Autonomous Robots and Multirobot Systems

In Conjunction with AAMAS 2011
Taipei, Taiwan, May 2, 2011

Editors
Gal Kaminka and Adriaan ter Mors

Organizing and Program Committee

Gal A. Kaminka	Bar Ilan University, Israel
Adriaan W. ter Mors	Delft University of Technology, The Netherlands
Alfons H. Salden	Almende BV, The Netherlands
Pierre Castagna	Université de Nantes, France
Petr Skobelev	Smart Solutions, Ltd., Russia
Ayanna Howard	Georgia Tech, USA
Emanuele Menegatti	Università degli Studi di Padova, Italy
Sonia Chernova	Worcester Polytechnic Institute, USA
Simon Parsons	Brooklyn College, USA
Pedro Lima	Instituto Superior Técnico, Portugal
Daniele Nardi	Sapienza Università di Roma, Italy
Erol Sahin	Middle East Technical University, Turkey
Elisabeth "Betsy" Sklar	Brooklyn College, USA
Paul Scerri	Carnegie Mellon University, USA
Lynne E. Parker	The University of Tennessee, USA
Naomi E. Leonard	Princeton University, USA
Laura Barbulescu	Carnegie Mellon University, USA
Lucia Pallottino	Università degli Studi di Pisa, Italy
Tatsushi Nishi	Osaka University, Japan
Rongxin Cui	National University of Singapore, Singapore
Koen Hindriks	Delft University of Technology, The Netherlands

AnySURF: Flexible Local Features Computation

Eran Sadeh-Or and Gal A. Kaminka
Computer Science Department
Bar Ilan University, Israel

Abstract. Many vision-based tasks for autonomous robotics are based on feature matching algorithm, finding point correspondences between two images. Unfortunately, existing algorithms for such tasks require significant computational resources and are designed under the assumption that they will run to completion and only then return a complete result. Since partial results—a subset of all features in the image—are often sufficient, we propose in this paper a computationally-flexible algorithm, where results monotonically increase in quality, given additional computation time. The proposed algorithm, coined AnySURF (Anytime SURF), is based on the SURF scale- and rotation-invariant interest point detector and descriptor. We achieve flexibility by re-designing several major steps, mainly the feature search process, allowing results with increasing quality to be accumulated.
We contrast different design choices for AnySURF and evaluate the use of AnySURF in a series of experiments. Results are promising, and show the potential for dynamic anytime performance, robust to the available computation time.

1 Introduction

The use of computer vision in autonomous robotics has been studied for decades. Recently, applications such as autonomous vision-based vehicle navigation [5], 3-D localization and mapping [17,6,3] and object recognition [16] have gained popularity due to the combination of increased processing power, new algorithms with real-time performance and the advancements in high quality, low-cost digital cameras. These factors enable autonomous robots to perform complex, real-time, tasks using visual sensors.

Such applications are often based on a local feature matching algorithm, finding point correspondences between two images. There are many different algorithms for feature matching, however in recent years there is a growing research on algorithms that use local invariant features (for a survey see [23,19]). These features are usually invariant to image scale and rotation and also robust to changes in illumination, noise and minor changes in viewpoint. In addition, these features are distinctive and easy to match against a large database of local features.

Unfortunately, existing algorithms for local feature matching [1,17,18] are designed under the assumption that they will run to completion and only then return a complete result. Many of these algorithms therefore require significant

computational resources to run in real-time. As we show in the experiments, this prohibits some of the algorithms from being used in current robotic platforms (where computation is limited). For instance, a Nao[1] humanoid robot computing the full set of features in an image of size 640×480 requires 2.4 seconds using a state-of-the-art implementation of the SURF algorithm [1,22].

Note, however, that for many robotics applications, even partial results—a subset of all features in the image—would have been sufficient (for example, to estimate the pose of the robot for obstacle detection). On the other hand, being able to invest computation time in getting higher-quality results is also important, e.g., in object recognition or in building accurate maps. Indeed, robots can benefit from computationally-flexible algorithms, where the computation time is traded for the accuracy requirements of the task. To do this, simply interrupting the algorithm when needed is not enough: We need to guarantee that the results of the algorithm would necessarily monotonically increase in quality, given additional computation time. This class of algorithms is called *Anytime* [26].

In this paper we present AnySURF, an anytime feature-matching algorithm, which can accumulate results iteratively, with monotonically increasing quality and minimal overhead. We achieve flexibility by re-designing several major steps in the SURF algorithm [1], mainly the feature search process and the order of interest point detection. We additionally discuss the design choices underlying AnySURF.

We evaluate the use of AnySURF in a series of experiments. We first demonstrate that non-anytime feature matching indeed suffers from significant computation time on limited platforms (including, in particular, the Nao humanoid robot). Then, we contrast different design choices for AnySURF, and analyze its performance profile under different image types. We also demonstrate the usability of AnySURF in computing approximate homography.

2 Related Work

Image matching using local features (or interest points) has been around for almost three decades – the term "interest point" was first introduced by Moravec in 1979 [20] who later proposed the use of a corner detector for stereo matching [21]. The Moravec detector was improved by Harris and Stephens [10]. Harris used it for efficient motion tracking and 3D structure from motion recovery [9]. The Harris corner detector has since been used widely for many other image matching tasks.

Although extensively used, the Harris corner detector is very sensitive to changes in image scale, so it does not provide a good basis for matching images of different sizes. There are many works that deal with representations that are stable under scale change, dating back to 1983 when Crowley and Parker [4] developed a representation that identified peaks and ridges in scale space and

[1] http://www.aldebaran-robotics.com

linked these into a tree structure which could be matched between images of different scales. More recently, Lindeberg conducted a comprehensive study of this problem [14] and suggested a systematic approach for feature detection with automatic scale selection [15].

A decade ago Lowe [16] introduced Scale Invariant Feature Transform (SIFT), which had a significant impact on the popularity of local features. SIFT descriptors are invariant to a substantial range of affine distortion, change in 3D viewpoint, noise and illumination differences. Robust matching is possible between different views of an object or scene, in the presence of clutter and occlusion. Since SIFT was published, several new algorithms inspired by SIFT have emerged, including PCA-SIFT [12], GLOH [18] and SURF [1].

SURF [1] is a state of the art algorithm for local invariant feature matching - a scale and rotation invariant interest point detector and descriptor. SURF is composed of three steps similar to SIFT, however it uses faster feature detection / extraction algorithms (approximation of the Hessian matrix and using the distribution of Haar-wavelet responses within the interest point neighborhood, relying on integral images to reduce computation time). SURF is faster to compute than SIFT, while allowing for comparable results.

SIFT, SURF and other algorithms are not anytime algorithms. Although several authors did accomplish complex real-time visual tasks such as Visual SLAM, using SIFT-like features [3] and correlation with reference templates [6], these implementations were tailored for a specific platform and are not computationally flexible. Therefore, they do not answer out research goals.

3 Methodology

The proposed AnySURF algorithm is based on SURF, which was selected over SIFT and SIFT-like algorithms since it is more suitable for an anytime implementation while also having an excellent quality/run-time ratio. It is more suitable for a flexible implementation because whereas SIFT begins with the computationally expensive operation of constructing several scale space representations (DoG, Difference of Gaussians approximation), SURF is based on a basic approximation of the Hessian matrix where an integral image is computed once for all scales and only the filter size changes when working on each scale. This means that in SURF there is very little overhead for working with specific scales or areas and this is very suitable for a flexible algorithm.

Since our algorithm is based on SURF, we start by explaining some basic concepts (integral image, Hessian matrix and Scale-Space), continue with a description of SURF (for a complete description of SURF see [1]) and finish with a detailed description of our proposed algorithm.

3.1 Basic concepts

An integral image is an image representation which is computed quickly from an input image and speeds up the calculation of any sized upright rectangular

area [24]. An integral image I_Σ of image I at point (x, y) is defined as the sum of pixel intensities of the rectangular region formed between the point and the origin. Formally it is defined as:

$$I_\Sigma(x, y) = \sum_{i=0}^{i \leq x} \sum_{j=0}^{j \leq y} I(x, y)$$

With I_Σ calculated, it only takes four additions to calculate the sum of the intensities over any upright, rectangular area, independent of its size. SURF makes excellent use of this property to perform fast convolutions of varying sizes of box filters at near constant time.

The Hessian matrix, H, is the matrix of partial derivatives of the function f :

$$H(f(x, y)) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}$$

The discriminant, determinant of this matrix, is:

$$det(H) = \frac{\partial^2 f}{\partial x^2} \frac{\partial^2 f}{\partial y^2} - \left(\frac{\partial^2 f}{\partial x \partial y} \right)^2$$

The discriminant can be used to find a local extremum of the function, via the second order derivative test. If the discriminant is positive then either both eigenvalues are positive or both are negative and so the point is classified as an extremum. Using this theory on images means that $f(x, y)$ is the image pixel intensity at $I(x, y)$. In order to calculate derivatives a convolution with an appropriate kernel can be used. SURF uses the Gaussian second order derivatives, approximated via box filters.

Scale-space is a continuous function which can be used to find extrema across scales [25]. Scale-space is usually divided into a number of octaves, with each octave containing a number of layers (response maps) covering a doubling of scale. SURF creates scale-space efficiently by applying kernels of increasing size to the original image, with the processing time being size invariant and without a need to sub-sample the image.

3.2 How does SURF works?

SURF [1] is composed of three steps: Detecting interest points, calculating descriptors and matching them (Alg. 1).

The first step, detection of interest points, starts with scale-space extrema detection: search over all scales and image locations is performed, using an approximation of the Hessian matrix to identify potential interest points that are invariant to scale and rotation. Calculation of the Hessian approximation relies on an integral image to reduce computation time. Interest points are first thresholded so that all values below a predetermined threshold are removed, then a

Algorithm 1 Generic SURF
(Input: image; Output: list of matched descriptors)

```
0.  Construct integral image
1.1 Over all octaves (Fine-to-Coarse)
1.2   Pre-calculate discriminants
1.3   Over inner octave layers
1.4     Over all pixels
1.5       Find interest point
2.1 Over all interest points
2.2   Calculate descriptor
3.1 Over all descriptors
3.2   Match descriptor
3.3   Add matched descriptor to list
4.  Return list of matched descriptors
```

non-maximal suppression is performed to find candidate points (each pixel is compared to its 26 neighbours, comprised of the 8 points in the native scale and 9 in each of the scales above and below) and finally they are localized in both scale and space by fitting a 3D quadratic.

The second step, calculation of the keypoint descriptors, is based on a distribution of Haar-wavelet responses within the interest point neighborhood, again relying on integral images for speed. The SURF descriptor describes how the pixel intensities are distributed within a scale dependent neighbourhood around each detected interest point. It is calculated by first assigning a repeatable orientation via Haar wavelet responses weighted with a Gaussian centered at the interest point, then a square oriented window is constructed around the interest point, divided into 4×4 regular sub-regions. For each sub-region 4 Haar wavelets responses are summed up (d_x, d_y, $|d_x|$, $|d_y|$), so a vector of length $4 \times 4 \times 4 = 64$ is produced.

The third step, matching different descriptors, is done via the Euclidean distance of their feature vectors. A fast nearest-neighbor algorithm is used that can perform this computation rapidly against large databases. SURF uses the sign of the Laplacian (the trace of the Hessian matrix) to distinguish bright features on dark background from the reverse situation. Since SURF's descriptor uses 64 dimensions, time for feature computation and matching is reduced.

3.3 Making SURF computationally flexible

In order to make SURF computationally flexible, several important design decision has to be made. These are: accumulating results iteratively, using a suitable search strategy and calculating the Hessian in-place. An in-depth explanation of these design decision follows. The impact of these decisions is presented in Section 4.

Guaranteeing Monotonically-Improving Descriptor List The first step in making an anytime version of SURF is trivial: Accumulate results iteratively. SURF divides the work to several large consecutive steps (get all interest points from all scales, compute descriptors for all interest points, match all descriptors against database - Alg. 1, steps 1–3) and so if the final stage is not reached – there might be no useful results. Contrary to this batch approach, we propose an iterative approach, where results are accumulated during the execution of the algorithm and are returned when the algorithm is interrupted.

This can be achieved by computing a full result, including a descriptor, in each iteration. The new descriptor can immediately be used to match against a database. This change is trivial yet vital as it guarantees good anytime functionality: usable results are generated such that the number of results is monotonically increasing.

Generating Descriptors Faster Now that we can guarantee that the list of matched descriptors will be monotonically-increasing in length, we can explore design choices that can make sure quality descriptors are generated faster. Below we discuss two such design choices.

Search strategy Detection of interest points (Alg. 1, step 1) is done by scanning the entire image in multiple octaves. This search usually starts with the smallest kernel and continues applying kernels of increasing size to the image. Since our flexible algorithm accumulates results iteratively, we have an opportunity to select an ordering on the search of octaves for interest points (Alg. 1, step 1.1), thereby allowing detection of more promising features earlier. Note that this search strategy need not be hard-coded, but can be changed according to the image or task at hand.

We considered two types of general search strategies: Coarse-to-Fine and Fine-to-Coarse. Coarse-to-Fine means we start with the largest filter size and continue to use smaller filter sizes so that we find larger features first and smaller ones later, while Fine-to-Coarse means the exact opposite. Note that if the algorithm runs to completion the search order does not matter and exactly the same features are found. Additional search strategies are also possible: order of going over inner octave layers (Alg. 1, step 1.3), order of going over pixels (Alg. 1, step 1.4), however we did not consider them here.

Selecting an appropriate search strategy according to the image type (e.g., blurry image) can maximize the number of features detected during the early phase of the search. However, sometimes the number of features is not what we prefer to optimize. For example, some vision tasks work better when the features have a good spatial distribution over the image (e.g., homography calculation [11]) or when coarse features are first matched and only then fine features are searched for in a limited area to complete the match (e.g., object recognition [16]). In such cases it might be preferable to use Coarse-to-Fine search, even if the initial number of features is smaller when compared to the Fine-to-Coarse strategy.

Calculating the Hessian discriminants in-place All SURF implementations we inspected (Pan-o-matic [22], OpenSURF [7], OpenCV [2]) pre-calculate the determinant of Hessian (discriminant) for each octave, over the entire image (Alg. 1, step 1.2). This step has high initial computational cost, however once calculated, results are faster to compute so the total running time is lower. Since we assume the algorithm might not run till completion, it might be preferable to sacrifice some of the running-time in order to get initial results sooner.

Memory consumption by the pre-calculated arrays is another issue to consider. Pre-calculating the determinant of Hessian requires several 2D arrays to be kept in memory. The SURF implementations we inspected (see above) use arrays the size of a full image to simplify coding (smaller arrays can however be used). So we have number of layers×image_width×image_height, which means multiple arrays each one the size of a full image are saved in memory. For large images or platforms with little memory available, this can be quite problematic. Obviously, when pre-calculation is not used and the determinant of Hessian is calculated in-place, there is no need to save multiple arrays in memory.

3.4 AnySURF - Anytime SURF

The following algorithm (Alg. 2), coined AnySURF (Anytime SURF), is a computationally flexible SURF algorithm. Results are accumulated iteratively, with a descriptor computed in each iteration. Octaves are searched in Coarse-to-Fine order and the determinant of Hessian is calculated in-place. We believe these design choices are appropriate for a generic Anytime SURF algorithm and an analysis of the Anytime performance profile is performed in Section 4.

A possible variant of AnySURF is to use pre-calculation. Compared to a batch approach such as panosurf, this alternative is more suitable to anytime since results are produced earlier yet the total computation time is exactly the same. Compared to the AnySURF without pre-calculation, the total computation time of this variant is lower yet first results are generated much later since pre-calculation has a high initial computational cost (see Figure 1).

4 Results

A flexible algorithm is required only when the non-flexible algorithm is slow and when partial / low accuracy results are useful. In this section we will show that both criteria are met in SURF. In addition, we analyze the design decisions explained in Section 3.3 and present an example of using AnySURF to approximate homography between 2 images.

To demonstrate that SURF is not fast enough for real-time full image feature search on current robotic platforms which have limited computational power, Table 1 shows computation time on multiple platforms for the same image in different sizes (QVGA: 320×240, VGA: 640×480, 3MP: 2048×1536). Evaluation was done using Pan-o-matic open-source SURF implementation [22] with

Algorithm 2 AnySURF
(Input: image; Output: list of matched descriptors)

```
0.  Construct integral image
1.  While not interrupted
2.1   Over all octaves (Coarse-to-Fine)
2.2     Over inner octave layers
2.3       Over all pixels
2.4         Find interest point
2.5         Calculate descriptor
2.6         Match descriptor
2.7         Add matched descriptor to list
3.  Return list of matched descriptors
```

Table 1. SURF detector-descriptor computation time (ms) on different image sizes and platforms

Platform	QVGA	VGA	3MP
Desktop PC (Intel Q9400 2.66GHz)	27	103	1021
Mini-ITX (Intel T7200 2.0GHz)	74	249	1599
Nao Robot (x86 AMD GEODE 500MHz)	560	2425	26367
Nokia N900 (ARM Cortex-A8 600MHz)	938	3656	442512

default parameters, which produces very similar results compared to the published SURF binary [8] to which source code is not available.

From Table 1 it is clear that in order to run real-time full-image feature search with SURF we need to work on a small resolution image coupled with a powerful platform. In addition, in this test the CPU and memory were devoted entirely to the SURF process, while in robotic applications additional non-vision tasks might also require processing time and memory usage (localization, mapping, motion generation, behavior selection, etc.).

Table 2 shows where the processing time is spent across the different major steps. The Intel Q9400 platform was selected for this test, to eliminate as many bottlenecks as possible and allow the optimal behavior of the algorithm show.

Table 2. Analysis of SURF detector-descriptor computation time (ms), on Intel Q9400 2.66 GHz

Image size	integral image	keypoints	descriptors
QVGA	1 (3.7%)	19 (70.4%)	7 (25.9%)
VGA	3 (2.9%)	80 (77.7%)	20 (19.4%)
3MP	32 (3.1%)	910 (89.1%)	79 (7.8%)

As can be seen in Table 2, calculation of the integral image is minor (~3.2%) and detection of keypoints takes most of the time (~79%). In the context of a flexible algorithm, since detecting keypoints takes most of the time, it seems beneficial to calculate the descriptor immediately upon keypoint detection, thus significantly shortening the time till partial results are available. We now turn to analyzing the impact of the various design choices of AnySURF (presented in Section 3.3).

The image database used in all following figures is a standard evaluation set, provided by Mikolajczyk [18]. It contains 48 images across 8 different scenes. All images are of medium resolution (approximately 800×640 pixels) and are either of planar scenes or the camera position is fixed during acquisition, so that in all cases the images are related by homographies (plane projective transformations). The scenes contain different imaging conditions: viewpoint changes, scale changes, image blur, JPEG compression and illumination changes.

Figure 1 shows the averaged rate of acquiring descriptors (%) as a function of run-time (%). Three alternatives are considered: First, a SURF implementation called Pan-o-matic [22] (henceforth will be referred to as panosurf). This implementation first detects all keypoints at all scales and only then calculates descriptors (as in Alg. 1). In addition, the determinant of Hessian is pre-calculated. Next, we tested two variants of the AnySURF algorithm (Alg. 2), where descriptors are computed immediately upon keypoint detection. In the first variant pre-calculation of the determinant of Hessian is used and in the other it is calculated in-place.

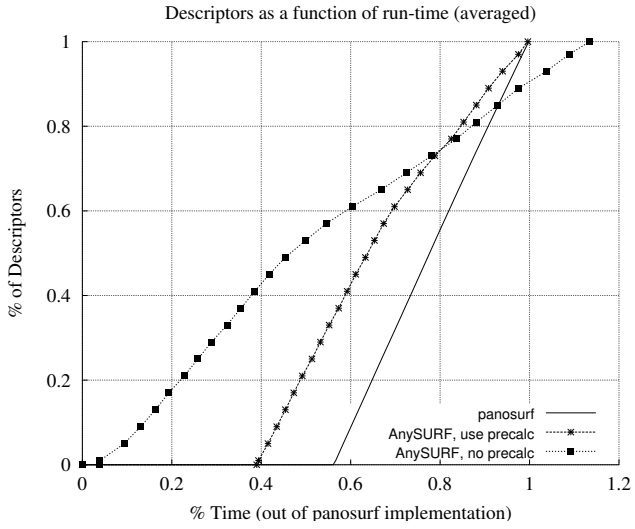


Fig. 1. The descriptors (%) vs. time (%) graph for different algorithms. Data is averaged across all (48) database images. Time (%) is compared to panosurf time (therefore can be > 1.0).

Figure 1 allows us to inspect the impact of calculating descriptors immediately upon keypoint detection and also the effect of pre-calculation. First, let us consider AnySURF with pre-calculation: calculating descriptors immediately is beneficial compared to the original panosurf approach since it does not adversely affect the total computation time while allowing results to start accumulating earlier (after 39% of time passed instead of 57% as in panosurf). Now, let us consider AnySURF without pre-calculation: although the algorithm does take longer to complete (13.5% more on average), we start getting results almost immediately (after 4% of time passed), with a near-linear acquire rate. Note that pre-calculation is only useful when all descriptors are needed or when it can be assumed that the algorithm will run to near completion (AnySURF with pre-calculation supersedes the no pre-calculation version after 80% of the time passed).

Next, let us inspect the impact of the search strategy. As explained in Section 3.3, since AnySURF accumulates results iteratively, we can select an ordering on the search for interest points. The following two figures are of specific images (1st image of "bricks" sequence, 4th image of "bikes" sequence), showing the number of descriptors as a function of run-time. All four combinations are shown (with/without pre-calculation, Coarse-to-Fine/Fine-to-Coarse search strategy), compared to the baseline panosurf.

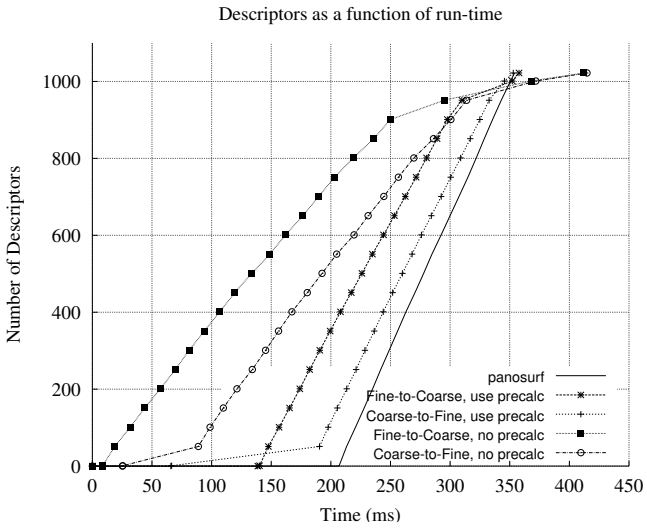


Fig. 2. 1st image of bricks sequence (see Figure 4), panosurf displayed as a baseline, AnySURF with Coarse-to-Fine and Fine-to-Coarse search strategies displayed with and without pre-calculation

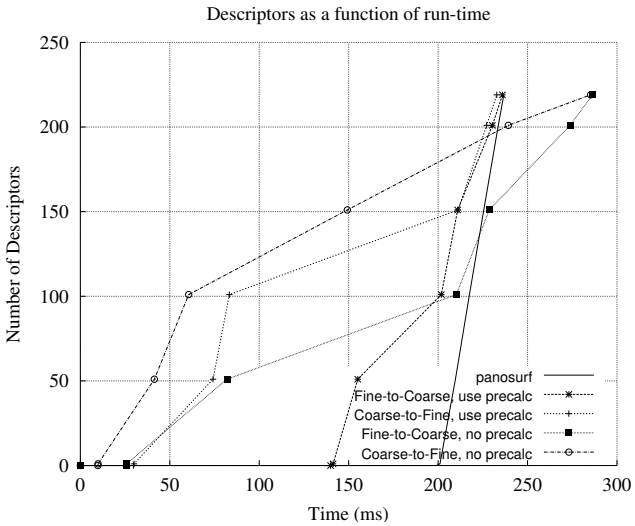


Fig. 3. 4th image of bikes sequence (see Figure 5), panosurf displayed as a baseline, AnySURF with Coarse-to-Fine and Fine-to-Coarse search strategies displayed with and without pre-calculation

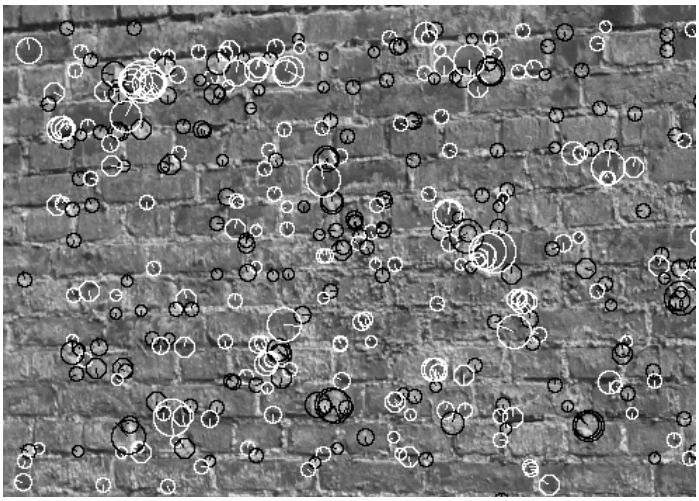


Fig. 4. A quarter of the 1st image of bricks sequence (see Figure 2)

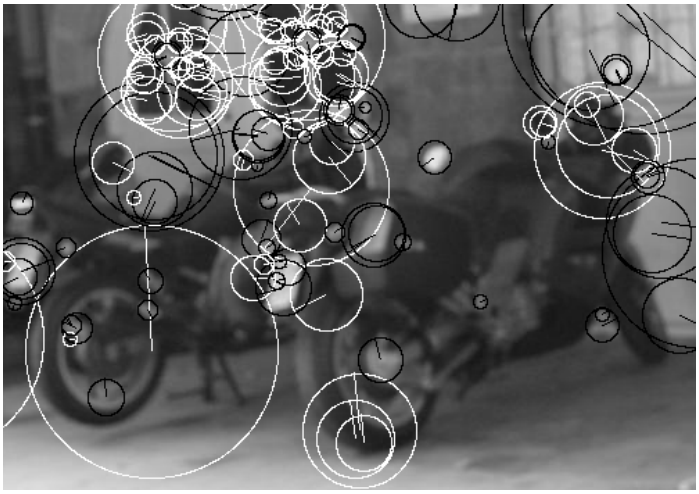


Fig. 5. A quarter of the 4th image of bikes sequence (see Figure 3)

In Figure 2, the Fine-to-Coarse search strategy produces results much faster than Coarse-to-Fine, while in Figure 3 the opposite is true. Note that in both figures results both start sooner and accumulate faster and the difference in number of descriptors is significant, up to an order of magnitude (e.g., after ~30ms). This means that appropriate selection of the search strategy is vital for an efficient Anytime performance.

As for the use of pre-calculation, it seems that our previous conclusion holds and for an anytime algorithm an in-place calculation is preferred. However, in Figure 3 the pre-calculated (Coarse-to-Fine) version supersedes the in-place (Fine-to-Coarse) version. This only stresses that selecting the appropriate search strategy is very important indeed: when selecting the correct search strategy, the no pre-calculation (Coarse-to-Fine) version triumphs again (at least until most of the descriptors are found, as explained earlier).

After witnessing a major difference in the above specific images according to the chosen search strategy, it is interesting how the Fine-to-Coarse vs. Coarse-to-Fine search strategies perform in our full image database, across the different scenes. To test this, Figure 6 shows the effect of search strategy in different scenes on the first 50ms of AnySURF (without pre-calculation).

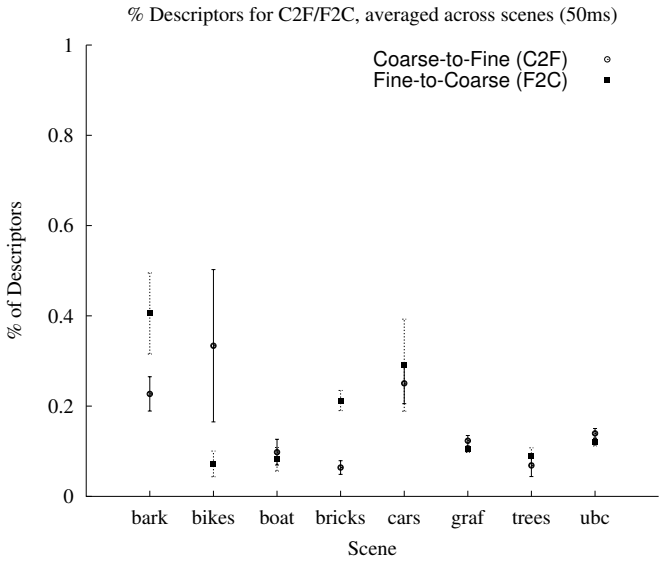


Fig. 6. The descriptors (%) for Coarse-to-Fine and Fine-to-Coarse tested on different scenes (first 50ms of AnySURF). The markers represent the mean for each scene, the error bars are two standard deviation units in length.

Figure 6 demonstrates that it is beneficial to select the appropriate searching strategy according to the type of image at hand if the number of descriptors is to be optimized. The image sequences "bark" and "bricks" are clearly more suited for Fine-to-Coarse search whereas "bikes" is more suited for Coarse-to-Fine search. The reason behind this is that "bikes" is a sequence of blurred images (so there are less fine features and processing time is wasted on searching for them, see Figure 5) while "bark" and "bricks" contain images with many fine features and few coarse ones (see Figure 4). It is also interesting to note that between 5% to 40% of all descriptors can be acquired within 50ms (however, higher percentages are usually for images with a lower total number of descriptors and a lower total computation time). The average number of descriptors acquired after 50ms on our image database is 119 for Coarse-to-Fine and 138 for Fine-to-Coarse.

A higher number of descriptors is not necessarily better. For example, coarse features are larger, fewer and usually more spread over the image so they might suit some tasks better than fine features. One such task is homography estimation [11]. Figure 7 shows the time passed till the homography between the 1st and 2nd images in each scene could be estimated (to within an order of magnitude from the optimal value). Homography was calculated via the RANSAC approach [11,13].

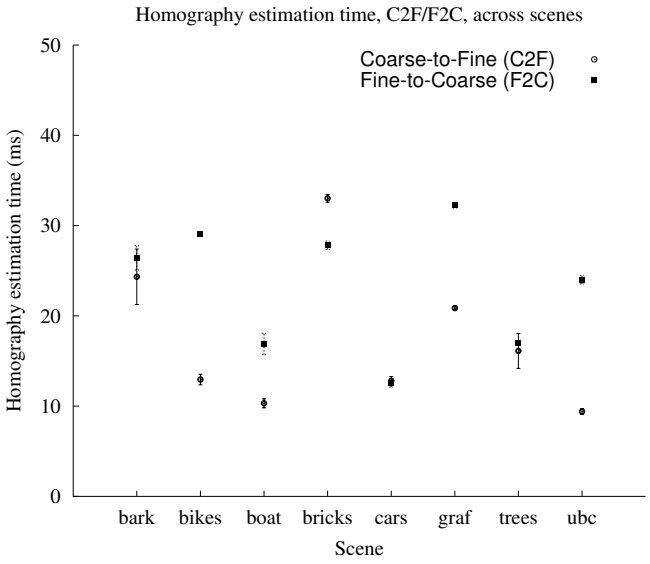


Fig. 7. Homography approximation time for Coarse-to-Fine and Fine-to-Coarse, tested across scenes. In each scene image 1 was processed fully. Image 2 was processed till the homography could be estimated (to within an order of magnitude from optimal value). The markers represent the mean for each sequence, the error bars are two standard deviation units in length.

Figure 7 demonstrates that coarse features enable a quicker homography estimation compared to fine features. The Coarse-to-Fine search strategy took equal or less time to estimate the homography in most scenes (7/8) and more time in just one scene (the bricks scene, which contains very few coarse features and so a lot of time is wasted searching for coarse features, see Figure 4). The results for the bark, cars and trees scenes do not differ significantly, while results for others do (two-tailed t-test, p=0.01). Also note that the homography could be estimated within a very short time (~20 ms).

Finally, we go back to evaluating the use of flexible local feature matching on the Nao robot platform. Prior to AnySURF, estimating a homography between two images on this platform took on average 4 seconds (averaged across all images in database, similar to Figure 7). This processing time was spent not on estimating the homography itself, but on computing all descriptors in the image. However, for estimating a homography a subset of the results suffice, so AnySURF can be used. Using AnySURF, this task is completed within 0.33 seconds, faster by an order of magnitude. Note that this homography can actually assist in computing the remaining descriptors faster, since we can now estimate their location.

5 Conclusion

We presented and analyzed AnySURF, a SURF-based anytime local feature matching algorithm, which can trade quality of results for computation time: It guarantees that the number of matched descriptors monotonically increases with computation. For robotics applications that can work with a subset of descriptors, this allows for much faster response times.

We discuss and carefully evaluate design choices in the feature search process, using several computational platforms. We demonstrate that changing the feature search order can significantly impact the rate at which descriptors are generated. Also, we show that surprisingly, avoiding pre-calculation steps that are intended to optimize the search process, leads to generating results at a faster rate. Future work will focus on the problem of dynamically managing AnySURF within the context of a real-time complex application (vision-based autonomous navigation system).

References

1. H. Bay, T. Tuytelaars, and L. V. Gool. SURF: speeded up robust features. In *Computer Vision – ECCV 2006*, pages 404–417. 2006.
2. G. Bradski. The OpenCV library. *Dr. Dobb's Journal of Software Tools*, 2000.
3. D. Chekhlov, M. Pupilli, W. Mayol-cuevas, and A. Calway. Real-time and robust monocular SLAM using predictive multi-resolution descriptors. *In 2nd International Symposium on Visual Computing*, 2006.
4. J. L. Crowley and A. C. Parker. *Representation for Shape based on Peaks and Ridges in the Difference of Low-pass Transform*. 1983.
5. DARPA. DARPA grand challenge. http://www.darpa.mil/grandchallenge/index.asp, 2007.
6. A. Davison, I. Reid, N. Molton, and O. Stasse. MonoSLAM: Real-Time single camera SLAM. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(6):1052–1067, 2007.
7. C. Evans. Notes on the OpenSURF library. Technical report, University of Bristol, Jan. 2009.
8. D. Gossow, D. Paulus, and P. Decker. An evaluation of open source SURF implementations. In *RoboCup 2010: Robot Soccer World Cup XIV*. 2010.
9. C. Harris. Geometry from visual motion. In *Active vision*, pages 263–284. MIT Press, 1993.
10. C. Harris and M. Stephens. A combined corner and edge detector. pages 151, 147, Manchester, 1988.
11. R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2 edition, Apr. 2004.
12. Y. Ke and R. Sukthankar. PCA-SIFT: a more distinctive representation for local image descriptors. *null*, 2:506—513, 2004.
13. P. D. Kovesi. MATLAB and Octave functions for computer vision and image processing. http://www.csse.uwa.edu.au/~pk/Research/MatlabFns, 2000.
14. T. Lindeberg. *Scale-Space Theory in Computer Vision*. Kluwer Academic Publishers, 1994.

15. T. Lindeberg. Feature detection with automatic scale selection. *Int. J. Comput. Vision*, 30(2):79–116, 1998.
16. D. G. Lowe. Object recognition from local Scale-Invariant features. In *Proceedings of the International Conference on Computer Vision - Volume 2*, page 1150. IEEE Computer Society, 1999.
17. D. G. Lowe. Distinctive image features from Scale-Invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, Nov. 2004.
18. K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(10):1615–1630, 2005.
19. K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. V. Gool. A comparison of affine region detectors. *Int. J. Comput. Vision*, 65(1-2):43–72, 2005.
20. H. P. Moravec. *Visual Mapping by a Robot Rover*. 1979.
21. H. P. Moravec. Rover visual obstacle avoidance. pages 785–790, Vancouver, British Columbia, Aug. 1981.
22. A. Orlinski. Pan-o-matic - automatic control point creator for hugin. http://aorlinsk2.free.fr/panomatic/.
23. T. Tuytelaars and K. Mikolajczyk. Local invariant feature detectors: a survey. *Found. Trends. Comput. Graph. Vis.*, 3(3):177–280, 2008.
24. P. Viola and M. Jones. Computer vision and pattern recognition, 2001. CVPR 2001. proceedings of the 2001 IEEE computer society conference on. volume 1, pages I–511–I–518 vol.1, 2001.
25. A. Witkin. Scale-space filtering: A new approach to multi-scale description. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP '84.*, volume 9, pages 150–153, 1984.
26. S. Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996.

Decoupling MDPs Step by Step from a POMDP

Chyon Hae Kim[1], Hiroshi Tsujino[1], Hiroyuki Nakahara[2]

[1] Honda Research Institutes Japan Co.,Ltd.
{tenkai,tsujino}@jp.honda-ri.com
[2] Integrated Theoretical Neuroscience RIKEN Brain Science Institute
hn@brain.riken.jp

Abstract. This paper addresses the problem, how an artificial agent decouples a partially observable Markov decision process (POMDP) to several Markov decision processes (MDPs) according to the dimensional hierarchy of the MDPs. We propose multi-layered reinforcement learning (MLRL) that selectively uses each layer to learn each MDP to reduce the learning cost. The MLRL separately learned two MDPs step by step in a simulated capture task. Also, the MLRL learned faster than SARSAs in the capture task and a simulated guiding task.

1 Introduction

This paper addresses the problem, how an agent learns for Markov decision processes (MDPs) that are inside a partially observable Markov decision process (POMDP).

Traditionally, many kinds of reinforcement learning (RL) systems for MDP and POMDP have been researched. MDP is a special case of POMDP, and there are many techniques to solve MDPs (e.g. SARSA and Q-learning are very famous formulations) [C89,C92,RA00]. On the other hand, there is no technique to solve general POMDPs, since POMDPs are often computationally intractable [LM98,Ke00].

To adapt to general POMDPs, an agent requires these capabilities.

1. Utilization for temporal sequence
2. Decision making that is based on statistical state models.

However, utilization for temporal sequence increases the complexity of learning, and statistical state models require a lot of observation data. In many practical scene, these problems result in large computational cost.

To relax the cost, we discuss how to find learnable MDPs inside a POMDP. If an RL system is able to decouple the MDPs from a POMDP, the system does not need to learn the MDPs using POMDP frameworks as traditional RL systems do. After the MDPs are learned, the remained POMDP will not require large cost to be learned. We propose a multi-layered RL formulation to decouple the MDPs step by step.

The structure of this paper is as follows: In Section 2, we formulate the proposed RL. In Section 3, we describe the experimental systems for a capture task and a guiding task. In Section 4, we show the results of the experiments. In Section 5, we mention our considerations. In Section 6, we conclude this paper.

2 Algorithm

2.1 Definition of a POMDP

POMDP frameworks are defined by a tuple (S, A, O, T, Ω, R), where S is a set of states, A is a set of actions, O is a set of observations, T is a set of conditional transition probabilities, Ω is a set of conditional observation probabilities, $R : S \times A \to R$ is the reward function. An agent that reaches the state $s' \in S$ from the state $s \in S$ using an action $a \in A$ observes $o \in O$ with probability $\Omega(o|s', a)$. In general, observation o is composed of partial observations y_is $o := \{y_1, y_2, \cdots, y_n\}$.

2.2 The least combination of elements to describe transition

In POMDP framework, we assume that state transitions follow MDP. The transition probability T is defined as follows:

$$T := T(s'|s, a) \tag{1}$$

Usually, the state transition is not observed directly, since observation o has smaller number of elements than $s = \{y_1, y_2, \cdots, y_n, \hat{y}_1, \hat{y}_2, \cdots, \hat{y}_n\}$. However, in some cases, s is redundant to describe transition probability T. In these cases, the state transition is observable. For example, in the case that $T(s'|s, a) = T(s'|o, a)$, elements $\hat{y}_1, \hat{y}_2, \cdots, \hat{y}_n$ are redundant, and o has sufficient kinds of elements to observe the state transition. Generally, each transition T has the least combination C of elements to be described. If the combination C is inside the elements of observation o while a transition, the transition is inside an MDP (sub-task) that is inside a POMDP (whole task). So, there is possibility that RL system adapts to the MDPs while learning a POMDP problem. This kind of adaptation will help the system to solve the POMDP, since each MDP is tractable using traditional RL framework. However, we need to consider the hierarchy of MDPs to divide MDPs from a POMDP.

2.3 Approach

To divide a POMDP into several MDPs, which are computationally tractable, and the a remained POMDP, we need to consider the hierarchy of observable information of an agent. We propose a step by step decoupling method that uses an RL system, which equips with multi-layers, based on the consideration. Each layer of the multi-layers is related to one of the hierarchy of MDPs.

In many cases, a POMDP includes many MDPs Ms as sub-tasks of the POMDP. These MDPs are categorized by the combination C of partial observations y_is. We define an MDP that requires only C to be learned as $M_C = M_{\{\cdots\}}$. Among them, the most simple MDPs are $M_{\{y_i\}}$s that include only one element y_i. An RL system RL_1 that learns the POMDP while assuming the POMDP as $M_{\{y_i\}}$ may converge to local optima. However, another RL system RL_2 that

learns the same POMDP while assuming the POMDP as $M_{\{y_i,y_j\}}$ may converge to a better solution than RL_1 (or the best solution of the POMDP. See [JS98] for the case when SARSA converges to an optimal policy of POMDP problems). Therefore, to find MDPs inside a POMDP, an RL system should select the best combination C of partial observations y_is or the best M for each sub-task inside a POMDP. We formulated an RL system for the problem, and examined the system to select Cs.

In our approach, an artificial agent decouples the POMDP to a simple MDP (e.g. $M_{\{y_i\}}$) and the remained POMDP ($\bar{M}_{\{y_i\}}$). To realize the first decoupling, we use the first layer that corresponds to $M_{\{y_i\}}$. This layer is able to learn only for $M_{\{y_i\}}$. To proceed the decoupling, the agent decouples $\bar{M}_{\{y_i\}}$ into an MDP of second level $M_{\{y_i,y_j\}}$ and the remained POMDP ($\bar{M}_{\{y_i,y_j\}}$) again. This layer is also able to learn only for $M_{\{y_i,y_j\}}$. This way, the proposed RL system decouples a POMDP into multi-layered MDPs.

2.4 Formulation

To explain the formulation of the proposed method, we show the formulation of a two layered RL at first, and extend that for a multi-layered RL.

Two Layered RL The two layered RL is composed of two RLs. The first layer has an RL, RL_1, that consists of SARSA or Q-learning. While an agent observes os according to its actions as, and receives rewards r. RL_1 learns the relationship between y_is, as, rs using its partial observations y_is. This means that RL_1 learns Q values for each state action pair $Q_1(y_i, a)$. If a POMDP has a sub-problem, MDP $M_{\{y_i\}}$, inside that, this RL system that has observation y_i is a good solution for the sub-problem.

As in the first layer, the second layer has an RL system, RL_2, that also consists of SARSA or Q-learning. RL_2 learns Q values for each state action pair $Q_2(y_i, y_j, a)$.

If an agent utilizes these two RL systems appropriately while learning a POMDP that includes $M_{\{y_i\}}$ and $M_{\{y_i,y_j\}}$ as sub-tasks, the agent will reduce the learning cost, since RL_1 and RL_2 are good solutions for sub-problems $M_{\{y_i\}}$, $M_{\{y_i,y_j\}}$.

We established the way to combine these two RL systems into an RL system. To utilize these systems, RL_1 and RL_2, we need these considerations.

1. How the whole RL system calculates the Q value from Q_1 and Q_2.
2. How the whole RL system learns Q values.

For the first problem, we use the following linear coupling formulation

$$Q = \sum_{k=1}^{2} w_k Q_k + w_{rem} Q_{rem} \tag{2}$$

where w_k is a weight for Q_k, w_{rest} is a weight for Q_{rest}, Q_{rest} is a Q value for an RL system that learns for a POMDP $\bar{M}_{\{y_i,y_j\}}$ that is the rest of $M_{\{y_i\}}$ and $M_{\{y_i,y_j\}}$. When we do not use an RL system for the remained POMDP, $w_{rem}Q_{rem}$ is 0. In general, an agent inside a POMDP has to consider sub-tasks of hierarchical MDPs. When an agent takes a state $S(y_i, y_j)$ in a MDP $M_{\{y_i,y_j\}}$, the agent takes a state $S(y_i)$ in another MDP $M_{\{y_i\}}$ simultaneously. So, transitions in these MDPs proceed at the same time. Therefore, the agent needs to consider the weight w of these tasks inside $M_{\{y_i,y_j\}}$ and $M_{\{y_i\}}$ to sum up the benefit of the sub-tasks.

For the second problem, if we assume the use of finite states for each layer, we are able to derive the learning formulations of the whole system as follows. When RL_1 observes y_i, RL_2 observes (y_i, y_j). We define the Q value of RL_1 as $Q_1 := Q_1(y_i)$ and define the Q value of RL_2 as $Q_2 := Q_2(y_i, y_j)$. When the RL system is in a sub-task $M_{\{y_i,y_j\}}$, we formulate the error as follows:

$$\begin{aligned} E = \ & \frac{1}{2}\sum_{y_i,y_j}\sum_{y'_i,y'_j} p^{\pi}_{y_i,y_j} P^{\pi}_{y_i,y_j,y'_i,y'_j}(r_{y_i,y_j,y'_i,y'_j} \\ & +\gamma Q'(y'_i, y'_j, \pi) - w_1 Q_1(y_i) - w_2 Q_2(y_i, y_j))^2 \end{aligned} \tag{3}$$

where $p^{\pi}_{y_i,y_j}$ is the probability where RL_2 observes (y_i, y_j), $P^{\pi}_{y_i,y_j,y'_i,y'_j}$ is the transition probability when an agent transits from the observation (y_i, y_j) to another observation (y'_i, y'_j) using an action selection policy π, r_{y_i,y_j,y'_i,y'_j} is the given reward in the transition, and Q' is the Q value of the selected action, which is based on a policy π and selected based on the next observation (y'_i, y'_j) when we apply SARSA type update.

We deduce update functions of each (y_i, y_j) from the error E using the steepest descent method, based on the two assumptions that the terms $r_{y_i,y_j,y'_i,y'_j} + \gamma Q'(y'_i, y'_j, a')$ to be the output targets of the learning system that are independent of y_i and y_j, and the symbols $p^{\pi}_{y_i,y_j}$ and $P^{\pi}_{y_i,y_j,y'_i,y'_j}$ are independent of Q_ns. Steepest descent method derives a formulation that is consistent to traditional RL theory as we mention later.

$$\begin{aligned} & \Delta Q_1(y_m) = -\alpha\frac{\partial E}{\partial Q_1(y_m)} \approx \alpha w_1 \sum_{y_j}\sum_{y'_i,y'_j} p^{\pi}_{y_m,y_j} P^{\pi}_{y_m,y_j,y'_i,y'_j} \\ & (r_{y_m,y_j,y'_i,y'_j}\gamma Q'(y'_i, y'_j, \pi) - w_1 Q_1(y_m) - w_2 Q_2(y_m, y_j)) \end{aligned} \tag{4}$$

$$\begin{aligned} & \Delta Q_2(y_m, y_n) = -\alpha\frac{\partial E}{\partial Q_2(y_m, y_n)} \approx \alpha w_2 \sum_{y'_i,y'_j} p^{\pi}_{y_m,y_n} P^{\pi}_{y_m,y_n,y'_i,y'_j} \\ & (r_{y_m,y_n,y'_i,y'_j} + \gamma Q'(y'_i, y'_j, \pi) - w_1 Q_1(y_m) - w_2 Q_2(y_m, y_n)) \end{aligned} \tag{5}$$

We show the obtained online update functions which are derived from these update functions.

$$\Delta Q_1 = \alpha_1(r + \gamma Q' - w_1 Q_1 - w_2 Q_2) \tag{6}$$
$$\Delta Q_2 = \alpha_2(r + \gamma Q' - w_1 Q_1 - w_2 Q_2) \tag{7}$$

When we change the formulations as follows, the formulations show that RL_1 and RL_2 learn separately for $TD\_Error - w_n Q_n$ as SARSAs. This means that each RL, RL_1 or RL_2, learns the rest of TD error that was learned by another RL.

$$\Delta Q_1 = \alpha_1((r + \gamma Q' - w_1 Q_1) - w_2 Q_2) \tag{8}$$
$$\Delta Q_2 = \alpha_2((r + \gamma Q' - w_2 Q_2) - w_1 Q_1) \tag{9}$$

In a special case when $w_1 = 1$ and $w_2 = 0$, This formulation is the

$$Q = Q_1 \tag{10}$$
$$\Delta Q_1 = \alpha_1(r + \gamma Q' - Q_1) \tag{11}$$

This result is consistent to traditional theory for SARSAs that consider single MDP. When the weight for the $M_{\{y_i,y_j\}}$, w_2, is 0, this multi-layered system negledts $M_{\{y_i,y_j\}}$. In this case, this system does not consider the POMDP as mixture of $M_{\{y_i\}}$ and $M_{\{y_i,y_j\}}$, but $M_{\{y_i\}}$. So, above mentioned formulation, which is consistent to SARSA, is reasonable to calculate reward estimation.

Multi-Layered RL We show the formulations of a multi-layered RL that was deduced the same as the two layered RL.

$$Q = \sum_{k=0}^{n} w_k Q_k \tag{12}$$
$$\Delta Q_i = \alpha_i((r + \gamma Q' - w_i Q_i) - \Sigma_{k \neq i} w_k Q_k) \tag{13}$$

3 Experimental Systems

We conducted two experiments, a capture experiment and a guiding experiment, to validate the proposed RL system.

3.1 Capture Experiment

We established a PC simulation. In this simulation, a learner (abstract robot) having a radius R captured the center mass of an agent in a half circle (Fig.1).

Robot The robot was equipped with the two layered proposed RL system. The robot observes the vertical relative position of the agent x, the horizontal relative position of the agent y, and the horizontal absolute velocity of the agent $\dot{y}$. We set (x, y) for the first layer's observation. The observation is related to an MDP $M_{\{x,y\}}$. We set $(x, y, \dot{y})$ for the second layer's observation. The observation is related to another MDP $M_{\{x,y,\dot{y}\}}$. We divided the input space into 100×4 (horizontal direction $\times$ vertical direction). For the vertical direction, the robot approaches an agent with a constant velocity v. For the horizontal direction, the robot selects its action among three actions: moving the half circle to the left by the length of Δ, moving the half circle to the right by the length of Δ, and remaining current position. The robot gets a reward value, 1, when it captures the agent. If the robot fails that, the robot gets a punishment value, -1.

Settings for the Agent We set two rules for the movement of the agent to make a POMDP environment for the robot. The first rule is that the agent moves left and right randomly using a normal random number $\phi(u, \sigma^2)$. If the robot learns ϕ, the robot improves capture performance. The second rule is that the agent changes the sign of u periodically. We noticed that the robot was not able to observe the sign of u, which decides the state of the agent. Therefore, the sign of u makes a POMDP environment for the robot.

We are able to control the difficulty to guess the rule of the agent by changing the parameter σ^2. This capturing task is easy when σ^2 is small, but is difficult when σ^2 is large. We show the flow of agent's motion as follows:

1. Decide the parameters of normal random numbers u and σ.
2. Add normal random number $\phi(u, \sigma^2)$ to the horizontal position of the agent y.
3. Invert the sign of u.
4. Continue Steps 2 and 3.

Parameters We set the initial position of the agent to be just above the position of the center of the robot. The robot continued to learn for one set (= 20,000 trials) using the same parameters $\Delta = 0.2$, $u = 0.2$, $\sigma^2 = 0.15$, and $R = 0.1$. At the start of the learning process, we initialized all Qs of the first layer and the second layer to zero. In order to obtain the initial bias of the Q value, we added a bias directly to the total Q ($Q = Q_1 + Q_2 + 0.007$). This bias makes optimistic selections of the actions [RA00]. We set the learning rate of each layer to 0.08.

Experiment We performed 100 experiments for each of the systems, SARSA1, SARSA2, and the proposed RL system. The capture rate values, which was obtained from the 100 experiments, were again averaged as a 2,000 trial moving average.

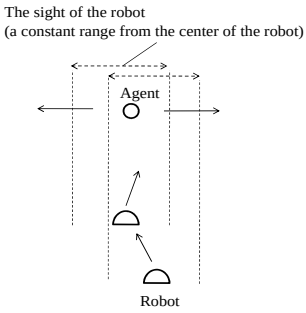


Fig. 1. Simulated capture experiment

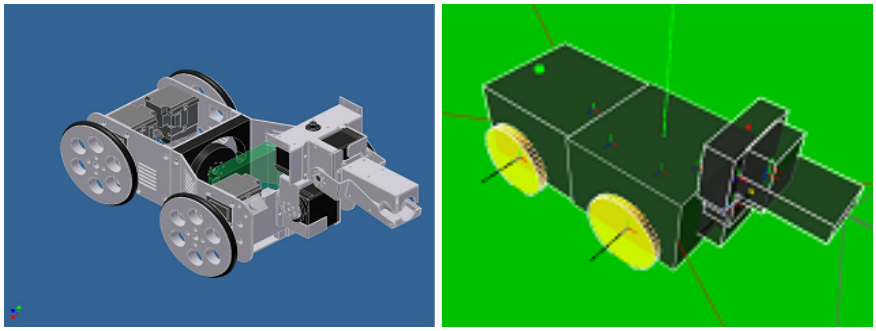


Fig. 2. Mechanical model (left) and computational model (right)

3.2 Guiding Task

We established another numerical experiment to examine the applicability of the proposed RL system for a more complicated POMDP environment. In this experiment, the learner (guiding robot) learns how to guide another robot (guided robot) to a goal. In previous studies, several researchers attempted such guiding tasks using traditional systems, such as a control system using a potential field [RN98,RN00], an evolutionary computation system [AJ96], and a classifier system [OP00]. The control system used by Vaughan gathered a flock of animals at a point using a feedback control [RN98,RN00]. This task is very useful to evaluate the proposed system, since experimenter is able to evaluate the effectiveness of RL systems from the speed to achieve the task, and the guided robot makes POMDP environment.

We modeled the hardware of the robots (Fig. 2 right) and the experimental environment (Fig. 3) on the Webots simulator [URL], which is based on a physical simulation engine called open dynamics engine (ODE). The specifications of the robots are shown in Table 1. We used the same model for both robots, the guiding robot and the guided robot.

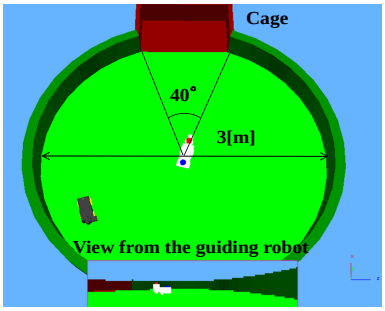


Fig. 3. Experimental environment

The guiding robot (black one) guides the guided robot (white one) into a cage. While the guidance, the guiding robot observes the guided robot using its head mount camera. The guided robot, which was equipped with two LEDs, shows the direction of its body to the guiding robot. The image from the camera is shown at the bottom of this figure.

Table 1. Specifications of the guiding and guided robots

Weight	Head		184.7 [g]
	Body (front)		370 [g]
	Body (back)		300 [g]
Size	Body	Width	120 [mm]
		Length	250 [mm]
	Wheels	Width	10 [mm]
		Radius	40 [mm]
DOF	Track Wheels		(D.O.F = 2)
	Waist		Roll (1)
	Neck		Pitch and Yaw (2)
	Jaw		Raises and lowers snout of robot (1)
Devices	Camera	Field of View	2 radians
		Resolution	128×32 pix.
	IR Sensor	Quantity	4
		Placement	30 degrees from side parallel
	Gyroscope		(not in the model)

Guiding Robot The software system of the guiding robot has three components, a pre-processing system, a learning system, and a behavior generation system.

[Pre-processing system]: This system process the information, which is obtained from the head-mounted camera image. The result is sent to the learning system (Table 2). From the image obtained by the head-mounted camera, the guiding robot extracts the weight centers of the guided robot, the cage, and the LEDs on the guided robot using the thresholds of their colors. The guiding robot then calculates the direction of the guided robot from the position of the LEDs. In addition, the guiding robot extracts the vertical edges of the cage using the Hough transform. The system normalized the horizontal weight centers of the guided robot and the cage, the sine and cosine of the direction vector of the guided robot, and the horizontal positions of the edges of the cage, to the range

Table 2. Observation of the learning system

Target	Information (dimension)	Range
Self (x)	Neck yaw (1)	$[-1, 1]$
Other agent (y)	Horizontal weight center (1)	$[0, 1]$ (detected)
	Rotation ($cos\theta$, $sin\theta$) (2)	-1 (not detected)
Cage (z)	Horizontal weight center (1)	$[0, 1]$ (detected)
	Horizontal corner position (2)	-1 (not detected)

of [0,1]. If the objects are out of view and the guiding robot fails to detect the objects, -1 is assigned to the value of the information. The angle of the guiding robot's neck , which is obtained from the encoder, is also normalized to the range of [0,1].

[Learning system] We applied the proposed two layered RL system with a predictor. The predictor predicts the velocity of the guided robot. We constructed the predictor, which has a mesh type function approximator, using an online learning process of the guiding robot. We set each cell of the mesh to output each prediction of $\tilde{y}$ for the corresponding observation of the robot. The predictor calculates an average value from the training data for $\tilde{y}$, and fixes the output of each cell to the value. We let the guiding robot move randomly using its action primitives (see the following subsection) around the guided robot in the experimental environment. Simultaneously, the predictor of the guiding robot was updated. We continued this update for 10 hours of the simulation time.

We set several rewards according to the state of the robots. The guiding robot rewarded its reinforcement learning system automatically with a reward of 0.1 when the guided robot and the cage overlapped on the image, which is obtained by the head-mounted camera of the guiding robot. From this state, if the guiding robot moved toward the guided robot, the learning system received a reward of 1. When the guiding robot successfully completed the guidance and the guiding robot confirmed the success by the head-mounted camera, the learning system received a reward of 10.

Action primitives We prepared eight action primitives (Table 3). The guiding robot executed one of the primitives that was selected by its learning system. Each action costs each time interval Δt. So, we used γ' instead of constant γ.

$$\gamma' = \gamma^{\Delta t} \tag{14}$$

Guided robot The guided robot moves according to its input from the infrared sensors (IR-sensors) and the force field that is set in the environment. The guided robot avoids obstacles and the guiding robot using its IR-sensors (Table 4). This avoidance has higher priority than movement according to the force field. So, while avoidance, the guided robot neglects the force field.

Table 3. Action primitives

Index	Time interval Δt [s]	Motion
A_0	1	Stay
A_1	1	Move toward the position of the guided robot
A_2	2	Turn clockwise around the guided robot
A_3	2	Turn counterclockwise around the guided robot
A_4	1	Move away from the position of the guided robot
A_5	1	Search for the guided robot
A_6	5	Move away from the cage
A_7	1	Search for the cage

Table 4. Collision Avoidance

Which sensors detect the objects	Command
Two front sensors	Turn left or right at random
Two rear sensors	Move forward
Right sensor only	Turn left
Left sensor only	Turn right

The guided robot follows the force field when nothing is detected by the IR-sensors. When the guiding robot is out of the 0.15 [m] radius from the center of the guided robot, the guided robot follows the force field shown in Fig. 4 (left) and moves toward the center of the field. If the guiding robot is in the circle, then this force field changes its flow, as shown in Fig. 4 (right). Fig. 4 (right) shows the force field when the guiding robot approaches the guided robot from the downward direction. Even if the relative positions of the robots are the same, the guided robot moves differently based on its absolute position in the field.

4 Results

4.1 Capture Task

The obtained capture rate is shown with the standard deviation error bars in Fig. 5. In addition, a 200 trial moving average is shown in Fig. 6 in order to provide detail. Based on the graphs shown in Figs. 5 and 6, we confirmed that the proposed system could achieve a higher success rate than the conventional systems, SARSA1 and SARSA2. The dotted lines in Fig. 5 show references for the performance when the robot performs optimal actions. The robot of Reference 1 does not observe the sign of u and follows the feedback control toward $y(t)$. Therefore, Reference 1 shows the maximum capture rate when the robot follows $M_{\{x,y\}}$. The robot of Reference 2 observes the sign of u and follows the feedback control to $y(t)+u$. Reference 2 shows the maximum capture rate when the robot follows $M_{\{x,y,\dot{y}\}}$.

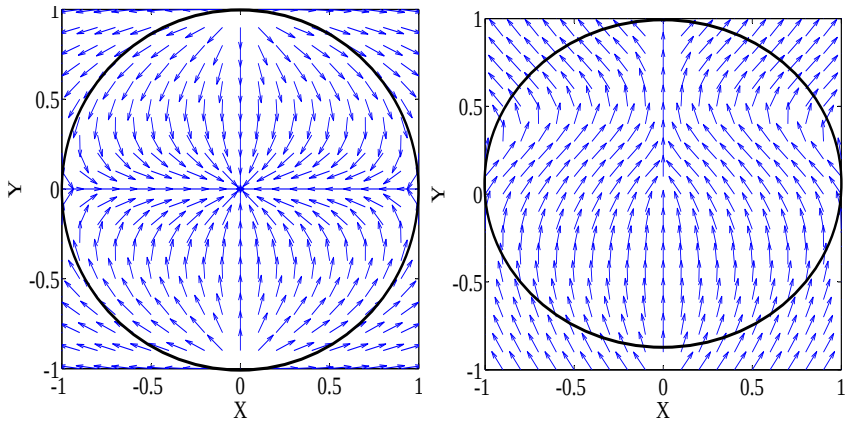


Fig. 4. Force field (left) and force field while avoiding the guiding robot (right)

In order to analyze the role of each layer of the proposed system, we focused on the domination of each layer. When the first layer dominates the action of the robot, the robot moves based only on input (x, y) because the first layer does not get input $\dot{y}$. In this case, the robot is following an MDP $M_{\{x,y\}}$. When the second layer dominates the action of the robot, the robot moves based on input $(x, y, \dot{y})$. In this case, the robot is following another MDP $M_{\{x,y,\dot{y}\}}$. We analyzed the dominations of the layers. We define the action selection of the first layer, $a_1(x, y, \dot{y})$, as whole system's action when Q_2 is neglected ($Q_2 = 0, Q = Q_1$). If the action selection of the first layer, a_1, is equal to the action selection of the whole system, a, then the first layer is the dominant controller of the whole system. Therefore, the whole system is using $M_{\{x,y\}}$ to capture the agent in this case. On the other hand, if $a_1(x, y, \dot{y})$ is modified by the second layer, $a \neq a_1$, then the first layer is no longer the dominant controller of the system. In this case, the whole system may use $M_{\{x,y,\dot{y}\}}$ to decide its action. To confirm the use of $M_{\{x,y,\dot{y}\}}$, we introduced Reference 1 of Fig. 5. We define dominance of the first layer, D_1, as follows:

$$D_1 = \sum_{\dot{y}} \delta_{a(x,y,\dot{y}),a_1(x,y)} \tag{15}$$

where δ is a Kronecker delta. We show the dominance of the robot in Fig. 7. We confirmed that the robot Learned to switch the dominance.

4.2 Guiding Task

We compared SARSA1, SARSA2, and the proposed system using the same learning parameters as those used in the capture task. The success rate of the proposed system tended to be higher than those of the other systems (Fig. 8).

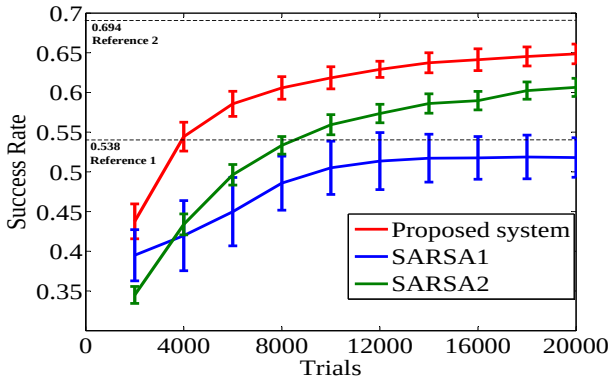


Fig. 5. Capture rate (2,000 trial moving average)

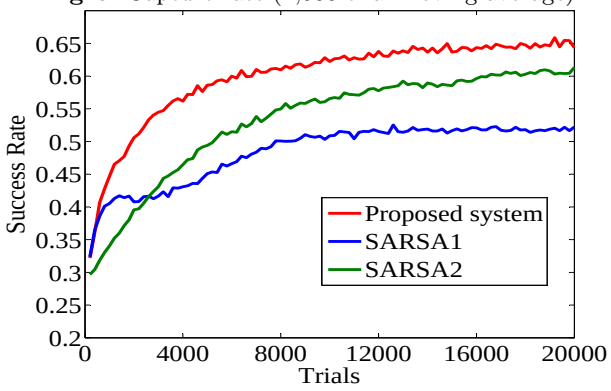


Fig. 6. Capture rate (200 trial moving average)

5 Discussion

5.1 Decoupling MDPs

At the early stage of the capture experiment, the first layer of the proposed RL dominated the action of the robot. So, the robot utilized $M_{\{x,y\}}$ at the early stage. At the last stage, the dominance of the first layer was weak when the agent was near the robot or was almost out of the sight of the robot. The robot utilized $M_{\{x,y\}}$ and $M_{\{x,y,\dot{y}\}}$ according to the situation. Generally, when a human captures an agent like the robot, the human needs the position y of the agent. If the agent is slow and far from us, y is sufficient information to approach the agent. However, if the agent is quick and near to the human, the human may need the speed $\dot{y}$ of the agent. We consider that the robot switched of $M_{\{x,y\}}$ and $M_{\{x,y,\dot{y}\}}$ appropriately according to the situation.

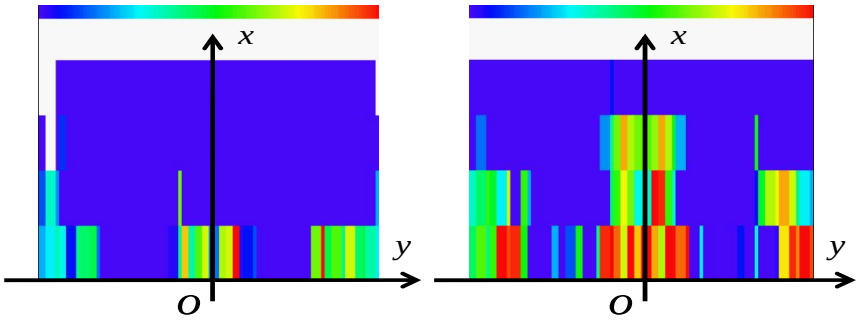


Fig. 7. Attention during the early stage (left) and attention during the last stage (right)

These images show the dominances of the first layer while the robot learns the capture task. We set O as the center position of the robot. Each colored pixel shows the dominance of the first layer. In the color bar at the top of the images, blue indicates that dominance of the first layer is strong (D_1 is high), and red indicates that dominance of the first layer is weak (D_1 is low). At the early stage of learning, the first layer dominated a wide area. This means that the robot focuses on $M_{\{x,y\}}$ and ignores the velocity $\dot{y}$ of the agent. During the final stage of learning, dominance of the first layer is weak at the center, on the left side, and on the right side. While the agent was far from the robot (top of the figure), the robot focused on $M_{\{x,y\}}$. However, when the agent was near the robot or around the limitation of the sight of the robot, the robot focused on the motion of the agent $\dot{y}$ and $M_{\{x,y,\dot{y}\}}$. The proposed learning system realized learning for the switch of two MDPs.

From the analysis for the robot's learning, $M_{\{x,y\}}$ was decoupled from the POMDP at first, and then $M_{\{x,y,\dot{y}\}}$ was decoupled. We consider that the reason why $M_{\{x,y\}}$ was learned faster than $M_{\{x,y,\dot{y}\}}$ is that $M_{\{x,y,\dot{y}\}}$ was more difficult problem, since it includes one more dimension than $M_{\{x,y\}}$.

We consider that the learning process of the robot was step by step as follows. The robot that learned $M_{\{x,y\}}$ followed a sub-optimal policy $\pi_{\{x,y\}}$ at the early stage of learning. The robot searched around $\pi_{\{x,y\}}$ using ϵ-Greedy search to improve the policy. In some observations, the robot found that $M_{\{x,y,\dot{y}\}}$ was more suitable to represent the problem. Then, the robot learned $\pi_{\{x,y,\dot{y}\}}$ during the observations. We may accelerate this learning process using A* search or other kind of model predictive searches, since ϵ-Greedy search is not the best one. However, to utilize model predictive searches, learning system has to make some model of the environment and/or agents while learning Q-values.

5.2 Consideration for a Traditional Theory

Pendrith et al. investigated the conditions for policy stability in non-Markov decision processes. Pendrith et al. mentioned that the TD style of credit assignment method is not guaranteed to have equilibrium points [MM98]. We agree on their consideration. However, in the case when a POMDP includes several MDPs Ms, the TD style of credit assignment method may have equilibrium points of the sub-problem M.

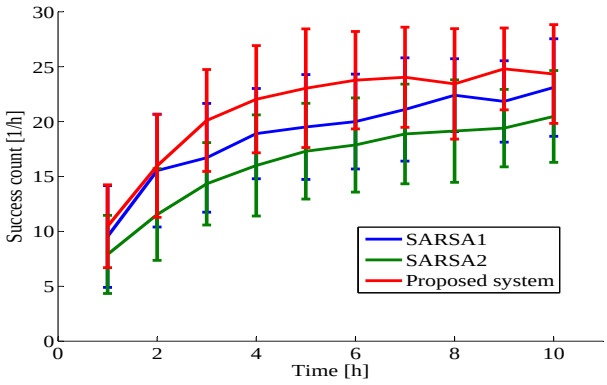


Fig. 8. Success rate of the guiding robot simulation

6 Conclusion

In this paper, we proposed multi-layered RL (MLRL) formulations that decouple a partially observable Markov decision process (POMDP) to several Markov decision processes (MDPs) step by step according to the dimensional hierarchy of the MDPs. From the results of the experiments, we confirmed that a two layered RL that is based on the proposed MLRL formulations decoupled two MDPs, $M_{\{x,y\}}$ and $M_{\{x,y,\dot{y}\}}$, step by step. Also, the MLRL got better success rate and success count than SARSAs in the experiments.

References

[C89] C. J. C. H. Watkins:"Learning From Delayed Reward," Ph.D. thesis of Cambridge University, (1989).

[C92] C. J. C. H. Watkins:"Q-Learning," Machine Learning, Vol. 8, pp. 279-292, (1992).

[RA00] Richard S. Sutton and Andrew G. Barto: "Reinforcement Learning," MIT Press, 55 Hayward Street Cambridge, MA 02142-1493 USA, (2000).

[LM98] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra :"Planning and Acting in Partially Observable Stochastic Domains," ARTIFICIAL INTELLIGENCE, (1998).

[Ke00] Kevin P.Murphy:"A Survey of POMDP Solution Techniques," Technical reports/ informal notes of the author, http://www.cs.ubc.ca/ murphyk/mypapers.html, (2000).

[JS98] John Loch and Satinder Singh:"Using Eligibility Traces to Find the Best Memoryless Policy in Partially Observable Markov Decision Processes," In Proceedings of International Conference on Machine Learning, (1998).

[MM98] Mark D. Pendrith and Michael J. McGarity:"An Analisis of Direct Reinforcement Learning in non-Markovian Domains," in Proceedings of the International Conference on Machine Learning, (1998).

[CT08] Chyon Hae Kim, Tetsuya Ogata, and Shigeki Sugano: "Reinforcement Signal Propagation Algorithm for Logic Circuit," Journal of Robotics and Mechatronics, (2008).

[ED04] Erfu Yang and Dongbing Gu: "Multiagent Reinforcement Learning for Multi-Robot Systems: A Survey," University of Essex Technical Report, (2004).

[Ge03] Gerald Tesauro: "Extending Q-Learning to General Adaptive Multi-Agent Systems," Advances in Neural Information Processing Systems, (2003).

[PG93] Peter Dayan and Geoffrey E. Hinton: "Feudal Reinforcement Learning," Advances in Neural Information Processing Systems, (1993).

[Mo91] Morgan Kaufmann: "Variable Resolution Dynamic Programming: Efficiently Learning Action Maps in Multivariate Real-valued State-spaces," In Proceedings of the International Conference of Machine Learning, (1991).

[RA01] Remi Munos and Andrew Moore: "Variable Resolution Discretization in Optimal Control," Machine Learning, (2001).

[Ri96] Richard S. Sutton: "Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding," Advances in Neural Information Processing Systems, (1996).

[Ha06] Hajime Kimura: "Reinforcement Learning in Multi-Dimensional State-Action Space using Random Tiling and Gibbs Sampling," Transaction of the Society of Instrument and Control Engineers, (2006) (in Japanese).

[MR98] Marco Wiering, Rafa L Salustowicz, and Jürgen Schmidhuber: "CMAC Models Learn to Play Soccer," In Proceedings of the International Conference on Artificial Neural Networks, (1998).

[RN98] Richard Vaughan, Neil Sumpter, Andy Frost, and Stephen Cameron: "Robot Sheepdog Project Achieves Automatic Flock Control," Proc. of the International Conference on Simulation of Adaptive Behavior, (1998).

[RN00] Richard Vaughan, Neil Sumpter, Jane Henderson, Andy Frost, and Stephen Cameron: "Experiments in Automatic Flock Control," Robotics and Autonomous Systems, Vol. 31, pp. 109-117, (2000).

[AJ96] Alan C. Schultz, John J. Grefenstette, and William Adams: "RoboShepherd: Learning a Complex Behavior," In Proceedings of the Robots and Learning Workshop, pp.105-113, (1996).

[OP00] Olivier Sigaud and Pierre Gérard: "Using Classifier Systems as Adaptive Expert Systems for Control," Lecture Notes in Computer Science, pp. 138-157, (2000).

[MS96] Minoru Asada, Shoichi Noda, Sukoya Tawaratsumida, and Koh Hosoda: "Purposive Behavior Acquisition for a Real Robot by Vision-based Reinforcement Learning," Machine Learning, Vol. 23, pp. 279-303, (1996).

[URL] http://www.cyberbotics.com/

The iCat as a Natural Interaction Partner
Playing Go Fish with a Robot

Koen Hindriks[1], Mark A. Neerincx[1], and Mirek Vink[1]

Delft University of Technology, The Netherlands. k.v.hindriks@tudelft.nl

Abstract. To be able to develop robots that naturally interact with humans it is important to gain a better understanding of the factors that shape this interaction. Although many aspects have already been studied in depth, few studies have been performed on the effect that socio-cognitive abilities may have on this interaction. We have developed a robot that shows intentional or proactive behavior and that can be used to conduct research on interaction that is shaped by cognitive abilities. We have used the iCat robot platform to perform experiments with children to test various hypotheses on perceived effects of socio-cognitive abilities. Two different versions were developed: a *socio-cognitive* iCat robot that behaves socially and takes the mood of the child into account, and an *ego-reactive* iCat robot that does not do so. These two robots were evaluated and compared with each other in a scenario where the robot plays the card game Go Fish with a child. Results indicate that children are more positive about the interaction with the socio-cognitive iCat than with the ego-reactive iCat.

1 Introduction

It is well-known that user interaction with a robot is shaped in part by the use of human-like features of the robot. For example, designing a robot with explicit anthropomorphic features such as a head with eyes and mouth may enhance human-robot interaction. A multitude of experiments and studies have been performed that explore and analyze various anthropomorphic and other mechanisms that support natural interaction [1].

As argued in [2], the design of robots is a matter of balance: finding the appropriate level of similarity with humans, and taking into account movement and appearance, and possibly many other factors. One factor that has not yet received much attention in human-robot interaction research concerns the socio-cognitive skills that shape the behavior of a robot. Our aim in this paper is to present and discuss some initial results related to the effects and contribution of such skills on human-robot interaction. To this end, we present the design of a cognitive robot endowed with some human-like socio-cognitive skills that shape its behavior and interaction with humans. Such skills involve among others capabilities such as decision-making, attention allocation, anticipation, planning, and taking mental states of others into account. Here, in particular, we will focus on aspects related to *goal setting in combination with emotion modeling.*

Our general motivation for performing this research is our interest in establishing whether, and, if so, how cognitive skills of a robot can enhance the anthropomorphic or intentional stance that a human takes towards a robot. In other words, the question that drives our research is whether a cognitive architecture can contribute to the effect of a human subject taking the intentional stance towards a robot and attributing mental attitudes to the robot. This research also contributes to the closely related area of so-called *believable social robots* [2]. [3, 4] argues that to be intentional, a robot must exhibit goal-directed behavior, which motivates our choice to focus on goal setting and proactive behavior here.

In this paper we report on a case study in which a robot with facial features that facilitate emotion expression and that demonstrates social, proactive behavior is compared with the same robot platform that makes rather different use of emotion expression and that does not show social proactive behavior. We call the first type of robot a *socio-cognitive* robot and the second one an *ego-reactive* robot. The socio-cognitive robot sets goals that contribute to becoming friends in the context of a game and adapts its behavior accordingly; moreover, it takes the mood of the human player into account. In contrast, the ego-reactive robot is only concerned with winning the game and does not respond to the human player's mood; this robot only reacts to game events. We have used the iCat robot platform (see Figure 1(b) below) and performed experiments with children that play a Go Fish card game with the robot to obtain experimental results about perceived differences. The main hypothesis is that the behavior of the socio-cognitive iCat is evaluated more positively than that of the ego-reactive iCat.

The paper is organized as follows. Section 2 briefly discusses some related work. Section 3 presents the *scenario design* and motivates the choice for the game that we have used. Section 4 discusses the *robot design*. Section 5 describes the *experimental design* and presents our findings. Finally, section 6 concludes the paper and discusses interesting future work.

2 Related Work

[5] argues that to interact socially with humans a robot must *convey intentionality* and describe an architecture implemented on top of Kismet that facilitates exhibiting so-called *proto-social responses* to this end. They argue that by means of these proto-social responses the robot is able to provide cues for interpreting its actions as intentional. Kismet is able to express a range of emotions but does not take the emotions of the human itself into account. The architecture also has a motivational component which manages drives. In our architecture instead of drives explicit goals that represent states the robot wants to realize are present and we focus on cognitive skills to convey intentionality.

Various researchers have used the iCat to study human-robot interaction and, as we also use this platform, in the remainder of this section we focus on this research. The iCat has been designed especially for studying interaction and turns out to be an excellent platform to this end since it is capable of generating

a range of facial expressions and because of its size and the way it looks, many humans feel immediately connected to it.

[6] reports on the influence of a robot's social abilities on acceptance by elderly users. Two iCats were used in a Wizard-of-Oz experiment where two conditions were compared: one which used a more socially communicative and one which used a less socially communicative interface. The more communicative iCat listened attentively, showed nice and pleasant facial expressions, remembered personal details like names and admitted mistakes. Results of this research show that participants who were confronted with the more socially communicative version felt more comfortable and were more expressive in communicating with the iCat. The work suggests that a more socially communicative iCat is more likely to be accepted as a conversational partner.

[7, 8] develops a social iCat that aims at helping children in their daily healthcare related activities. The goal of this research is to use the iCat as a medium to enable diabetic self-care for children. Three support roles with corresponding behaviors were developed labeled *motivator*, *educator* and *buddy* respectively. Results indicate that children value the support roles positively, in particular the buddy role. In follow-up research reported on in [9], two different versions of the iCat were developed: one that shows interest in the human (demonstrated e.g. by remembering names) and displays social behavior at appropriate times in contrast to the other version that only expresses its ego-centric emotions. Children were asked to perform various tasks during an experiment such as answering questions about health-related movies and playing a game of Tic-Tac-Toe. The results show that the more social robot is rated as more empathetic than the other robot.

[10] reports on an iCat that is able to play chess. This chess-playing iCat is different in that it expresses emotions which are influenced by the move the opponent makes. The results suggest that the childrens' learning experience of chess is enhanced by the iCat's expressions which is explained by the fact that the children are able to recognize when they made a good or bad move based on the expressions displayed by the iCat.

Our experimental scenario is also based on a game since a game structures the interaction and provides for more control with respect to various parameters of the research. As discussed above, games have been used often in robot research for similar reasons. However, not every game is suitable for our purposes. For example, the Tic-Tac-Toe game that has been used by [7, 8] has too limited game play, game play is too short and the game induces virtually no speech if at all to be useful for our purposes. Although the game play of the chess robot of [10] is much more interesting and challenging, in a game of chess the focus again is more on strategy than on interaction. In the next section we will describe and motivate our choice for a different game as the basis for an interaction scenario.

3 Scenario Design: Playing Go Fish with a Robot

Partly based on a review of the literature and partly based on requirements on the type of interaction we are interested in, we decided to use a game-based scenario. Such a choice has several benefits. Given a specific scenario based on a game and a human that acts according to the rules of the scenario, it is possible to more or less reliably predict the actions of humans, what they will expect, and to create matching behavior and speech. A fixed scenario has benefits also from an experimental point of view since it provides more control over variables that potentially impact what we want to measure and it is easier to measure effects and identify the causes of these effects in a structured environment. Finally, a scenario based on a game also naturally limits the tasks and speech acts that are needed to socially interact. This thus greatly eases the design task of both the environment and the robot.

In order to select a specific game, multiple criteria were used. First of all, since we planned to perform experiments with children, we need a game that is suitable for children in the age range from 9 to 12. The game needs to be sufficiently challenging, but also relatively easy to grasp for children. Moreover, we need a game that can be played at a sufficiently challenging level of profiency by the iCat and in particular does not require capabilities that cannot be supported by the iCat platform or other state of the art technology (e.g. no advanced dialogue capabilities should be required). The game should not be too easy so that the iCat would win all games, but it should not lose all games either; ideally the iCat can be tuned so as to play at the level of an average human player. One of the main factors determining whether a game is captivating, according to [11], is the *challenge factor*. The challenge factor in turn is determined by two other factors: the activity has a *clear goal* and *the outcome* of the game *is uncertain*.

We have selected the card game named "Go Fish" [12] in which various players take turns and have to ask for cards to collect complete sets of cards (which are called *ranks* and typically consist of four cards). Go Fish has been selected because it is a fairly easy game that is easy to explain and can be mastered in a relatively short amount of time. Still Go Fish has the challenge factor. Go Fish is a game with a clear goal, i.e. to collect the largest number of card sets. But the outcome of Go Fish is also uncertain because Go Fish is highly dependent on chance, and even when it looks like one player is clearly winning, it can still swiftly turn around when the other player receives excellent cards from the deck. One of the most important aspects of the game for our purposes, moreover, is that the game requires a significant amount of interaction. Although the game also requires conversation between players, the set of speech acts that is minimally needed for playing the game is rather limited.[1]

We briefly discuss the rules of the game. At the start of the game each player receives seven cards from a shuffled deck of cards. The remaining cards remain

[1] In line with the rules, all that is strictly required is being able to ask for a specific card and to say "Go Fish". To make the game fun, however, a little more text seems required and has been built into the repertoire of the iCat. See below for more details.

facing down in the deck. For the first game the player that begins is chosen randomly; for all games thereafter, the player who lost the last game starts. If it is a player's turn, he or she may ask for a *specific* card of another player. The rules of the game dictate that a player can only ask for a card if he or she already holds at least one card of the same rank. If the card requested is owned by the player that is being asked, the card must be handed over to the player that asked for the card and that player continues and asks for another card; otherwise, the player that was asked for the card says "Go Fish!", the player whose turn it is draws a card from the deck and the turn shifts to the player who was asked for a card. The game ends when all card sets have been collected, typically by different players; the player with the most card sets wins.

In our experiments, we use a card set consisting of 36 cards (9 ranks) with zoo animal pictures (monkey, lion, etc.) each of which occurs in four different colors (red, yellow, green, blue). Moreover, in the experiments we conducted only two players play the game: one human and one iCat.

4 Robot Design: A Socio-Cognitive & Ego-Reactive iCat

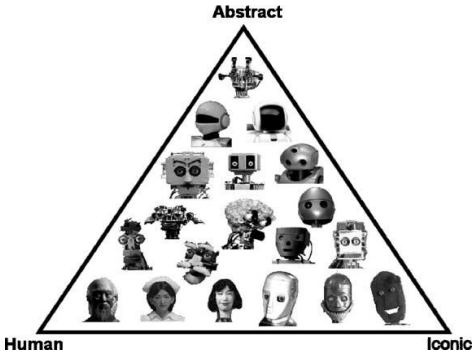
In order to evaluate the main hypothesis that the behavior of a socio-cognitive iCat is evaluated more positively than that of an ego-reactive iCat we have to specify the two different architectures that correspond with these types. This specification should clarify what kind of behavior both types of robot will produce. Before we describe the architecture design in more detail in this section, however, we first introduce and discuss the robot platform that we used. Next the probability model on which the strategic game-play is based is discussed. We then continue and discuss the two key components in our study, the design of a cognitive component that enables goal setting and decision making, and the design of an emotion model. Finally, we then briefly describe the design of various modalities that are also relevant to the interaction including facial expression (emotion, mirroring), speech, and eye contact.

4.1 Choice of Robot Platform: The iCat

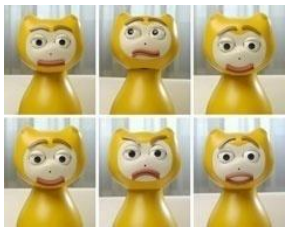
Social robots differ in their purpose from other robots in that they are specifically designed to be used in interaction scenarios with humans. As comics designers have known for decades, the particular representation used to portray characters in a comic can influence dramatically the way people identify and sympathize with its characters. Humans, for example, are more likely to identify with Dilbert then with Albert Einstein. The reason is that people can more easily identify with iconic universal characters instead of with a unique individual with a particular biography and personality [2].

Figure 1(a) depicts a triangle known as the Design Space. At the lower left *Human* correlates to as human as possible. *Iconic* is more like a cartoon figure with a minimum set of characteristics but which is still expressive. The *Abstract* corner corresponds with more mechanical and functional designs of the robot,

with minimal human-like aesthetics. The Design Space was originally designed by McCloud for comics, but may usefully be applied to robotics as well [2].



(a) Robot Head Design Space



(b) Facial Expressions of the iCat

Fig. 1. Robot Head Design

For our research, we have selected the iCat robot platform illustrated in Figure 1(b). The iCat is a research robot, designed by Philips, to stimulate research on human-robot interaction. An important reason for selecting it has been the iconic face of the iCat that is capable of mechanically rendering facial expressions. Emotion expression is an important aspect of social interaction that is supported by the iCat. Since we target children as our main subjects, an iconic head also may be preferable over more human-like heads since [13] suggests that fear would be increased in young children in that case.

An iconic face more easily can represent the human user which makes it easier for humans to communicate and identify with an iconic robot, whereas a human-like face is taken to represent somebody else (with a different personality). Humans apply a social model to robots and will often approach interactions with robots holding a set of preconceived expectations based on their experiences of interactions with other humans. If these social equivalences extend to the interpretation of human-like body language displayed by a robot, it is likely that there will be corresponding benefits associated with enabling robots to successfully communicate in this fashion [14].

The iCat supports human-like facial expressions. It is 38cm tall and equipped with several servos that control different parts of the face, such as the eyebrows, eyes, eyelids, mouth and head position. It also incorporates touch sensors in its ears and feet, a web cam in its nose, stereo microphones and loudspeakers.

4.2 Generating Strategic Game-Playing Behavior

As a start, to ensure a minimum of natural interaction with a human player, it is important that the robot is able to play a game of Go Fish at a sufficient level of proficiency. The robot is supposed to play Go Fish by asking *specific* cards (as opposed to asking for a rank or class of cards as is also usual). It seems clear that inconsistent or very bad play (e.g. by asking for a card that is known to be owned by the robot itself, or by always asking random cards) will not be perceived by a human opponent as natural (assuming that an opponent is expected to play such as not to lose quickly). Moreover, if the iCat aims at winning, asking the opponent for a lot of different cards also means to give away a lot of information which may not be strategic.

Go Fish is a game with incomplete information. Cases where a player knows exactly which cards the other player is holding are very rare. It is also not known exactly which cards are left in the deck. Good game play nevertheless requires a player to ask for the "best" card from an opponent. Of course, the fact that it is not known which cards are held by the opponent and which are still in the deck complicates the task of determining what is the best card to ask. In order to provide the iCat with appropriate game-playing skills, a probabilistic model is used to determine the probability that the human opponent holds a particular card. In addition, an opponent model is used to estimate what information the human opponent has about the hand of the iCat. These models are used to decide which card to ask for: The card with the highest probability of being owned by the opponent.

The probabilistic model that the iCat maintains assigns a probability to each card of either being held by the opponent, being part of the deck, or being owned by the player (iCat) itself; the latter, of course, has either a probability of 1 or 0 because the player knows which cards it is holding. To initialize the model, first a probability of 1 is assigned to each card as being part of the deck. After distributing the initial cards to players, the probability of cards owned by the player itself are assigned a probability of 1 but now as being owned by the player. All other cards are assigned a probability of 7/29 of being held by the opponent and 22/29 of being part of the deck (here, it is assumed that a deck of 36 cards is used and there are only 2 players).

The probabilistic model is updated after each of the following game events: *A card is requested*, *a card is received*, *a card is given away*, *a card is taken from the deck* and *a rank has been completed* (ranks need to be set aside by players when they are complete). As cards being received, cards being given away, and ranks that are completed are public events, it is obvious how to update probabilities associated with these events.

We briefly explain how the remaining events are handled. If a card is requested and the iCat owns the card, the iCat hands over the card; probabilities for the other cards of that rank then are updated and set to $P = 1/(3-R)$ where R is the remaining number of cards of the rank the player itself owns (note that we must have $R < 3$). If a card is requested and the iCat does not own the card, however, two things happen. First, the card being asked for is assigned

probability 1 as being part of the deck (it is not allowed to ask for cards that a player owns). Second, since it is now known that the opponent has at least one card from the rank of the card requested, the probability P that a card of that rank is being owned by the opponent is updated (assuming that $P \neq 1$ for all of these cards) and set to $P = 1/(4 - R)$ where R is the number of cards of the rank the player itself owns.

Finally, whenever an opponent draws a card from the deck the probability of a card being held by the opponent needs to be raised. Of course, this needs to be done only for those cards which are not part of a completed rank or are otherwise known to be owned by that player. A simple update is applied to all other cards and $1/D$ is added to the current probability where D is the number of cards still in the deck; thereafter, probabilities are renormalized again.

The model just discussed is not perfect and based on some simplifying assumptions. However, the model has been tested during an initial pilot and it was established that the iCat played Go Fish at a reasonable level while using it. The model was adapted however to compensate for the fact that by just asking for the card with the highest probability the iCat would sometimes ask cards in a strange order. A good example is when there are multiple cards with the same probability spread over multiple ranks. It is then possible that the iCat first asks, for example, the red chimpanzee, then the blue camel, and then the blue chimpanzee and by doing so obtains a complete card set of chimpanzees. This type of behavior is perceived as strange because a human player typically likes to first secure a complete card set before asking cards from another rank, and introduces the risk of not being to complete a rank and invite obvious counterplay of the opponent. A preference was built into the iCat for selecting cards from the same rank whenever multiple cards have the same probability.

4.3 Generating Intentional Behavior

To clarify the range of behavior that exists in the Go Fish scenario, we first list some of the action alternatives that are available and have been implemented. During the game, the iCat among others can *ask for cards*, *provide hints*, *stimulate the opponent to perform an action*, *express various emotions*, *move its head*, and *make eye contact*. As will become clearer below, only some of these actions will be performed by the ego-reactive iCat whereas all of the actions may be performed by the socio-cognitive iCat.

The idea is that the ego-reactive robot only shows *reactive behavior*. Its interaction is solely based on its intent to win the game (strategic play by reacting to the opponent's move) and the only emotions it displays are related to its own, ego-centric emotions. That is, the ego-reactive robot will express happiness if it is winning and sadness if it is losing, but it will not take into account the emotion of the human player. In fact, the ego-reactive robot has been designed to not even register the human's emotions. The ego-reactive robot will react to the same (game) event in the same manner and the robot has been designed as if it is living in its own world while almost completely disregarding the opponent.

In contrast, the idea of the socio-cognitive robot is that it will take the state of the human into account while selecting actions. The main goal (which is implicit by design) of the socio-cognitive robot is to become friends with the human. It will try to - of course only up to a certain extent - mimic some of the intentional behavior that would be displayed by humans to this end. In our context, the socio-cognitive robot will take the game score and the emotions of the human into account. Of course, the socio-cognitive robot also is able to play the game at a reasonable level of proficiency and in fact extends the action repertoire of the ego-reactive robot. Because the socio-cognitive robot can do whatever the ego-reactive robot can do, we can view the ego-reactive robot as a *base-line* for the other robot. We like to note in particular that both iCats have the same capabilities for winning the game (see also the description of strategic game-playing behavior below).

One of the main differences between the socio-cognitive and ego-reactive iCat concerns the *goals adopted throughout the game*. This difference has been explicitly designed and has a significant impact on the behavior produced. It is one of the key factors that makes the socio-cognitive robot different from the ego-reactive. Whereas the ego-reactive iCat will stay focused on winning, the socio-cognitive iCat will adopt various other goals *depending on the game state and estimated emotional state of its opponent.*[2] For example, the socio-cognitive robot will adopt a goal to cheer up the opponent by saying something like "Cheer up, I'm sure you'll win the next one!" and to tone down his emotions when the opponent has been losing for a while. Another example is that the socio-cognitive iCat will adopt a goal to provide hints to its opponent when the opponent is losing and/or sad. These hints will suggest to the opponent which card to ask for (knowing, of course, which cards are in the hand of the iCat the robot can give away useful information). The iCat will not suggest a particular card but say, for example, "If I were you I would try a different color". A third example concerns intentionally loosing a card set. A goal to this end may be adopted if the opponent is badly losing. The iCat can do this when it knows that it is possible to get a complete set. The iCat's behavior will then change in a way that it will ask by accident a card it already asked for; it then will become clear to the opponent that it can ask these cards back and gain a complete set. The goal setting behavior and the resulting game playing behavior of the socio-cognitive iCat thus is significantly different from that of the ego-reactive iCat. Intentional behavior is based on the intentions or goals of a person. We believe that by explicitly separating goals from the behavior that is selected to achieve these goals will help create the perception that behavior is intentional and thus will increase the intentional stance towards the robot. Below we show that the difference between the socio-cognitive and ego-reactive robot is also perceived by subjects in the experiments we conducted.

It is important to realize that the socio-cognitive iCat may have conflicting goals. The first goal set to win the game conflicts with the other goals set to make

[2] Mood detection of the human opponent is rated by a co-experimenter in a Wizard-of-Oz setup; see Section 5.

friends and enjoy a nice game. Of course, to achieve the latter goals it is still important to balance the behavior these goals induce with behavior to win the game. That is, usually you do not make friends by simply losing severely which is not very natural in the first place. Balancing these goals has been achieved by making sure that the iCat does not play badly throughout the whole game; i.e. by interleaving and balancing "friendly" behavior to achieve the goal to make friends with game-playing behavior to win the game a more natural interaction is realized.

A cognitive architecture that supports goal setting, decision making and action selection has been built on top of the iCat using the cognitive programming language GOAL [15]. The choice for this language is in part based on its support for goal-directed decision making. Using GOAL it is easy to make the robot explicitly adopt and drop goals based on e.g. the game state or the mood of the opponent. The cognitive language GOAL provides explicit support for updating, maintaining and acting based on goals. The language also provides support for managing the knowledge and beliefs of the robot. Knowledge is used to represent basic facts about the game (e.g. 4 cards of a type form a set) whereas beliefs are used to keep track of the state of the (gaming) environment.

To explain the main components of the cognitive architecture, we very briefly discuss each of the components of a GOAL program. The interested reader can find more details in [15]. For representing informational attitudes, a *knowledge* and a *belief* base are used. A knowledge base consists of concept definitions and/or domain knowledge represented using Prolog. The main difference between the knowledge and belief component is that the former is static whereas the latter can be modified at runtime. The belief base consists of beliefs that keep track of the current state (e.g. which cards are being held by whom in our case). Motivational attitudes are maintained in the *goal base* which consists of goals that represent the state the agent wants to be in. The knowledge, belief and goal base make up the agent's *mental state*. The remaining sections concern actions and action selection. A so-called *program* section consists of a set of *action rules* and/or *modules* that define a strategy or policy for action selection. The *action specification* section consists of a specification of the pre- and post-conditions (effects) for each action that the robot can perform. Finally, a special section deals with events and consists of so-called *percept rules*. A percept rule is used to determine how percepts received via sensors from the environment are used to modify the mental state of the agent.

As a final note, we would like to remark that even though a cognitive architecture is used some of the behavior of the socio-cognitive iCat may also be triggered by particular events and thus may be generated *reactively*. For example, when an opponent asks for a card and the iCat does not have it, the socio-cognitive iCat may say something like "Better luck next time!" if the estimated valence (emotion) of the opponent is low.

4.4 Emotion and Mood Model

A second important difference between the two iCats is that the socio-cognitive robot uses a model that estimates the emotional state of the human whereas the ego-reactive robot does not. Both maintain a model of the emotion of the iCat itself. The emotion model of the ego-reactive robot only takes into account the current game state and how well the robot does itself; basically, this means that the robot is happy when it is doing well and sad when it is doing badly. The emotions of the socio-cognitive robot, however, are also influenced by the estimated affective state of the human. The emotion and mood models are used to determine when and what kind of emotion expression the iCat will display at various points during the game. The socio-cognitive robot also uses its emotion model in its decision mechanism to select actions; e.g. the socio-cognitive iCat may provide hints when the opponent is unhappy.

We have used a *valence-based emotion model* [16]. This model is relatively simple but turned out to work well in practice. Emotion expressions are not generated only by means of valences which may fluctuate too much to appear natural but are also determined by *mood*. Emotions are different from moods. The latter are more stable and therefore more useful for determining which emotions to express. Mood is computed as an *exponential moving average* - a method also used in e.g. [10] - over the last 20 emotions (valences), a number which was determined experimentally and which we found provides good results. The advantage of using an exponential moving average over a simple moving average is that it gives more weight to recent events.

In this approach it is important to associate valences with events, in our case game events, to establish current emotion valences. For instance, giving a card away will generate a valence of a certain value. We have determined the values for these valences experimentally and balanced things so as to give sufficiently expressive emotions that can be recognized by human subjects but which result in more or less stable moods as well. The details can be found in Table 1.

Table 1. Valence Associated with Game Events

Game Event	Valence
iCat requests a card	30
Player refutes requested card	-30
Player requests a card	-30
iCat says Go fish	30
Player confirms requested card	70
iCat gives requested card	-70
Waiting for player to request a card	0
iCat has a set complete	90
Player has a set complete	-90
iCat won the game	100
iCat lost the game	-100

Emotion expressions are generated based on the mood whenever there is no active valence value; this implies that emotion expressions remain stable in that case. Otherwise, for a brief interval of time (we used 3 seconds) the iCat will express an emotion based directly on the valence value.

4.5 Facial Mirroring

Facial mirroring occurs because we share the emotions of others. If someone sees someone else is smiling at him, this may trigger a positive response and even a feeling of happiness within the observer. One typical response is to produce a smile as well. The same principle also applies to other emotions [17]. The principle of facial mirroring has been partly implemented in the socio-cognitive iCat, in which the cognitive structure tries to mimic this behavior. This can be observed during a game, for example, when neither player is winning and the human is expressing happiness. In that case, the iCat will adapt is own emotions and also become more happy which in turn will show in its facial expressions.

4.6 Speech

In the game of Go Fish it is important to be able to use speech when aiming for natural interaction. By design, the iCat will produce game and scenario related speech. As human subjects likely feel confused or disturbed by a combination of clearly-human speech with clearly-non-human face [18] we have used a simple text-to-speech synthesizer.

A pre-configured database of sentences is used to generate speech. These sentences have been selected so as to fit the Go Fish game setting. The game setting naturally induces a set of sentences that are typically uttered during game play. This greatly helps shape the database and limits the amount of work needed to create such a database. The database contains different sentences related to different events in the game. As it is important to avoid the impression that the iCat is very static and robot like multiple sentences for each event were added to the database which were randomly varied.

The database contains sentences to allow both iCats to introduce themselves and to ask the opponent for his or her name. After the opponent has told his or her name, the iCat will say that it thinks the name is very nice. In case the human subject would start conversing on a quite different topic, the iCat would produce a sentence such as “My designer only programmed me to play Go Fish”. Finally, the iCat will also ask the human subject at the end of a game if he enjoyed his time with the iCat and will ask him to fill in questionnaires.

For the socio-cognitive iCat, additional sentences have been inserted that are related to some of the intentional behavior discussed above including, for example, sentences for cheering up (“Next time I am sure you will get a set!”) and giving hints about cards. Special care has been taken to limit the number of additional sentences that the socio-cognitive iCat will produce compared to the ego-reactive iCat to avoid excessive focus on speech which might induce a

preference in the human subject for one iCat over the other based on the speech interaction.

4.7 Eye Contact and Face Tracking

Listeners look more at the speaker in order to show responsiveness and interest, typically looking at the speaker about 75% of the time in glances lasting 1-7 seconds. If, as a listener, you want to make a verbal contribution, it is important that eye contact is reestablished with the speaker. During the experiment the experimenter determined when to look at the child, as it turned out that automatic eye contact behavior was hard to implement. During parts of the scenario the iCat automatically looked at the child, but during the game the experimenter recognized speech so this would also indicate that the child was talking and that the iCat should look in the direction of the child.

The iCat by design also looks at its cards when it is its turn to emulate the behavior of somebody that looks at his cards trying to decide what to ask for.

5 Experimental Design: Setup and Results

We discuss the experimental design and findings but start by discussing the software used, the experimental setup and the methods for evaluation we used.

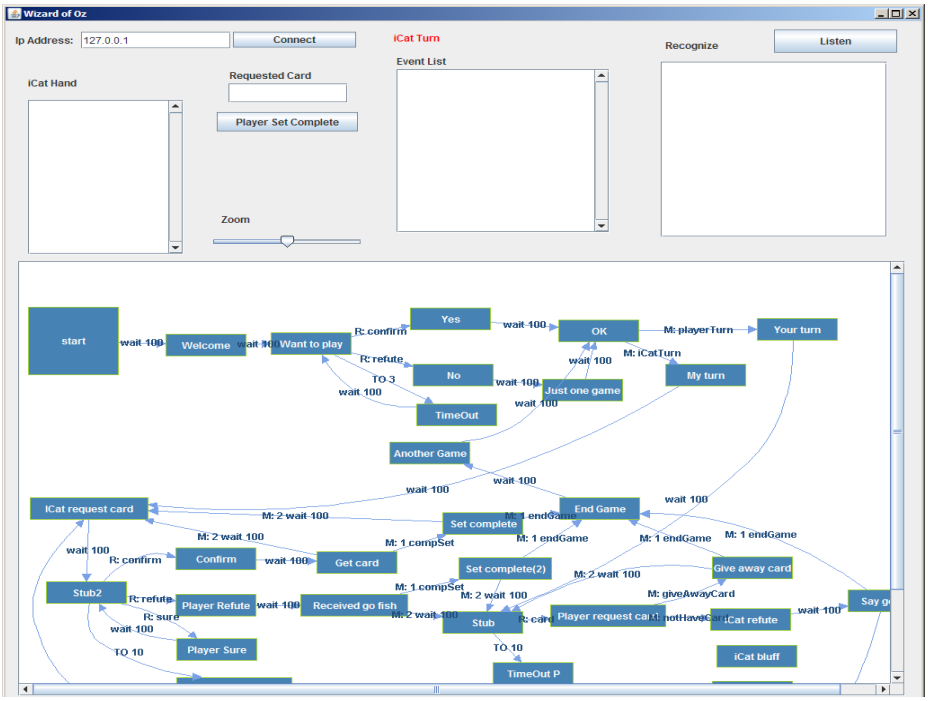
5.1 Game Play

A game shell was developed using Java to graphically represent the game and to handle all game events. This game shell handles the flow of the game and implements the basic rules of Go Fish. The game shell also keeps track of various other aspects of the game. The card deck that has been used has nine ranks including a Bat, Camel, Chimpanzee, Dog, Giraffe, Goldfish, Shark, Lion, and Rhinoceros. The interface allows a human player to see his or her own hand, the game score, how many cards the iCat is holding, whose turn it is and facilitates to freely rearrange cards in a designated area in order to sort them.

5.2 Wizard-of-Oz Interface

One of the two main reasons for introducing a Wizard-of-Oz setup as part of the experimental design is to be able to reach a high level of correctly recognized speech. Speech recognition is still in many ways a challenging problem and not the focus of this research which explains why we have chosen for this setup. The second reason concerns emotion detection. Current state of the art technology does not provide easy access to reliably recognize human facial expressions of emotion. A dedicated GUI was developed to support the main experimenter in performing these tasks (see Figure 2(a)). Additionally, a co-experimenter has been introduced to perform some related tasks. Introducing a co-experimenter ensured that the main experimenter is not overloaded. The

co-experimenter helped with gathering some statistics; this included how often a child laughed, how much time a child talked with each iCat and how often a child looks at the iCat. The co-experimenter used a simple GUI (Figure 2(b)). Note that the items rated by the co-experimenter do not have any effect on the behavior of the robots but only registers some aspects of this behavior.



(a) Wizard-of-Oz Interface

(b) Mood Detection GUI

Fig. 2. Wizard-of-Oz Interface

5.3 Subjects

For our experiments, we selected subjects in the age range of 9-13. One reason why children are a particularly interesting subject group is that the emotion

and the mood of a child are easier to determine than that of an adult. Adults have a tendency to camouflage their emotions which makes it more difficult to obtain reliable data. The children that participated are going to school and the experiment was conducted at the school they go to. In each class, the experiment was briefly explained and the reason why participants were needed for the experiment was explained. A selection of 36 children were picked by a lottery to participate. These children then received a letter of consent to be signed by the parents; 27 children returned signed consent forms. Finally, 24 children participated and the data of 20 of these could be used. The 20 participants were aged 9-13 (Median age = 11, SD = 1). They were awarded with a small present (eraser, sticker, etc) and a photo of the participant with the 2 iCats for taking part in the experiment, which lasted about 30 minutes.

5.4 iCat Names

Both iCats have been given unisex names. The socio-cognitive iCat has been named Robin and the ego-reactive iCat has been named Kim. Although few conclusive results have been published about gender impact on perception of robot personality, believability and engagement, we believe it is better to stay on the safe side and not force a gender on the iCat.

5.5 Hypotheses

A number of different hypotheses were formulated in order to verify whether the socio-cognitive iCat is considered to behave more to the liking of human subjects than the ego-reactive iCat. The first hypothesis is that *the socio-cognitive robot will be perceived as a friendlier robot than the ego-reactive one*. A reason for this difference in perception would be that the ego-reactive robot does not respond to the mood of the opponent at all while the socio-cognitive robot does. Moreover, the socio-cognitive robot gives hints and asks questions such as whether the child is (still) having a good time.

The second hypothesis is that *the ego-reactive robot will be perceived as handling losing worse than the socio-cognitive robot*. A reason for this difference is that when the ego-reactive robot is losing it will express a quite negative mood by means of facial expressions. The socio-cognitive robot presumably will take losing better as its reaction also takes the mood of the opponent into account.

The third hypothesis is that *the socio-cognitive iCat will be considered as more fun to play with than the ego-reactive one*. A reason for this is that the socio-cognitive robot will try to help the child and will try to cheer up the child whenever that seems appropriate.

The fourth hypothesis is that *human subjects will not perceive a significant difference in strategic (i.e. with the aim of winning) game-playing*. This hypothesis seems reasonable considering that both iCats use the same strategy and even though the socio-cognitive iCat sometimes deviates from best game-play on purpose and gives hints this will most likely not be perceived as bad game-play either. This hypothesis has been added to verify that the implementation

of social behavior did not result in bad game-playing behavior which would spoil the fun in playing the game for most human subjects.

The fifth hypothesis is that *the socio-cognitive iCat will make the opponent smile and laugh more than the ego-reactive iCat.* A reason for this is that the ego-reactive robot does not care at all about the mood of the child.

The sixth hypothesis is that *with the socio-cognitive iCat more speech events will be observed from the human subject towards the iCat than with the ego-reactive robot.* A reason to expect this is that the ego-reactive iCat will only ask for a certain card and shows otherwise very consistent overall behavior focused on winning the game.

Finally, the seventh hypothesis is that *a human subject will look more often at the socio-cognitive iCat than at the ego-reactive iCat.* A reason for this is that the socio-cognitive iCat will say more and different things than the ego-reactive iCat, and, according to hypothesis one and three the socio-cognitive robot is more friendly and fun to play with.

5.6 Experimental Design

A pilot was performed to obtain initial feedback about the experimental design. This resulted in various improvements related to game play, strategy, speech and emotion expression, as well as with respect to emotion and mood detection. In particular, aspects that felt unnatural according to participants during the pilot were modified (cf. the example concerning the strategy for asking cards above).

A within-subjects design was employed in which children played a game with 2 different iCats, meaning that each child plays against the socio-cognitive as well as the ego-reactive iCats. Experiments were counterbalanced and half of the children first played with the ego-reactive and then with the socio-cognitive iCat whereas the other half played first with the socio-cognitive and then with the ego-reactive. A game of Go Fish typically took about 10 minutes to finish.

The experiment was conducted in a single room with which the children were acquainted. In order to give some privacy to both the child and the experimenters they were separated by a screen and the children were facing in the opposite direction of the screen. The entire experiment was recorded on video for accountability and for statistical analysis purposes. The setup of the game environment, the strategy of the iCat, and the type of speech produced were controlled for throughout the experiment.

The procedure used was as follows. Each participant enters the room and is greeted by the main experimenter and directed to his seat. It is then explained that he will play a game of Go Fish against each iCat. The gaming interface for Go Fish is explained and it is explicitly explained to the participant that the iCat has its own screen to watch Go Fish on. The participant then is informed that the experimenters will be sitting behind him in order to monitor whether things go well. The participant is then asked whether he is already acquainted with the rules of Go Fish. Depending on the answer more or less time is spent on explaining the rules. Next the interface and the use of the mouse to control the game is explained. A piece of paper is put near the screen which shows which

colors a complete set consist of; on the paper also the rule that you are only allowed to ask for a card of a rank if you have another card of that same rank is printed. The questionnaire is also briefly discussed to check whether there are any problems with understanding any of the words and to explain that there are 3 different questionnaires.

5.7 Results

Various methods have been used to verify whether the socio-cognitive iCat is rated more positively than the ego-reactive iCat. Three questionnaires were developed which subjects filled in at different stages of the experiment. These include questions about e.g. the friendliness and how much fun it was to play with an iCat. Subjects were asked to rate questions on a 5-point Likert scale. One open question asking for an explanation of a preference for either one of the iCats was used. Additionally, various observations were made such as how often the subject looked at the iCat, how often the subject smiled or laughed, and the percentage of time a subject (or the iCat) was having the upper hand (higher game score). Finally, various events from the game have been logged, including among others who won and how often a subject cheated by refuting a card it did own.

To obtain results from the data collected, one-tailed significance is measured. A result indicates a *trend* whenever significance is below 0.10 and indicates a *significant result* whenever significance is below 0.05. Due to space limits, we only present the main results. Detailed findings are available in [19].

Variable	Ego-Reactive	Socio-Cognitive	Neg. Rank	Pos. Rank	Ties	Significance
Friendliness	4.50	4.50	4	4	12	1.000
Loss handling	4.50	4.30	5	2	13	0.412
Fun	4.60	4.75	1	3	16	0.129
Observed Valence	5.25	5.90	5	10	5	.028
Observed Arousal	5.10	5.45	5	10	5	.090

Table 2. Wilcoxon Signed Rank Tests

Tables 2 and 3 summarize our main findings. The first three rows in Table 2 concern the perception of the iCat; results were obtained by a questionnaire. A clear ceiling effect can be observed. It is interesting, however, to see that there is a trend in measured expected fun. Children rated on a five-point Likert scale how much fun they expected to have while playing with both iCats and rated again afterwards how much fun it actually was. The average before was 4.45 (SD=.605) and afterwards 4.70 (SD=.657). The significance is .096 which indicates that there is a trend related to fun.

The most interesting result is that the observed valence of the child while playing with the socio-cognitive iCat is significantly higher than with the ego-reactive iCat. This is a clear indication that supports our hypothesis that the

Variable	Ego-Reactive	Socio-Cognitive	Significance
Laughing	.30 (.801)	.70 (1.593)	.060
Talking time	34.7 (11.4)	36.3 (10.9)	.164
Looking behavior	20.30 (13.6)	26.45 (11.5)	.037
Subject winning time	13.0 (19.0)	23.3 (20.2)	.027
iCat winning time	31.4 (23.3)	25.9 (25.6)	.199

Table 3. T-Test

socio-cognitive iCat is evaluated more positively than the ego-reactive one. A trend was measured with respect to observed arousal of a child. This can possibly be explained because children get more aroused when they have a higher chance of winning during the game (see also the item *Subject winning time* in Table 3).

There is not much difference in how much time the children talk to each iCat. This may be explained because practically all of the speech that the children produced is related to the game. Looking behavior, however, is significantly different. Children look more often at the socio-cognitive iCat than at the ego-reactive iCat. The co-experimenter who observed the children during the experiment mentioned that children seem to look more often at the iCat if they are happier. When placing these results next to the observed valence this does seem to be the case, but after formal analysis the results remain inconclusive and this claim cannot be confirmed.

5.8 Discussion

We conclude that there is evidence that supports our main hypothesis that the socio-cognitive iCat is evaluated more positively than the ego-reactive iCat. Strong evidence in support stems from the fact that children looked happier when playing against the socio-cognitive iCat than against the ego-reactive iCat. Another relevant observation concerns the fact that on average children were doing better when playing against the socio-cognitive robot. The hint system may have contributed to this performance. Moreover, it is also clear that the gaming experience is more positive with the socio-cognitive iCat compared with the ego-reactive iCat.

On average, the socio-cognitive iCat tried 0.55 times to lose a set, gave 2.05 hints and 1.2 cheer-ups per child as opposed to the ego-reactive iCat which does not show this behavior. The data shows a child who got nine “cheer ups”. Interestingly, this child first played against the ego-reactive iCat and won and then played a game against the socio-cognitive iCat and lost. Another interesting observation concerns a child who received 7 hints from the socio-cognitive iCat. He did lose in the end, but again the observed valence was higher than with the ego-reactive iCat.

During the experiment, it was interesting to see that some children started to copy sentences the iCat said. Children started to ask cards in the same way, refute cards in the same way, etc. This type of mirroring of the iCat’s behavior by the children can be seen as an attempt to understand the iCat’s actions and

emotions. It is not known whether this mirroring is done more with one iCat than with the other, but this can perhaps be interpreted as a sign of observed intentional behavior by the children.

6 Conclusion

Two robots have been developed which both have the capabilities to play Go Fish and interact with a child in a semi-autonomous way. A platform has been developed that can be (re)used for research on socio-cognitive human-robot interaction. The experimental design is based on a Wizard-of-Oz setup since automatic speech and emotion recognition is not yet able to provide the required levels of recognition. Future work is needed to investigate extensions to fully automate the recognition tasks of the robot.

The behavior repertoire of the socio-cognitive iCat is an extension of the behavior of the ego-reactive iCat. The ego-reactive iCat bases its interaction solely on his own state of mind whereas the socio-cognitive robot also takes the human with which it interacts into account. Experimental results support our hypothesis that the behavior of the socio-cognitive iCat is evaluated more positively than that of the ego-reactive iCat. The evidence that supports the main hypothesis is derived primarily from the observation that there was a significant difference between how happy the children appeared to be when they looked at each iCat. While playing with the socio-cognitive iCat, the children were smiling more and their overall mood was better than with the ego-reactive iCat. The results suggest that children have relatively high expectations of the robot, but also indicate that after the experiment their expectations are more than met. At the start of the experiment, before playing with the iCat, children estimate their fun with the iCat to be 4.55 on average, on a 5 point scale. After playing two games with the two different iCats, children rated it to be a 4.70 average. These results indicate a trend but more research is needed to show these results are significant.

An interesting setting to study in future work concerns the more involved scenario of more than two players. During our studies many people asked whether they could play against both iCats.

The emotion model used in this research has been based on a valence range. This is a one dimensional emotion mapping. There are other and more advanced models to represent emotions, for instance, the well-known Pleasure, Arousal, Dominance model. This model allows to map possible emotions onto a three dimensional space.

Concluding, we believe that our results show that the software platform has been and in the future can successfully be applied and used as a research platform to study effects of socio-cognitive abilities of robots on human-robot interaction.

References

1. Fong, T., Nourbakhsh, I., Dautenhahn, K.: A survey of socially interactive robots. Robotics and Autonomous Systems **42** (2003) 143–166

2. Dautenhahn, K.: Design spaces and niche spaces of believable social robots. In: Proceedings of the International Workshop on Robots and Human Interactive Communication. (2002) 192–197
3. Kozima, H., Yano, H.: In search of ontogenetic prerequisites for embodied social intelligence. In: Proceedings of the Workshop on Emergence and Development on Embodied Cognition; International Conference on Cognitive Science. (2001) 30–34
4. Kozima, H., Yano, H.: A robot that learns to communicate with human caregivers. In: Proceedings of the International Workshop on Epigenetic Robotics. (2001) 47–52
5. Breazeal, C., Scassellati, B.: How to build robots that make friends and influence people. In: Proceedings of the International Conference on Intelligent Robots and Systems. (1999)
6. Heerink, M.: The influence of a robot's social abilities on acceptance by elderly users. In: RO-MAN, Hertfordshire. (2006) 521–526
7. Looije, R., Neerincx, M.A., de Lange, V.: Children's responses and opinion on three bots that motivate, educate and play. Journal of Physical Agents **2**(2) (2008) 13–20
8. Kessens, J.M., Neerincx, M.A., Looije, R., Kroes, M., Bloothooft, G.: Facial and vocal emotion expression of a personal computer assistant to engage, educate and motivate children. In: Third IEEE International Conference on Affective Computing and Intelligent Interaction (ACII 2009). (2009)
9. Looije, R., Neerincx, M.A., Cnossen, F.: Persuasive robotic assistant for health self-management of older adults: Design and evaluation of social behaviors. International Journal of Human-Computer Studies **68** (2010) 386–397
10. Leite, I., Martinho, C., Pereira, A., Paiva, A.: iCat: an Affective Game Buddy Based on Anticipatory Mechanisms (Short Paper). In: Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008). (2008) 1229–1232
11. Malone, T.W.: Heuristics for designing enjoyable user interfaces. lessons from computer games. In: Proceedings of the 1982 Conference on Human factors in Computing Systems. (1982) 63–68
12. Wikipedia. http://en.wikipedia.org/wiki/Go_fish (2011)
13. Minato, T., Shimada, M., Ishiguro, H., Itakura, S.: Development of an android robot for studying human-robot interaction. In: Proceedings of the 17th International Conference on Innovations in Applied Artificial Intelligence. (2004) 424–434
14. Brooks, A.G., Arkin, R.C.: Behavioral overlays for non-verbal communication expression on a humanoid robot. Autonomous Robots **22**(1) (2006) 55–74
15. Hindriks, K.V.: Programming Rational Agents in Goal. In: Multi-Agent Programming. Springer (2009) 119–157
16. Schlossberg, H.: Three dimensions of emotion. Psychology Review **61** (1954)
17. Rizzolatti, G., Craighero, L.: Mirror neuron: a neurological approach to empathy. Manuscript (2005)
18. Lee, C., Lesh, N., Sidner, C.L., Morency, L.P., Kapoor, A., Darrell, T.: Nodding in conversations with a robot. In: Proceedings of the Conference on Human Factors in Computing Systems. (2004) 785–786
19. Vink, M.: The iCat as a Natural Interaction Partner: Play Go Fish with a Robot. Master's thesis, Delft University of Technology (2009)

Individual Localization and Tracking in Multi-Robot Settings with Dynamic Landmarks

Anousha Mesbah and Prashant Doshi

THINC Lab, Dept. of Computer Science
University of Georgia
Athens, GA 30602
anousha@uga.edu, pdoshi@cs.uga.edu

Abstract. Accurate and robust localization is a key challenge toward achieving successful robot navigation and path planning in new environments. Previously, particle filtering has been shown to be an effective approach for meeting this challenge. However, the contexts have typically been single robot settings or involved the presence of other agents such as humans who have been modeled naively. In this paper, we present a new method that generalizes particle filtering to allow for the presence of other robots, possibly non-cooperative, sharing the environment, and who may disturb the locations of landmarks previously known to the subject robot. Our approach results in good localization of the subject robot and tracking of the other robot in these complex settings, and serves to make localization more robust and practically applicable.

1 Introduction

A class of approaches that have shown promise in localizing are probabilistic and use particles (samples) to indicate the potential pose of the mobile robot [14]. The particles are projected using a technique representative of the Bayes filter [11] as the robot moves and observes. Particle filtering leads to good localization performance despite limitations such as needing a large number of particles and difficulty in capturing low probability outcomes. However, its use has been limited to single robot settings or those involving other agents such as humans which are modeled as random noise in the environment. While variants of particle filtering have been fielded in museum tour guide robots [4, 5], these have focused mostly on achieving good performance in single robot localization and mapping while modeling others minimally.

In this paper, we generalize particle filtering to multi-robot settings in order to localize the subject robot in a partially observable environment using landmarks and simultaneously track the uncertain location of the other robot(s). We assume that the exact locations of the landmarks are known to the subject robot. Our focus is on the subject robot's localization at its own level in the presence of others who may not be cooperative. Consequently, our approach differs from previous work in multi-robot settings, which has predominantly focused on joint localization by multiple cooperating robots [6, 8, 10, 12]. In this context, we introduce a nested set of particles to track the subject robot and others, and recursively project these particles as the subject robot moves and makes observations. Because the robots sharing the environment need not

be part of a team, communication between them may not be possible. Consequently, the subject robot attributes a behavioral model to the other in order to predict its actions.

We extend Rosencratz et al.'s *laser tag* domain [9] for our experimentation. In particular, we generalize the problem by assuming that the subject robot is itself not localized. Motivated by the challenges faced in search and rescue, we require the subject robot to tag the other and then seek to reach the opponent's base. On being tagged, the opponent robot may move the nearest landmarks in order to confound the subject's localization. This is analogous to independent rescue robots moving obstacles while searching for victims. This has the effect of delaying the robot's approach toward the base. We simulate the laser tag domain and perform our experiments in a 3D environment using Microsoft's Robotics Developer Studio. Despite the presence of dynamic landmarks, we demonstrate that the subject robot using our approach localizes well and tracks the other robot satisfactorily. While previous approaches isolate dynamic landmarks and do not use them for localization, our approach continues to utilize them particularly because landmarks may be few. The divergence between the particle-based distribution and the actual locations decreases progressively as the game proceeds.

Rest of the paper is structured as follows. We briefly discuss related work in Section 2. Localizing a single robot using the standard particle filter is described in Section 3. We generalize this particle filter to multi-robot settings using nested particle sets in Section 5. The algorithm for implementing our approach is outlined in Section 6. We provide the experimental evaluation of the localization performance in Section 7, and conclude the paper with a discussion in Section 8.

2 Related Work

Research on localization has traditionally focused on single robot settings where the particle filter applied under the name *Monte Carlo localization* has proved to be successful [14, 16]. Other types of localization include Markov localization [5] and localization using the extended Kalman filter [13]. Analogous to Monte Carlo localization, Markov localization utilizes the Bayesian filter to update the pose uncertainty related to the robot. However, it maintains a full probability distribution over the poses of the robot thereby requiring compact ways to represent large state spaces. The extended Kalman filter permits the application of the exact Kalman filter in non-linear environments by essentially linearizing the state transition and observation models about the current estimate. In contrast to the latter approaches, particle filters maintain a discrete set of hypothesized poses and project these across time.

In the multi-robot setting, research has predominantly focused on cooperative localization and mapping. Kurazume and Nagata [6] describe a cooperative positioning system in which other agents act as landmarks. Robots alternate in moving in the environment and remaining stationary to be considered as a landmark for other robots while they move. The robots communicate their positions relative to other agents' positions. A robot uses a Kalman filter to estimate its own position and of other agents due to noisy sensory information. Building on this work, Navarro-Serment et al. [8] developed a system of localizing and mapping for a team of robots in which a group of robots serve as beacons. Analogous to this work, Tully et al. [12] propose a leap-frog path

planning algorithm that uses the cooperative positioning method described previously in order to improve the accuracy of localization. All of these approaches exploit cooperation between the robots and focus on jointly localizing multiple robots. In contrast, we focus on the individual localization of a subject robot when faced with an environment involving multiple robots who may be non-cooperative or neutral.

Research has also studied localization with dynamic landmarks by maintaining a separate data structure for objects that are identified as transient, and localizing using stationary objects only [5, 17]. Another line of research seeks to localize in the presence of humans thereby posing the problem as involving simultaneous localization and people tracking [5]. Simple Brownian motion models are utilized to track the movement of people in the environment while localization is performed with respect to stationary landmarks. While previous research discriminates between fixed and dynamic landmarks, our approach allows us to use both types of landmarks toward localization, partly by modeling the movement of the landmarks due to actions of other robots.

Recursive projection of nested particle sets is analogous to the principled approach in the interactive particle filter [1, 2]. In the latter, nested particle sets represent the other agent's possible beliefs, and the approach predicts likely observations of the other agent in order to project the particles. In comparison, the subject robot's observations of the other are used to project the nested particles, which represent possible locations of the other robot. Consequently, our approach could be seen as adapting the interactive particle filter toward robotic applications.

3 Background: Localization Using Particle Filter

We briefly review traditional particle filtering (PF) for localizing a robot in partially observable settings in the context of noisy sensors. This approach has also been called Monte Carlo localization previously [16]. We focus on localizing using landmarks whose exact locations are known *a priori* to the robot.

Each particle, $\mathbf{x}^{(n)}: \quad n = 1 \ldots N$, represents a possible pose of the robot, $\mathbf{x}^{(n)} = \langle x, y, \theta \rangle$, where x, y are the x and y coordinates of the center of the robot in a two dimensional plane respectively, and θ is the angle of its orientation w.r.t. the positive x axis. Traditional PFs project a particle as the robot acts and observes using the three steps of *propagation*, *weighting* and *resampling*. We illustrate this process in Fig. 1 and describe the steps below:

• *Propagation*: Given an action, a, which causes the robot to move for time $\triangle t$ with given translational velocity v and rotational velocity, w, we utilize the motion model to propagate each particle:

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} \leftarrow \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{\hat{v}}{\hat{w}} sin\theta + \frac{\hat{v}}{\hat{w}} sin(\theta + \hat{w}\triangle t) \\ \frac{\hat{v}}{\hat{w}} cos\theta - \frac{\hat{v}}{\hat{w}} cos(\theta + \hat{w}\triangle t) \\ \hat{w}\triangle t + \hat{\gamma}\triangle t \end{pmatrix} \tag{1}$$

where $\hat{v} = v + \epsilon_v$ and $\hat{w} = w + \epsilon_w$ are the actual translational and rotational velocities, respectively. $\hat{\gamma}$ is the final rotation that the robot makes when it arrives at its destination. We sample the error terms, ϵ_v and ϵ_w, from a zero-mean Gaussian density with variance

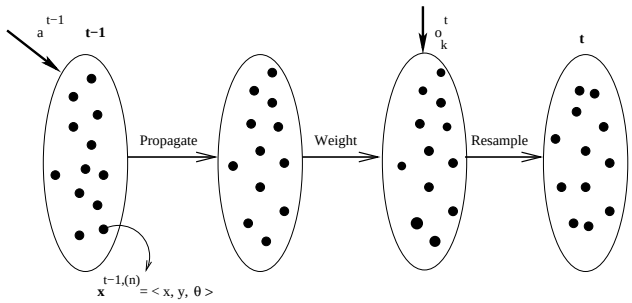


Fig. 1. Conceptual illustration of PF (Monte Carlo localization). Each particle representing a hypothesized pose of the robot is propagated, weighted and resampled in order to obtain its projection at time t.

that is a function of the motion noise and the velocities. We denote this motion model as, $p(\mathbf{x}^t|u^t, \mathbf{x}^{t-1})$, where $\mathbf{x}^{t-1}$ is the pose at time $t-1$ and u^t is the velocity vector $(v, w)^T$. The propagated particle is thus, $\mathbf{x}^{t,(n)} \sim Pr(\mathbf{x}^t|u^t, \mathbf{x}^{t-1,(n)})$.

• *Weighting*: On observing a landmark k distinguished by the feature vector, $o_k^t = (r_k^t, \phi_k^t, c_k^t)$, we weight each particle with the likelihood of the observation if the robot's pose matched that in the particle. Here, r_k^t is the average range of the detected landmark obtained from a laser range finder, ϕ_k^t is the difference in angle between the orientation of the particle when the robot made the observation and the bearing of the observed landmark (from the hypothesized position of the robot) and c_k^t is the observed color of the landmark.

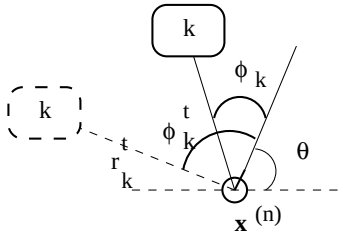


Fig. 2. Illustration of ϕ_k^t and ϕ_k for a particle. θ is the orientation of the hypothesized pose of the robot. The observed range, r_k^t, and the angular range, ϕ_k^t, places the observed landmark at the location of the dashed rectangle. The solid rectangle is the true location of the landmark.

Let the true feature vector be $f_k = (r_k, \phi_k, c_k)$, where r_k is the true range based on the exact location of the landmark, ϕ_k is the difference between the orientation of the particle and the bearing of the landmark based on its exact location, and c_k is the landmark's true color. We illustrate ϕ_k^t and ϕ_k in Fig. 2. In order to weight a particle,

$\mathbf{x}^{t,(n)}$, we compute the following:

$$\begin{aligned} \hat{r} &= \sqrt{(k_x - x)^2 + (k_y - y)^2} \\ \hat{\phi} &= atan2(k_y - y, k_x - x) \end{aligned} \tag{2}$$

where k_x, k_y is the exact location of the landmark, k, and x, y is obtained from the particle, $\mathbf{x}^{t,(n)}$. The weight is then, $\mathcal{N}(r_k^t - \hat{r}, \sigma_r) \times \mathcal{N}(\phi_k^t - \hat{\phi}, \sigma_\phi) \times \mathcal{N}(c_k^t - c_k, \sigma_c)$, where $\mathcal{N}(\mu, \sigma)$ represents the Gaussian distribution with mean, μ, and deviation, σ. We denote the likelihood as $Pr(o_k^t | f_k, \mathbf{x}^{t,(n)}, m)$, where m is the map giving the exact locations of the landmarks.

• *Resampling:* We utilize a simple scheme for generating unweighted particles at time t by resampling them using the normalized weights as the sampling distribution. This method leads to more particles in the high likelihood regions of the state space, as determined by the observations.

On performing these three steps, we obtain a set of projected particles which are representative of the probable poses of the robot at the next time step, t.

4 Multi-Robot Laser Tag

Motivated by the challenges posed by other independent robots who could be moving potential landmarks, we modify Rosencratz et al.'s [9] laser tag problem domain.

We adopt the perspective of a robot i whose task is to tag another robot j and then proceed to reach j's base within a certain amount of time steps. The physical environment shared by both robots is populated by multiple objects of different colors and sizes. Two of these objects are distinguished and serve as bases of each robot. We show the laser tag environment in Fig. 3. As we focus on localization in this paper, we assume that robots i and j know the exact locations of these objects. However, they are unaware of their individual or each other's locations in the environment and do not have access to any device, which can inform them about this. [1] Consequently, the objects may serve as landmarks whose observations could be used by robot i for localizing itself in the environment. Unfortunately, the presence of multiple objects with the same color means that the localizing information is often ambigious.

We equip each robot with a laser range sensor which provides range information about objects in front of the robot, a color sensor whose image data is segmented into the major color patches and colors of these patches identifi ed, and a bump sensor which signals if a robot hits an obstacle. Each robot's set of actions consist of moving straight for some time, $\triangle t$, and rotating left or right by 90 degrees. Embedded within the move action is a collision avoidance procedure that activates if the robot anticipates colliding with an obstacle during the course of its action. On being tagged, the collision avoidance of robot j is suspended and it may proceed to push an object if it hits it. We accomplish this by making the objects suffi ciently light. Robot j is considered tagged if it is detected by i and its range is within a small threshold.

[1] Despite possibly having GPS devices, robots are often inhibited by the fact that these devices do not work well inside buildings.

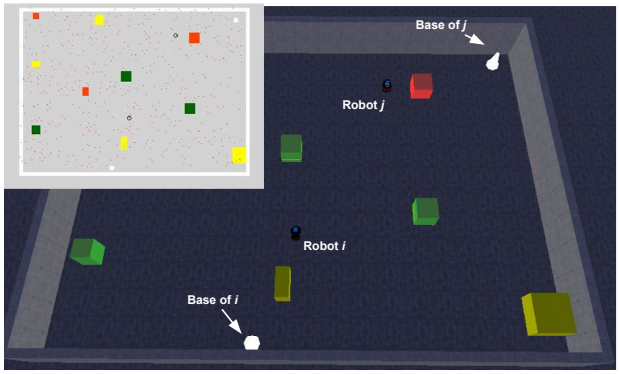


Fig. 3. Our laser tag environment simulated in Microsoft Robotics Developer Studio consisting of two robots (AmigoBots) along with differently colored objects. The two cones are the bases of robot i and j, respectively. The top view of the 3D environment (map) is shown embedded.

Robot j utilizes a mixed strategy behavioral model that is in play until j is tagged. The behavioral model takes as input the hypothesized pose of a robot and prescribes moving straight with a probability of 0.75, and rotating left or right with a probability of 0.125 respectively. If robot j is tagged, i then uses simple path planning to move toward j's base while j proceeds toward the nearest object with the aim of displacing it thereby possibly disturbing i's localization and slowing its progress toward j's base.

5 Individual Localization in Multi-Robot Settings

Within the laser tag domain described in Section 4, it is benefi cial for robot i to predict j's action at each time step in order to track j. This will allow i to anticipate j's pushing of the landmark and subsequently i could update its map with the new potential location of the landmark. Consequently, i may continue using observations of the dynamic landmarks toward its localization.

5.1 Nested Particle Filtering

We generalize the basic PF of Section 3 in order to apply it to settings shared with others. For each of its hypothesized pose (particle), robot i maintains a set of hypothesized poses for j. This differs from a joint pose for both i and j, which is applicable when both robots must localize and an objective or external view of the multi-robot setting is taken. *In contrast, we adopt a subjective view from the perspective of* i. Hence, each particle of i contains i's pose and a set of particles reflecting j's possible poses. Formally, a particle for robot i is, $\mathbf{x}_i^{t-1,(n)} = \langle (x,y,\theta)_i, \mathbf{x}_j^{t-1} \rangle$, where $\mathbf{x}_j^{t-1}$ is the set of j's particles. Each particle, $\mathbf{x}_j^{t-1,(n)} \in \mathbf{x}_j^{t-1}$ is $\langle x,y,\theta \rangle_j$ representing the pose of j. The nested PF targets settings where subject robot i itself is not localized and simultaneously seeks to track j. We illustrate this nested set of particles in Fig. 4.

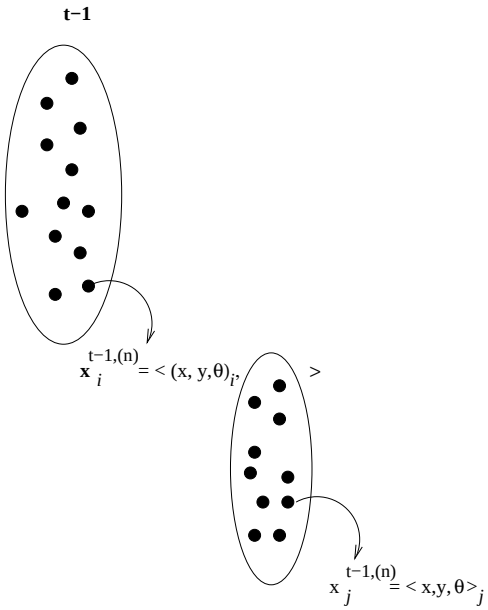


Fig. 4. Nested set of particles for robot i.

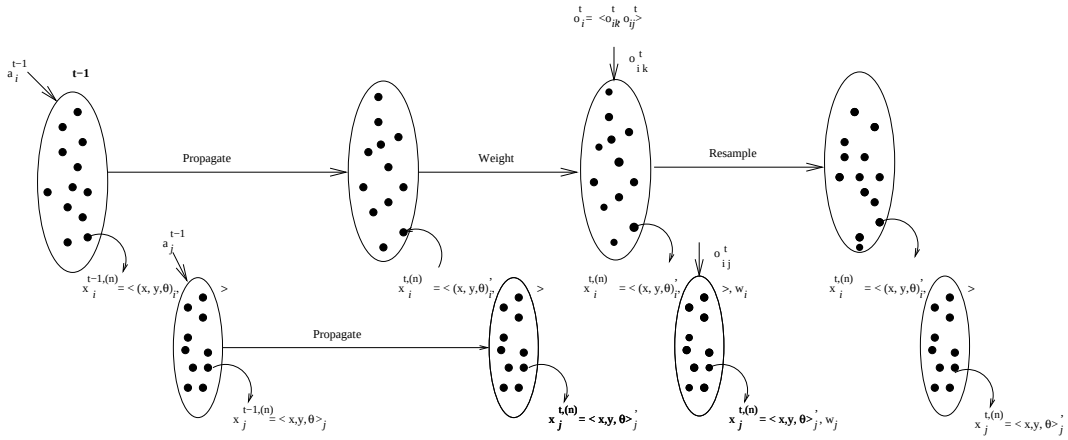


Fig. 5. Recursively propagating, weighting and resampling j's particles as part of the particle filtering process for i. This allows robot i to maintain an updated particle set for j as well.

If the problem setting is highly interactive, the nesting could continue further with each of j's particles containing a set of i's particles and so on. This is useful if i thinks that j may not be localized either and that j thinks that its localization depends on i. In this paper, we focus on a single level of nesting by assuming that j is perfectly aware of its own location and does not track i, at any time in the laser tag environment.

As robot i acts and observes, it projects the nested set of particles (Fig. 4) across time. Let $m_j = \triangle(A_j)$ where A_j is the set of j's possible actions, be a mixed-strategy behavioral model that i believes j possesses. Let a_j^{t-1} be an action sampled from the distribution, $a_j^{t-1} \sim m_j$. During the propagation step, j's particles are propagated as well based on its predicted action, a_j^{t-1}, in addition to propagating i's pose. The propagation for j is analogous to the procedure described in Section 3, $\mathbf{x}_j^{t,(n)} \sim Pr(\mathbf{x}_j^t | u_j^t, \mathbf{x}_j^{t-1,(n)})$.

Robot i's observation is now two-fold, $o_i^t = \langle o_{ik}^t, o_{ij}^t \rangle$: it may observe a landmark, o_{ik}^t, and robot j, o_{ij}^t, using its laser range finder and color sensor. Observations of robot j are used to weight the nested set of j's particles for each particle of i. Finally, the resampling of i's particles includes resampling j's particles using the particle weight as the sampling distribution. Each of the three steps, propagation, weighting and resampling may be carried out recursively in nesting depth for j's set of particles. The final outcome is a set of i's particles each of which containing a nested set of j's particles, which have been projected to the next time step. Thus, robot i maintains an updated set of j's possible poses for each of its own hypothesized pose. We illustrate the recursive particle filtering in Fig. 5.

Because weighting j's particles using i's observations of j is not straightforward, we provide some details about this procedure next.

5.2 Weighting Other's Particles

Intuitively, if o_{ij}^t is indicative of j being spotted, then all particles of j for some particle of i, $\mathbf{x}_i^{t,(n)}$, that are in the vicinity of i whose pose is given by the particle should receive high weights. Otherwise, the weight should be low. Specifically, each particle of j contained in $\mathbf{x}_i^{t,(n)}$ is weighted by the likelihood, $Pr(o_{ij}^t \mid \mathbf{x}_j^{t,(n)}, \langle x, y, \theta \rangle_i)$, where $\langle x, y, \theta \rangle_i$ is i's pose in the particle.

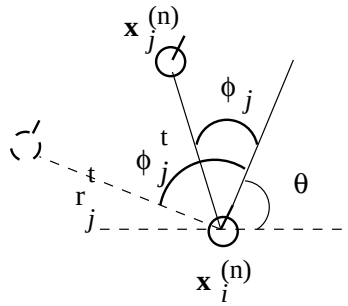


Fig. 6. Weighting j's particle given an observation of j by i.

Let $o_{ij}^t = \langle r_{ij}^t, \phi_{ij}^t, c_{ij}^t \rangle$ be the observed vector of information about j. Although i is unaware of the true location of j (but knows the color of j to facilitate detection), the

likelihood is calculated analogously to Fig. 2. As we show in Fig. 6, we use the pose in j's particle to obtain ϕ_j and the observed range and bearing of j to obtain ϕ^t_{ij}. Next, we calculate:

$$\hat{r} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$$
$$\hat{\phi} = atan2(y_j - y_i, x_j - x_i)$$

where (x_i, y_i) and (x_j, y_j) are the x and y locations in i's and j's particles, respectively. The weight associated with j's particle, $\mathbf{x}_j^{t,(n)}$ is then, $\mathcal{N}(r^t_{ij} - \hat{r}, \sigma_r) \times \mathcal{N}(\phi^t_{ij} - \hat{\phi}, \sigma_\phi) \times \mathcal{N}(c^t_{ij} - c_j, \sigma_c)$.

Finally, we point out that we utilize negative observations as well for weighting both i's and j's particles. For example, if no landmark is detected, particles with poses of i near landmarks receive smaller weights than those with poses from which no landmark could be detected. For particles near landmarks, the observation likelihood is obtained analogously to Fig. 2, except that in the observation vector, r^t is the minimum of all the ranges returned by the laser range finder, ϕ^t is the bearing of this range and c^t is the observed color of the background, and k is the landmark nearest to the particle. For particles away from any landmark, the errors are zero and a high likelihood is obtained from the Gaussian densities.

6 Algorithm

We show the algorithm for the PF generalized to multi-robot settings in Fig. 7. It takes as input a nested particle set, $\mathbf{x}_i^{t-1}$, the action of robot i, a_i^{t-1}, and its possible observation of a landmark or the other robot j and generates the projected nested particle set, $\mathbf{x}_i^t$. The initial particle set is generated by randomly picking i's pose N_i times and for each i's particle, picking j's pose randomly N_j times.

LOCALIZATION ($\mathbf{x}_i^{t-1}$, a_i^{t-1}, o_i^t=$\langle o_k^t, o_{ij}^t \rangle$) **returns** $\mathbf{x}_i^t$

1: $\mathbf{x}_i^t \leftarrow$ PROPAGATE ($\mathbf{x}_i^{t-1}$, a_i)
2: $\mathbf{x}_i^t, \hat{\mathbf{w}}_i \leftarrow$ WEIGHT ($\mathbf{x}_i^t$,o_i^t)
3: $\mathbf{x}_i^t \leftarrow$ RESAMPLE ($\mathbf{x}_i^t, \hat{\mathbf{w}}_i$)
4: **return** $\mathbf{x}_i^t$

Fig. 7. Particle filtering in a multi-robot setting using generalized propagation, weighting and resampling.

Each of the three steps in the PF, propagation, weighting and resampling, differ from those in Section 3 in that they deal with particles for the other robot as well. As we show in Figs. 8, 9 and 10, each of the methods recursively invokes the corresponding procedure on the nested sample sets. Therefore, in projecting i's particles, j's particles are projected across time as well.

If we consider N particles for both robots i and j, then the complexity of this approach is due to projecting $\mathcal{O}(N^2)$ particles. We may mitigate the complexity by al-

PROPAGATE ($\mathbf{x}_i^{t-1}$,a_i^{t-1}) **returns** $\mathbf{x}_i^t$

for $\mathbf{x}_i^{t-1,(n)} \in \mathbf{x}_i^{t-1}$ **do**
  Propagate the pose, $(x, y, \theta)_i^{t-1}$, in $\mathbf{x}_i^{t-1,(n)}$ using Eq. 1 to obtain $(x, y, \theta)_i^t$
  if $\mathbf{x}_i^{t-1,(n)}$ contains a nested particle set, $\mathbf{x}_{-i}^{t-1,(n)}$ **then**
    Predict action, a_{-i}^{t-1}, using model, m_{-i}
    $\mathbf{x}_{-i}^t \leftarrow$ PROPAGATE ($\mathbf{x}_{-i}^{t-1}$, a_{-i}^{t-1})
    $\mathbf{x}_i^{t,(n)} \leftarrow \langle (x, y, \theta)_i^t, \mathbf{x}_{-i}^t \rangle$
  end if
  $\mathbf{x}_i^t \stackrel{+}{\leftarrow} \mathbf{x}_i^{t,(n)}$
end for
return $\mathbf{x}_i^t$

Fig. 8. Recursively propagating the nested particle sets. Here, $-i$ denotes the robot j in a two-agent setting.

WEIGHT ($\mathbf{x}_i^t$,$o_i^{t-1} = \langle o_{ik}^t, o_{i-i}^t \rangle$) **returns** $\mathbf{x}_i^t$, $\hat{\mathbf{w}}$

for $\mathbf{x}_i^{t,(n)} \in \mathbf{x}_i^t$ **do**
  Calculate, $\hat{r}$,$\hat{\phi}$ using Eq. 2
  Weight $\mathbf{x}_i^{t,(n)}$: $\hat{w}_i^{(n)} \leftarrow \mathcal{N}(r_k^t - \hat{r}, \sigma_r) \times \mathcal{N}(\phi_k^t - \hat{\phi}, \sigma_\phi) \times \mathcal{N}(c_k^t - c_k, \sigma_c)$
  if $\mathbf{x}_i^{t,(n)}$ contains a nested particle set, $\mathbf{x}_{-i}^t$ **then**
    $\mathbf{x}_{-i}^t, \hat{\mathbf{w}}_{-i} \leftarrow$ WEIGHT ($\mathbf{x}_{-i}^t$, o_{i-i}^t)
    $\mathbf{x}_i^{t,(n)} \leftarrow \langle (x, y, \theta)^t, (\mathbf{x}_{-i}^t, \hat{\mathbf{w}}_{-i}) \rangle$
  end if
  $\mathbf{x}_i^t \stackrel{+}{\leftarrow} \mathbf{x}_i^{t,(n)}$
end for
return $\mathbf{x}_i^t$, $\hat{\mathbf{w}}_i$

Fig. 9. Robot j's particles are weighted by recursively invoking the procedure on j's particle sets for each particle of i.

locating a lesser number of particles for tracking j thereby giving priority to i's own localization.

7 Experiments

We implemented the algorithm for the nested particle fi ltering outlined in Section 6 in a 3D environment and evaluate it in the context of the non-cooperative multi-robot laser tag environment described in Section 4. As mentioned before, the robots are equipped with 2D laser scanners, cameras for the purpose of color detection and tactile sensors. Note that the observation of the subject robot i based on its laser readings as well as its motion controls are subject to noise. While we did not add noise explicitly to the

RESAMPLE ($\mathbf{x}_i^t$,$\hat{\mathbf{w}}_i$) **returns** $\mathbf{x}_i^t$

1: **for** $\mathbf{x}_i^{t,(n)} \in \mathbf{x}_i^t$ **do**
2: **if** $\mathbf{x}_i^{t,(n)}$ contains a nested particle set **then**
3: $\mathbf{x}_{-i}^t \leftarrow$ RESAMPLE ($\mathbf{x}_{-i}^t$,$\hat{\mathbf{w}}_{-i}$)
4: $\mathbf{x}_i^{t,(n)} \leftarrow \langle (x, y, \theta)^t, (\mathbf{x}_{-i}^t) \rangle$
5: **end if**
6: **end for**
7: Normalize the weights, $\hat{\mathbf{w}}_i$
8: **for** $n \leftarrow 1$ to N_i **do**
9: Sample $\mathbf{x}_i^{t,(n)}$ using the associated normalized weights as the distribution
10: $\mathbf{x}_i^t \stackrel{+}{\leftarrow} \mathbf{x}_i^{t,(n)}$
11: **end for**
12: **return** $\mathbf{x}_i^t$

Fig. 10. Normalizing and then resampling both robots' particles using a recursive invocation.

simulator, we noticed that the simulator does not respond perfectly to our commands, thereby simulating noise as in a physical robot.

Within the laser tag environment, our evaluation scenario involves robot i making observations of the green landmark near the center of the environment and then moving with the aim of tagging robot j. As we mentioned previously, tagging is accomplished by identifying robot j when it is in close proximity of i. On being tagged, robot j proceeds to the nearest landmark to push it (in this case the red landmark near its base) while i proceeds to j's base. On its way to the base, i may observe the pushed landmark. Until j is tagged, j utilizes the mixed strategy behavioral model described in Section 4, which is known to i. Robot i's task is complicated by the presence of j who may push landmarks. We compare the performance of our localization with two other approaches that represent alternate ways of dealing with dynamic environments. Both these candidate approaches deal with the dynamism in different ways. The first is the simple method of remaining oblivious to the dynamism. In our context, robot i although being aware that landmarks may be pushed continues to utilize the original map of the landmarks for its localization. We label this approach as Original Map. The second is the method of Wolf and Sukhatme [17], which essentially seeks to identify those landmarks that are dynamic in order to avoid using them for localization. Within our context, we use j's particles to ascertain which landmark could be pushed by j and do not use that landmark(s) toward robot i's localization anymore. We label this approach as Wolf. Consequently, we comprehensively evaluate the performance of our approach, which is labeled as Nested PF.

We show the laser tag environment, simulated in Microsoft's Robotics Developer Studio, when i tags j, in Fig. 11. Additionally, we show the map with i's particles in red and a map showing all of j's particles – for every i's particle – in blue at this point in the game. Notice the two distinct groups of i's particles, which implies that i hasn't localized well up to this point. However, one of these groups of particles reflects i's ac-

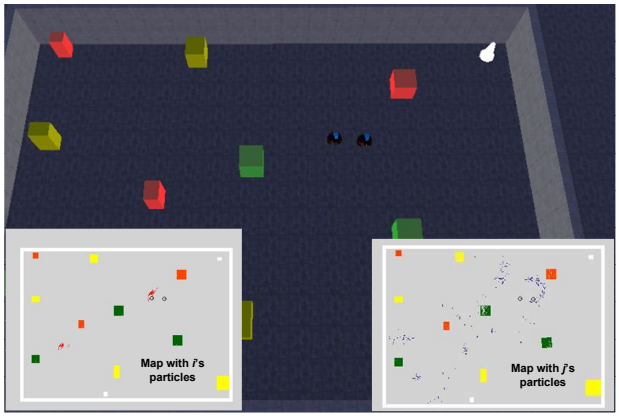


Fig. 11. The laser tag environment phyiscally simulated in Microsoft Robotics Developer Studio when robot i tags j. We show the map with i's particles (in red) bottom left and the map with all of j's particles (in blue) bottom right. Notice the group of j's particles around the actual pose of j. This is due to the range information for j obtained during the tagging.

tual pose. On the other hand, tagging robot j leads to many particles for j being around j's actual pose. However, groups of particles continue to appear elsewhere because i itself is not well localized.

What remains to be clarified is how we update the map of landmarks that i refers to during localizing if it thinks that j has moved a landmark. First, note from the description of the laser tag environment that j possibly pushes landmarks only after it has been tagged. On tagging j, i predicts which of j's particles are sufficiently close to a landmark such that j could be hypothesized to move that landmark. For particles that could have pushed landmark(s), we compute the new locations of the landmarks. For all other particles, the new locations are deemed to be identical to the original landmark locations. In order to update the map, i averages the new locations of all the landmarks across all particles of j for every i's particle. This technique is analogous to unweighted model averaging [3] and helps reduce the error in arriving at the updated map. Observe that if j's particles reflect j's actual pose well, then we may expect a reasonably accurate estimation of the pushed landmark's new location. On the other hand, if j's particles are not well localized, map averaging mitigates the error due to the uncertainty. In Fig. 12, we show the simulated environment after j has pushed a landmark (the red landmark in the top right corner near j's base) and the map with i's particles showing updated positions of all the landmarks. Notice that the positions of all landmarks except for the pushed one remain unchanged. Location of the landmark that is pushed is altered although less than the amount by which it is actually pushed.

We evaluate the performance of our recursive localization with map update in comparison to the two alternative methods mentioned previously. Our hypothesis is that by explicitly tracking j using particles and estimating pushed landmarks, our approach localizes better in comparison to the others. We measure the mean squared error (MSE) between the robot i's particles and its actual pose at different time steps in the simula-

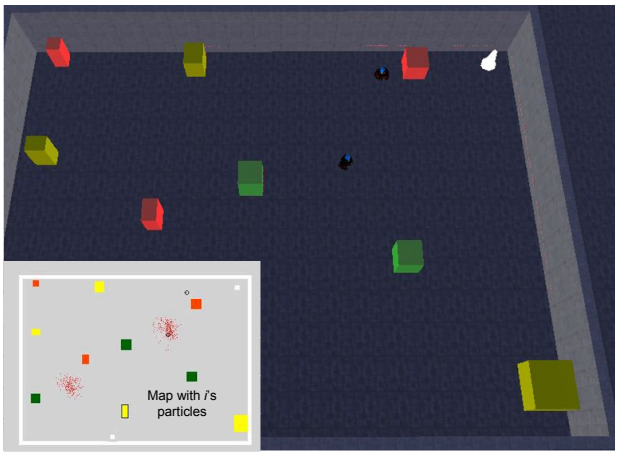


Fig. 12. The laser tag environment after robot j pushes the red landmark near the top. We show the updated map with i's particles. Observe that the landmark in the updated map does not accurately reflect the new position of the actual landmark due to map averaging and some error in localizing robot i.

tion. We do this for varying numbers of i's particles (N_i) and j's nested particles (N_j). In Fig. 13(a)–(e), we show the MSE for different settings of N_i and N_j across time steps in our scenario. The robots move asynchronously, and almost continuously, and our simulations take place in real-time. We show the standard deviation error in Fig. 13 at discrete time steps for simplicity. Each data point is the average of 3 runs of each of the three approaches on approximately similar trajectories of i. The paths taken by i and j may differ between particle settings. Within each plot, we indicate the time step when j is tagged and when it pushes the landmark. Note that i may move for some time before it observes the pushed landmark. We are currently running more simulations in order to get more data.

Observe that as we increase the number of particles allocated to robot i (N_i), the average MSE of its localization across all time steps reduces. For example, the MSE across all time steps when N_i=250,N_j=50 is 3.0 in comparison to 2.43 when N_i=500,N_j=50, and this further drops down to 2.15 when N_i=1,000,N_j=50. This is characteristic of approaches that use Monte Carlo localization as the number of particles are increased, although in our case N_j plays a role as well. More importantly, as we increase the number of j's particles (N_j) from 20 to 50, we note that the MSE of the Nested PF approach drops further in the last few time steps after the landmark has been pushed and i observes it. This is because the greater number of j's particles help in tracking j better, which leads to a better estimation of the new location of the pushed landmark. Indeed, the average MSE for Nested PF dropped from 2.43 to 2.05 when N_j increased from 20 to 50 while N_i was fixed at 500.

While there is no significant difference in the performance of the three approaches until the landmark is pushed as we may expect, Wolf does consistently worse subsequently. Specifically, Wolf does not utilize the dynamic landmark for observations any-

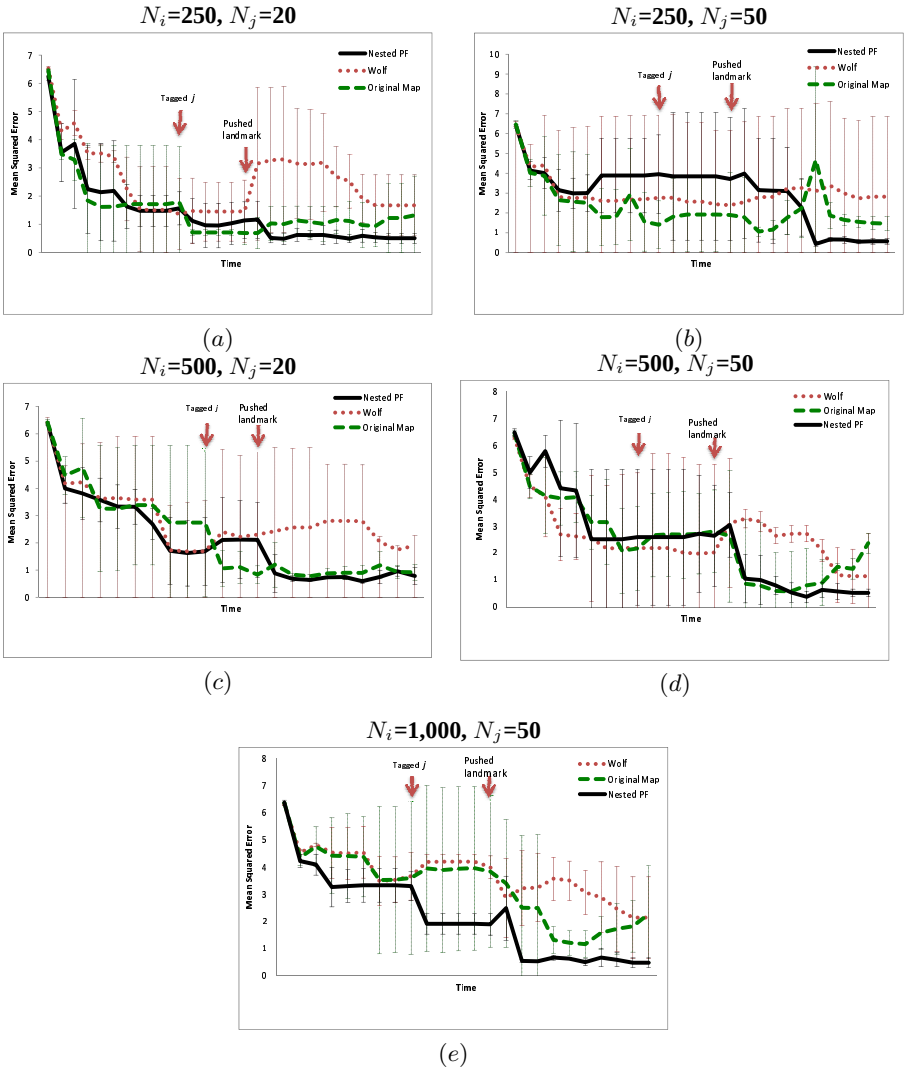


Fig. 13. Localization performance of robot i as the particles allocated to i (N_i) and nested particle set (N_j) change ((a)–(e)). Lower MSE indicates better performance. In each plot, we indicate when i tags j and when j pushes a landmark. Robot i may move for some time before it observes the pushed landmark. Performance after j has pushed the landmark is of interest. Observe that Nested PF consistently localizes better than the other approaches in the time steps of interest.

more, due to which the MSE of robot i does not change much from its previous values. Clearly, a robot utilizing Wolf for its localization would have diffi culties in reaching the base. Perhaps surprisingly, Original Map's performance compares with Nested PF when N_j=20 (see Fig. 13(c)). This is because in this case the estimated landmark is

approximately as far away from the actual pushed landmark as its original position is from the pushed position. Hence, 20 particles may not be sufficient to track j satisfactorily in this environment. However, on increasing N_j to 50, the performance of Nested PF is distinctly better than Original Map's performance in the last couple of time steps (see Figs. $13(a, b)$ and (c, d)). Observe that the Nested PF consistently outperforms the other approaches once the landmark has been pushed by j, and it performs the best among all particle settings when N_i=1,000 and N_j=50. Thus, robot i when using the Nested PF is significantly better localized as it closes in on j's base.

Favorable performance of the Nested PF approach comes at a cost. The approach took about 6 secs to perform one projection step – propagation ($\sim$3s), weighting and resamplng ($\sim$3s) – on an Intel Core 3 with 2.27GHz processor, 4GB RAM and Windows Vista when N_i=500 and N_j=50. When we increased N_i to 1,000 and N_j remained at 50, a single projection step took 10 secs. Larger environments may require more particles allocated to both i and j for satisfactory localization, which further increases the computational time.

In summary, although the nested particle filtering is computationally intensive, it provides a way to better track other robots in the environment whose actions could impact the localization of the original robot.

8 Discussion

Application domains such as search and rescue in disaster areas exhibit characteristics such as multiple other independent robots and movement of obstacles, which motivate more robust localization techniques. We introduced a recursive localization approach based on particle filtering in which particles for the other robot are nested within the particles of the subject robot. We showed how the particle filtering may be performed recursively to propagate, weight and resample the nested particles. The advantage of this approach is that it allows us to track the other robot explicitly in comparison to implicit approaches that marginalize the other robots as noise in the environment. Consequently, we may better predict the other robot's actions such as move a landmark, given its behavioral model. In the context of a modified laser tag environment, we evaluated the localization accuracy of our approach in a dynamic environment. We demonstrated that maintaining more information about the other robot leads to better localization in this complex environment, in comparison to remaining oblivious of the dynamism or not utilizing dynamic landmarks. This is especially useful if the environment exhibited few landmarks to begin with.

On the other hand, the total number of particles quickly increases thereby requiring more computational resources. This could be problematic if the approach is implemented on board within the bounded resources available to a mobile robot. Hence, heuristics that seek to intelligently allocate particles gain importance. Furthermore, because many particles are lost during the filtering, our next step is to investigate particle replenishment that replaces lost particles without significantly affecting the performance of the localization. While we utilized simple map averaging to estimate the updated map, techniques that concurrently build the map and localize [7, 15], could be suitable provided that they do not assume that the environment is static. In adversar-

ial environments the behavioral model of the other robot or a close estimate may not be available. In this case, a Bayesian approach in which the subject robot maintains a belief about the candidate models of the other robot and updates it based on its observations seems promising. Finally, a logical extension of this work would be to augment the localization with simultaneous mapping, which would be useful when the position of the landmarks is not known a priori.

References

1. P. Doshi and P. Gmytrasiewicz. Monte carlo sampling methods for approximating interactive pomdps. *Journal of Artificial Intelligence Research*, 34:297–337, 2009.
2. P. Doshi and P. J. Gmytrasiewicz. Approximating state estimation in multiagent settings using particle filters. In *Autonomous Agents and Multi-agent Systems Conference (AAMAS)*, pages 320–327, Utrecht, Netherlands, 2005.
3. D. Draper. Assessment and propagation of model uncertainty. *Journal of the Royal Statistical Society B*, 57:45–97, 1995.
4. D. Fox, W. Burgard, H. Kruppa, and S. Thrun. A probabilistic approach to collaborative multi-robot localization. *Autonomous Robots on Heterogenous Multi-Robot Systems*, 8(3), 2000.
5. D. Fox, W. Burgard, and S. Thrun. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research (JAIR)*, 11:391–427, 1999.
6. R. Kurazume and S. Nagata. Cooperative positioning with multiple robots. In *International Conference on Robotics and Automation*, volume 2, pages 1250–1257, 1994.
7. F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, 4:333–349, 1999.
8. C. J. P. Luis E. Navarro-serment and P. K. Khosla. ‘a beacon system for the localization of distributed robotic team. In *International Conference on Field and Service Robotics*, pages 232–237, 1999.
9. M. Rosencrantz, G. Gordon, and S. Thrun. Locating moving entities in indoor environments with teams of mobile robots. In *International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 233–240, 2003.
10. S. I. Roumeliotis and I. M. Rekleitis. Propagation of uncertainty in cooperative multirobot localization: Analysis and experimental results. *Autonomous Robots*, 17:41–54, 2004.
11. S. Russell and P. Norvig. *Artificial Intelligence, A Modern Approach*. Prentice Hall, 2nd edition, 2002.
12. G. K. S. Tully and H. Choset. Leap-frog path design for multiple robot cooperative localization. *Advanced Robotics*, 62:307–317, 2010.
13. H. W. Sorenson, editor. *Kalman Filtering: Theory and Application*. IEEE Press, New York, 1985.
14. S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2005.
15. S. Thrun, D. Fox, and W. Burgard. A probabilistic approach to concurrent mapping and localization for mobile robots. *Autonomous Robots*, 5:253–271, 1998.
16. S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust monte carlo localization for mobile robots. *Artificial Intelligence*, 128:99–141, 2001.
17. D. Wolf and G. Sukhatme. Mobile robot simultaneous localization and mapping in dynamic environments. *Autonomous Robots*, 19:53–65, 2005.

Adaptive Multi-Robot Team Reconfiguration using a Policy-Reuse Reinforcement Learning Approach

Ke Cheng[1], Prithviraj Dasgupta[1], and Bikramjit Banerjee[2]

[1] Computer Science Department, University of Nebraska, Omaha, USA
[2] Computer Science Department, University of Southern Mississippi, USA

Abstract. We consider the problem of dynamically adjusting the formation and size of robot teams performing distributed area coverage, when they encounter obstacles or occlusions along their path. Based on our earlier formulation of the robotic team formation problem as a coalitional game called a weighted voting game (WVG), we show that the robot team size can be dynamically adapted by adjusting the WVG's quota parameter. We use a Q-learning algorithm to learn the value of the quota parameter and a policy reuse mechanism to adapt the learning process to changes in the underlying environment. Experimental results using simulated e-puck robots within the Webots simulator show that our Q-learning algorithm converges within a finite number of steps in different types of environments. Using the learning algorithm also improves the performance of an area coverage application where multiple robot teams move in formation to explore an initially unknown environment by $4-8\%$.

Keywords: multi-robot formation, Q-learning, coalition game

1 Introduction

Multi-robot formation control is an important problem in distributed multi-robot systems. Formation maintenance among autonomous vehicles or robots is used for various applications such as convoying high-security objects safely [8, 19], mobilizing a team of robots with heterogeneous suite of sensors for extra-terrestrial applications [7] and carrying heavy payloads to support combat-forces in military applications. A principal challenge in multi-robot formation is to allow a robot team to dynamically split or merge into new teams that can continue to move in formation, after encountering obstacles or occlusion in its path. In our previous work on multi-robot formation[9], we have described a coalition game theory based algorithm to solve this problem. Our technique consists of a lower-level *controller layer* that handles the basic navigation tasks for a group of robots while moving in formation by using a flocking-based technique, along with an upper *game theoretic layer* that handles more cognitively involved tasks such as merging and splitting teams using a coalition game called weighted voting game (WVG). However, in this technique, the maximum size that a robot

team can have is kept fixed throughout the operation of the algorithm. Changing the maximum size of a team is an important factor for efficient robotic team formation because a large team can have difficulty in navigating in tight spaces. On the other hand, having a small team to perform a task in place of a larger team that could have operated in the same situation, results in lower operational efficiency in terms of increased time and energy expended to complete the task. Developing a mechanism that allows robot teams to autonomously adapt their maximum size would solve this problem. Therefore, it makes sense to investigate techniques that can be used to dynamically adjust the size of robot teams within the framework of the multi-robot formation problem. In this paper, we investigate a policy reuse-based reinforcement learning approach to propose a solution to this dynamic team size adjustment problem. To illustrate the operation of our team formation algorithm, we have used a distributed area coverage scenario where multiple mobile robots, organized as teams are placed in an initially unknown environment. The objective of the robots is to entirely cover the free space of the environment while reducing the overlap between regions covered by different robots. To achieve this, the robot teams have to dynamically reorganize their team structure when they encounter obstacles in their path while covering the environment, using the algorithm described in this paper. Experimental results using teams of 5 e-puck robots within the Webots robot simulator show that when robot teams are able to dynamically adapt their team sizes using our algorithm, their performance of the area coverage improves by $4 - 8\%$.

2 Related Work

Much of the research on formation control with multi-robot teams [1, 15, 18] has been based on Reynolds' model for the mobility of flocks[20]. Reynolds prescribes three fundamental operations for each team member to realize flocking - *separation, alignment* and *cohesion*. In [2], the authors describe three reactive behavior-based strategies for robot teams to move in formation, viz., unit center-referenced, neighbor-referenced, or leader-referenced. In contrast to these approaches, Fredslund and Mataric[14] describe techniques for robot team formation without using global knowledge such as robot locations, or the positions/headings of other robots, while using little communication between robots. Complementary to these approaches [23, 13] have used a combination of graph theory and control theory-based techniques to effect multi-robot formations. Our previous work on multi-robot formation [4, 10] uses techniques where a team simply reverses its direction to avoid obstacles or other teams, without dynamically reforming, merging with other teams or splitting into smaller teams, while in [5, 6, 9], we have described a weighted voting game [22] based algorithm for determining the best partitions among a team of robots and dynamically re-configuring their formation after the robot team encounters obstacles.

Reinforcement learning [25] is a popular technique in the design of multi-robot systems [3, 17, 24, 26] with a survey of recent results given in [27]. The complexity of the state and joint action spaces of the robots and the credit

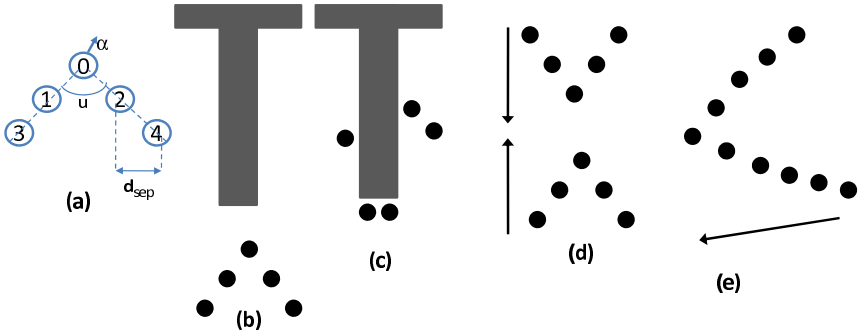


Fig. 1. (a) A robot team showing the position identifiers of each robot. The angular separation in the team is u, the separation between adjacent robots is d_{sep} and α is the heading of the team. (b)-(c) A scenario where a single team in formation encounters a T-shaped obstacle and needs to split. (d)-(e) A scenario where two teams in close proximity of each other encounter each other and need to merge.

assignment problem of determining the rewards for each robot [17, 11] pose challenging problems to adapt reinforcement learning from single to multiple robots. To improve the learning process in multi-robot scenarios, policy reuse [12] provides an attractive mechanism when the state and action spaces of the robots do not change, but only the environment changes. Recently, policy reuse has been combined with spatial hints [21] for use in simulated multi-robot scenarios for learning navigation tasks. Our work in this paper extends the concepts of reinforcement learning and policy reuse within the framework of a coalition game to improve the dynamic restructuring and reconfiguration robot teams that are required to move in formation, after the teams encounter obstacles or occlusions along their path.

3 Machine Learning For Improving Coalitional Game-Based Multi-robot Team Formation

The fundamental operation in robotic team formation is to maintain a specific configuration or shape among the robots in a team, even when the team is in motion. For our setting, robots are deployed into an initially unknown environment in teams of a pre-determined size. The leader of a team navigates the team while maintaining a wedge-shaped formation as shown in Figure 1(a), while using a leader-referenced flocking technique [4, 10]. Most of the previous work on multi-robot formation control however do not directly address the problem of reconfiguring and restructuring robot teams after they encounter obstacles or other teams, as illustrated in Figures 1 (b) and (d). In [9], we developed an

algorithm called DYN-REFORM that uses a form of a coalition game called a *weighted voting game* (WVG) to enable robot teams to dynamically reconfigure by splitting or merging into new teams after encountering an obstacle or other teams, while maintaining formation - as illustrated in Figures 1 (c) and (e). The essential features of the WVG for a multi-robot team formation problem are discussed below. The main parameters of a WVG are the following:

1. R: Set of players or robots interested in forming a coalition.
2. w_i: Weight of robot $r \in R$ that is calculated from its individual efficiency in performing its operation or role while participating in the team.
3. Q: Quota or threshold of the WVG. A set of robots becomes a winning coalition if the sum of the weights of the robots exceeds the quota. In [5, 9], we have defined the value of the quota Q of a WVG to $Q = \lceil q_f \times \sum_i w_i, \rceil$ where w_i is the weight of robot i that is participating in the WVG, and $q_f \in [0, 1]$ is called the *quota fraction*. $q_f = 0$ results in no coalition (all robots participating in the WVG move individually), $q_f = 1$ results in the grand coalition (all robots participating in the WVG move together), while intermediate values of q_f result in coalitions of intermediate sizes.

The solution of a WVG is to find the smallest set of players whose weights, taken together equal to or exceed the quota value Q. This set of players is called the **minimum winning coalition (MWC)**. A simple example is given below to illustrate the WVG-based robot team formation process: consider a set of 4 robots A, B, C and D with weights $4, 2, 1$, and 1 respectively, which need to determine the subset of robots to form a team. Let quota $Q = 5$, that is any coalition must have a combined weight of at least 5 to be a winning coalition. The set of MWCs for this WVG are $\{A, B\}, \{A, C\}, \{A, D\}$. Because the robotic team formation problem requires a unique set of robots to be identified as a winning coalition, we developed a heuristic called BMWC (best minimum winning coalition) to break the tie between multiple MWCs [6]. We have used our DYN-REFORM algorithm for multi-robot dynamic team reconfiguration in a multi-robot area coverage application and shown that dynamically reforming robot teams improves the efficiency of the area coverage operation by $5 - 12\%$ in simulations and $5 - 8\%$ with physical e-puck robots.

While the WVG based framework provides a structured technique to dynamically determine stable partitions among a set of robots, it does not provide any mechanism of adapting the size of the partitions based on current operational and environmental conditions. Dynamically adapting the team size is important because the efficiency that a robot team gets by performing its operation is determined to a large extent by the team's size. Within the WVG framework, the crucial parameter that determines the size of a robot team is the quota fraction q_f. Because q_f determines the quota Q which provides the threshold value by specifying the minimum number of robots required to form a team.

In our previous work, the value of the quota fraction q_f was kept fixed throughout the duration of the robots' operation. However, a fixed value of q_f fails to dynamically adapt team sizes based on the perceived operation conditions

in the environment. To illustrate the problem of keeping q_f fixed we consider an example in an area coverage application where a set of 5 robots with individual weights $1, 1, 1, 1, 1$ run a WVG with $q_f = 0.9$, giving $Q = 0.9 \times 5 = 4.5$. The BMWC that will be output in this case is the grand coalition of 5 robots because this is the only set of robots whose weights add up to 5, which is greater than the desired quota $Q = 4.5$. Now, consider that after some time this team of 5 robots moves into a region (e.g., a narrow channel) where a team 5 robots does not have enough space to navigate and a team of 3 robots would provide the most efficient coverage. Because the team of 5 robots is not able to navigate under the current conditions, the individual efficiency of each robot reduces from the earlier value of 1. Suppose the new weights of the robots are $0.9, 0.8, 0.7, 0.6, 0.6$. If q_f is kept fixed at 0.9, giving $Q = 0.9 \times (0.9 + 0.8 + 0.7 + 0.6 + 0.6) = 3.24$, then all 5 robots are still required to form a coalition to reach the desired quota value. Navigating 5 robots in a space that can accommodate 3 robots impedes their motion and aggravates their efficiency. The problem of forming the large, 5-robot team could have been avoided if the sizes of the robot teams could be adjusted by dynamically updating the value of q_f based on the operation and environment conditions. We propose to use machine learning techniques that allow dynamic updates to the quota value depending of the sensory information about the environmental and operational conditions. Our proposed learning mechanism for the quota value proceeds in two parts. In the first part, the leader robot within each team uses an ε-greedy iterated policy selection strategy to select an action at every step. In the second part, leader robots of teams adapt their policies using a policy-reuse technique based on the perceived environment features in their vicinity, so that their operations can continue to receive high rewards. A schematic of the controller of a multi-robot team showing the learning mechanism and the underlying coalition game and controller layers, and their respective functionalities, is shown in Figure 2.

3.1 Q-Learning for Updating the q_f Parameter

The efficiency of a robot team performing area coverage is given by the ratio between the area of the previously uncovered region covered by the team and the total amount of the area covered by the team. The highest value of this coverage efficiency can be 1.0 when the entire team covers previously uncovered region. On the other hand, if any team member covers region that has been previously covered by the robots from the same team or another team, the coverage efficiency reduces. Specifically, when some robots in a team encounter an obstacle while navigating, the robots that encounter the obstacle have to backtrack and re-cover their own trail, and thereafter, follow behind other team members that did not encounter the obstacle. Both of these movements cause the robots that encountered the obstacle to re-cover the regions already covered by themselves or by other team members, and ultimately reduces the overall team's coverage efficiency. The learning problem we consider here is how to dynamically determine the size of the team so that the team can continually maintain a high coverage efficiency.

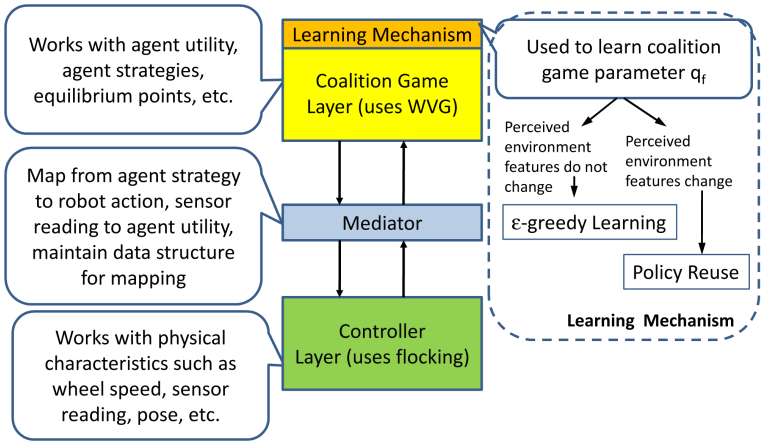


Fig. 2. A layered controller for controlling multi-robot teams.

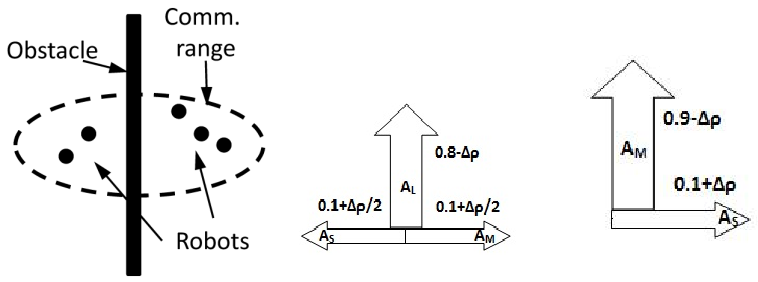
The coverage efficiency that a robot team gets is determined to a large extent by the team's size. The team size in turn is determined by the quota fraction q_f. The problem of maintaining a high efficiency while performing coverage can therefore be viewed as a problem of updating the quota fraction q_f dynamically based on the current operational and environment conditions perceived by the robots in the team. This problem is non-trivial because the noise in a robot's distance sensors and the error in localizing the robots results in uncertainty in the actions performed by a robot. We have modeled the dynamic update of the quota fraction as a reinforcement learning problem [25]. Reinforcement learning problems are typically formalized by Markov Decision Processes(MDPs). An MDP is a tuple $< S, A, T, R >$, where S is the set of states, A is the set of actions, $R : S \times A \longrightarrow R$ is a reward function. $T : S \times A \times S \longrightarrow [0, 1]$ is a stochastic state transition function. The solution for an MDP is a policy $\Pi : S \longrightarrow A$ that assigns one action in A for each state in S. To find the optimal policy, we use the well-known Q-learning function $Q^*(s, a) = R(s, a) + \gamma \sum_{s'} T(s, a, s^{'}) \max_{a'} Q(s^{'}, a^{'})$, where γ is a discount factor, while following a greedy action selection strategy to determine the policy.

Using this formulation, we have used each state for our MDP model to represent the coverage efficiency received by a robot team over its recent history. We discretize this coverage efficiency, such as S_1 represents a coverage efficiency value of 0.1, S_2 represents a value of 0.2, etc., and S_{10} represents a value of 1.0. We define three actions on this state space corresponding to different quota fraction q_f values. Action A_L corresponds to $q_f = 0.9$ which sets the quota value Q to 90% of the combined weights of the robots (players) participating in the WVG. This results in the team size remaining close to the grand coalition. Correspondingly, we define two more actions A_M corresponding to $q_f = 0.5$ which results in the team size becoming about half the grand coalition, and, A_S cor-

responding to $q_f = 0.2$ which results in the team size becoming about 20% of the grand coalition. However, because of inaccuracies in the perceived environment resulting from distance sensor range limitations and noise, and localization error of the robots, there exists some uncertainty in performing these actions. For example, as illustrated in Figure 3(a), while attempting to perform action A_L that should result in most of the robots joining the same coalition, some of the robots perceive an obstacle after they decide to come together as a team. Repeated attempts by the robots to enter into formation do not alleviate the situation. This results in the robots ending up in smaller sized teams or coalitions, corresponding to what would result from $q_f = 0.5$ (action A_M). Figures 3 (b) and (c) show the uncertainty while performing actions A_L and A_M in our MDP model. As shown in the figure, both of these actions, A_L and A_M, can result in the formation of smaller teams with a certain probability.

To select an action, we use an ε-greedy strategy [16]. This strategy allows a trade-off between exploitation and exploration by selecting the action recommended by the optimal policy using Q-learning with a probability ε while selecting a random action with probability $1 - \varepsilon$.

The reward at each state of our MDP is calculated as a linear function of the coverage efficiency corresponding to that state and is denoted by $R(S_i) = \rho\times$ coverage efficiency of the robot team corresponding to state S_i.



(a) (b) (c)

Fig. 3. (a) Five robots that are within communication range of each other do not perceive the obstacle when they run a WVG and decide to form a team. The obstacle impedes their motion when they try to get into formation. Probabilities showing the uncertainty in performing the action (b) A_L and (c) A_m corresponding to setting a high and medium value respectively of the quota fraction.

3.2 Policy Reuse for Adapting to Different Environment Features

Following the example from Section 3 of dynamically adjusting the size of a robot team depending on the space available in the team's immediate vicinity,

Algorithm 1 Iterated policy selection strategy

Input: $s_{final-lastEpisode}$, π_i, ε_0, H
Output: $Q^{\pi_{new}}$

$p \longleftarrow 1.0$
$s_{curr} \longleftarrow s_{final-lastEpisode}$
$\varepsilon \longleftarrow \varepsilon_0$
for $h = 1$ to H **do**
  if randomly generated number $> p$ **then**
    $action \longleftarrow \pi_i(s_{curr})$
  else
    $action \longleftarrow \varepsilon - greedy(\pi_{new}(s_{curr}))$
  end if
  Execute(action), record $s_{next} = T(s_{curr}, action)$ and reward $r(s_{curr}, action)$ as given by MDP model
  $Q^{\pi_{new}}(s_{curr}, action) \longleftarrow (1-\alpha)Q^{\pi_{new}}(s_{curr}, action) + \alpha[r(s_{curr}, action) + \gamma \max_a Q^{\pi_{new}}(s_{next}, action)]$
  $p \longleftarrow p - \frac{1}{H}$
  $s_{curr} \longleftarrow s_{next}$
  $\varepsilon \longleftarrow \min(1, \varepsilon + \Delta\varepsilon)$
end for

the quota fraction derived from a policy which results in a desired efficiency dynamically varies depending on environment features such as the presence of obstacles and occlusions in the robot team's path. This means that the optimal policy of the MDP underlying the Q-learning process must be updated based on the environment features. Following [12, 21], we characterize the environment by a finite set of features represented by the set of states S, set of actions A a transition function $T : S \times A \times S$. Each such tuple $< S, A, T >$ is referred to as domain D. Depending on the domain, different robot team sizes (resulting from different q_f values or different actions in the MDP) yield different rewards. For example, a large team can get a high coverage efficiency and hence a high reward in an environment where its motion is not impeded by obstacles, while the same team can get a reduced reward owing to reduced efficiency from obstacles in its path, in an environment with significant number of obstacles. Based on this observation, each domain yields a different reward function. Finally, a task corresponds to maintaining a team size that results in the best coverage efficiency (reward) in a domain. It can be represented as a combination of a domain and its associate reward function, $\Omega =< D, R_\Omega >$. The optimal policy corresponding to each domain $\{D_1, D_2, D_3, ...\} \in D$ is maintained as a policy library $L_{D_i} = \{\pi_1, \pi_2, \pi_3, ...\}$. The robot team then selects the appropriate policy corresponding to the domain (environment feature) it perceives around its vicinity. The transition models for our MDP are shown in Figures 4(a)-(d). As shown in Figure 4(b), when the robot team perceives no obstacles in its vicinity, increasing the size of the team improves the efficiency of the team. Therefore, in this domain, performing the action A_L improves the efficiency. Similarly, in an environment where the robot team perceives 20% of its vicinity occupied

by obstacles, the action A_M makes the team smaller and enables the robots to avoid obstacles, thus improving the efficiency - as shown in Figure 4(c). Finally, as shown in Figure 4(d), when the robot team perceives 40% of its vicinity occupied by obstacles, the action A_S reduces the team's size rapidly and enables the robots to avoid obstacles, thus improving the efficiency.

The leader robot of a robot team executes the Q-learning and policy reuse algorithms at intervals of H time steps. The duration of each such interval of H time steps is called an *episode*. At the end of each episode, the coverage efficiency achieved by the robot team is calculated by the leader robot to determine the current state corresponding to the coverage efficiency. We define the expected average reinforcement from the episode set at the end of K episodes, each of length H as $W(E)_K = \frac{1}{K}\sum_{k=1}^{K}\sum_{h=1}^{H}\gamma^h r_{k,h}$ where $\gamma^h \in [0,1]$ is the discount factor for rewards during step h within an episode, and $r_{k,h}$ defines the normalized value of the actual coverage efficiency derived by the team during step h of episode k. The values of $W(E)_K$ and the corresponding policy π_k during that episode are recorded by the leader robot.

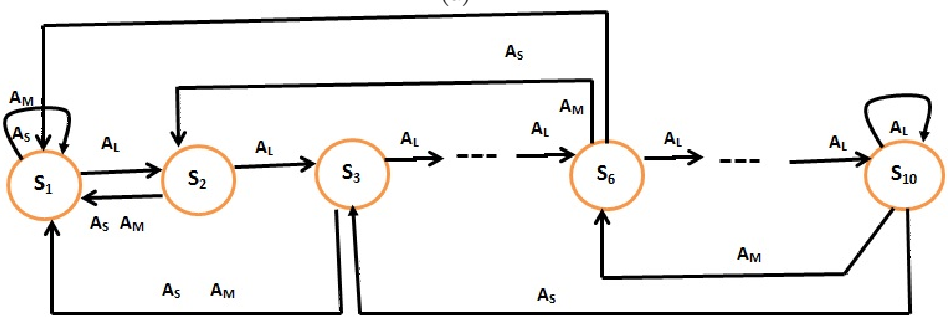
Algorithm 2 Policy-Reuse Algorithm

Input: Domain *currentDomain*, H, *numEpisodes*,
Output: π_{Ω^*}
$h \leftarrow 1$
for $k = 1$ to *numEpisodes* **do**
  while $h <= H$ **do**
    $W(E)_k \leftarrow W(E)_k + \gamma^h \times r_{k,h}$
    $h \leftarrow h + 1$
  end while
  Add last policy $\pi_{numEpisodes}$ to policy library L of domain *currentDomain*
  if currently perceived domain = *currentDomain* **then**
    $\pi_{\Omega^*} \leftarrow \pi_{numEpisodes}$ (last policy used)
  else
    $\pi_{\Omega^*} \leftarrow$ policy π_k with maximum value of $W(E)_k$ in randomly selected domain other than *currentDomain*
  end if
end for
return π_{Ω^*}

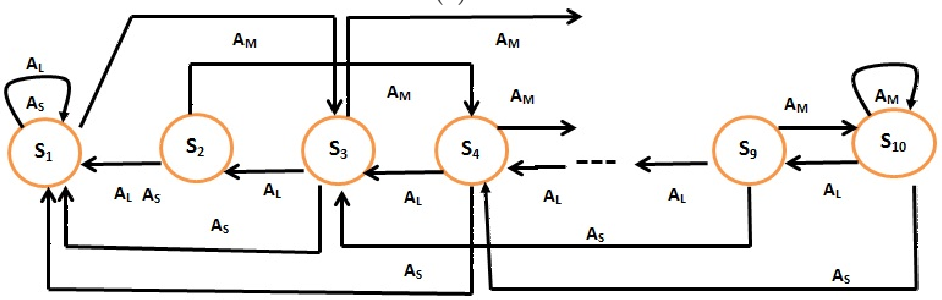
Algorithm 2 shows the policy reuse algorithm used by a team's leader robot. The algorithm first updates the value of the discounted reward for each episode. It then adds the most recent policy to the policy library for the current domain and checks to see if the domain perceived by the robots is the same as the current domain. If the domain has not changed, the most recently used policy is reused. On the other hand, if the domain has changed, the policy that corresponds to the maximum rewards obtained for episode in that domain is selected as the new policy to use.

Domain	Action	Effect	Transition
0% of perceived environment has obstacles	A_L	***Improves*** coverage efficiency (reward) by 10%	$T(S_i, A_L) \rightarrow S_{i+1}$
	A_M	Reduces coverage efficiency (reward) by 40%	$T(S_i, A_M) \rightarrow S_{i-4}$
	A_S	Reduces coverage efficiency (reward) by 70%	$T(S_i, A_S) \rightarrow S_{i-7}$
20% of perceived environment has obstacles	A_L	Reduces coverage efficiency (reward) by 10%	$T(S_i, A_L) \rightarrow S_{i-1}$
	A_M	***Improves*** coverage efficiency (reward) by 20%	$T(S_i, A_M) \rightarrow S_{i+2}$
	A_S	Reduces coverage efficiency (reward) by 60%	$T(S_i, A_S) \rightarrow S_{i-6}$
40% of perceived environment has obstacles	A_L	Reduces coverage efficiency (reward) by 40%	$T(S_i, A_L) \rightarrow S_{i-4}$
	A_M	Reduces coverage efficiency (reward) by 20%	$T(S_i, A_M) \rightarrow S_{i-2}$
	A_S	***Improves*** coverage efficiency (reward) by 10%	$T(S_i, A_S) \rightarrow S_{i+1}$

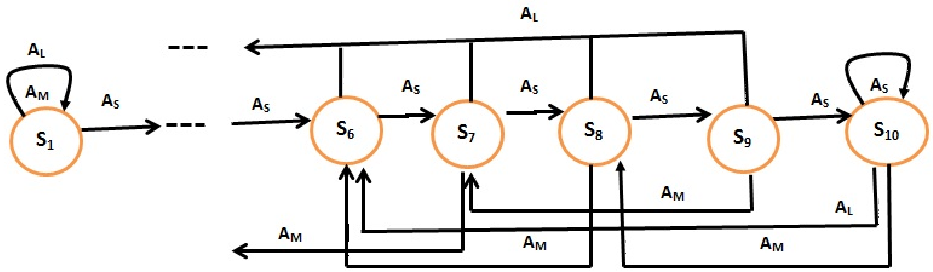
(a)



(b)



(c)



(d)

Fig. 4. (a) Effect of performing different actions A_L(set $q_f = 0.9$), A_M(set $q_f = 0.5$) and A_S(set $q_f = 0.2$) in different domains on the state of MDP(team coverage efficiency of robots). (b)-(d) Corresponding state transition diagrams of the MDPs for the different domains.

4 Experimental Results

4.1 Parameter configuration and simulation setup

We have tested our algorithms on the Webots robot simulation platform. Webots allows realistic modeling of robots and environments including the parameters of different sensors on robots and the physics of the environment. Each robot in our simulated system is modeled as an e-puck robot with added on-board GPS and compass for localization[1]. The speed of each wheel was set to 2.8 cm/sec, which results in a robot covering an area of $0.07 \times 0.07 meter^2$ under its own footprint during each time step. Unless otherwise stated, the experiments are repeated over 10 times with simulated physical robots for 30 minutes to 2 hours, and we collected the average results as well as the maximum and minimum values. As shown in Figure 5 (a)-(c), we tested our algorithm in the $2 \times 2\ m^2$ environment with 0%, 10% and 20% of the area of the environment occupied by obstacles.

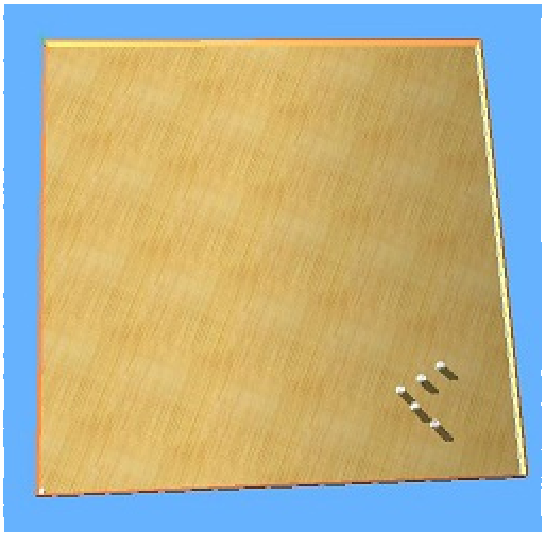
For the Q-learning parameters, we set the discount factor $\gamma = 0.95$, and the learning rate $\alpha = 0.05$. The ε-greedy strategy was configured with $\varepsilon_0 = 0$ and $\triangle\varepsilon = 0.001$. The number of time steps per episode $H = 100$.

4.2 Performance evaluation

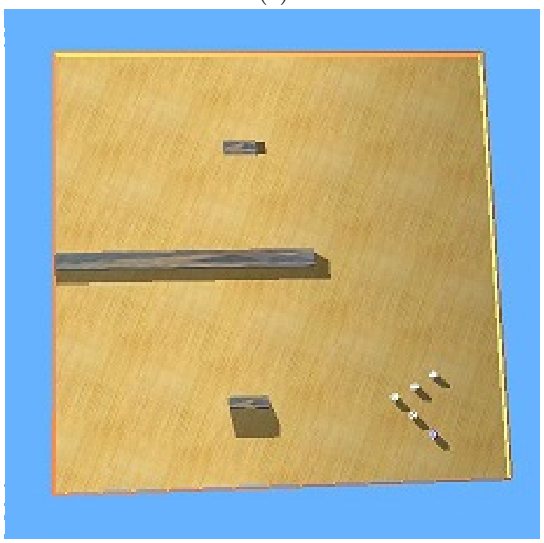
The first set of experiments, we evaluated the performance of our algorithm learning over time in the three scenarios of Figure 5 (a)-(c). As shown in Figure 6, we test our policy reuse algorithm in an unknown environment without obstacles. After 70 episodes, the accumulated average reward becomes unchanged around 0.86, which is a normalized value in the interval$[0, 1]$. In other words, the team can achieve a high level percentage of the coverage after 100 episodes. Around 50 episodes, the system even achieved $W(E) = 0.92$, which corresponds to a reasonably high coverage efficiency of the robot team. The value goes down, for the environment is boundary by walls at four sides. Figure 7 and Figure 8, the results are shown for the environments where 10% and 20% of the free space from Figure 4 is occupied by obstacles. The accumulated average rewards of both these environments become stable roughly around 0.84. As shown in Figure 8, the curve becomes flat earlier than the other two cases, because the robot team encounters obstacles earlier and learns to reuse the policies towards the starting episodes. In all these three scenarios the robot team can gradually achieve a high level percentage of the coverage after 100 episodes. In other words, our Q-learning algorithm converges within a finite number of steps in different type of environments.

The second set of experiments shown in Figure 9 give the improvement in coverage achieved using the policy reuse and reinforcement learning algorithms by a set of 5 robots, initially configured as a team. The robots are placed within $2 \times 2\ m^2$ with 20% of the environment occupied by obstacles. The experiment results were collected after half an hour, an hour, an hour and a half, and two

[1] For physical robots we use an overhead camera based system for localizing the robots.



(a)



(b)

(c)

Fig. 5. The environment is $2 \times 2\ m^2$ arena (a) with no obstacles in it. (b) with 10% of the arena's area occupied by obstacles, (c) with 20% of the arena's area occupied by obstacles.

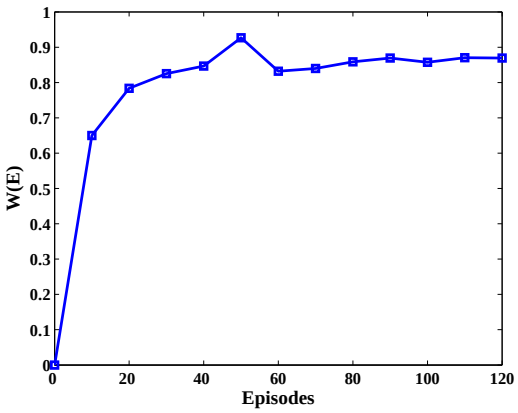


Fig. 6. The accumulated average reward per episode in an environment without obstacles.

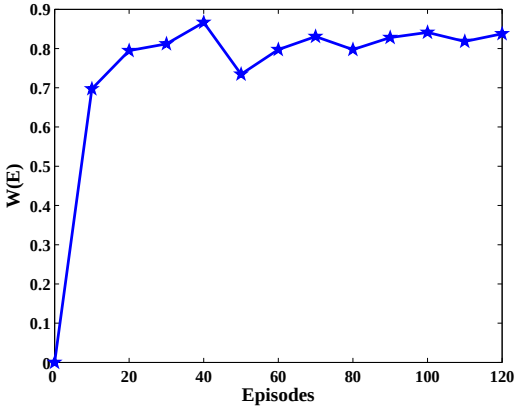


Fig. 7. The accumulated average reward per episode in an environment with 10% obstacles.

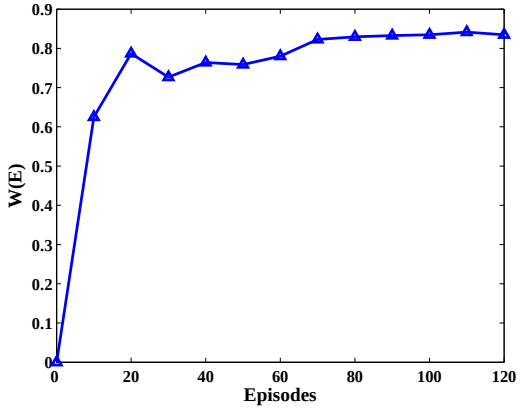


Fig. 8. The accumulated average reward per episode in an environment with 20% obstacles.

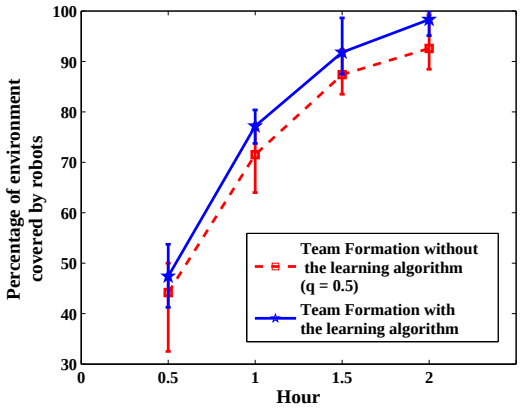


Fig. 9. Percentage of the environment covered by a set of 5 robots initially configured as a team without and with the reinforcement learning algorithm. The environment is $2 \times 2\ m^2$ with 20% of the environment occupied by obstacles. The results are collected during half an hour, an hour, an hour and a half, and two hours

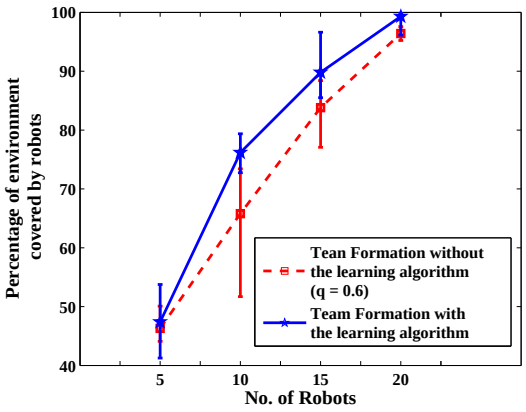


Fig. 10. Percentage of environment covered by different numbers of robot with and without the reinforcement learning algorithm. Each experiment was run for a period of 30 minutes with 20% of the environment occupied by obstacles.

hours of simulation. The graph was drew with error bars at each point. We observe that the robots using the policy reuse algorithm are able to improve coverage by about $4-8\%$ at different time periods.

Finally, we quantify the effect of the team formation approach without and with the reinforcement learning algorithm described in this paper, as shown in Figure 10 with error bars. The compared approach uses fixed value of the quota fraciton $q_f = 0.6$. We report results from experiments with 5, 10, 15, 20 robots each of which were deployed as teams of 5 robots within $2 \times 2\ m^2$ arena with 20% of the arena's area occupied by obstacles. Overall, the results of this experiment suggest that robots that are running a policy reuse algorithm among themselves produce better results for coverage if they can dynamically choose the optimal policy from an earlier learning process.

5 Conclusions and Future Work

We have introduced a policy reuse reinforcement learning approach to address the challenge for adaptive multi-robot team reconfiguration. First, this algorithm learns to perform a task by using policies from the policy repository which is built by former tasks in the same domain. Second, we integrated this online reinforcement learning model with our former designed weighted voting game mechanism for multi-robot team formation. By using this combined high level intelligent hierarchy, a group of robots can not only choose who should be in the team, but also find the best size of the team in an initially unknown complex

environment. We have implemented this approach on Webots robot simulation and experiment results show it improve the coverage efficiency about $4 - 8\%$ than the weighted voting game mechanism with the predefined quota value. Future ongoing research includes extension of this policy reuse model to multiple teams of robots. We envision multiple robot teams can share and reuse policies previously learned in different sub regions of an environment to learn a suitable policy for a previously unvisited environment.

Acknowledgment

The research by the authors at the University of Nebraska, Omaha has been sponsored by the Office of Naval Research as part of the COMRADES project, grant no. N000140911174.

References

1. E. Bahceci, O. Soysal and E. Sahin, "Review: Pattern formation and adaptation in multi-robot systems," CMU Tech. Report no. CMU-RI-TR-03-43, 2003.
2. T. Balch and R. Arkin, "Behavior-based formation control of multi-robot teams," IEEE Transactions on Robotics and Automation, vol. 14, no. 6, 1998, pp. 926-939.
3. M. Bowling and M. Veloso, "Simultaneous adversarial multi-robot learning," Proc. 18th International Joint Conference on Artificial Intelligence (IJCAI), 2003, pp. 699-704.
4. K. Cheng, P. Dasgupta and Y. Wang, "Distributed Area Coverage Using Robot Flocks," World Congress on Nature and Biologically Inspired Computing (NaBIC '09), 2009, pp.678-683.
5. K. Cheng, P. Dasgupta "Weighted Voting Game Based Multi-robot Team Formation for Distributed Area Coverage," AAMAS 2010 Workshop on Practical Cognitive Agents and Robots, Toronto, Canada, 2010.
6. K. Cheng, P. Dasgupta "Multi-Agent Coalition Formation for Distributed Area Coverage: Analysis and Evaluation," Second International Workshop on Collaborative Agents Research and Development (CARE), Toronto, Canada, 2010, pp.
7. P. Clark, M. Rilee, S. Curtis, C Cheung, W. Truszkowski, G. Marr and M. Rudisill, "PAM: Biologically Inspired Engineering And Exploration Mission Concept, Components, And Requirements For Asteroid Population Survey," Proc. 55th Intl. Astronautical Congress, Vancouver, Canada, IAC-04-Q5.07, 2004.
8. P. Cook, "Stable control of vehicle convoys for safety and comfort," IEEE Trans. on Automatic Control, vol. 52, no. 3, 2007, pp. 526 - 531.
9. P. Dasgupta and K. Cheng, "Robust Multi-robot Team Formations using Weighted Voting Games", 10th International Symposium on Distributed Autonomous Robotics Systems (DARS 2010), EPFL, Switzerland, 2010.
10. P. Dasgupta, T. Whipple and K. Cheng, "Effects of Multi-robot Team Formations on Distributed Area Coverage", (accepted at), International Journal of Swarm Intelligence Research, 2010.
11. F. Fernandez, D. Borrajo and L. Parker, "A Reinforement Learning Algorithm in Cooperative Multi-Robot Domains", Journal of Intelligent and Robotic Systems, v.43, no. 2-4, 2005, pp. 161-174.

12. F. Fernandez and M. Veloso, “Probabilistic Policy Reuse in Reinforcement Learning Agent,” Proc. 5th Intl. Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), 2006 pp.
13. R. Falconi, S. Gowal, A. Martinoli, “Graph Based Distributed Control of Non-Holonomic Vehicles Endowed with Local Positioning Information Engaged in Escorting Missions,” ICRA 2010, Anchorage, AK, pp. 3207-3214.
14. J. Fredslund and M. Mataric, “A general algorithm for robot formations using local sensing and minimal comm,” IEEE Trans. on Rob. and Auton., vol. 18, no. 5, 2002, pp. 837-846.
15. F. Gokce and E. Sahin, “To flock or not to flock: the pros and cons of flocking in long-range migration of mobile robot swarms,” AAMAS 2009, pp. 65-72.
16. E. Gomes and R. Kowalczyk, ”Dynamic analysis of multiagent Q-learning with ε-greedy exploration” ,Proc. of the 26th Annual International Conference on Machine Learning, Montreal, Canada, 2009, pp. 369-376.
17. M. Mataric, “Reinforcement learning in the multi-robot domain,” Autonomous Robots, vol. 4, 1997, pp. 73-83.
18. R. Olfati Saber, ”Flocking for Multi-Agent Dynamic Systems: Algorithms and Theory,” IEEE Trans. on Automatic Control, vol. 51, no. 3, 2006, pp. 401-420.
19. I. Rekleitis, G. Dudek and E. Milios, “Multi-Robot Collaboration for Robust Exploration,” Annals of Mathematics and Artificial Intelligence, vol. 31, no. 1-4, pp. 7-40, 2001.
20. C. Reynolds, “Flocks, herds and schools: A distributed behavioral model,” Computer Graphics, vol. 21, no. 4, 1987, pp. 25-34.
21. B. N. Silva and A. Machworth, ”Using Spatial Hints to Improve Policy Reuse in a Reinforcement Learning Agent”, Proc. 9th Intl. Joint Conf. on Autonomous Agents and Multi-Agent Systems, (AAMAS), 2010, pp.
22. Y. Shoham and K. Leyton-Brown. “Multiagent Systems: Algorithmic, Game Theoretic and Logical Foundations”, Cambridge University Press, 2009.
23. B. Smith, M. Egerstedt, and A. Howard, “Automatic Generation of Persistent Formations for Multi-Agent Networks Under Range Constraints,” Mobile Networks and Applications Journal, Vol. 14, pp. 322-335, 2009.
24. W. Smart and L. Kaelbling, “Effective reinforcement learning for mobile robots,” Proc. International Conference on Robotics and Automation(ICRA), 2002, pp. 3404-3410.
25. R. Sutton and A. Barto, Reinforcement Learning, Cambridge MA USA, 1998.
26. C. Touzet, “Distributed Lazy Q-learning for cooperative mobile robots,” International Journal of Advanced Robotic Systems, vol 1., no. 1, pp. 5-13, 2004.
27. E. Yang, and D. Gu, ”Multi-robot systems with agent-based reinforcement learning: evolution, opportunities and challenges,” International Journal of Modelling, Identification and Control, Vol. 6, No. 4, 2009, pp. 271 - 286.

Ship Patrol: Multiagent Patrol under Complex Environmental Conditions

Noa Agmon, Daniel Urieli, and Peter Stone

Department of Computer Science
The University of Texas at Austin
{agmon,urieli,pstone}@cs.utexas.edu

Abstract. The problem of multiagent patrol has gained considerable attention during the past decade, with the immediate applicability of the problem being one of its main sources of interest. In this paper we concentrate on frequency-based patrol, in which the agents' goal is to optimize a frequency criterion, namely, minimizing the time between visits to a set of interest points. We consider multiagent patrol in environments with complex environmental conditions that affect the cost of traveling from one point to another. For example, in marine environments, the travel time of ships depends on parameters such as wind, water currents, and waves. We demonstrate that in such environments there is a need to consider a new multiagent patrol strategy which divides the given area into parts in which more than one agent is active, for improving frequency. We show that in general graphs this problem is intractable, therefore we focus on simplified (yet realistic) cyclic graphs with possible inner edges. Although the problem remains generally intractable in such graphs, we provide a heuristic algorithm that is shown to significantly improve point-visit frequency compared to other patrol strategies. For evaluation of our work we used a custom developed ship simulator that realistically models ship movement constraints such as engine force and drag and reaction of the ship to environmental changes.

1 Introduction

The problem of multiagent patrol has gained considerable attention during the past decade [6, 10, 3, 8, 4, 2, 1], with the immediate applicability of the problem being one of its main sources of interest. The problem is formally described as repeatedly visiting some interest points in order to monitor them. The points may either be in a discrete environment, a continuous 1-dimensional environment (along a line), or a continuous 2-dimensional environment (inside an area).[1] The problem is usually divided according to the perspective of the agents. In *multiagent frequency-based patrol*, the agents' goal is to optimize some point-visit criterion, for example minimizing the maximal time between visits to a point (e.g. [6, 8]). In *multiagent adversarial patrol* the agents' goal is to maximize their chances of detecting an adversary that tries to penetrate through their patrol path undetected (e.g. [4, 1]).

[1] Of course higher dimensions are also possible.

In this paper we concentrate on the continuous 2-dimensional frequency-based multiagent patrol problem, with discrete points of interest, in complex environmental conditions. In this problem, we are given a graph $G = (V, E)$, and we need to define patrol paths for a team of k agents that will minimize the maximal time some vertex of the graph is left unvisited. The complexity of the environment is expressed via the cost of travel between each pair of vertices of the graph.

Consider the problem of ship patrol, i.e., patrol by agents (ships) in marine environments. When designing algorithms for ships in such environments, it is critical to consider the impact of the environment and the specifications of the ship on the behavior of the ship that might also change over time. In our case, we incorporate the shape and engine power of the ship, and environment conditions such as water currents and winds in modeling the environment as graphs, namely in its affect on the cost of travel between vertices of the graph.

Current strategies for multiagent patrol offer, roughly, two alternatives for agents' patrol paths. The first strategy, denoted herein as SingleCycle, is to create one simple cyclic path that travels through the entire area (graph), and to let all agents patrol along this cyclic path while maintaining uniform distance between them [8, 6]. The second strategy, denoted herein by UniPartition, is to partition the area (graph) into k distinct subareas, where each agent patrols inside one area.

We suggest a third, general, strategy, denoted by MultiPartition, in which the graph is divided into m subgraphs, $m \leq k$, such that a subteam of agents jointly patrols in each subgraph. We define the problem of finding k (possibly overlapping) paths for the agents such that the maximal time between any two visits at a vertex is minimized, and show that the problem is $\mathcal{NP}$-Hard. The SingleCycle and UniPartition strategies, as special cases of MultiPartition, are also intractable in general graphs.

An additional version of the problem, in which the graph is to be divided into m *disjoint cycles*, where the k agents are divided among the cycles, is also intractable in general graphs. We therefore investigate the problem on a special family of graphs, which are cyclic graphs with non intersecting shortcuts (diagonals), called *outerplanar graphs* [5]. This simplified, yet realistic, family of graphs have some characteristics that can be of help when looking for optimal solutions to the multiagent patrol problem. For example, an optimal SingleCycle strategy is unique and can be found in linear time. Unfortunately, the time complexity of the general problem of finding an optimal MultiPartition strategy even in such graphs appears to be intractable as well. We therefore suggest a heuristic algorithm HeuristicDivide for finding a partition of the graph into disjoint cycles in the outerplanar marine environment, and a partition of the k agents among those cycles.

For evaluation of our work we used a custom developed ship simulator, OurSim, that was designed to realistically model ship movement constraints in marine environments. OurSim simulates the specifications of the ship, namely, the weight, engine power, and shape of the ship; and how it is influenced by the environmental conditions, including water, currents and winds. We first

show that in a simple scenario in which the optimal MultiPartition strategy is easily computable, it outperforms the other two strategies (SingleCycle and UniPartition). We then show that in a more complex environment, our heuristic algorithm HeuristicDivide, following the MultiPartition strategy, performs significantly better than the tractable SingleCycle strategy.

2 Related Work

The problem of multiagent patrol can be roughly divided into two problems: multiagent frequency-based patrol (e.g. [10, 6, 8]), and multiagent patrol in adversarial environments (e.g. [1, 4]). The problems differ in the objective function that should be optimized, namely optimizing frequency-based criteria or optimizing probability of detecting events controlled by an adversary (respectively). In this paper we focus on the problem of frequency-based patrol, in which we aim at minimizing the time between two visits at a point.

Mechado *et al.* [10] were the first to define the problem of multiagent patrol in graph environments, and introduced the notion of *idleness*, meaning the time between two visits in a vertex of the graph. They consider environments with uniform length edges, and perform an empirical evaluation of various architectures for multiagent patrol in different graphs. Generally, they distinguish between reactive and cognitive agents, where the former are locally-driven agents using minimal coordination (if any), and the latter might try to use global state while deciding their next move. They did not theoretically define nor evaluate the multiagent patrol problem on graphs, nor did they consider complex environments.

The first *theoretical* analysis of the problem of multiagent patrol was given by Chevaleyre [6]. Chevaleyre refers mainly to the *worst idleness* criterion, which is the largest amount of time that some vertex remained unvisited throughout the execution of the patrol algorithm. He discusses two possible strategies: a *Cyclic* strategy, in which one cyclic path travels through the entire graph, and all agents follow this path (denoted by SingleCycle) and a *Partition-based* strategy, in which the graph is partitioned into k distinct subgraphs (k being the number of agents), where each agent visits one subgraph in a cyclic tour (denoted by UniPartition). He analyzes the idleness criterion in each of these strategies, using an approximation algorithm to the Traveling Salesman Problem (TSP) under the assumption that the triangle inequality holds. In our work we redefine the multiagent patrol problem in a more general form, in which the graph is possibly partitioned into disjoint subgraphs, however agents can share a subgraph (denoted by MultiPartition). In addition, we do not assume that the triangle inequality holds (as in many realistic scenarios this assumption is not true). This general definition includes also the two strategies proposed by Chevaleyre as subcases, i.e., $\mathsf{SingleCycle}, \mathsf{UniPartition} \subseteq \mathsf{MultiPartition}$.

Ahmadi and Stone [2] investigated the multiagent patrol problem in prioritized environments, i.e., where different areas require different attention from the agents. They suggest a new, learning-based method for determining the optimal patrol path for each robot, which is adapted to the different constraints of

the environment. In this paper we consider nodes with uniform priority, i.e., all nodes should have minimal possible idleness.

Multi-robot patrol in areas was considered by Elmaliach *et al.* [8], which offered an optimal patrol algorithm using a cyclic strategy, i.e., one cyclic path with minimal cost passes through the entire area, and all robots coordinatedly travel along this path. Their solution assumes that the area and the size of the robots meet several constraints, allowing them to find an optimal solution (minimal cost cyclic path) in polynomial time.

Elmaliach *et al.* [9] considered the problem of frequency-based multi-robot patrol along an open fence, where they evaluated their patrol algorithm according to different frequency criteria. They offer a model for determining the patrol path of the robots in this asymmetric environment, which takes into account the motion model of the robots (acceleration and velocity changes, and error in motion). This model results in a realistic cost of travel along the fence. In our work we consider the case of uncertain cost of travel that depends on the environment, and is updated during the patrol execution (rather than fixed in advance given the physical constraints of the robots). Moreover, both the general graph model and the restricted outerplanar graph model pose a considerable challenge compared to the linear environment of a fence, due to the number of new possibilities of patrol paths for the robots.

Recent work by Marier *et al.* [11] describe a solution to multiagent patrol on graphs with non-uniform weights on the vertices of the graph, corresponding to the importance of the node. They offer two algorithms for patrolling. The first is reactive (based on consequences calculated for a very limited horizon) and the second is based on an online heuristic algorithm for solving POMDPs. They describe the problem as information gain (rather than idleness), and examine the performance of their heuristic algorithms with respect to the known (or unknown) duration of the patrol. They do not consider uncertainty in travel time, nor do they refer to the graph theoretic problem.

3 Motivation - Ship Patrol and Marine Environment

As surveyed in Section 2, the problem of multiagent patrol has become a canonical problem in multiagent (and specifically multi-robot) systems in the past several years. In this paper, we investigate this problem in a realistic ship simulator that we have designed in our lab and that introduces important new travel-time constraints to the problem. (a technical description of the simulator is given in Section 5). The general problem defined in graph environments requires a team of k agents to repeatedly visit all N nodes of the given graph while minimizing the longest time a node has remained unvisited by some robot. We first look into the simplest scenario found in the literature, namely patrol in circular environments. In such environments, the patrol path is linear and the algorithm that optimizes point-visit frequency was shown to be an algorithm that requires all robots to travel in a coordinated manner and maintain uniform (time) distance between each neighboring pair of robots along the path (e.g. [6, 8]).

After implementing a simple scenario, in which three ships patrol along a cyclic path in order to optimize point-visit frequency over a set of 10 points (see Figure 1), we discovered several interesting phenomena. First and foremost, we saw that in environments in which the cyclic path is not along a perimeter of some closed structure (for example airports, prisons and military bases), it is necessary to consider paths that create shortcuts between point of interest, and allow traveling from one point to another that are not necessarily along the cyclic path. Moreover, even in such environments (especially military bases, factories and airports), there usually are roads going through the closed area, creating shortcuts in transitions between points along the perimeter. Second, when we applied different water current and wave conditions, the cost of traveling (corresponding to travel time) along some edges of the graph became very high, encouraging the use of the shortcuts for traveling between points of interest. We examined solutions that exist in the literature for defining optimal patrol paths for a team of robots, and found only the SingleCycle and UniPartition solutions, which consider the entire cyclic path, or divide the patrol paths into k areas, each under the responsibility of a single agent (respectively).

In the example illustrated in Figure 1, we examined two scenarios. In the first we had no currents or winds (a "clean" environment), and in the second we introduced winds and currents, specifically currents between points p_2 and p_3, and between points p_6 and p_7. The travel time between p_2 and p_3 and between p_6 and p_7 (in both directions) is, therefore, very high. The *worst idleness* results we describe here are calculated by averaging the idleness of each point along a 10 minute execution of the patrol in the simulator, and choosing the point with highest average idleness value. In the first (clean) environment, when the three ships executed the SingleCycle strategy, we got an average worst idleness of 651 seconds. When dividing the set of 10 points among the three ships, where one ship patrols along points p_3, p_4, p_5, p_6 in a circular path, and the other two ships divide the remaining points between them (the UniPartition strategy) we get worst idleness of 786 seconds. However, when looking closely at this example, it can be clearly seen that there exists another possibility: letting one ship patrol along p_3, p_4, p_5, p_6, and having the other two share the cycle $p_1, p_2, p_7, p_8, p_9, p_{10}$ and patrol, coordinatedly, with uniform time distance between them (the MultiPartition strategy). By executing this algorithm, we got worst idleness of 614 seconds, a major improvement compared to the previous two strategies. This improvement becomes more substantial when we examine the situation with currents. Now, the SingleCycle strategy yields worst idleness of 795 seconds, the UniPartition yields worst idleness of 792 seconds, and the MultiPartition strategy yields worst idleness of 613 seconds (note that in the last two cases there is no significant change from the clean environment, as the ships did not travel through the stormy weather, i.e., where the strong currents are).

This example, along with other similar phenomena we have viewed in our simulator, motivated us to redefine the problem of multiagent patrol in a more general form, denoted as MultiPartition, and discuss possible solutions to the problem in circular environments, but with additional shortcuts between the points of interest.

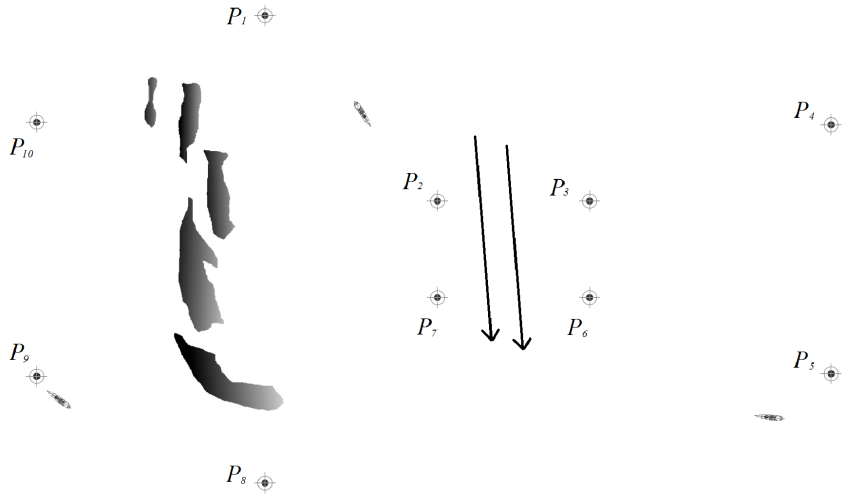


Fig. 1: An example of a scenario handled by the simulator. The circles represent the points of interest (nodes of the graph), and the drop shapes are the ships. The large grey shapes are obstacles, and the drawn arrows indicate the direction of the water current.

4 Problem definition and complexity

In this section, we define the general problem of multiagent frequency based patrol on general graphs. We describe the decision version of the problem, where the input is the graph $G = (V, E)$ ($|V| = N$), an integer $k < N$ that corresponds to the number of agents, and an integer f which is the maximal worst idleness, i.e., the maximal requested idleness from all vertices of the graph (similar to the definition in [6]). Formally, if f_i denotes the idleness of a vertex v_i, then the worst idleness of the graph G, wi(G), guaranteed by an Algorithm $\mathcal{A}$ is defined as $\text{wi}(G) = \max_{1 \leq i \leq N}\{f_i\}$.

Note that real world constraints dictate modeling the world with directed graphs, i.e., the travel time from a vertex v to a vertex u is not necessarily the same as that from u to v. However, we assume that the graph is *symmetric*, i.e., if an edge exists from v to u, then an edge exists also from u to v (not necessarily of the same cost). We therefore describe the general problem on undirected graphs. Once a cycle is defined, the algorithms will decide whether to go clockwise or counterclockwise along the cycle, depending on the direction that has lower cost.

4.1 Multiagent patrol in general graphs

Definition: Multiagent Graph Patrol (MGP)
Given a graph $G = (V, E, C)$ where $|V| = N$, and $\forall(v_i, v_j) \in E, c_{ij} \in C$ is the associated cost of the edge, an integer $k < N$ denoting the number of agents, and a desired maximal worst idleness target f, is there a division of V into $m \leq k$

cyclic paths $V_1, V_2, \ldots, V_m$ (not necessarily simple), each assigned with k_i agents such that all k_i agents visit all vertices in V_i and $\sum_{i=1}^{m} k_m = k$, such that the worst idleness wi(G) is at most f?

In the following theorem we show that the MGP problem is $\mathcal{NP}$ complete for general k's.

Theorem 1. *The* MGP *problem is* $\mathcal{NP}$ *complete.*

Proof. First, the MGP problem is in $\mathcal{NP}$, since given a solution, i.e., a division of the graph into m paths, it is possible to verify whether wi(G) is indeed f using a variation of the Algorithm AssignKAgents. MGP is $\mathcal{NP}$-Hard for general k by a simple Turing reduction from the decision version of the graphical traveling salesman problem ($GTSP$). Specifically, we can find whether there exists a minimal tour of size at most f that travels through all nodes in the graph at least once by solving the MGP problem given f as input, and $k = 1$.

In our work, we would like to consider a special case of the MGP problem, in which each path $V_1, \ldots, V_m$ is a *simple* cycle, i.e., it is a closed path with no repeated vertices. Moreover, we restrict our attention to sets of distinct paths that do not share any vertices, i.e, $V = V_1 \bigoplus V_2 \bigoplus \ldots \bigoplus V_m$ (distinct simple cycles). This problem handles restrictions that are more suitable for realistic robotic environments, in which two requirements are met:

1. Two robots will not meet during the execution of the algorithm, thus will not interfere with one another during the patrol.
2. Robots will not need to interact outside of their subteam, i.e., the patrol algorithm requires only local coordination (unless the environment changes the optimality of the current patrol algorithm). Moreover, if different human operators observe each subteam, it does not require continuous coordination among the human operators.

The formal definition of the problem is as follows.

Definition: Multiagent Cyclic Graph Patrol (MCGP)
Given a graph $G = (V, E, C)$ where $|V| = N$, and $\forall (v_i, v_j) \in E, c_{ij} \in C$ is the associated cost of the edge, an integer $k < N$ denoting the number of agents and a desired maximal worst idleness target f, is there a division of V into $m \leq k$ *distinct simple* cycles $V = V_1 \bigoplus V_2 \bigoplus \ldots \bigoplus V_m$, each cycle V_i assigned with k_i agents coordinatedly traveling along V_i and $\sum_{i=1}^{m} k_m = k$, such that the worst idleness wi(G) is at most f?

The MCGP is a special case of the MGP, in which the cyclic paths are required to be disjoint, and each cycle is simple (with no repeated vertices). The $\mathcal{NP}$-Hardness proof resembles the proof for the MGP problem, thus we conclude the following.

Corollary 1. *The* MCGP *problem on general graphs is* $\mathcal{NP}$*-Hard.*

We can define the worst idleness in this problem as follows. If k' agents visit a cyclic path, denoted by V^C, where $V^C = \{v_{i_1}, v_{i_2}, \ldots, v_{i_l}\}$, $v_{i_j} \in V(G)$,

$(v_{i_j}, v_{i_{j+1 mod l}}) \in E(G)$, and denote the total weight of edges in the cycle by $w(V^C) = \sum_{h=1}^{l} c_{i_j i_{(j+1 mod l)}}$, then $\forall v_{i_j} \in V^C$, $f_{i_j} = \frac{w(V^C)}{k'}$. Therefore if G is divided into m distinct cycles, where each cycle V_i^C is visited by k_i agents, then $\mathrm{wi}(G) = \max_{1 \leq i \leq m}\{\frac{w(V_i^C)}{k_i}\}$.

Algorithm *AssignKAgents* (described below) is given as input m cyclic paths, an integer k corresponding to the number of agents, and a maximal idleness f, and has to answer the question of whether k agents are sufficient to guarantee a maximal idleness of f on the given graphs. It returns the assignment of number of agents per graph ($K = \{k_1, \ldots, k_m\}$ such that $\sum_{i=1}^{m} k_i = k$ and k_i agents are necessary to visit G_i in order to guarantee minimal idleness f) and the maximal idleness guaranteed by this assignment (f_{loc}). Denote the edges along the cyclic path G_i in clockwise direction by G_i^{cw} and in the counterclockwise direction by G_i^{ccw}. The algorithm will work for either symmetric directed graphs (in which it will refer to the direction with minimal cost — either going clockwise or counterclockwise) or undirected graphs (in which $w(G_i^{cw}) = w(G_i^{ccw})$ where $w()$ is the cycle weight, or length, function).

Algorithm 1 $< K, f_{loc} >$ = Algorithm AssignKAgents$(\{G_1, \ldots, G_m\}, k, f)$

$C \leftarrow 0$, $K \leftarrow \emptyset$
for $i \leftarrow 1, \ldots, m$ **do**
  $w_i \leftarrow min\{w(G_i^{cw}), w(G_i^{ccw})\}$
  $c_i \leftarrow argmin_{G_i^{cw}, G_i^{ccw}}\{w(G_i^{cw}), w(G_i^{ccw})\}$
  $k_i \leftarrow \lceil w_i / f \rceil$
  if $k_i > k$ **then**
    Return $\emptyset$
  end if
  $K \leftarrow K \bigcup k_i$, $C \leftarrow C \bigcup c_i$
  $k \leftarrow k - k_i$
end for
$f_{loc} \leftarrow \max_{1 \leq i \leq m}\{w(C[i])/K[i]\}$
Return K, f_{loc}

4.2 The multiagent patrol problem in outerplanar graphs

Motivated by the problem of multi-robot *perimeter patrol* (e.g. [1]), we examine the MCGP problem in circular environments. However, we add more realistic considerations to the environment, namely adding possible *shortcuts* between vertices that pass inside the circle. To avoid possible interference by agents that travel along the edges, we require the inner edges not to intersect one another. The resulting graph is planar, and moreover, it is a *biconnected outerplanar* graph [5], i.e., it is a planar graph that is cyclic, and there are no nodes that are inside the cycle (all nodes in the graph are on the same outer face).

An example for such a graph is shown in Figure 2. In this example, if an edge existed between v_4 and v_{11}, then the graph would not be planar (as the

edge (v_5, v_{11}) crosses the edges (v_6, v_{12}) and (v_8, v_{12})). Also, if an edge existed between v_{13} and v_{11}, then the graph would remain planar, but would not be outerplanar (as v_{12} is not adjacent to the outer surface anymore).

In the family of outerplanar graphs, several hard problems become very easy to solve. For example finding a Hamiltonian cycle is done in linear time, as the only possible simple cycle that visits all nodes in the graph is the external cycle. Therefore also finding the optimal SingleCycle strategy is done in linear time, as the solution is unique. Another interesting characteristic of outerplanar graphs is that every subgraph of an outerplanar graph is outerplanar, thus a biconnected component of a subgraph of an outerplanar graph also has a unique Hamiltonian cycle ([5]).

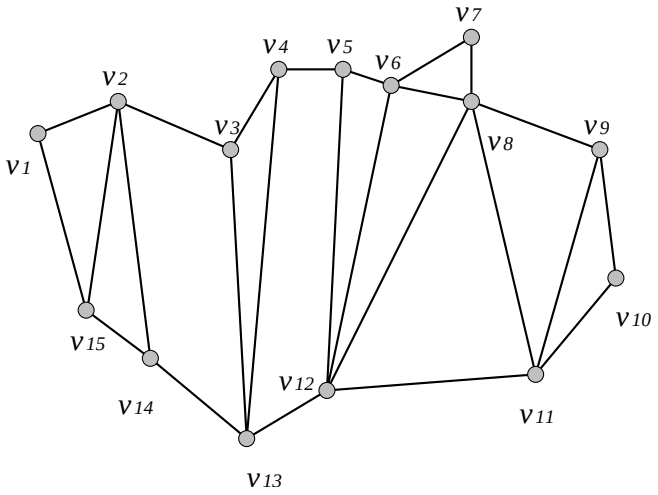


Fig. 2: An outerplanar graph: The vertices of the graph (points of interest for the agents along the patrol) are all adjacent to one external face, and the inner edges (shortcuts) do not intersect.

We draw a connection between disjoint cycles and biconnected components in Lemma 1. Generally, a biconnected component in an outerplanar graph has a unique Hamiltonian cycle, which is the outer-cycle that visits all vertices. We would therefore like to find a way to use this property in order to find disjoint cycles, as defined in the MultiPartition strategy. As a first step, we look at the case of dividing the graph into two disjoint cycles. We show in the lemma that in order to find such disjoint cycles, it is sufficient and required (in the general case) to consider all biconnected components that are created by the removal of two edges from the graph. We later use this property in the heuristic algorithm for solving the MCGP problem in outerplanar graphs.

Consider the outerplanar graph $G = (V, E)$ in Figure 2. We will demonstrate why the removal of only one edge might not result in all disjoint cycles, and

how we can achieve either two or three disjoint cycles by removing every pair of edges. First, by removing only one edge $e \in E$ and computing the biconnected components in the remaining graph $G = (V, E \setminus \{e\})$ it would be impossible to get the division of the graph into the two disjoint cycles $V_1^C = (v_9, v_{10}, v_{11})$ and $V_2^C = G \setminus V_1^C$: Removing (v_8, v_9) would result in the biconnected components $\{v_9, v_{10}, v_{11}\}$ and $\{v_1, \ldots, v_8, v_{11}, v_{12}, \ldots, v_{15}\}$, which are not disjoint, thus their Hamiltonian cycles are not disjoint. The removal of (v_8, v_{11}) results in one biconnected component (thus cycle) G, and the removal of (v_{11}, v_{12}) results in the cycles $\{v_8, v_9, v_{10}, v_{11}\}$ and $\{v_1, \ldots, v_7, v_8, v_{12}, \ldots, v_{15}\}$, which are again not disjoint. Therefore by removing only one edge we could not get the disjoint cycles V_1^C and V_2^C. However, by removing (v_8, v_9) and (v_8, v_{11}), this division is achieved. Note that the removal of the pair of edges (v_2, v_3) and (v_4, v_5) results in three disjoint biconnected components (thus cycles): $\{v_1, v_2, v_{14}, v_{15}\}$, $\{v_3, v_4, v_{13}\}$ and $\{v_5, v_6, \ldots, v_{12}\}$.

Lemma 1. *Given a biconnected outerplanar graph $G = (V, E)$, each division of G into two disjoint biconnected components can be achieved by removing one pair of edges and computing the biconnected components of the remaining graph. If removing one pair of edges, the number of remaining disjoint biconnected components (excluding disconnected vertices) will not exceed* 3.

Proof. We first show that two disjoint biconnected components $V_1^C = \{v_1, \ldots, v_l\}$ and $V_2^C = \{u_1, \ldots, u_h\}$ in an outerplanar graph G can be connected by either two or three edges. First, assume that V_1^C and V_2^C are connected by only one edge (v_i, u_j), $v_i \in V_1^C$ and $u_j \in V_2^C$. Therefore both v_i and u_j are articulation vertices (their removal disconnects the graph), contradicting the nonseparability characteristic of the biconnected graph G.

We now show that V_1^C and V_2^C cannot be connected by more than three edges. Without loss of generality, assume that both V_1^C and V_2^C vertices are ordered clockwise in ascending order, and that V_1^C is left of V_2^C. Therefore, since G is outerplanar, two edges connecting V_1^C and V_2^C are necessarily (v_i, u_j) and (v_{i+1}, u_{j+1}). Moreover, there cannot be any other edge (v, u) connecting V_1^C and V_2^C such that $v \notin \{v_i, v_{i+1}\}$ and $u \notin \{u_j, u_{j+1}\}$, otherwise some vertex $v_i' \in V_1^C$ and/or $u_j' \in V_2^C$ are not adjacent to the outer face, contradicting the outerplanar definition of G. Therefore the only possible edges connecting the two disjoint components are between vertices v_i, v_{i+1} and u_j, u_{j+1}. Since an outerplanar graph cannot have a subgraph that is a clique of size 4 [5], and since necessarily $(v_i, v_{i+1}) \in E$, $(u_j, u_{j+1}) \in E$ and we've shown that $(v_i, u_j), (v_{i+1}, u_{j+1}) \in E$, then there could exist only one more edge connecting the two components: either (v_i, u_{j+1}) or (v_{i+1}, u_j), but not both.

Since two disjoint biconnected components can be connected by at most three edges, by removing every possible pair of edges from the graph, at some point we will necessarily remove two of the connecting edges of V_1^C and V_2^C, resulting in two disjoint biconnected components. Moreover, the removal of one edge can result in dividing the graph into two disjoint components only if these components are connected by only two edges. Therefore, removing a pair of edges can result in up to three biconnected components (cycles) and no more than that.

This lemma results in the fact that finding two disjoint cycles (and possibly 3) in a graph can be done efficiently in time complexity of at most $\binom{|E|}{2}$. Since finding the partition of k into two (or three) components is done efficiently as well, the MCGP problem can be solved optimally in polynomial time if m is restricted to be 2.

Corollary 2. *In an outerplanar graph $G = (V, E)$, finding a division of the graph into up to two disjoint simple cycles V_1^C and V_2^C such that $V = V_1^C \bigoplus V_2^C$ and $wf(G)$ (for any value of k) is minimized can be done in polynomial time, using Algorithm* DivideTo2Cycles.

Algorithm DivideTo2Cycles receives as input the graph $G = (V, E)$ and the maximal frequency criterion f that should be met, and returns the best division of the graphs into two components such that the division results in maximal idleness of at most f. If no such division exists, it returns the empty set. Note that in order to get all possible divisions of G into two disjoint cyclic paths, the algorithm should be given the value $f = w(G)/k$. The algorithm removes all possible pairs of edges from the original graph, and computes the biconnected components of the remaining graph. For those biconnected components, it checks whether there is an assignment of k into those biconnected components such that the maximal idleness of the graph is at most f, using Algorithm AssignKAgents. By Corollary 2, the algorithm examines all possibilities of dividing the graph into two cycles (which has time complexity of $\mathcal{O}(|E|^2)$). Since checking all possibilities of partitioning the number k into at most 3 components is polynomial in k, and determining the idleness is linear in $|E|$, then the total time complexity of Algorithm DivideTo2Cycles is $\mathcal{O}(|E|^3)$.

Algorithm 2 $S =$ Algorithm DivideTo2Cycles$(G = (V, E), f, k)$

$S \leftarrow \emptyset$
for Every pair of edges $e_i = (v_i, u_i)$ and $e_j = (v_j, u_j)$, $e_i, e_j \in E$ **do**
  $E' \leftarrow E \setminus \{e_i, e_j\}$
  $U \leftarrow$ biconnected components of $G' = (V, E')$
  if $< K, f_{loc} >=$ AssignKAgents$(U, k, f)! = \emptyset$ **then**
    $f_{loc} \leftarrow$ optimal assignment of k agents to U
    $S \leftarrow S \bigcup \{< U, K, f_{loc} >\}$
  end if
end for
Return $S[i]$ for which f_{loc} is minimal

As shown by de Mier and Noy [7], the number of cycles in a maximal outerplanar graph is exponential in the number of vertices of the graph, thus if we need to examine all possible sets that generate a direct sum of V it will result in at least an exponential time complexity. We therefore believe (although not proven) that the MCGP problem, also in the simple biconnected outerplanar environment, is intractable, as the number of all possibilities of the division of the graph into up to k subgraphs grows exponentially with k.

We therefore describe a heuristic algorithm, Algorithm HeuristicDivide, for finding a division of the graph into disjoint cycles.

4.3 Heuristic algorithm for multiagent patrol in outerplanar graphs

We describe in this section a heuristic algorithm, Algorithm HeuristicDivide, for finding a set of any number of disjoint cycles in an outerplanar graph (based on Algorithm DivideTo2Cycles), and dividing the k agents between these disjoint cycles in a way that aims to find a low overall maximal idleness.

The algorithm applies algorithm DivideTo2Cycles once, then further applies itself recursively on each element of the set of disjoint biconnected components that yield lowest idleness. In this way it performs a greedy heuristic search and halts once it completes all possible divisions up to k cycles. The algorithm receives as input the graph G, the number of agents k, and $f = \frac{w(G)}{k}$.

Note that once the cycles, the direction of travel along the cycles, and the division of the agents among the cycles are determined, it is only left to distribute the agents along the cycles (number of agents per cycle as determined by the algorithm), and require the agents in each cycle to maintain uniform distance between them and continue traveling coordinatedly along their circular path.

Algorithm 3 Algorithm HeuristicDivide$(G = (V, E), f, k)$

1: $ResSet \leftarrow$ DivideTo2Cycles(G, f, k)
2: **if** $ResSets = \emptyset$ **then**
3: Return G
4: **end if**
5: $CurChoice = argmin_{<U_i, K_i, f_i> \in ResSet}\{f_i\}$
6: Return $\bigcup_{G_i \in U(CurChoice)}$ HeuristicDivide(G_i, K_i, f_i)

Time complexity of Algorithm HeuristicDivide**:**
The time complexity of HeuristicDivide is defined by two components: The depth of the search and the time to process each level of the search tree. Since at each step we deepen the recursion we lose at least one vertex (as the minimal division to two distinct cycles is to a vertex v and to a cyclic graph $G \setminus \{v\}$), the depth of the tree is at most $k - 1$. The time complexity of AssignKAgents is linear in the number of edges, and the complexity of finding biconnected components is also linear in outerplanar graphs. Therefore the complexity of examining each pair of edges is $\mathcal{O}(|E|)$. When we go down in the recursion, if E is divided into up to three disjoint biconnected components E_1, E_2 and E_3, then we have complexity of $\mathcal{O}(|E_1|^2 + |E_2|^2 + |E_3|^2)$ where $0 \leq |E_1| \leq |E_2| \leq |E_3| < |E|$ and $|E_1| + |E_2| + |E_3| = |E|$. Therefore, since $\mathcal{O}(|E_1|^2 + |E_2|^2 + |E_3|^2) < \mathcal{O}(|E|^2)$ it leads to a total time complexity of $\mathcal{O}(k|E^2|)$.

5 Empirical evaluation

We evaluated HeuristicDivide under realistic marine environment using our novel OurSim Simulator. In the following section we describe the simulator, the experimental setting and the empirical results from executing the patrol algorithms described above.

5.1 The OurSim Simulator

For our experiments, we use a custom-designed naval surface navigation simulator that will soon be open-source, that we call OurSim.[2] The OurSim simulator is written as a supporting platform for research on autonomous sea navigation. It uses realistic $2D$ physical models of marine environments and sea vessels, and runs both in GUI and in non-GUI modes. Figures 1 and 3 show snapshot of the simulator's GUI (with blue highlights added), taken during a real-time simulation, which animates navigating ships from a top view.

The simulator's core contains three main modules: a *Sea Environment* module, a *Ship* module, and a *Decision-Making* module. The sea environment module includes models of winds, water currents, waves, and obstacles. The ship module models all relevant aspects of a ship, including the ship's physical properties, sensing capabilities, and ship actuators. The decision making-module implements an agent that controls a ship autonomously. At each time step, the agent receives the perceptions sensed by the ship, processes them to update its current world state, and decides on control actions for the ship based on its current world state and its decision-making strategy.

All of the above components are plug-and-play: each one can easily be replaced by a component of the same type that uses alternative modeling. This includes the agent's perception processing algorithms, world model, and decision making strategy; the ship's model; and the different environmental models.

5.2 Experimental Setting and Results

In our experiments, we examined several different environments. Due to space constraints, we describe one interesting environment, in which the graph is outerplanar with a large number of possible division of the graph into disjoint cycles. The marine environment was implemented in OurSim, and illustrated in Figure 3 (distances are shown in meters). In this environment, $N = |V| = 36$, and the number of ships (k) varies from 1 to 30. Although the points of interest are arranged in two straight lines, this environment can become equivalent to many realistic cases by controlling the currents between nodes thus controlling the edge lengths. Moreover, this environment can represent a man-made group of points of interest, for instance a sequence of oil rigs.

We examined four different scenarios, where in each one the sea conditions were different. In the first environment (denoted as *Sea Condition* 0), there were no currents and the cost of traveling between two points correlates to the

[2] Name removed for blind review.

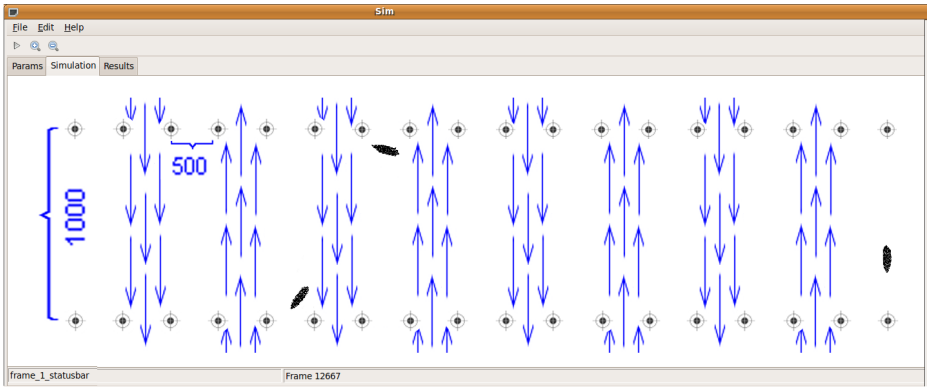


Fig. 3: The evaluation environment.

geographical distance. We then gradually added more currents to the system with three different strengths - *Sea Condition* $1, 2$ *and* 3, for weak, medium and strong, respectively, where their directions was as shown in Figure 3. We ran Algorithm HeuristicDivide initially with the worst idleness of ∞ and let it find the best division it can. We then simulated the patrol-division returned by the algorithm for 20,000 seconds (333 minutes), and reported the worst node-idleness, i.e., the highest average time between consecutive node visits, for any node. Note that in all the results, lower values are *better*.

Figure 4 shows the worst idleness resulting from HeuristicDivide when running on sea conditions 0–3. Note that in Sea Condition 0 (no currents) the algorithm always returned the SingleCycle strategy, which is indeed the best division for the case of no currents. As expected, the worst idleness becomes higher as the sea conditions become rougher.

In order to evaluate the performance of our algorithm, we compared its worst idleness with the worst idleness of the following:

1-Loop The result of the SingleCycle strategy, i.e., a patrol algorithm that instructs all ships to patrol around the cycle while maintaining uniform distance between adjacent pairs.

k+1 Incremental change. In this case, we assumed the HeuristicDivide algorithm was computed for k ships, and upon the arrival of a new ship it is added to the cycle with highest worst idleness (for the best improvement in idleness). This is compared to running HeuristicDivide with $k+1$ ships, and the goal is to check whether HeuristicDivide, as a heuristic algorithm, does better than just a straightforward increment.

k-1 Decremental change. In this case, we assumed the HeuristicDivide algorithm was computed for $k+1$ ships, and a ship needs to be removed from the system. We assume that in this case a ship is removed from the cycle with best worst idleness (for minimal increase in the worst idleness). This is compared to running HeuristicDivide with $k-1$ ships, again with the goal of checking whether HeuristicDivide does better than just a straightforward adjustment.

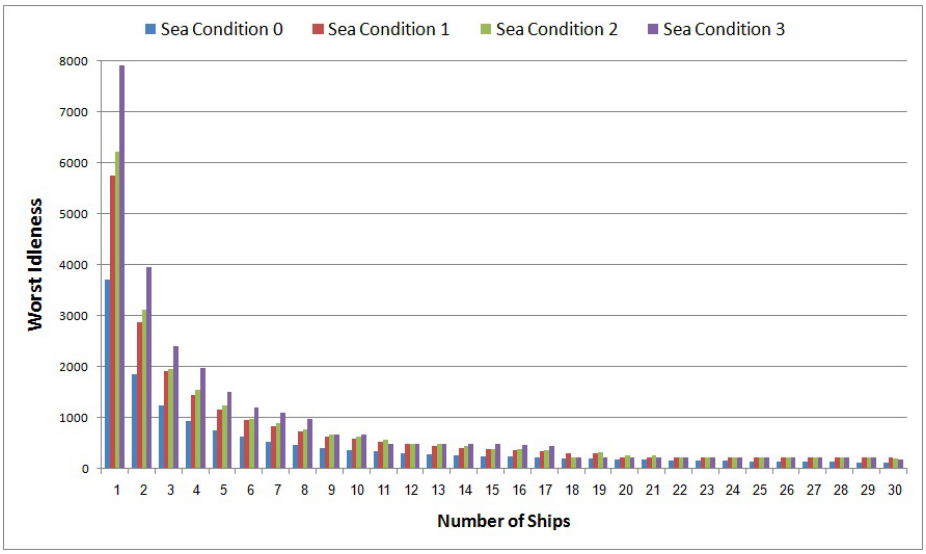


Fig. 4: The worst idleness returned from Algorithm HeuristicDivide in the four different sea conditions examined.

The results are shown in Figures 5, 6 and 7 for Sea Conditions 1, 2 and 3, respectively. Note that since for Sea Condition 0 HeuristicDivide's solution is always the SingleCycle strategy, for every given k HeuristicDivide's solution is identical to the 1−loop, $k+1$ and $k-1$ solutions.

In Sea Conditions 2 and 3, algorithm HeuristicDivide performs statistically significantly (using paired t-test) better than the 1−loop algorithm and the incremental $k+1$ and decremental $k-1$ cases, with p-values < 0.002 for Sea Condition 3 (in all cases) and p-value < 0.04 for Sea Condition 2 (in all cases). In Sea Condition 1, although the results are generally better, no significant results are achieved. Interestingly, there are some cases in which 1−loop slightly outperforms HeuristicDivide, even though HeuristicDivide (theoretically) does not divide a cycle into smaller cycles unless it improves the worst idleness. The reason for that lies in the fact that deciding whether to break a big cycle into smaller cycles is done using an estimation of the cost of traveling along an edge, by averaging across simulations of ships patrolling along the edges of arbitrary paths, which are usually different then the paths found by HeuristicDivide's solution. For instance, consider the case where the final solution requires a 180° turn towards the next point. The physics of the ship movement dictates the ship to travel in an arc, which makes the path to the next point longer than its estimate. We leave the incorporation of the cost of traveling along angular paths in HeuristicDivide to future work.

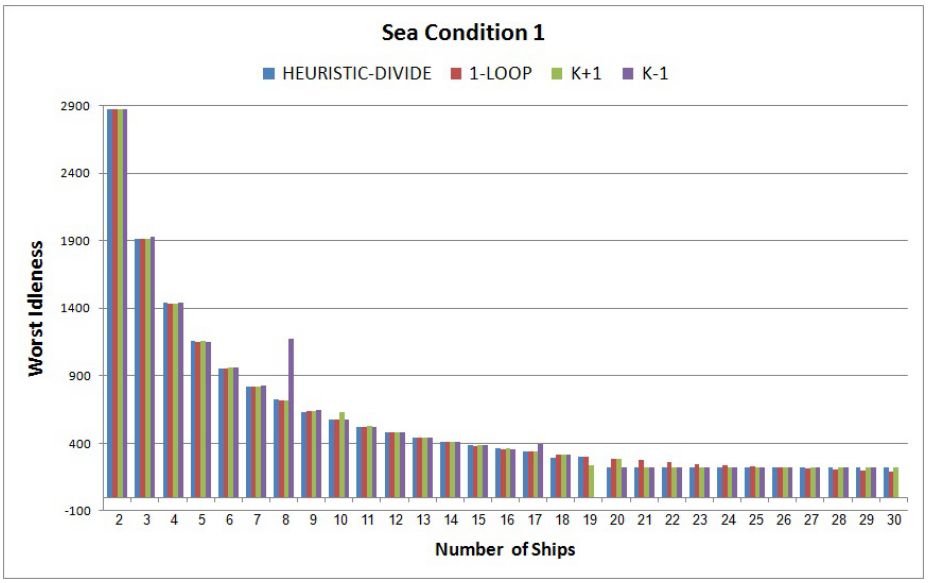


Fig. 5: The results of HeuristicDivide, 1−loop, $k+1$ and $k-1$ in Sea Condition 1 (weak currents).

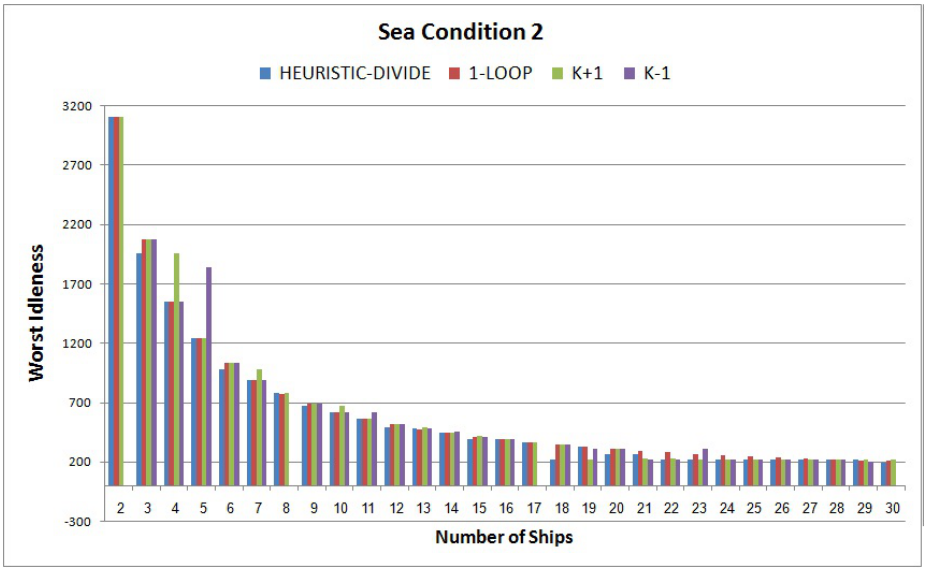


Fig. 6: The results of HeuristicDivide, 1−loop, $k+1$ and $k-1$ in Sea Condition 2 (medium currents).

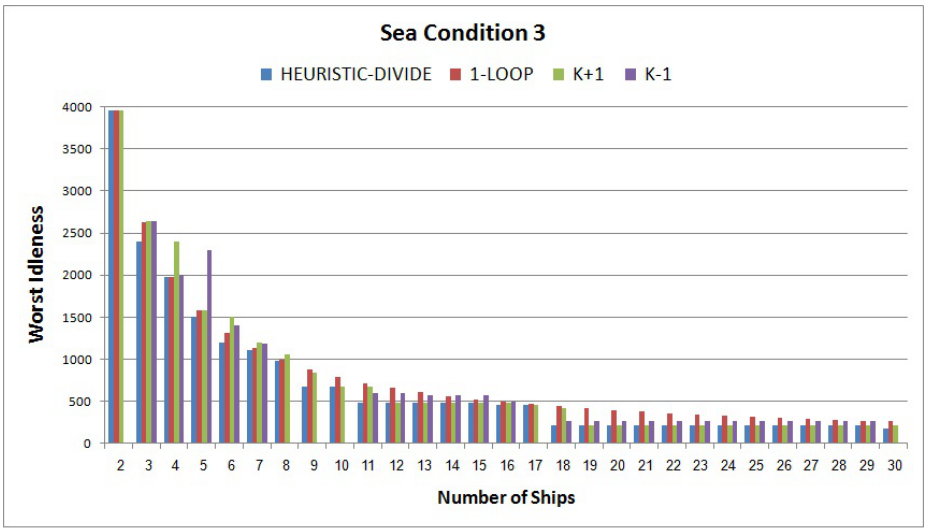


Fig. 7: The results of HeuristicDivide, $1-$loop, $k+1$ and $k-1$ in Sea Condition 3 (strong currents).

6 Conclusions and Future Work

In this paper we introduced a new class of strategies for the frequency-based multiagent patrol problem suitable for multiagent patrol in complex environmental conditions. In this new strategy class, which we call MultiPartition, a graph is divided into disjoint cycles in which a subteam of the agents travel coordinatedly such that the maximal time between visits to interest points is minimized. This strategy class is a generalization of existing strategies that either create one cyclic path throughout the entire graph (SingleCycle strategies) or divide the graph into k disjoint subgraphs (k being the number of agents), where each agent patrols in its own subgraph—the UniPartition strategy. We showed that finding an optimal strategy to the problem is $\mathcal{NP}$-Hard in the general case, and also intractable in a realistic simplified family of outerplanar graphs. We then introduced a heuristic algorithm that divides the outerplanar graph into disjoint cycles. We evaluated our heuristic algorithm in a custom-developed ship simulator that realistically models ship movement constraints such as engine force and drag, and reaction of the ship to environmental changes. Results indicate that this algorithm significantly improves the frequency of visits compared to other known patrol strategies.

This paper opens up several directions for future work. First, it would be interesting to consider the problem of multiagent patrol in prioritized environments, i.e., where vertices of the graph should be visited with different frequencies. Second, we intend to add more learning methods for determining the cost of travel, especially in prioritized environments. From a larger perspective, this

paper raises the challenge of finding effective heuristics for the MultiPartition problem on general graphs.

References

1. N. Agmon, S. Kraus, and G. A. Kaminka. Multi-robot perimeter patrol in adversarial settings. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2008.
2. M. Ahmadi and P. Stone. A multi-robot system for continuous area sweeping tasks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2006.
3. A. Almeida, G. Ramalho, H. Santana, P. Tedesco, T. Menezes, V. Corruble, and Y. Chevaleyr. Recent advances on multi-agent patrolling. *Lecture Notes in Computer Science*, 3171:474–483, 2004.
4. N. Basilico, N. Gatti, and F. Amigoni. Leader-follower strategies for robotic patrolling in environments with arbitrary topologies. In *AAMAS*, pages 57–64, 2009.
5. G. Chartrand and F. Harary. Planar permutation graphs. *Annales de l'institut Henri Poincare' (B) Probabilite's et Statistiques*, 3(4):433–438, 1967.
6. Y. Chevaleyre. Theoretical analysis of the multi-agent patrolling problem. In *Proceedings of Agent Intelligent Technologies (IAT-04)*, 2004.
7. A. de Mier and M. Noy. On the maximum number of cycles in outerplanar and series-parallel graphs. *Electronic Notes in Discrete Mathematics*, 31:489–493, 2009.
8. Y. Elmaliach, N. Agmon, and G. A. Kaminka. Multi-robot area patrol under frequency constraints. *Annals of Math and Artificial Intelligence journal (AMAI)*, 57(3—4):293—320, 2009.
9. Y. Elmaliach, A. Shiloni, and G.A. Kaminka. A realistic model of frequency-based multi-robot fence patrolling. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-08)*, pages 63–70, 2008.
10. A. Machado, G. Ramalho, J.D. Zucker, and A. Drogoul. Multi-agent patrolling: An empirical analysis of alternative architectures. In *Proceedings of the Third International Workshop on Multi-Agent-Based Simulation (MABS02)*, pages 155–170, 2003.
11. J.S. Marier, C. Besse, and B. Chaib-draa. Solving the continuous time multiagent patrol problem. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 941–946, 2010.

Robot Navigation with Weak Sensors

Noa Agmon[1] and Yehuda Elmaliah[2]

[1] The University of Texas at Austin, USA agmon@cs.utexas.edu
[2] College of Management Academic Studies, Israel elmaliahy@colman.ac.il

Abstract. The problem of robot navigation is a fundamental problem for every mobile robotic system: How to make a robot travel from point A to point B on a given map with maximal efficiency. The problem is easily solvable if the robot has means to determine its current location with respect to the world. However, when this is not the case, for example for robots with weak sensing capabilities, this problem becomes non trivial and even impossible to solve. In this paper we consider the problem of robot navigation with weak sensors, for example sonar. If a robot travels without sensing its destination, it will not necessarily reach its target, but arrive within some proximity to it. We model this error in movement, and along with an evaluation of the environment produce a stochastic graphical representation of the possible ways the robot can travel in the given area. Based on this graph, we determine the *optimal navigation path* for the robot, i.e., a path that maximizes the probability that the robot will reach its destination.

1 Introduction

The problem of robot navigation is a fundamental problem for every mobile robot: How to make a robot travel from point A to point B on a given map with maximal efficiency. Solving the problem of robot navigation can be trivial if the robot has means to determine its position in the world at any time by using, for example, reliable sensors. However, in some cases localization means are nonexistent (for example the use of a GPS in indoor environments) or costly (for example the use of laser sensors). In these cases, the problem of robot navigation becomes far more complicated, even when a map is given, and in some cases impossible. The main objective of this paper is to determine a quantitative measure for determining the possibility of navigating in indoor environments given a map for a robot without perfect localization, and to find a navigation path that maximizes the chances of arriving at the destination point safely.

The growing use of robots in indoor domains, for instance vacuum cleaning and robotic toys, requires the use of inexpensive components in the robots, to make them financially accessible for mass production and retail. In addition, robots might be required to have low power consumption in other robotic applications such as military use or search and rescue. Specifically, the robots should be able to perform their tasks with low-quality sensors, for example sonar/IR sensors. Consequently, algorithms for such robots are generally straightforward and

do not require the robots to exercise navigation skills. By developing navigation skills for such robots, the possible use of the robots could enhance significantly, potentially resulting in more efficient algorithms for general tasks such as area sweeping.

For a robot that has poor sensing capabilities, moving from a point p to point q that is not under its sensorial range, will usually be done with movement error, i.e., the robot will not necessarily arrive at q, but will reach some point that could be anywhere from a close proximity to q to a completely distant location. Our goal is, therefore, to determine a path for a robot from a given point A on the map to a destination point B such that the probability that we can assure it will arrive at B is maximized.

We describe a probabilistic movement error model of the robot that captures its actual movement model, along with its errors. We then use this model to find the optimal direct destination point for each pair of points (p, q) in the map, that maximizes the probability of arriving at q (thus minimizing the probability that the robot will get lost along the way from the origin to the destination point). After determining the optimal direct link from a point to its destination, we can determine a path from the origin source point A to the destination point B that maximizes the probability of safe arrival of the robot, i.e., solve the navigation problem optimally.

The solution to the problem is, therefore, threefold: First, we model the environment as a graph based on the movement error model of the robot (this is done for the environment once, regardless of the robotic model). Second, we determine the optimal destination point for each edge in the graph, based on the error model of the robots' movement (this is done per robot, regardless of the source and destination nodes in the graph). Last, we find a path from a source node s to a destination node d in the graph that maximizes the probability that the robot will arrive at its destination.

This paper reports the theoretical contribution of our work. Initial results using real robots seem promising, and we are currently pursuing massive empirical evaluation of the work in both simulation and in real robots.

2 Related Work

The general subject of robot navigation, due to its immense impact on almost every robotic missions, has received considerable attention in the literature. The main problem that arises in navigation from a source point to a destination point is the question of localization, i.e., where is the robot?

There are various approaches for determining the paths a robot should take when navigating to a destination, where the approaches also differ in the localization aspect, i.e., how to know where the robot is located along the path. In outdoor environments the concerns for planning a navigation are different, as the localization aspect is usually more trivial, using a GPS. We therefore concentrate on navigation in indoor environments, where the use of a GPS is usually impossible.

The most common way used in robotic systems for navigation are Simultaneous localization and Mapping (SLAM) techniques (e.g. [3, 10]), in which a robot simultaneously construct the map of the environment, and localize itself in the map according to landmarks it can relate to. Specifically, in indoor navigation usually laser based or vision based SLAM methods are used (e.g. [4, 10]). In these methods, the robots can identify their location using laser sensors or cameras. However, both methods are are expensive and have high power consumption, thus irrelevant for use if designing low-price or low energy consuming robots. Moreover, the main objective in mapping methods is to efficiently cover the entire area while merging the constructed maps. Our goal is to efficiently navigate, *given a map*, hence we are not interested in neither exploration nor unified map construction.

Other methods for indoor navigation include the use of cameras. Recent work by Chrysanthakopoulos and Shani [1] describe an approach for indoor navigation using a camera. In their work, they describe a generalized appearance-based localization approach that uses a combination of several techniques including POMDPS, hierarchical state representation an more to be used by a navigating robot. In their work, the robot goes through a learning phase after which it is possible for the robot to use their methods for navigation. In our work, however, we assume the map is given, however the robot does not have to go through any training phase. In addition, we can provide guarantees as for the probability of arriving at the destination and plan alternative paths based on the price the system is willing to pay (in time vs. probability of safe arrival).

O'Kane and S. M. LaValle [7] examined the problem robot localization without sensors. They prove that a robot equipped with only a compass, a contact sensor an a map of the environment can successfully localize itself in it. Although their world assumption are similar to our work, they do not consider the problem of navigation, and moreover, they do not assume a stochastic model of the robots movement. Okane [6] also considered the problem of robot localization using odometry, by alternating rotations and forward translations in the known space. His model could potentially account for uncertainty in the robot's world model, however his evaluation is not analytic with respect to the probability of movement, but he refers to the problem as a discrete time planning problem.

Erickson *et al.* [2] offer a solution to the problem of localization of a blind robot in a known environment (given a map), where they distinguish between active and passive localization. In active localization the robot's mission is to localize itself, where in passive localization the robot should maintain a probability distribution over a set of possible positions in the environment while performing some other task. In both cases, the problem is handled as a set of possible states the robot might be in, with a probability distribution over them. Their approach differs from our solution, as they offer a continuous solution (updates of state), whereas we suggest to create a-priori a path that mill minimize the probability that the robot will lose its localization.

3 The environment and error models

In the following section, we describe the modeling of the error in the robot's movement and the representation of the environment. These models are used as base for computing the optimal navigation path, i.e., a path that maximizes the probability of guaranteed arrival at the destination point. A path is composed of a set of edges, i.e., pairs of points (p, q) in the plane. We call the point q an *immediate* destination point from p. Generally, if a robot cannot view its immediate destination point due to poor sensorial capabilities, it might not reach that specific point but some point around it. We therefore base our path on moving from a source point to an immediate destination point that lies along a wall. In this case, we are able to eliminate some uncertainties in the movement, specifically, the possibility that the robot will accidently stop before of after the destination point, and relate only to the probability of reaching points along the wall to the left or to the right of the destination point. We assume that if a robot travels from p and does not arrive along the wall it intended to arrive along, it is *lost*. Thus we want to minimize the probability that this will happen. We assume that a robot, even with poor sensing capabilities, is able to walk along a wall without wandering around and losing its way.

3.1 The robotic error movement model

In order to model the actual point in which the robot arrived with respect to the target point towards which it intended to travel, we use *triangle error model*, described in this section, which defines how far from either side of the destination point the robot will arrive.

We assume the robot is guaranteed to arrive within some spectrum surrounding its actual trajectory (see Figure 1). Assume robot R travels from a point p towards a point q which is on a wall. It does not see q from p, hence it will either arrive exactly at q or along the wall from either of its sides. The possible range in which R will arrive is defined as the *error angle*, denoted by α, divided into α_l and α_r, which are the angles to the left and right of the course towards q (respectively) within which it is guaranteed to arrive. We relate to these two angles separately, as it is very common for a robot to diverge differently towards each direction (depending, for example, on its actuators or motors).

The probabilites of arriving within α_l and α_r are not necessarily equal (realistically, they rarely are). Therefore we associate a probability of arrival within α_l and α_r, denoted by P_l and P_r (where $P_l + P_r = 1$). The triangle created by p, q, α_l, α_r and the wall q resides on, is referred to as the *error triangle* of the robot. Note that Thrun et al. [9, 8] describe various models for estimating the actual final position of the robot given a map. Generally, the motion model of the robots create a banana-like possible destinations, where the center of the banana is the most probable actual destination points. When we limit our destination points to points along a wall, we get the triangular shaped error range, rather than a banana shape.

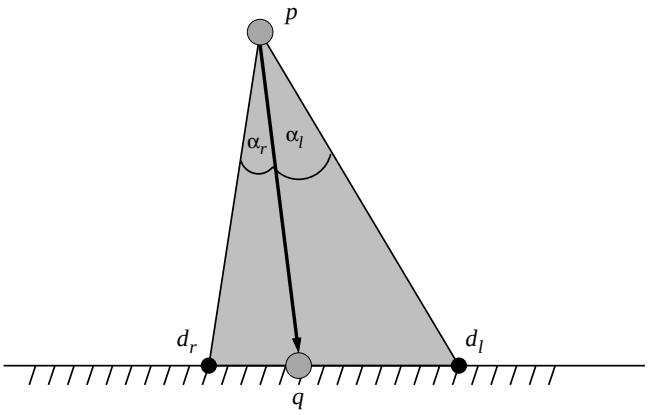


Fig. 1. *An illustration of the triangle error model - the robot is going from p to q, yet could arrive between d_l and d_r along the wall.*

3.2 Initial environment representation

The input to our procedure is a map, from which we extract all obstacles and represent them as *simple polygons* $P = \{P_1, P_2, \ldots, P_M\}$, i.e., each polygon P_i corresponds to an obstacle in the given area (where the area inside the polygon is unreachable by the robot). Every polygon includes a set of sorted vertices $P_i = \{v_1^i, v_2^i, \ldots, v_{q_i}^i\}$. The graph $G' = (V', E')$ is initialized as follows: $\forall v_j^i \in P_i$,, add v_j^i to V' ; $(v, u), (v, u) \in E'$ if $v = v_j^i, u = v_{(j+1)mod(q_i+1)}^i$ or the segment $\overline{vu}$ does not intersect any polygon $P_i \in P$ *and* is not internal to some polygon $P_i \in P$. Note that the graph G' is an unweighted graph. In the next section we describe how G' is modified to become a weighted directed graph with associated cost and probability of traveling along the edges of the graph.

4 Determining immediate optimal destination point

After creating the initial graph representation of the environment (the unweighted directed graph G'), we wish to determine the optimal way to travel from a point p to an immediate neighbor q in the graph. As stated previously, if the robot has perfect localization means, allowing it to travel from p to q without any errors, then it will set its course directly to q. However, in most cases it is impossible to guarantee arrival at the destination point with 100% accuracy. Thus we modify the immediate destination point to some point q^o that will maximize the probability that the robot will arrive at q. Our goal is to find q^o for each edge in the graph.

An additional challenge rises from the fact that once the robot arrives at a wall, it does not necessarily know whether q is to its left or to its right. Therefore, in order to guarantee that the robot will be oriented with respect to

q, we examine two different actual destination points: one along the wall to the left of q (denoted by q_l^o), and one along the wall to the right of q (denoted by q_r^o). These destination points should be chosen such that the probability of arriving along the wall, if setting its course towards this point, is maximized.

The cost of traveling from p to q_l^o or q_r^o is set to the length of the segment $\overline{pq_l^o}$ and $\overline{pq_r^o}$, respectively. The probability of arrival at q_l^o or q_r^o is the weighted average of the base of the error triangle (set by pq_l^o and pq_r^o, respectively) and the length of the wall these points lie on. Therefore, in order to determine the optimal destination points q_l^o and q_r^o, we must first calculate the length of the wall these point lie on, that is visible from the point p. We call this area the *physical triangle*. The left base point of the physical triangle to the left of q is denoted by q_l and the right base point of the physical triangle to the right of q is denoted by q_r. We refer to the segments $\overline{qq_l}$ and $\overline{qq_r}$ as the *left* and *right wings* (respectively).

If the error triangle is smaller than the physical triangle, then q_r^o (q_l^o) is simply the one that places the error triangle's leftmost point (or rightmost) at q (see Figure 2(a)). If the physical triangle is smaller than the the error triangle, then the chosen destination point is the one that maximizes the congruence of the triangles with respect to the probability of arrival at the respective wing (see illustration in Figure 2(b) and (c)).

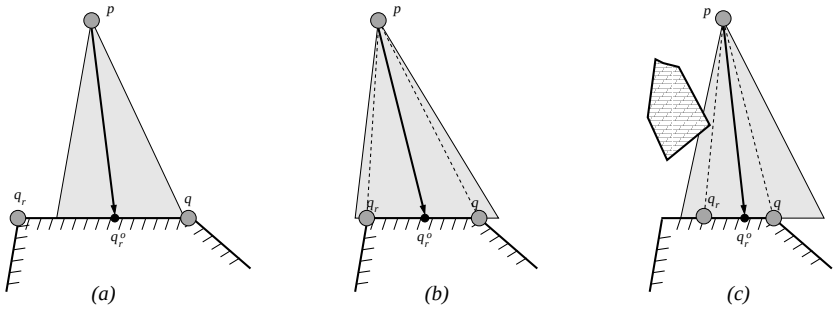


Fig. 2. *An illustration of the change in the optimal destination point when going from p to q - when the physical triangle is larger than the error triangle (a), or when it is smaller (b)*

Algorithm InitializeWorld creates the graphical model corresponding to the environment, i.e., given the polynomials that exist in the target area, it does the following:

1. Creates the graph $G = (V, E)$ including all vertices from the polygon V, where $(u, v) \in E$ if and only if there is a straight line between u and v that is not met by any other polygon in the area, i.e., if there is a straight line of sight between u and v.

2. Determines the left and right wings - the physical triangle - of each vertex $v \in V$ such that $(u, v) \in E$.
3. Determines the best *actual* destination point for each edge, update this point in the graph (by adding a new vertex) and adjust the new graph accordingly (by adding new edges and removing old ones).
4. Stores the cost of traveling along each edge and the probability of actually arriving at the end vertex.

Algorithm 1 Algorithm InitializeWorld

Find all edges E' in the graph (sets of points p, q in the plane that can be reached directly from one another)
for every edge $(p, q) \in E'$ **do**
  Compute the left and right wings as follows.
  Set $q_l \leftarrow$ left neighbor of q in its polygon.
  Set $q_r \leftarrow$ right neighbor of q in its polygon.
  for all $\{q' | (p, q') \in E'\}$ **do**
    if $q' \in \triangle p, q, q_l$ **then**
      $q_l \leftarrow$ intersection point of the line $\overline{pq'}$ and $\overline{qq_l}$
    end if
    if $q' \in \triangle p, q, q_r$ **then**
      $q_r \leftarrow$ intersection point of the line $\overline{pq'}$ and $\overline{qq_r}$
    end if
  end for
  Compute left optimal point $(q_l^o, P^l) \leftarrow \mathsf{FindOptDestP}(p, q, q_l)$
  Compute right optimal point $(q_r^o, P^r) \leftarrow \mathsf{FindOptDestP}(p, q, q_r)$
  Update G with new vertices and edges:
  if $q_l^o \neq q$ **then**
    Add (p, q_l^o) to E with $P_{(p,q_l^o)} = P^l$
    Remove (q, u_l) from E
    Add (q, q_l^o) with cost $\mathrm{dst}(q, q_l^o)$ and $P_(q, q_l^o) = 1$
    Add (q_l^o, u_l) with cost $\mathrm{dst}(q_l^o, u_l)$ and and $P_(q_l^o, u_l) = 1$
  end if
  if $q_r^o \neq q$ **then**
    Add (p, q_r^o) to E with $P_{(p,q_r^o)} = P^r$
    Remove (q, u_r) from E
    Add (q, q_r^o) with cost $\mathrm{dst}(q, q_r^o)$ and $P_(q, q_r^o) = 1$
    Add (q_r^o, u_r) with cost $\mathrm{dst}(q_r^o, u_r)$ and $P_{(q_r^o, u_r)} = 1$
  end if
  if $q_l^o \neq q$ AND $q_r^o \neq q$ **then**
    Remove (p, q) from E
  end if
end for

Recall that the head angle of the error triangle is denoted by α, where α_l is the error angle to the left of the destination point with probability P_l and α_r and P_r to its right. We describe the calculation of the probability of arrival and the

optimal destination point to the right of q, i.e., P_r and q_r^o, and the calculation to the left of q is symmetric. Let q_r denote the rightmost point in the physical triangle, denote the head angle of the physical triangle by β, the angle $\angle_{q_r^o pq}$ by β_l and $\angle_{q_r^o pq_r}$ by β_r (see Figure 3).

Denote the left point of the error triangle (exceeding q) by l, and the right point of the error triangle (exceeding q_r) by r. The expected probability of arriving at the line $\overline{qq_r}$ is the area of the triangle $pq_r^o q$ divided by the area of triangle $pq_r^o l$ multiplied by the probability of arriving at that triangle (P_l) plus the area of the triangle $pq_r^o q_r$ divided by the area of triangle $pq_r^o r$ multiplied by P_r. Formally,

$$P(\textbf{arriving at } \overline{qq_r}) = P(\overline{qq_r}) = P_l \times \frac{\mathrm{dst}(q_r^o, q)}{\mathrm{dst}(q_r^o, l)} + P_r \times \frac{\mathrm{dst}(q_r^o, q_r)}{\mathrm{dst}(q_r^o, r)}$$

We represent all the unknown variables as a function of $\epsilon = \angle_{plq_r}$. The following can be calculated from only $\mathrm{dst}(p,q), \mathrm{dst}(p,q_r)$ and $\mathrm{dst}(q_r,p)$ using the sine theorem or the law of cosines.

$$\gamma = \angle_{pqq_r} = \cos^{-1}\left(\frac{\mathrm{dst}(p,q)^2 + \mathrm{dst}(q,q_r)^2 - \mathrm{dst}(p,q_r)^2}{2\mathrm{dst}(p,q)\mathrm{dst}(q,q_r)}\right)$$
$$\theta = \angle_{pq_rq} = \sin^{-1}\left(\frac{\mathrm{dst}(p,q)\sin(\gamma)}{\mathrm{dst}(p,q_r)}\right)$$

We can represent all the unknown variables as function of ϵ, thus we get:

$$P(\overline{qq_r}) = f(\epsilon) =$$
$$P_l \times \frac{\sin(\alpha_l - \gamma + \epsilon)\sin(\epsilon)}{\sin(\alpha_l - \gamma + \epsilon)\sin(\epsilon) + \sin(\gamma - \epsilon)\sin(\alpha_l + \epsilon)} +$$
$$P_r \times \frac{\sin(\beta - \alpha_l - \epsilon + \gamma)\sin(\alpha + \epsilon)}{\sin(\beta + \gamma - \alpha_l - \epsilon)\sin(\alpha + \epsilon) + \sin(a_l + \epsilon)\sin(\alpha - \beta + \epsilon - \gamma)} \quad (1)$$

If we derive $f(\epsilon)$, we can determine the value of ϵ yielding maximal $f(\epsilon)$, hence maximal $P(\overline{qq_r})$. From that we can determine the optimal destination point q_r^o as follows, where $\beta_l = \alpha_l - \gamma + \epsilon$ and $\delta = \pi - (\alpha_l + \epsilon)$.

$$\mathrm{dst}(q, q_r^o) = \frac{\sin(\beta_l)\mathrm{dst}(p,q)}{\sin(\delta)} \quad (2)$$

The time complexity of the procedures is polynomial in the map size. Specifically, if the number of points in the original map is N (nodes of the polygons describing the obstacles in the environment), then the number of edges in G' is at most N^2. Creating the physical triangles (left and right wings) require at most N^3 operations. Last, each edge in G' is replaced by two edges in G, therefore the number of edges in G is $2N^2$, and the number of vertices is $3N$ (for each vertex, we add two new vertices - q_l^o and q_r^o). Note that this algorithm can be computed a priori, and not during run time.

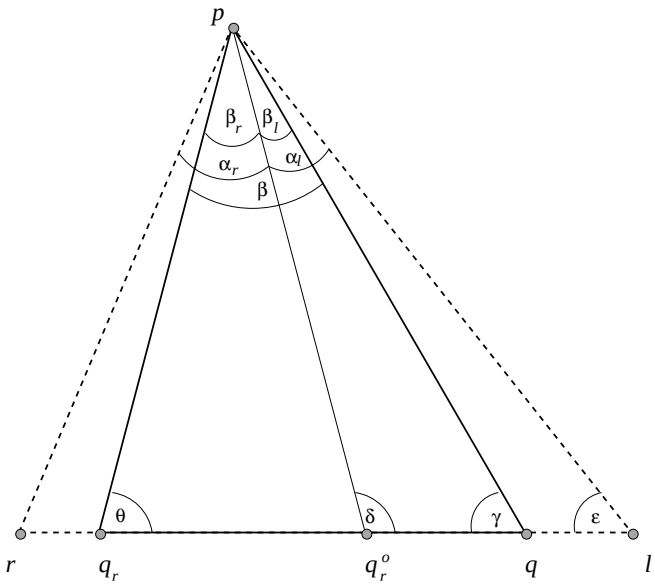


Fig. 3. *Illustration of computing q_r^o and P_r*

Algorithm 2 Algorithm FindOptDestP(p, q, q_w)

$a \leftarrow \frac{\text{dst}(p,q)\sin(\alpha)}{\sin(\theta)}$
if $a \leq \text{dst}(q, q_w)$ (Physical triangle is larger than ET) **then**
    $\text{dst}(q, q_w^o) \leftarrow \frac{\text{dst}(p,q)\sin(\alpha_l)}{\sin(\gamma+\alpha_l)}$
    $P^w \leftarrow 1$
else
    $\epsilon^* \leftarrow \epsilon$ that maximizes equation 1 (maximal point between 0 and π)
    $P^w \leftarrow$ Equation 1, given ϵ^*
    $\text{dst}(q, q_w^o) \leftarrow$ Equation 2, given ϵ^*
end if

4.1 Probabilistic robot navigation - determining the optimal path

The main goal of the navigation problem is to determine the optimal *path* for the robot from its source to its destination point. In the previous section we have determined the immediate destination point that will maximize the probability of arriving at this immediate destination (point p to point q) for every pair of visible points in the map, and creates a graph G with the costs of traveling along edges and the probability that the robot will successfully travel along the edge. Given this graph with the probabilities of arrival on each edge, we can determine the probability of arriving at the global destination.

Note that the probability of reaching a destination point is the multiplication of the probability associated with each of the edges the robot travels through. Therefore, using a shortest path on the probabilities (summing up the probabilities) as the weight of each edge *will not* bring us to the desired actual probability of arrival, and we need to use an algorithm for maximizing the multiplicity of the probabilities on all traveled edges.

We therefore use a method commonly used in QOS of networks (e.g. [5]) for determining a path that minimizes the probability of packet loss along the network. Recall that the probability of arriving at the immediate destination point when traveling along an edge $e \in E$ is P_e. In this method, we assign a weight to each $e \in E$, $w(e)$, which is set to the absolute value of the logarithm of P_e, i.e., $w(e) = |\log(P_e)|$. The advantage of using this method is twofold. First, since $0 \leq P_e \leq 1$, then as P_e is higher the weight of the edge ($|\log(P_e)|$) is lower (encouraging the use of edges with higher probability of arrival). Second, if we run Dijkstra's shortest path algorithm on the logarithms of $P_e, \forall e \in E$, then we get a path $(e_1, \ldots, e_k)$ minimizing the values $\log(P_{e_1}) + \log(P_{e_2}) + \ldots + \log(P_{e_k})$, hence (by logarithmic characteristics) minimizes $\log(P_{e_1} \times P_{e_2} \cdots \times P_{e_k})$, which maximizes $P_{e_1} \times P_{e_2} \cdots \times P_{e_k}$, i.e., maximizes the probability of arrival at the destination point. The formal algorithm is described by Algorithm OptimalPath (Algorithm 3) with s as the source node in G and d the destination point.

Algorithm 3 Algorithm OptimalPath(G, s, d)

1: Compute $W(G)$ as follows:
2: **for** Each $e \in E(G)$ **do**
3: Set $w(e) \leftarrow |\log(P_e)|$
4: **end for**
5: Run Dijkstra's shortest path algorithm with s, d on graph G with $W(G)$

4.2 Possible Extensions

Finding the optimal path that maximizes the probability of arrival at the destination point uses only one aspect of the information we gathered: the probability associated with traveling along each edge. It is possible that we are interested

in a taking the risk of traveling along an edge with low probability of arrival, but that is physically shorter, i.e., has lower cost (similar to a risk seeking strategy). In this case, we can omit the probability factor, and run Dijkstra's shortest path algorithm on our graph with only the cost of travel (yet still walking along edges that individually maximize the probability of reaching successfully the immediate destination point).

However, a more risk neutral approach can also be adopted. In this case, we can eliminate edges with probability of arrival lower than some value P_{limit} we are willing to accept. Another possibility is to calculate the *expected cost* of traveling along an edge, where we multiply the probability of *not* arriving at the endpoint of the edge and the cost of the edge (yielding an expected cost complying to lower is better). By finding the shortest path algorithm in this case, we combine both cost and probability of arrival, however we do not account for probabilities of not reaching the endpoint of the edge that is carried from one edge to another.

We plan on evaluating the performance of these possible extensions in our empirical work.

5 Conclusions and Future Work

In this paper we set a theoretical basis for determining an optimal navigation path for a robot with (possibly) weak sensors. We describe a polynomial time algorithm for determining a path from a source point A to a destination point B, given a map, such that the probability that the robot will arrive at B is maximized.

As ongoing work, we are pursuing massive empirical evaluation of the theoretical results in both simulation and real robots. There are various possible direction for future work. First, we would like to generalize our solution to stochastic and/or dynamic environments. Specifically, it could be possible that the map we base our path upon is outdated, but the probabilities that a blockage occurs in certain places is known. Moreover, in office environments there are stationary objects (walls) that are not likely to move, whereas there are objects that tend to be moved around (tables, closets). Therefore accounting for the possibility of changes in the environment is important. Another possible extension of the work includes sensor-dependent navigation, i.e., given a (not necessarily weak, but not perfect) sensor model of a robot, find the optimal navigation path. In addition, there is growing interest in removing the dependency on GPS in outdoor navigation, thus applying these methods to outdoor navigation could be interesting as well.

References

1. G. Chrysanthakopoulos and G. Shani. Augmenting appearance based localization and navigation using belief update. In *AAMAS*, 2010.

2. L. H. Erickson, J. Knuth, J. M. O'Kane, and S. M. LaValle. Probabilistic localization with a blind robot. In *Proceedings IEEE International Conference on Robotics and Automation*, 2008.
3. J. J. Leonard and H. F. Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. In *Proceedings of IEEE Int. Workshop on Intelligent Robots and Systems*, pages 1442–1447, 1991.
4. J. V. Miro, W. Zhou, and G. Dissanayake. Towards vision based navigation in large indoor environments. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006.
5. H. De Neve and P. Van Miegham. Tamcra: A tunable accuracy multiple constraints routing algorithm. *Computer Communications*, 23:667–679, 2000.
6. J. M. O'Kane. Global localization using odometry. In *Proceedings IEEE International Conference on Robotics and Automation*, 2006.
7. J. M. O'Kane and S. M. LaValle. Localization with limited sensing. *IEEE Transactions on Robotics*, 23(4):704–716, August 2007.
8. S. Thrun, W. Burgard, and D. Fox. A probabilistic approach to concurrent mapping and localization for mobile robots. *Autonomous Robots*, 5(3-4), 1998.
9. S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.
10. S. Thrun and M. Montemerlo. The GraphSLAM algorithm with applications to large-scale mapping of urban structures. *International Journal on Robotics Research*, 25(5/6):403–430, 2005.

Fast Frontier Detection for Robot Exploration

Matan Keidar, Eran Sadeh-Or, and Gal A. Kaminka

MAVERICK Group, Department of Computer Science, Bar-Ilan University

Abstract. Frontier-based exploration is the most common approach to exploration, a fundamental problem in robotics. In frontier-based exploration, robots explore by repeatedly computing (and moving towards) *frontiers*, the segments which separate the known regions from those unknown. However, most frontier detection algorithms process the entire map data. This can be a time consuming process which slows down the exploration. In this paper, we present two novel frontier detection algorithms: *WFD*, a graph search based algorithm and *FFD*, which is based on processing only the new laser readings data. In contrast to state-of-the-art methods, both algorithms do not process the entire map data. We implemented both algorithms and showed that both are faster than a state-of-the-art frontier detector implementation (by several orders of magnitude).

1 Introduction

The problem of exploring an unknown territory is a fundamental problem in robotics. The goal of exploration is to gain as much new information as possible of the environment within bounded time. Applications of efficient exploration include search and rescue [13], planetary exploration [1] and military uses [11].

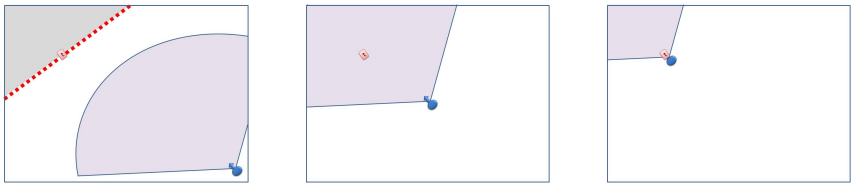
The most common approach to exploration is based on *frontiers*. A frontier is a segment that separates known (explored) regions from unknown regions. By moving towards frontiers, robots can focus their motion on discovery of new regions. Yamauchi [22, 23] was the first to show a frontier-based exploration strategy. His work preceeded many others (e.g, [6, 15, 16, 5]).

Most frontier detection methods are based on edge detection and region extraction techniques from computer vision. Thus, to detect frontiers, they process the entire map data with every call to the algorithm. State of the art frontier detection algorithms can take a few seconds to run, even on powerful computers. If a large region is explored, the robot actually has to wait in its spot until the frontier detection algorithm terminates. Therefore, many exploration implementations call the frontier detection algorithm only when the robot arrives at its destination.

Thus, a real-time frontier detection can shorten the exploration time. We present two examples:

A Single-Robot Example. A common situation of single-robot exploration can be seen in Figure 1: Figure 1(a) shows a robot exploring its environment and has

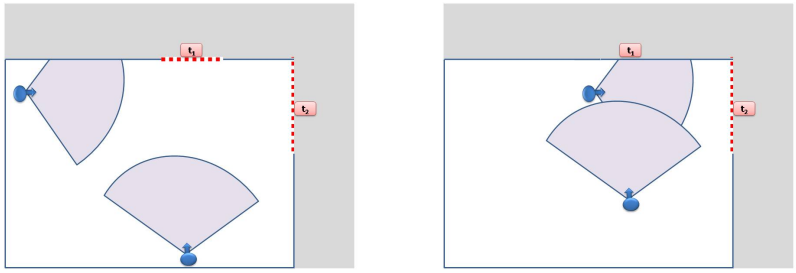
just decided navigating to a target. Figure 1(b) shows that the target has been covered by the robot's sensors and it does not have any reason to keep moving. Figure 1(c) shows that because of the lack of real-time frontier computation, the robot moved to its target, unnecessarily.



(a) (b) (c)

Fig. 1. Single-robot example: 1(a) the robot is heading towards the target on the frontier. 1(b) the target is being covered by the robot's sensors. 1(c) the robot has reached the frontier.

A Multi-Robot Example. A common situation of multi-robot exploration can be seen in Figure 2: Two robots, R_1 and R_2 which are located on bottom and top, respectively, are exploring the environment. Their start position is as in Figure 2(a). Figure 2(b) describes the world state after a while when each robot was heading to its target. R_2 has covered with its sensors the frontier of t_1. Therefore, there is no need for R_1 to go to t_1. If R_1 does not calculate frontiers in real-time, it would continue moving towards target t_1.



(a) top robot, R_2 is heading towards right target, t_2 and bottom robot, R_1 is heading towards top target, t_1.

(b) top robot has reached its target.

Fig. 2. Multi-robot example.

In this paper, we introduce two algorithms for fast frontier detection: The first, *WFD* (Wavefront Frontier Detector, Section 4) is an iterative method that performs a graph-search over already-visited map points. The advantage over state of the art methods is that *WFD* does not have to scan the entire map, only the regions that have already been visited by the robot. The second, *FFD* (Fast Frontier Detector, Section 5) is a novel approach for frontier detection. *FFD* processes only the new received laser readings. It can be much faster, but requires interfacing with the mapping algorithms and data structures, so that frontiers are maintained even when they are no longer within sensor range.

In Section 6, we compare these algorithms to a state of the art edge-detection method for frontier detection. The results shows that *WFD* and *FFD* are faster by at least two orders of magnitude than previous methods. Moreover *FFD* is faster than *WFD* by an order of magnitude.

2 Related Work

An outline of the exploration process can be described as follows: while there is an unknown territory, allocate each robot a target to explore and coordinate team members in order to minimize overlaps. In frontier-based exploration, target are drawn from existing *frontiers*, segments that separate known and unknown regions (see Section 3 for definitions).

There are two aspects that are often tackled in existing literature on exploration: deciding on next target to be explored and coordinating team members in order to minimize overlaps. The latter is not related to this paper and so, we focus on the former.

To the best of our knowledge, all of the following works utilize a standard edge-detection method for computing the frontiers. They therefore recompute target locations whenever one robot has reached its target location or whenever a certain distance have been traveled by the robots or after a timeout event.

Yamauchi [22, 23] developed the first frontier-based exploration methods. The robots explore an unknown environment and exchange information with each other when they get new sensor readings. As a result, the robots build a common map (occupancy grid) in a distributed fashion. The map is continuously updated until no new regions are found. In his work, each robot heads to the *centroid*, the center of mass of the closest *frontier*. All robots navigate to their target independently while they share a common map. Frontier detection is performed only when the robot reaches its target.

Burgard et al. [5, 6] focus their investigation on probabilistic approach for coordinating a team of robots. Their method considers the trade-off between the costs of reaching a target and the utility of reaching that target. Whenever a target point is assigned to a specific team member, the utility of the unexplored area visible from this target position is reduced for the other team members. In their work, frontier detection is carried out only when a new target is to be allocated to a robot.

Wurm et al. [21] proposed to coordinate the team members by dividing the map into segments corresponding to environmental features. Afterwards, exploration targets are generated within those segments. The result is that in any given time, each robot explores its own segment. Wurm [20] suggests to call frontier detection every time-step of the coordination algorithm. Moreover, he claims that updating frontiers frequently is important in a multi-robot teams since the map is updated not only by the robot assigned to a given frontier but also by all of the robots in the team. In the real world the algorithm should be executed every 0.5-1m or every second or whenever a new target is requested.

Stachniss [17] introduced a method to make use of background knowledge about typical structures when distributing the team members over the environment. In his work, Stachniss computes new frontiers when there new target are needed to be allocated. This happens whenever one robot has reached its designated target location or whenever the distance traveled by the robots or the elpased time since last target assignment has exceeded a given threshold.

Berhault et al. [2] proposed a combinatorial auction mechanism where the robots bid on a bunch of targets to navigate. The robots are able to use different bidding strategies. Each robot has to visit all the targets that are included in his winning bid. After combining each robot's sensor readings, the auctioneer omits selected frontier cells as potential targets for the robots. Frontier detection is performed when creating and evaluating bids.

Visser et al. [19] investigated how limited communication range affect multi-robot exploration. They proposed an algorithm which takes into account wireless constraints when selecting frontier targets. Visser [18] suggests recomputing frontiers every 3–4 meters, which on his opinion, has positive effect.

Lau [15] presented a behavioral approach. The authors assume that all team members start from a known location. The team members follow the behavior and spread in the environment while updating a shared map. Frontier detection is called when the robot plan its next direction of movement.

Many other works omit details of their frontier detection timing. For example, Sawhney et al. [16] presented an exploration method which uses a novel visibility per-time metric that can reduce exploration time. Bouraqadi et al. [3] proposed a flocking-based approach for solving the exploration problem, where robots act according to the same set of rules. One of their rules (R5) makes the robot navigate towards the nearest frontier. Ko et al. [14] presented a decision-theoretic approach to the mapping and exploration problem. Their approach uses an adopted version of particle filters to estimate the position in the other robot's partial map.

One previous work [7] mentions frontier detection algorithm that utilizes breadth-first search, similar to one of the algorithms that we present here (WFD). However, it does not provide details of the algorithm and so exact similarities and differences cannot be assessed.

3 Frontier-Based Exploration: Definitions and Terms

In this section we define and explain the terms that are used in the following sections. We assume the robot in question uses an occupancy-grid map representation in the exploration process (Figure 3) within the map:

Unknown Region is a territory that has not been covered yet by the robot's sensors.

Known Region is a territory that has already been covered by the robot's sensors.

Open-Space is a *known region* which does not contain an obstacle.

Occupied-Space is a *known region* which contains an obstacle.

Frontier is the segment that separates known (explored) regions from unknown regions. Frontier is a set of *unknown* points that each have at least one *open-space* neighbor.

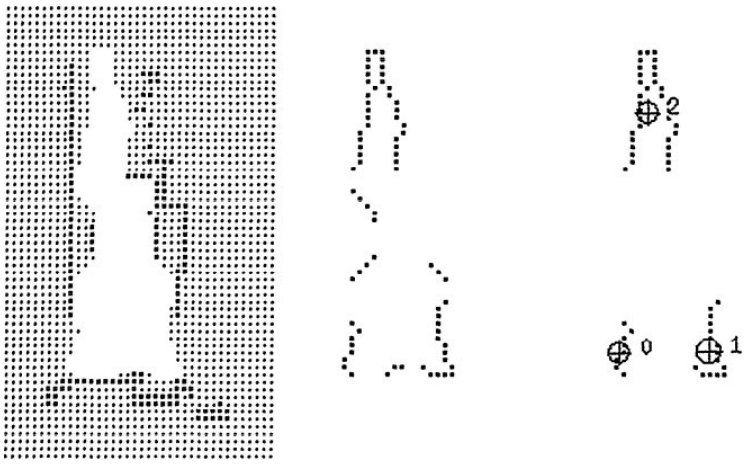


Fig. 3. Image taken from [23]: evidence grid, frontier points, extraction of different frontiers (from left to right).

Existing algorithms for frontier detection rely on edge-detection methods. The algorithms systematically search for frontiers all over the occupancy-grid, i.e., both in known and unknown regions.

4 Wavefront Frontier Detector (WFD)

We present a graph search based approach for frontier detection. The algorithm, *WFD* (Algorithm 1), processes the points on map which have already been scanned by the robot sensors and therefore, does not always process the entire map data in each run, but only the known regions.

WFD is based on Breadth-First Search (BFS). First, the occupancy-grid point that represent the current robot position is enqueued into $queue_m$, a queue data-structure used to determine the search order (Lines 1– 3).

Next, a BFS is performed (Line 4–39) in order to find all frontier points contained in the map. The algorithm keep scanning only points that have not been scanned yet and represent open-space (Line 33). The above scanning policy ensures that only known regions (that have already been covered by the robot's sensors) are actually scanned. The significance of this is that the algorithm does not have to scan the entire occupancy-grid each time.

Because frontier points are adjacent to open space points, all relevant frontier points will be found when the algorithm finishes (Line 39). If a frontier point is found, a new BFS is performed in order to extract its frontier (Lines 14–30). This BFS is searching for frontier points only. Extracting the frontier is ensured because of the connectivity of frontier points.

At the end of the frontier extraction process (Line 30), the extracted frontier data is saved to a set data-structure that stores all frontiers found in the algorithm run.

In order to avoid rescanning the same map point and detecting the same frontier reachable from two frontier points, *WFD* marks map points with four indications:

1. Map-Open-List: points that have already been enqueued by the outermost BFS (Line 34)
2. Map-Close-List: points that have already been dequeued by the outermost BFS (Line 5)
3. Frontier-Open-List: points that have already been enqueued by the frontier extraction BFS (Line 23)
4. Frontier-Close-List: points that have already been dequeued by the frontier extraction BFS (Line 15)

The above marks indicate the status of each map point and determine if there is a need to handle it in a given time.

The key innovation in *WFD* is that it prevents scanning unknown regions, since frontiers never appear there. However, it still searches all known space.

Algorithm 1 WFD

Require: $queue_m$ // queue, used for detecting frontier points from a given map
Require: $queue_f$ // queue, used for extracting a frontier from a given frontier cell
Require: $pose$ // current global position of the robot

```
1: queue_m ← φ
2: ENQUEUE(queue_m, pose)
3: mark pose as "Map-Open-List"

4: while queue_m is not empty do
5:    p ← DEQUEUE(queue_m)

6:    if p is marked as "Map-Close-List" then
7:       continue
8:    end if

9:    if p is a frontier point then
10:      queue_f ← φ
11:      NewFrontier ← φ
12:      ENQUEUE(queue_f, p)
13:      mark p as "Frontier-Open-List"

14:      while queue_f if not empty do
15:         q ← DEQUEUE(queue_f)

16:         if  q is marked as {"Map-Close-List","Frontier-Close-List"} then
17:            continue
18:         end if

19:         if q is a frontier point then
20:            add q to NewFrontier
21:            for all w ∈ adj(q) do
22:               if  w  not  marked  as  {"Frontier-Open-List","Frontier-Close-List",
                  "Map-Close-List"} then
23:                  ENQUEUE(queue_f,w)
24:                  mark w as "Frontier-Open-List"
25:               end if
26:            end for
27:         end if

28:         mark q as "Frontier-Close-List"
29:      end while

30:      save data of NewFrontier
31:   end if

32:   for all v ∈ adj(p) do
33:      if v not marked as {"Map-Open-List","Map-Close-List"} and v has at least
         one "Map-Open-Space" neighbor then
34:         ENQUEUE(queue_m,v)
35:         mark v as "Map-Open-List"
36:      end if
37:   end for

38:   mark p as "Map-Close-List"
39: end while
```

5 Fast Frontiers Detector

Unlike other frontier detection methods (including *WFD*), our proposed algorithm (Algorithm 2) only processes new laser readings which are received in real time. It therefore avoids searching both known and unknown regions. The reason for such an approach lies within the characteristics of new frontiers, as can be seen in Figure 3.

New frontiers are never contained within known (scanned) regions: According to Yamauchi's frontier definition [22, 23], a frontier cell is an unscanned cell which has at least one neighbor which was previously scanned and represents an open space.

Also, new frontiers are never wholly within unknown (unscanned) regions: frontiers represents the boundaries between the known and unknown regions of the environment. Hence, scanning all unknown regions is definitely unnecessary and not time-efficient.

The *FFD* algorithm contains four steps (see Algorithm 2), and can be called with every new scan.

5.1 Sorting

The first step sorts laser readings based on their angle, i.e., based on the polar coordinates with the robot as the origin. Normally, laser-readings are given as a sorted set of polar coordinated points. However, if this is not the case, a sorting is needed to be applied on the received laser readings because next steps of *FFD* relies on an internal order of the received laser readings.

In this case, we assume that a laser reading is a set of Cartesian coordinated points, which consists of the locations of laser hits ($\{(x_0, y_0), \ldots, (x_n, y_n)\}$ where n is the number of readings), sorted by the angle and distance from the robot as the origin. The naive method for converting Cartesian coordinates to polar coordinates uses two CPU time-consuming functions: *atan2* and *sqrt*. Therefore, we use a cross product [8] in order to avoid using the above and still get a result of sorted Cartesian points according to polar coordinates.

Cross Product. Given 3 Cartesian coordinated points:

$$P_0 = (x_0, y_0), P_1 = (x_1, y_1), P_2 = (x_2, y_2)$$

the *cross product* is defined as:

$$(p_1 - p_0) \times (p_2 - p_0) = (x_1 - x_0) \cdot (y_2 - y_0) - (x_2 - x_0) \cdot (y_1 - y_0)$$

If the result is positive, then $\overrightarrow{P_0P_1}$ is clockwise from $\overrightarrow{P_0P_2}$. Else, it is counter-clockwise. If the result is 0, then the two vectors lie on the same line in the plane.

Therefore, by just examining the sign of the cross product, we can determine the order of the Cartesian points according to polar coordinates, without calculating their actual polar coordinate value; only by applying five subtractions

Algorithm 2 FFD

Require: $OldFrontiers$ // data-structure that contains last known frontiers
Require: $pose$ // current global position of the robot
Require: lr // laser readings which were received in current iteration. Each element is a 2-d cartesian point

// polar sort readings according to robot position
$sorted \leftarrow SORT\_POLAR(lr, pose)$

// get the contour from laser readings
$prev \leftarrow POP(sorted)$
$contour \leftarrow \phi$

for all Point $curr \in sorted$ **do**
  $line \leftarrow GET\_LINE(prev, curr)$
  for all Point $p \in line$ **do**
    $contour \leftarrow contour \cup \{p\}$
  end for
end for

// extract new frontiers from contour
$NewFrontiers \leftarrow \phi$ // list of new extracted frontiers
$prev \leftarrow POP(contour)$

if $prev$ is a frontier cell **then** // special case
  create a new frontier in $NewFrontiers$
end if

for all Point $curr \in contour$ **do**
  if $curr$ is not a frontier cell **then**
    $prev \leftarrow curr$
  else if $curr$ and $prev$ are frontier cells **then**
    add $curr$ to last created frontier
    $prev \leftarrow curr$
  else
    create a new frontier in $NewFrontiers$
    add $curr$ to last created frontier
    $prev \leftarrow curr$
  end if
end for

// maintainance of previously detected frontiers
$MAINTAIN\_FRONTIERS(NewFrontiers, OldFrontiers)$

and two multiplications which are far less time-consuming than calling *atan2* and *sqrt*.

5.2 Contour

In this step we use the angle-sorted laser readings. The output of the contour step is a contour which is built from the sorted laser readings. The algorithm computes the line that lies between each two adjacent points from the set. The line is computed by calling the function GET_LINE. In our implementation we use *Bresenham's line algorithm* [4]. Next, all points that are represented by all the lines (including the points from the laser readings set) are merged into a contour (Figure 4).

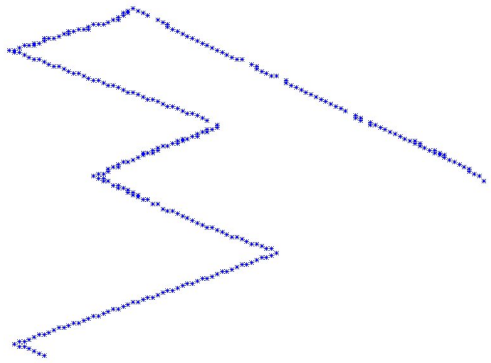


Fig. 4. example of produced contour.

5.3 Detecting New Frontiers

In this step the algorithm extracts new frontiers from the previously calculated contour. There are three cases correspond to each two adjacent points in the contour:

1. **Current scanned point is not a frontier cell:** therefore, it does not contribute any new information about frontiers and can be ignored.
2. **Current and previous scanned points are frontier cells:** therefore, both points belong to the same frontier and current scanned point is added to last detected frontier.
3. **Current point is a frontier cell but the previous is not:** a new starting point of a frontier was detected. Hence, the algorithm creates a new frontier and adds the new starting point to it.

5.4 Maintaining Previously Detected Frontiers

FFD gains its speed by processing the laser readings only, rather than entire regions of the map. However, if the robot navigates towards a specific frontier, other previously detected frontiers are no longer updated because they are not covered by the robot's sensors. In this step, in order to get complete information about the frontiers, the algorithm performs maintenance over previously detected frontiers which are no longer covered in the range of the sensors.

Maintaining FFD In order to keep in memory all available frontiers, *FFD* has to run in the background, in contrast to other approaches that can be executed in a certain time, and only then. However, because of its high speed, in our opinion, keeping *FFD* running in background is preferable over waiting for a few seconds for other frontier detector to finish.

Particle Switching *FFD* requires the previously detected frontiers to be robust against map orientation changes caused by loop-closures of the mapping algorithm. In a *Particle Filter* based SLAM infrastructures, changes in active particles probably occur. Hence, because particles do not share maps, previously detected frontiers by *FFD* cannot be easily maintained.

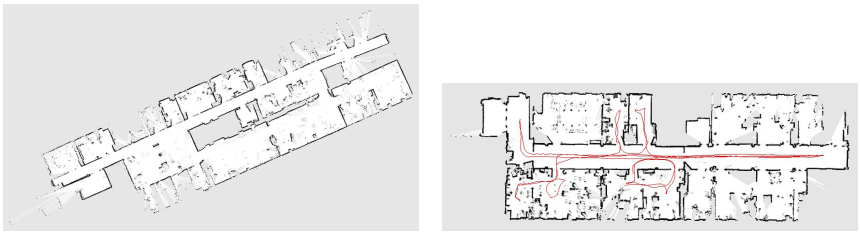
The situation is different in *Extended Kalman-Filter* (EKF) based SLAM infrastructures. These infrastructures have one map that is updated. Hence, data can be stored within a map in EKF SLAM infrastructures because the information about changing map orientation is available (in contrast to particle-based systems in which every particle is independent from the other particles). We find *Kalman-Filter* (EKF) based SLAM implementations best for integrating *FFD*. In Section 7, we suggest a solution to integrate *FFD* into Particle-Filter based SLAM implementations.

6 Experimental Results

We have fully implemented *WFD* and partially implemented *FFD* (all steps except maintenance) and performed testings on data obtained from the Robotics Data Set Repository (Radish) [12]. Figure 5 shows a few of the environments used for the evaluation. *WFD* and *FFD* were compared with a state of the art frontier detection algorithm[1], denoted *SOTA* (state of the art).

To evaluate the algorithms, we integrated them into a single-robot exploration system. The system is based on GMapping, an open-source SLAM implementation [9, 10]. We integrated our code into the *ScanMatcher* component which is contained inside *gsp thread* (Grid SLAM Processor). By the time that a new MapEvent is raised, all frontier detection algorithms are executed according to current world state. Execution times are measured by Linux system-call

[1] We thank Kai M. Wurm and Wolfram Burgard for providing us with their own implementation.



(a) Cartesium Building, University of Bremen

(b) Freiburg, Building 079

Fig. 5. some of the testing environments

getrusage, which measures the CPU-process time. We used a desktop computer containing Intel Q9400 CPU with clock speed of 2.66 GHz and and Random Access Memory (RAM) in size of 4 GB.

We used several environments taken from Radish[2] [12]:

- Edmonton Convention Centre (site of the AAAI 2002 Grand Challenge), marked (A)
- Outdoor dataset recorded at the University of Freiburg, marked (B)
- Freiburg, Building 079, marked (C)
- 3rd Floor of MIT CSAIL, marked (D)
- Cartesium Building, University of Bremen (E)

FFD is called every-time a new laser reading is received. Therefore, in order to compare *FFD* execution time to other algorithms correctly, we accumulate *FFD*'s execution times between calls to other algorithms. In other words, if we call *WFD* in time-stamps t_i and t_{i+1}, then *FFD*'s accumulated execution time is calculated by:

$$\sum_{x=t_i}^{t_{i+1}} ExecutionTime_{FFD}(x)$$

Figure 6 shows the results of the comparison. Each group of bars represent a separate run. For each algorithm, we calculate the average execution time. Y axis measures the calculated execution time in microseconds, on a *logaritmic scale*.

Figure 6 shows that *WFD* is faster than *SOTA* by two orders of magnitude. Furthermore, *FFD* is faster than *WFD* by an order of magnitude, which means it is much faster than state-of-the-art frontier detection algorithm. In our opinion, one can boost *FFD*'s execution time by not executing it on every received laser reading. The reason is that the frequency of receiving new laser readings is higher than the speed of processing and updating the map.

[2] Thanks go to Cyrill Stachniss, Giorgio Grisetti and Nick Roy for providing this data.

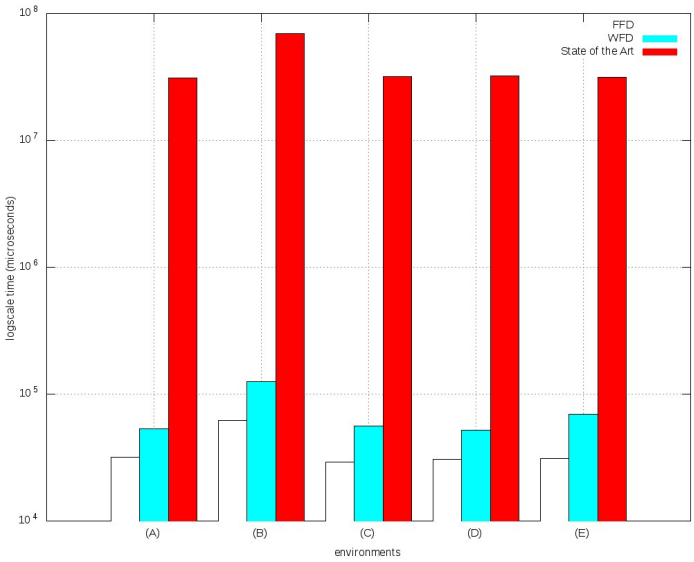


Fig. 6. Comparing WFD and FFD to State of the Art algorithm.

7 Conclusions and Future Work

Frontier-based exploration is the most common approach to solve the exploration problem. State of the art frontier detection methods process the entire map data. The result is a frontier detection which hangs the exploration system for a few seconds.

We introduced two novel faster frontier detectors, *WFD* and *FFD*. The first, a graph based search, processes the map points which have already been scanned by the robot sensors and therefore, does not process unknown regions in each run, in contrast to state of the art frontier detection methods. The second, a laser-based approach for frontier detection, only processes new laser readings which are received in real time. Thus, eliminating also much of the known area search. However, maintaining previous frontiers knowledge requires tight integration with the mapping component, which may not be straight-forward.

In future, we plan to address efficient methods for maintaining frontiers in *FFD*. In addition, in order to integrate *FFD* into particle-based systems, we suggest executing *FFD* on all particles concurrently, which is feasible given its runtime. We intend to test the suggested solution.

References

1. Apostolopoulos, D., Pedersen, L., Shamah, B., Shillcutt, K., Wagner, M., Whittaker, W.: Robotic antarctic meteorite search: Outcomes. In: IEEE International

Conference on Robotics and Automation. pp. 4174–4179 (2001)
2. Berhault, M., Huang, H., Keskinocak, P., Koenig, S., Elmaghraby, W., Griffin, P., Kleywegt, A.: Robot exploration with combinatorial auctions. In: Proceedings of the International Conference on Intelligent Robots and Systems. pp. 1957–1962 (2003)
3. Bouraqadi, N., Doniec, A., de Douai, E.M.: Flocking-Based Multi-Robot Exploration. In: National Conference on Control Architectures of Robots (2009)
4. Bresenham, J.: Algorithm for computer control of a digital plotter. IBM Systems Journal 4(1), 25–30 (2010)
5. Burgard, W., Moors, M., Fox, D., Simmons, R., Thrun, S.: Collaborative multi-robot exploration. In: IEEE International Conference on Robotics and Automation. Vol. 1. pp. 476–481 (2000)
6. Burgard, W., Moors, M., Stachniss, C., Schneider, F.: Coordinated multi-robot exploration. IEEE Transactions on Robotics 21(3), 376–378 (2005)
7. Calisi, D., Farinelli, A., Iocchi, L., Nardi, D.: Multi-objective exploration and search for autonomous rescue robots: Research articles. J. Field Robot. 24, 763–777 (August 2007)
8. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT Press (2001)
9. Grisetti, G., Stachniss, C., Burgard, W.: Improving grid-based SLAM with Rao-Blackwellized particle filters by adaptive proposals and selective resampling. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA). pp. 2443–2448 (2005)
10. Grisetti, G., Stachniss, C., Burgard, W.: Improved techniques for grid mapping with Rao-Blackwellized particle filters. IEEE Transactions on Robotics 23, 34–46 (2007)
11. Hougen, D.F., Benjaafar, S., Bonney, J., Budenske, J., Dvorak, M., Gini, M.L., French, H., Krantz, D.G., Li, P.Y., Malver, F., Nelson, B.J., Papanikolopoulos, N., Rybski, P.E., Stoeter, S., Voyles, R.M., Yesin, K.B.: A miniature robotic system for reconnaissance and surveillance. In: ICRA. pp. 501–507 (2000)
12. Howard, A., Roy, N.: The robotics data set repository (RADISH) (2003), http://radish.sourceforge.net/
13. Kitano, H., Tadokoro, S., Noda, I., Matsubara, H., Takahashi, T., Shinjou, A., Shimada, S.: Robocup rescue: Search and rescue in large-scale disasters as a domain for autonomous agents research. In: IEEE International Conference on Systems, Man, and Cybernetics. pp. 739–746. IEEE Computer Society (1999)
14. Ko, J., Stewart, B., Fox, D., Konolige, K., Limketkai, B.: A practical, decision-theoretic approach to multi-robot mapping and exploration. In: Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 3232–3238 (2003)
15. Lau, H., NSW, A.: Behavioural approach for multi-robot exploration. In: Australasian Conference on Robotics and Automation (ACRA), Brisbane, December (2003)
16. Sawhney, R., Krishna, K.M., Srinathan, K.: On fast exploration in 2D and 3D terrains with multiple robots. In: Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems. Vol. 1. pp. 73–80 (2009)
17. Stachniss, C.: Exploration and Mapping with Mobile Robots. Ph.D. thesis, University of Freiburg, Department of Computer Science (2006)
18. Visser, A.: personal communication. Email (January 4th, 2011)

19. Visser, A., Slamet, B.A.: Including communication success in the estimation of information gain for multi-robot exploration. In: Proceedings of the 6th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt 2008). pp. 680–687. IEEE Publishing (April 2008)
20. Wurm, K.M.: personal communication. Email (January 20th, 2011)
21. Wurm, K., Stachniss, C., Burgard, W.: Coordinated multi-robot exploration using a segmentation of the environment. In: Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Nice, France (Sep 2008)
22. Yamauchi, B.: A frontier-based approach for autonomous exploration. In: Proceedings of the 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation. pp. 146–151. IEEE Computer Society, Washington, DC, USA (1997)
23. Yamauchi, B.: Frontier-based exploration using multiple robots. In: Proceedings of the Second International Conference on Autonomous Agents. pp. 47–53 (1998)

Lazy auctions for multi-robot collision avoidance and motion control under uncertainty

Jan-P. Calliess[1], Daniel Lyons[2] and Uwe D. Hanebeck[2]

[1] Dept. of Engineering Science, University of Oxford, Parks Road, OX1 3PJ, UK.
jan@robots.ox.ac.uk
[2] Intelligent Sensor-Actuator-Systems Lab, Karlsruhe Institute of Technology,
Kaiserstr. 12, D-76128 Karlsruhe, Germany

Abstract. We present an auction-flavored multi-agent planning mechanism where coordination is to be achieved on the occupation of atomic resources. Introducing virtual obstacles, we show how this approach can be applied to particle-based multi-robot control, offering a decentralized, auction-based alternative to previously established centralized methods. Furthermore, we discuss conditions under which it is possible to prove our method achieves coordination in a finite number of iterations in discrete environments. Finally, we link our method to particle-based open-loop control and illustrate the effectiveness of our new approach by presenting simulations for typical spatially-continuous multi-robot path-planning problems with collision avoidance under uncertainty.

1 Introduction

Due to its practical importance, multi-agent coordination has been subject to ever increasing research efforts over the past decades. One of its subfields, multi-robot coordination focusses on problems that reflect the specific nature of robotic agents and their environment. In contrast to strategic settings, a multi-robot mechanism designer can typically afford to assume obedient agents and hence, does not need to burden herself with ensuring design goals such as incentive compatibility or strategyproofness. On the other hand, this freedom should be much welcomed considering that robots typically interact in a complex and uncertain physical world and typically can choose from a continuum of control signals (actions).

While in principle, slightly abstracted models of such problems could often be solved optimally with centralized optimization techniques (eg cf. [22]) these are known to suffer from substantial architectural and tractability deficiencies, which rule them out for many real-world sized applications. Therefore, distributed coordination mechanisms have been heavily investigated in various areas of application [11, 9].

Auction methods have been identified as a particularly promising methodology that typically requires little communication bandwidth [12, 2]. They are based on the observation that many multi-robot problems can be stated as coordination of resource consumptions.

We consider the following multi-robot path planning problem (**MRPPP**): *A team of robots* $1, ..., A$ *desires to find individual plans* $p^1, ..., p^A$ *that translate to collision-free paths in free-space such that each robot's path leads from its start position to a predefined goal location.*

In our setting, an individual robot's plan could translate to a sequence of locations in free-space. Interpreting locations as non-divisible (*atomic*) resources an agent's plan (path) corresponds to a set of resource claims.

How do we coordinate the planning process such that the joint outcome of plans is *socially optimal*, i.e. that the sum of the robot path costs over all robots is minimized?

With our resource interpretation in mind, we could set up a combinatorial (VCG-)auction in the course of which each robot needs to compute a bid for any possible combination of plans which guarantees social optimality [21]. However, it should come as no surprise that the winner determination problem for combinatorial VCG-mechanisms is known to be NP-hard and difficult to approximate [8] even for a finite number of resources.

Of course, in a continuous planning space, trying to compute the costs (on which the bids would have to be based) seems computationally hopeless, because the number of combinations each robot would have to asses is uncountably infinite. Therefore, the multi-robot auction literature is presently restricted to a finite number of resources and hence, assumes prior spatial discretization.

In general, even for a finite number of resources, variations of such problems are known to be NP-hard. Hence, we cannot hope to find a coordination mechanism that is simultaneously computationally efficient and optimal. Therefore, most threads of works have focussed on developing suboptimal coordination mechanisms whose performance was evaluated experimentally (e.g. [25, 24]) .

In this work, we present two contributions that address problem *MRPPP*.

First, we present a distributed auction mechanism for this problem that can cope with a finite number of resources (locations). Our mechanism is *lazy* in the sense that, instead of asking for bids on all conceivable combinations of plans, all agents plan independently and bidding only takes place for resources that turn out to be overbooked (i.e. which two or more robots plan to use simultaneously). Thereby, their coordinated paths are guaranteed to be collision-free, while at the same time the exponential blow-up resulting from considering all combinations is avoided.

Second, we show how this mechanism can be combined with particle-methods for open-loop control [4] which have been successfully applied to single-agent path planning and control problems in continuous spaces with obstacles. The result of this merger is a distributed multi-robot path planning mechanism that (with adjustably high certainty) generates collision-free paths without prior space-discretization and which can take uncertainty into account (which may be desirable due to sensor noise and model-inaccuracies).

2 General Model and Task Description

Let $\mathcal{A} = \{1, ..., A\}$ denote the index set of our robotic agents we desire to coordinate. Each robot $a \in \mathcal{A}$ needs to find a plan p^a that corresponds to a path starting at its start state $S(a)$ and ending at its destination $D(a)$.

Since our approach is motivated by multi-robot path planning, we interpret a plan as being in a one-to-one relationship with a path in free space. For instance, a plan could be a sequence of control inputs that linearly relates to a trajectory of locations (resources) in an environment. For simplicity of exposition, we will from now on assume that plan p^a *is* a time-indexed sequence $(p^a_t)_{t\in\mathbb{N}}$ where p^a_t corresponds to a decision specifying which resource to consume at time t. However, we will lift this assumption again in Sec. 4 where the plans are indeed control inputs that linearly relate to locations.

Obviously, the agents need to make sure that plans are *legal*, that is they adhere to the laws of the environment. We call the set of all legal plans the *global feasible set* G.

For example, consider a routing scenario in a graph with edges E and vertex set V. A plan could be to find a path through the network represented as a sequence of vertices that respects the graph's topology. To enforce this, we could specify a global feasible set as a subset of $\{(p_t)_t | \forall t : (p_t, p_{t+1}) \in E\}$. The global feasible set is global in the sense that the constraints it enforces apply to all agents in the system.

By contrast, each agent a may desire to enforce individual constraints upon the plans it generates. We can represent them as a *local feasible set* L^a. For instance, in the routing example, agent a may wish to ensure that he finds a path that leads from its start location to its destination: $p^a \in L^a \subset \{p^a = (p^a_t)_t | p^a_0 = S(a), \exists k \forall t \geq k : p^a_t = D(a)\}$.

Depending on the environment, there might be many (possibly infinitely many) plans that are both legal and locally feasible. In most applications however, robots may have a preference over different plans implied by a local cost function $c^a : G \to \mathbb{R}$ that assigns a cost to different plans. (For instance, $c^a(p^a)$ may quantify the path length.) So, if robot a could plan independently, he would like to execute the solution to optimization problem:

$\min_{p^a \in G \cap L^a} c^a(p^a)$.

Unfortunately, this is not possible in environments with multiple agents as they need to avoid collisions (i.e. plans where two agents simultaneously use the same non-divisible resource). Let $p^{\neg a} = (p^r)_{r\in\mathcal{A}-\{a\}}$ denote the collection of plans of all agents except a. If a knew fixed $p^{\neg a}$, he could react to it by solving

$$\min_{p^a \in G \cap L^a \cap R(p^{\neg a})} c^a(p^a) \tag{1}$$

where $R(p^{\neg a})$ is the set of all paths that are not in conflict with the paths generated by $p^{\neg a}$. If $p^{\neg a}$ is a tentative, we can interpret $R(p^{\neg a})$ as the set of all plans that do not use any resource that are already used by any agent in $\mathcal{A} - \{a\}$ based on the current belief that all other resources will be available.

For all a, $R(p^a)$ is unknown a priori and hence, the individual optimization problems are undetermined (since the feasible sets are interdependent). This is where the necessity for coordination arises.

We can now restate the overall task description (comprising (MRPPP) as a special case) in general terms:

TASK: *Assume each agent a $(a = 1, ..., A)$ can choose a plan $p^a \in G \cap L^a$. Coordinate the planning process such that the overall outcome $(p^1, ..., p^A)$ of plans is conflict free (i.e. $\forall t : p_t^a \neq p_t^r, \forall a, r \in \mathcal{A}, a \neq r$) and such that the social cost $\sum_{a \in \mathcal{A}} c^a(p^a)$* is small.

The socially optimal solution can be stated quite easily as the solution of the centralized optimization problem

$$\min_{(p^1;...;p^A) \in G^A \cap \times_a L^a \cap I} \sum_{a=1}^{A} c^a(p^a) \tag{2}$$

where I is a set defined by inter-agent constraints that prohibit collisions. In other words, I is the set of all overall plans $p = (p^1; ...; p^A)$ such that all plans p^a, p^r use distinct resources (for $a, r \in \mathcal{A}, r \neq a$).

Unfortunately, such centralized approaches are known to scale poorly in the number of agents, even in expectation. They are NP-hard in the worst case and are limited by the typical architectural down-sides of multi-agent systems that rely on centralized planners. For example, central planners constitute computational and communication choke-points and a single points of failure (cf. e.g. [7]).

Since the centralized optimization problem acc. to (2) scales poorly, we will seek to replace it by iteratively solving a sequence of individual, tractable problems acc. to (1). Due to the hardness of the original problem we will have to be satisfied if the ensuing overall solution is not always socially optimal.

3 Mechanism

We propose an iterative mechanism that proceeds as follows:

In each iteration, agents plan independently based on their current beliefs of available resources. Initially each agent assumes all resources are available. The planning process in each agent r is done solving an opt. problem of the form (1).

Whenever a conflict is detected, the conflicting agents participate in an auction for the contested resource. The winner is allowed to proceed as if no conflict had occurred while the losers add new constraints preventing them from using the lost resource at the specific time t where the conflict occurred in future iterations (i.e. they update their beliefs about the available resources as encoded by R). Conflicts are resolved in time step order. That is, a conflict that would lead to a collision at time t is resolved before a detected conflict that would lead to a collision at time step $t' > t$. If we define the auction horizon to be the largest time step t where a conflict has been resolved then this horizon increases monotonically from coordination iteration to iteration until no more conflicts arise.

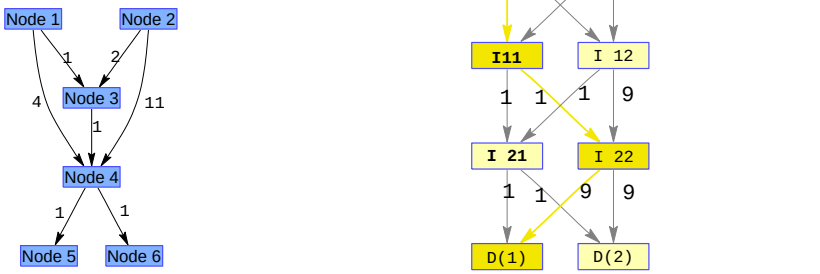
Whenever an agent has won a resource for a certain time step t in past iterations that she does not need anymore in her current plan, she releases it for t and informs the other agents of this event. Once all conflicts are resolved, the agents can execute their final plans.

Winner determination of an auction proceeds as follows: *All agents who simultaneously (at the same coordination iteration $i \in \mathbb{N}_0$) plan to use a resource at the same time step t submit a bid. The bid $b^r(i)$ that each contestant r submits equals $l^r(i) - s^r(i)$. Here, $l^r(i)$ is the cost a expects to experience (given its current belief in i of the available resources) if it would lose the resource. And, $s^r(i)$ is the cost r expects (given its current belief of the available resources) to incur if it can keep using the contested resource. The winner is determined to be the agent who submits the highest bid. If multiple agents have greater or equal high bids than all the other ones ($|\arg\max_{a\in\mathcal{A}} b^a(i)| \geq 2$), the robot with the highest index wins.*
To gain an intuitive motivation for the bidding rule, notice the bid quantifies the regret an agent expects to have for losing the auction (given its current belief of the availability of resources). Acknowledging that $s^{winner}(i) + \sum_{a\in losers} l^a(i)$ is the estimated social cost (based on current beliefs of available resources) after the auction, we see that the winner determination rule greedily attempts to minimize social cost: $\forall r : b^w(i) \geq b^r(i) \Leftrightarrow \forall r : s^r(i) + \sum_{a\neq r} l^a(i) > s^w(i) + \sum_{a\neq w} l^a(i)$.

Notice, there are several degrees of freedom regarding the architectural implementation of the mechanism. For instance, to detect a conflict, all agents communicate their their current plans to all other agents. With broadcast messages the communication effort per coordination iteration is hence in $O(A)$ where A is the number of agents. Then each agent would be responsible to detect the next conflict and arrange an auction with the other agents. Alternatively, the mechanism designer could set up a number of additional dedicated conflict detectors and auctioneers (e.g. one for a set of time steps or a set of resources).

As an illustration, consider a simple graph routing example. Two agents desire to find low-cost paths in a graph with transition costs as depicted in Fig. 1(a). Agent 1 desires to find a path from Node 1 to 5, Agent 2 from Node 2 to 6.

In the first iteration ($i = 1$), Agent 1 and Agent 2 both assume they can freely use all resources (nodes). Solving a binary linear program they generate their shortest paths as $p^1 = (1\,3\,4\,5\,5...)$ and $p^2 = (2\,3\,4\,6\,6...)$, respectively. Detecting a conflict at time step 2 and 3, the agents enter an auction for contested Node 3. Agent 1's estimated "detour cost" for not winning Node 3 (assuming he will be allowed to use all other nodes in consecutive time steps) is 2 which he places as a bid $b^1(i) = 2$. On the other hand, Agent 2's detour cost ist $b^2(i) = l^2(i) - s^2(i) = 12 - 4 = 8$ and hence, she wins the auction. Having lost, Agent 1 adds a constraint to his description of his feasible set (more precisely to R) that from now on prevents it from using Node 3 in time step 2. Replanning results in updated plans $p^1 = (1\,4\,5\,5\,...)$ and $p^2 = (2\,3\,4\,6\,6...)$. Being conflict-free now, these plans can be executed by both agents.



(a) Ex. 1.

(b) Ex. 2. S(a)/D(a): start/destination of agent a. Coordinated plans depicted in yellow (Agent 1) and cream (Agent 2) which happens to be socially optimal.

Fig. 1. Two examples. Numbers next to the edges denote the transition costs.

Notice how the laziness of our method protected us from unnecessary computational effort: the initial conflict at time 3 (Node 4) was implicity resolved by the first auction without the need to set up an explicit auction for Node 4 or bidding on all combinations of availability of Nodes 3 and 4.

Of, course, this positive effect of laziness may not always bear fruit - in several situations resolving a collision at one node may not prevent collisions from happening (or, trigger new ones) at other nodes. As an example consider Ex. 2 in Fig. 1(b) and assume Agent 2's initial plan visits Vertex I11 - after this conflict is resolved there will be a second at Vertex I21.

Nonetheless, Ex. 1 was designed to provide an intuition that it often can. In Sec. 4, we provide an experimental investigation of the number of collisions triggered in a typical multi-robot path planning scenario.

4 Coordination in a spatially continuous world and under uncertainty

4.1 Preliminaries- Sampling-based control and obstacle avoidance

Multi-robot motion planning and control problems in continuous maps have been addressed with mixed-integer linear programming (MILP) techniques [22]. Typically they rely on time-discretization only, without prior space-discretization. However, they are commonly solved with a centralized planner and typically do not take uncertainty into account. Recently, stochastic control methods have been suggested for single-robot path planning that accommodate for uncertainty in the effect of control signals. For instance, Blackmore et. al. [4] discuss a

particle-based method that can be used to generate a low-cost trajectory for a vehicle that avoids obstacles with adjustably high confidence. In their model, the plans p^a are time-discrete sequences of control inputs. The spatial location x_t^a of Robot a at time t is assumed to be a linear function of all previous control inputs plus some iid random perturbations $\nu_0, ..., \nu_{t-1} \sim \mathcal{D}$. So, given plan p^a, drawing n samples of perturbations for all time steps generates N possible sequences of locations (*particles*) $(x_t^{a,(j)})_t$ $(j = 1, ..., N)$ Robot a could end up in when executing his plan.

Formally, $x_t^{a,(j)} = f_t(x_0^{a,(j)}, u_0^a, ..., u_{t-1}^a, \nu_0^{(j)}, ..., \nu_{t-1}^{(j)})$ $(j = 1, ..., N)$ where f_t is a linear function and $u_0^a, ..., u_{t-1}^a$ is a sequence of control inputs as specified by Robot a's plan. Due to this functional relationship we can constrain Robot a's MILP's search for optimal control inputs by adding constraints on the particles.

Let T be the number of time steps given by the time horizon and temporal resolution. That is, $t \in \{1, ..., T\}$. Furthermore, let F be the free-space, i.e. the set of all locations that do not belong to an obstacle. Obstacle avoidance is realized by specifying a chance constraint $Pr((x_t^a)_{t \in T} \notin F) \leq \delta$ on the actual location of the robot. The chance constraint can be approximated by the constraints $\frac{1}{N}|(x_t^{a,(j)})_{t \in T} \notin F, i = 1, ..., N| \leq \delta$ [4] which we add to Robot a's individual MILP.

If $\mathcal{D}$ is a unimodal and light-tailed distribution, the particles $x_t^{a,(1)}, ..., x_t^{a,(N)}$ for a at time step t typically form a cluster mostly centered around the mean.

Note that the uncertainties due to the random perturbations accumulate over time. Hence, the standard error of the particle clusters along a robot's trajectory can be expected to increase with t.

4.2 Multi-Robot motion control under uncertainty

As collision-free plans are found by solving a MILP we could combine both approaches to a multi-robot stochastic control mechanism: Integrating the individual MILP's into one large central MILP (cf. to Eq. 2 in Sec. 2) we could then add an appropriate inter-robot constraint for each combination of particles in order to avoid collisions. Unfortunately, the number of integer constraints would grow superlinearly in the number of particles and even exponentially in the number of robots, rendering this approach computationally intractable.

Instead, we propose to apply our mechanism as follows: Each robot solves its local MILP to find a plan that corresponds to sequences of n particle trajectories. When two (or more) robots $a, r, ..$ detect their particle clusters $\{x_t^{a,(1)}, ..., x_t^{a,(N)}\}$, $\{x_t^{r,(1)}, ..., x_t^{r,(N)}\}$,.. 'come too close', they suspect a conflict and participate in an auction. The winner gets to use the contested region, while the losers receive constraints that correspond to a *virtual obstacle* (that is valid for time step t) and replan. Notice, for notational convenience, we omit the explicit mention of the coordination iteration i in our notation throughout the rest of the section.

Next, we will explain the application of our mechanism to the continuous path planning problem in greater detail. Every robot employs the path planning algorithm as described in [4] to generate a particle-trajectory that is optimal for

him. As explained in Sec. 3 the mechanism requires the robots to exchange their plans in every coordination iteration. However, they do not need to exchange all particles constituting their trajectories – it suffices only to exchange the optimal control inputs that lead to the particle trajectories (alongside the state or seed of their own pseudo-random-generator with which they drew their disturbance parameters).[3]

With this knowledge all the other robots are able to exactly reconstruct each others' particle trajectories. Now each robot locally carries out a test for collision by calculating the probability of a collision for each plan of every other robot.

Let $\{x_t^{a,(1)}, \ldots, x_t^{a,(N)}\}$ be the particle cluster that probabilistically describes the desired position of Robot a at time step t. Furthermore, let $\{x_t^{r,(1)}, \ldots, x_t^{r,(N)}\}$ be the particle cluster of Robot r. Let ϵ be a predetermined parameter representing the minimum distance allowed between two robots. For instance, we could set $\epsilon = 2d$ where d is the diameter of the robots which is a reasonable choice when defining a robot's location as the cartesian coordinates of his center point.

The probability of a collision of Robot a and Robot r at time step t is

$$\Pr(\|x_t^a - x_t^r\| < \epsilon) = \mathbb{E}_{x_t^a,\ x_t^r}\{\chi_C\} \tag{3}$$

$$= \int\int \chi_C(x_t^a, x_t^r) f(x_t^a) f(x_t^r) dx_t^a dx_t^r \tag{4}$$

$$\approx \frac{1}{N^2} \sum_{k=1}^{N} \sum_{j=1}^{N} \chi_C(x_t^{a,k}, x_t^{r,j}) \ , \tag{5}$$

where $f(x_t^a)$ and $f(x_t^r)$ are the densities representing the uncertainty regarding Robot a's and Robot r's locations, respectively, given the histories of their control inputs and where

$$\chi_C(x_t^a, x_t^r) := \begin{cases} 1 & , \text{ for } \|x_t^a - x_t^r\| < \epsilon \\ 0 & , \text{ otherwise.} \end{cases} \tag{6}$$

Therefore, the probability of a collision of Robot a and Robot r at time step t is approximated by their respective particle representations. If this approximated probability is above a predefined threshold δ, the robots engage in an auction for the contested spatial resource, as described in previous sections. The resource in this case corresponds to the right to pass through. We propose its denial to be embodied by a new *virtual obstacle* the loser of the auction, say Robot r, will have to avoid (but only at time t). By placing the virtual obstacle around the winner's location estimate at time step t, we will reduce the chance of a collision. We represent the new obstacle by a square (if planning takes place in higher dimensions a hypercube) $B_{\alpha+\epsilon}(\mu_t^a)$ with edge length $\alpha+\epsilon$ and centered at the sample mean μ_t^a of Robot a at time step t. The choice of this representation

[3] Note, the approach is still distributed as the computationally demanding task of computing the individual, optimal control sequences are still left to the individual robots.

is motivated by the fact that the chance constraints for a square-obstacle can be encoded by merely four linear and a few additional integer constraints [3, 4].

Obviously, the larger the virtual obstacle, the lower the probability of a collision between the robots. On the other hand, an overly large additional obstacle shrinks the free-space and may unsuitably increase path costs or even lead to deadlocks. Next, we will derive coarse mathematical guidelines for how to set the size of the virtual obstacle in order to avoid a collision with a predefined probability.

Let t be a fixed time step. Let $C := \{(x_t^a, x_t^r) | (\|x_t^a - x_t^r\| < \epsilon)\}$ be the event of a collision and $E := \{(x_t^a, x_t^r) | \|x_t^a - \mu_t^a\| \leq \alpha\}$ the event that the true position of Robot a at time step t deviates no more than α from the mean of its position estimate given by sample mean μ_t^a. By introducing a chance constraint with threshold $\frac{\delta}{2}$,

$$Pr[x_t^r \in B_{\epsilon+\alpha}(\mu^a)] < \frac{\delta}{2} \tag{7}$$

we enforce a bound on the collision probability. Introduction of the virtual obstacle to Robot r's constraints induces his planner to adjust the control inputs such that the fraction of particles $(x_t^{r,(j)})_{i=1,\ldots,N}$ that are inside the square box $B_{\epsilon+\alpha}(\mu^a)$ with edge length $\alpha + \epsilon$ around sample mean μ_t^a is bounded (and by particle approximation of the chance constraint, hence also the (approximated) probability that Robot r is inside the box). Parameter α needs to be specified after the desired δ is defined and we will now discuss a proposal how this can be done.

Let K be the event $\{x_t^r | x_t^r \in B_{\epsilon+\alpha}(\mu^a))\}$.

We have $\Pr(C) = \Pr(C \cap E) + \Pr(C \cap \neg E) = \Pr(C \cap E \cap K) + \Pr(C \cap E \cap \neg K) + \Pr(C \cap \neg E) = \Pr(C \cap E \cap K) + \Pr(C \cap \neg E)$ where the last equality holds since $\Pr(C \cap E \cap \neg K) = 0$. Furthermore, $\Pr(C \cap E \cap K) \leq \Pr(K)$ and $\Pr(C \cap \neg E) \leq \Pr(\neg E)$. Hence,

$$\Pr(C) \leq \Pr(K) + \Pr(\neg E) \tag{8}$$

Due to chance constraint (7) we know that control inputs are found that (for sufficiently large N) ensure that $\Pr(K) < \frac{\delta}{2}$. Hence, all we are left to do is to determine box parameter α such that

$$\Pr(\neg E) \leq \frac{\delta}{2}.$$

Let the distributions of Robot a be an isotropic Gaussian with covariance matrix $\Sigma = \sigma^2 I$ where I is the identity matrix. We can then control $\Pr(\neg E)$ by computing the σ-bounds of the normal distribution (considering the masses of its tails). For instance, an upper bound $\frac{\delta}{2} = 0.05$ on the collision probability can be achieved by setting $\alpha := 2\sigma$ and a bound of 10 percent by setting $\alpha := 1.64\sigma$.

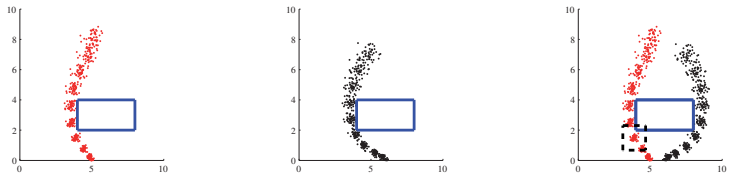
4.3 Experiments

We consider three different path planning scenarios, all with planning horizon of length ten, in our simulations:

– A simple example with only two robots to illustrate the very basic functionality of the mechanism.
– A quantitative evaluation of the average runtime behaviour for an increasing number of robots in an environment with a fixed number of obstacles.
– A quantitative evaluation of the average number of conflicts to be resolved by the mechanism in an increasingly complex environment for a fixed number of robots.

In all simulations the sample distribution for the robots was chosen as isotropic zero-mean white Gaussian noise with standard deviation $\sigma = 0.01$.

For an illustration, consider the simulations of a two-robot planning scenario depicted in Fig 2. Here two robots 1 and 2 start at locations at the bottom of



(a) Robot 1's initial plan.

(b) Robot 2's initial plan.

(c) Updated conflict-free plans.

Fig. 2. Simple example. Blue box: obstacle. Dashed box: virtual obstacle for Robot 2 for time step 3 (after he lost an auction against Robot 1).

a map. When generating paths to destinations at the far side of the map, they desire to avoid the obstacle (blue rectangle). Planning independently with the particle-control method, they find their individually cost-optimal trajectories as depicted in Figs. 2(a) and 2(b). Note, how the spread of their particle clusters increases as the uncertainties accumulate over time. Coming too close to each other on time step three (i.e. causing our coll. probability estimate to exceed our threshold δ), Robot 1 is determined to be the winner in the invoked auction. Hence, Robot 2 gets a virtual obstacle (dashed box) for time step 3 which increases the length for the path to its destination on the left to the (real) obstacle enough to induce him to take the (initially longer) way around on the right hand side of the obstacle (Fig. 2(c)).

It should be expected that the number of iterations of our mechanism depends on the number of collisions during coordination, which in turn, should increase with the number (and size) of obstacles (or decrease with available free-space) and the number of robots in the system. To develop an intuition for the dependence of run-time on these factors we conducted randomized experiments (with varying robot destinations and obstacle placements) in which run-time and number of collisions were recorded. The results for ten robots with varying starts, destinations and obstacles are depicted in the left part of Fig. 3.

In a third round of simulations, the obstacles were placed at fixed positions together with fixed, equally spaced, starting positions for the robots. In order to provoke potential conflicts, the agents' goals were drawn at random from a uniform distribution. We iteratively added more agents to the planning scenario and set up the mechanism to calculate conflict-free plans for varying numbers of robots. The results are depicted in the right plot of Fig. 3.

The simulations were implemented in MATLAB, with no particular emphasis on run-time optimization and all experiments were executed on a standard desktop computer. In summary, Fig. 3 illustrates that both the number of coordination iterations (collisions) and run-time increased moderately with increasing problem complexity.

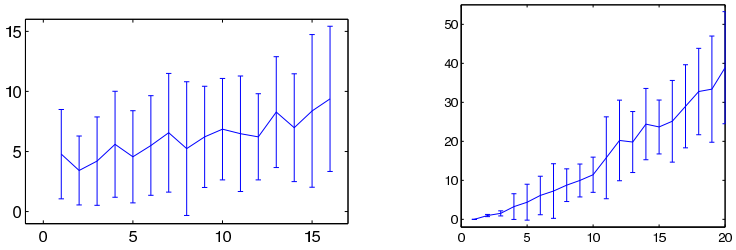


Fig. 3. Right: Runtime in seconds vs. number of robots. Left: Number of arising conflicts vs. varying number of obstacles. Plots show averages over 50 Monte-Carlo runs of randomized problems.

5 Related Work

Multi-robot coordination is a broad topic with numerous strands of works. The approach we present to collision avoidance and control is germane to a number of these strands comprising both approaches designed to operate in both continuous and in discrete worlds. It is beyond the scope of this paper to present an exhaustive survey of the extensive body of previous work that ranges across various disciplines. For surveys focussing on market-based approaches refer to [9, 14].

As a rather coarse taxonomy, present methods can be divided into centralized and decentralized approaches. *Centralized approaches* (e.g. [22][20]) typically rely on combining the individual agents' plans into one large, joint plan and optimizing it in a central planner. Typically, they are guaranteed to find an optimal solution to the coordination problem (with respect to an optimality criterion, such as the sum of all costs). However, since optimal coordination is NP-hard it is not surprising that these methods scale poorly in the number of participating agents and the complexity of the planning environment. With worst-case computational effort growing exponentially with the number of robots, these methods

do provide the best overall solutions, but are generally intractable except for small teams.

In contrast, decentralized methods distribute the computational load on multiple agents and, combined with approximation methods, can factor the optimal problem into more tractable chunks.

There are two classes of *decentralized coordination mechanisms*. The first class imposes local interaction rules designed to induce a global behavior that emerges with little or no communication overhead. For instance, based on a specific robot motion model, Pallottino et. al. [19] propose interaction policies that result in guaranteed collision avoidance and can accommodate new robots entering the system on-line. Furthermore, under the assumption that robots reaching their goals vanish from the system, the authors prove that eventually all robots will reach their respective destination locations. While in its present version uncertainty is not explicitly taken into account, it may be worthwhile endowing their method with an explicit error model and performing a similar analysis as we provide in Sec. 4.

The second class focusses on the development of mechanisms where coordination is achieved through information exchange succeeding the distributed computations.

Distributed optimization techniques have been successfully employed to substitute the solution of a centralized optimization problem by solving a sequence of smaller, decoupled problems (e.g. [5], [16], [17], [2] and [13]). For example, Bererton et. al. [2] employ Dantzig-Wolfe Decomposition [6] to decentralize a relaxed version of a Bellman MIP to compute an optimal policy. However, due to the relaxation of the collision constraints, collisions are only avoided in expectation. Many of these algorithms have a market interpretation due to passing Lagrangian multipliers among the subproblems.

Generally, *market-based approaches* have been heavily investigated for multi-robot coordination over the past years [23] [12] [9]. Among these, *auction mechanisms* allow to employ techniques drawn from Economics. They are attractive since the communication overhead they require is low bandwidth due to the fact that the messages often only consist of bids. However, as optimal bidding and winner determination for a large number of resources (as typically encountered in multi-robot problems) is typically NP-hard, all tractable auction coordination methods constitute approximations and few existing works provide any proof of the social performance of the resulting overall planning solution beyond experimental validation. An exception are *SSI auctions* [14, 15]. For instance, Lagoudakis et. al. [15] propose an auction-based coordination method for multi-robot routing. They discuss a variety of bidding rules for which they establish performance bounds with respect to an array of team objectives, including social cost. While multi-robot routing is quite different from the motion control problem, we consider some of their bid design to be related in spirit to ours. It may be worthwhile considering under which circumstances one could transfer their theoretical guarantees to our setting. One of the main obstacles here may be the fact that in SSI auctions, a single multi-round auction for all existing resources

(or bundles) is held. This may be difficult to achieve, especially if we, as in Sec. 4, desire to avoid prior space discretization and take uncertainty into account.

Among all multi-robot path planning approaches, *fixed priority methods* are perhaps the most established ones. In its most basic form introduced by Erdmann and Lozano-Perez [10], robots are prioritized according to a fixed ranking. Planning is done sequentially according to the fixed priority scheme where higher ranking robots plan before lower ranking robots. Once a higher ranking robots is done planning, his trajectories become *dynamic obstacles*[4] for all lower ranking robots, which the latter are required to avoid. If independent planning under these conditions is always successful, coordination is achieved in A planning iterations that spawn the necessity to broadcast $A-1$ messages in *t*otal (plans of higher priority agents to lower priority ones) where A is the number of robots.

By contrast, in our mechanism, A such messages need to be sent *per coordination iteration*. Although our results indicate that the number of these iterations scale mildly in the number of robots and obstacles in typical obstacle avoidance settings, such an additional computation and communication overhead needs to be justified with better coordination performance.

Note, our mechanism also incorporates an (in-auction) prioritization (as expressed by the robots' indices) that becomes important for winner determination whenever $|\arg\max_{a\in\mathcal{A}} b^a(i)| \geq 2$.

For a simple example where our method outperforms the fixed priority method reconsider Ex. 2 in Fig. 1(b). Regardless of whether Agent 1 had higher priority than Agent 2 or vice-versa the social cost would be $(1+1+1)+(1+9+9) = 22$. In contrast, the coordination with our mechanism achieves a social cost of $(1+1+9)+(1+1+1) = 14$.

As a first systematic performance test, we pitted the simple fixed priority method against our mechanism in 1000 randomized graph planning problems. In each trial, the planning environment was a forward directed graph similar in structure to the one in Fig. 1(b). Each graph had a random number of vertices (TxM - graphs where number of layers T unif$(\{3,...,10\})$ and number of nodes per layer M unif$(\{3,...,10\}$) and randomized vertex-transition costs drawn from unif$([1,...,100])$. The coordination task was to find a cost optimal path for each agent through the randomized graph where both agents had a randomized start location in the first layer and a destination vertex in the last layer. The results are depicted in Fig. 4. Let $S_p(\tau)$ be the social cost when coordination is found with the fixed priority scheme for the τth randomized graph problem and $S_l(\tau)$ be the social cost incurred when coordination was achieved with our lazy mechanism using agent indices that coincided with the priorities of the fixed priority algorithm. Each point on the plot in Fig. 4 represents the difference in social cost $(S_p(\tau) - S_l(\tau)/T_\tau)$ where T_τ was the number of forward layers in the τth

[4] The notion *dynamic obstacle* loosely corresponds to our *virtual obstacles* (cf. Sec. 4). The difference is that our virtual obstacles are only present at a particular time step whereas the dynamic obstacles span the whole range of all time steps. Furthermore, we described how to adjust the box-sizes to control the collision probability in the presence of uncertainty.

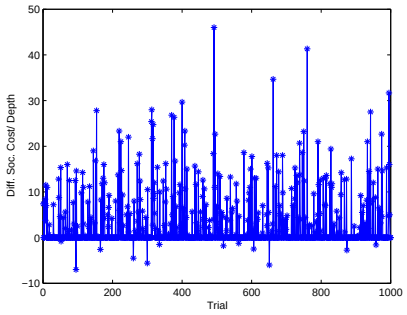


Fig. 4. Differences in social cost of priority method and lazy auction mechanism over 100 randomized graph problems. Our lazy auction method outperforms the fixed priority algorithm on the problem sample.

graph. The division was done to have comparable results since larger graphs would otherwise correspond to higher average path costs.

As can be seen from the plot, in the vast majority of trials, the social cost achieved with our mechanism was at least as low as the one achieved with the fixed priority method. In particular, our method performed strictly better than the fixed priority method on 296, equally good on 692 and poorer only on 12 randomly generated problem instances. These results do look promising and may serve as an indication that our mechanism can indeed outperform the basic fixed priority method on a larger number of randomized problems. Nonetheless, these first experiments where somewhat limited in scope and further research needs to be done to elucidate the exact nature of the problems where our method is guaranteed to outperform the fixed priority method and where it is not.

In priority methods, the overall coordination performance depends on the choice of the ranking and a number of works have proposed methods for a priori ranking selection (e.g. [1]). Conceivably, it is possible to improve our method further by optimizing its in-auction prioritization (robot indexing) with such methods. Exploring how to connect our mechanism to extensions of priority methods, such as [18], could have the potential to improve the communication overhead. Investigating the feasibility of such extensions will have to be done in the course of future research efforts as well.

6 Conclusions

In this paper, we presented a distributed, auction-flavoured multi-robot coordination mechanism. It is lazy in the sense that the agents only coordinate when necessary (due to conflicting resource usage), generating conflict free plans. We applied the mechanism to the MILP formulation of a multi-robot path planning problem, taking uncertainty about the robots' positions into account. Using our distributed mechanism in this scenario can be expected to be computationally more attractive, since the complexity of a single-agent MILP is significantly lower

than that of a centralized model. Our simulations suggest that the effort and communication overhead scales well in the number of agents and the complexity of the environment.

Since this paper reflects work in progress it can be the outset for several research questions that can be addressed in the context of future work. For instance, can we identify problem classes where our mechanism can provably be expected to be efficient and iterate to socially optimal paths? Our initial experiments support the intuition that our approach mostly outperforms simple fixed priorities, justifying an additional overhead in communication and computation. In future work, we would like to aim at deriving a theoretical explanation for which problem classes this can always be guaranteed to be the case. Complementing our experimental results in Sec. 4, we believe to able to establish an upper bound on the number of collisions that can arise during coordination under reasonable assumptions. This could provide theoretical underpinning for our observed experimental results that indicate that our coordination overhead does not suffer from a combinatorial blow-up. Moreover, we would hope to be able to establish distribution independent collision bounds that relate our confidence in collision avoidance to the number of particles.

At the bottom line, we hope this work succeeded in introducing our method to the reader and in convincing the research community that this work could form the basis for various fruitful future research endeavors.

References

1. M. Bennewitz, W. Burgard, and S. Thrun. Finding and optimizing solvable priority schemes for decoupled path planning techniques for teams of mobile robots. *Robotics and Autonomous Systems*, 41(2):89–99, 2002.
2. C. Bererton, G. Gordon, S. Thrun, and P. Khosla. Auction mechanism design for multi-robot coordination. In *NIPS*, 2003.
3. C. A. Bererton. *Multi-Robot Coordination and Competition Using Mixed Integer and Linear Programs.* PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 2006.
4. L. Blackmore, M. Ono, A. Bektassov, and B.C. Williams. A probabilistic particle approach to optimal, robust predictive control. *IEEE Trans. on Robotics*, 26(5), 2010.
5. Jan-P. Calliess and Geoffrey J. Gordon. No-regret learning and a mechanism for distributed multiagent planning. In *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS'08)*, 2008.
6. G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Oper. Res.*, 8:101–111, 1960.
7. R.K. Dash, N.R. Jennings, and D. C. Parkes. Computational mechanism design: A call to arms. *IEEE Int. Syst.*, 2003.
8. S. de Vries and R. Vohra. Combinatorial auctions: A survey. *INFORMS J. Computing*, 2003.
9. M. B. Dias, R. M. Zlot, N. Kalra, and A. Stentz. Market-based multirobot coordination: a survey and analysis. *Proceedings of the IEEE*, 2006.
10. M. A. Erdmann and T. Lozano-Perez. On multiple moving objects. *Algorithmica*, 1987.

11. Joan Feigenbaum, Christos H. Papadimitriou, and Scott Shenker. Sharing the cost of multicast transmissions. *J. Comput. Syst. Sci.*, 63(1):21–41, 2001.
12. B. Gerkey and M. Mataric. Sold!: Auction methods for multirobot coordination. *IEEE Transactions on Robotics and Automation*, 19(5):758–768, 2002.
13. C. Guestrin and G. Gordon. Distributed planning in hierarchical factored mdps. In *UAI*, 2002.
14. Sven Koenig, Pinar Keskinocak, and Craig A. Tovey. Progress on agent coordination with cooperative auctions. In *AAAI*, 2010.
15. M. Lagoudakis, V. Markakis, D. Kempe, P. Keskinocak, S. Koenig, A. Kleywegt, C. Tovey, A. Meyerson, and S. Jain. Auction-based multi-robot routing. In *Int. Conf. on Robotics: Science and Systems*, 2005.
16. T. Nishi, M. Ando, and Masami Konishi. Distributed route planning for multiple robots using an augmented lagrangian decomposition and coordination technique. *IEEE Trans. on Robotics*, 2005.
17. T. Nishi, M. Ando, and Masami Konishi. Experimental studies on a local rescheduling procedure for dynamic routing of autonomous decentralized agv systems. *Robotics and Computer-Integr. Manuf.*, 2006.
18. K. Sycara P. Velagapudi and P. Scerri. Decentralized prioritized planning in large multirobot teams. In *IROS'10*, 2010.
19. L. Pallottino, V. G. Scordio, E. Frazzoli, and A. Bicchi. Decentralized cooperative policy for conflict resolution in multi-vehicle systems. *IEEE Trans. on Robotics*, 23(6):1170–1183, 2007.
20. D. Parsons and J. Canny. A motion planner for multiple mobile robots. In *ICRA*, 1990.
21. T. Sandholm. Algorithm for optimal winner determination on combinatorial auctions. *Artif. Int.*, 2002.
22. T. Schouwenaars, B. De Moor, E. Feron, and J. How. Mixed integer programming for multi-vehicle path planning. In *European Control Conference*, pages 2603–2608, 2001.
23. A. Stentz and M. B. Dias. A free market architecture for coordinating multiple robots. Technical Report CMU-RI-TR-99-42, Carnegie Mellon, Robotics Institute, Pittsburgh,PA,USA, 1999.
24. C. Tovey, M. Lagoudakis, S. Jain, and S. Koenig. The generation of bidding rules for auction-based robot coordination. In *Multi-Robot Systems: From Swarms to Intelligent Automata*, volume 3, pages 3–14, 2005.
25. R. Zlot, A. Stentz, M. B. Dias, and S. Thayer. Multi-robot exploration controlled by a market economy. In *ICRA*, 2002.

Multi-Robot Path Planning with the Spatio-Temporal A* Algorithm and its Variants

Wenjie Wang, Wooi-Boon Goh

School of Computer Engineering,
Nanyang Technological University,
Singapore 639798
wang0570@e.ntu.edu.sg , aswbgoh@ntu.edu.sg

Abstract. This paper presents the design of an offline collision-free path planning algorithm for multiple mobile robots travelling simultaneously on a 2D gridded map. We first solved this problem by extending the traditional A* algorithm into 3D, namely two spatial and one time dimensions. This 3D approach proved computational costly and this led to the development of a novel and faster Spatio-Temporal (S-T) A* algorithm. This is a modified A* algorithm, which uses discrete time stamps and a temporal occupancy table to communicate previously planned routes and potential collision among robots. We further adapted the S-T A* algorithm to allow robots to stop and wait near nodes where potential collision is detected in order to increase their probability of finding a viable path to their destination. Using a time-based objective function that requires all robots in the environment to reach their respective destination in the shortest possible time, this decoupled planning strategy was done using a fixed priority based on the slowest robot first. Another variant using an adaptive priority scheme was then introduced to improve the success rate of finding a viable path for all robots as the number of robots in the fixed-sized environment increased. We present experimental results comparing the performance of the various path planning and priority schemes.

Keywords: Multi-robot path planning, A* algorithms, Offline path planning.

1 Introduction

A tangible interactive system for teaching children how to spell words in the English language is shown in Figure 1. This system consists of a set of passive and active cubes marked with letters of the English alphabets, which the child can arrange to form words. The active cubes are intelligent autonomous agents in the shape of mobile cube-like robots that can interact and assist the child by maneuvering themselves to appropriate localities so as to provide meaningful contextual scaffold to the child while he is forming a word. Overseeing the entire workspace is a video camera that is able to track the position and orientation of each individual letter block and mobile robot, essentially providing an instantaneous map of the entire operating environment. The central host PC uses this map to plan the traversal path of relevant mobile robots so that they can travel to appropriate positions in the map to form a

required word. Critical to the success of this interactive system is the need for a speedy offline path planning algorithm for the multiple mobile robots, which is the subject of this paper.

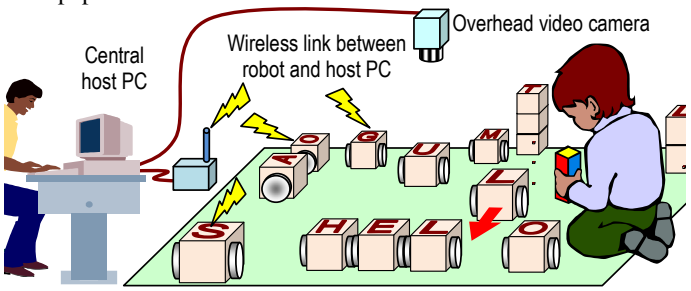


Fig. 1. An interactive educational system for teaching children spelling. A swarm of cube-like mobile robots helps the child by rearranging themselves to form words (e.g. 'HELLO'). An overhead camera observes the instantaneous location and orientation of each robot in the environment.

Algorithms for path planning have been widely researched. In our context, the problem of path planning could be described simply as follows: given a starting point and a target point in an environment with static and moving obstacles, the path planner is required to determine an optimal path between these two points based on some associated cost functions related to the path and the motion sequence of the agent. There are of course many other examples where path planning algorithms are relevant. For instance, a delivery truck needs to move from city A to city B, while there are several different paths between A and B, each path has an associated cost (e.g. distance, traffic lights, etc). So the truck driver needs to choose the shortest path to minimize fuel cost associate with the delivery of his goods to a client in city B.

Path planning algorithms can generally be divided into global or local. In global path planning, the information regarding the environment is known in advance. On the other hand, in local path planning, only the information related to the immediate vicinity surrounding the current robot is known (e.g. in autonomous navigation in unchartered terrains). In the context of our work, the overhead camera provides us a global view of the environment. Approaches using global path planning are able to determine a feasible solution if it exists but complex scenarios with large search spaces can be computationally costly if a guaranteed optimal solution is desired. Most practical solutions are sub-optimal and may suffice for applications such as ours. Computational speed is of reasonable importance even though the problem is resolved in an offline manner since an interactive system needs to be responsive to the actions of the user.

Many methods have been proposed to solve the global path planning problem. These include the Dijkstra algorithm, A* algorithm, variants of A* algorithm and so forth. These algorithms are guaranteed to find an optimal path if it exists. However, path planning for multiple robots is still an active research area with it many unresolved problems. Not only is there a need to determine a optimal path for each individual robot, there is also the need to coordinate the motion sequences of the robots to avoid collision when their paths intersect. Approaches for path planning of

multiple robots can be generally divided into coupled or decoupled. In a coupled approach, all robots plan their path simultaneously using a centralized planner to avoid colliding into one another. The advantage of a coupled approach is that its solution is complete. However, the dimension of this approach is the sum of degree of freedom of all robots. This means its computational time increases exponentially with the robot count. An example of a coupled approach is by Svestka and Overmars [1], who used a super graph method to coordinate the path for all robots. It combines the workspace of all robots into one workspace and each robot plans its path simultaneously. But only one robot can move at a time. This approach is probabilistically complete but does not scale well with increasing robot count.

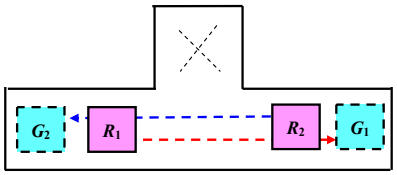


Fig. 2. A deadlock situation in prioritized planning.

The alternative approach is a decoupled approach that reduces the dimension of path planning by making each robot plan its path individually. Associated with the decoupled approach are the issues related to prioritized planning and path coordination. In prioritized planning, each robot is given a priority. The robot with the highest priority plans its path first and its resulting path influences the way the next highest priority robot would plan its path and so on. In path coordination, each robot searches its path independently and then adopts some strategy such as speed modification or stop-and-wait delays to avoid collisions. Guo and Parker [2] proposed a distributed approach for path planning of multiple robots. In this approach, the D* algorithm is used to search the path for each robot independently. The coordination between robots is realized using a simple priority scheme where the robot with a lower priority has to give way when there is an imminent collision with another with a higher priority. However, this approach can suffer from deadlocks such as that shown in Figure 2. In a distributed approach, robots R1 and R2 will plan shortest paths denoted by red and dashed lines respectively. No matter which one has the higher priority, when one chooses the most efficient path, the other will not be able to reach its destination. Such situations could be solved using a coupled planner that searches for a solution that will move one robot to the place denoted by × first, so that the other one can pass through to its destination. Marchese [3] adopted a potential field method for path planning of multiple robots. However, this approach suffers from local minima in which the cooperation between attractive force and repulsive force is equal to 0. Lee [4] proposed a collision map for path planning of two robots. Time delay or speed change strategies are used for coordinating robots to avoid collisions. Ji et al. [5] extended this idea to multiple robots. However, this decoupled method also suffers from deadlocks.

In this paper, we introduce a modification of the standard A* algorithm called the Spatio-Temporal (S-T) A* algorithm for path planning of multiple mobile robots. We have adopted a decoupled approach, where a centralized planner uses the proposed S-

T A* algorithm to find the lowest cost path for each robot in a sequentially order based on its assigned priority. A novel adaptive priority re-assignment scheme is proposed to improve the probability of path solution for all the multiple robots.

2 Related Work

The problem of path planning for multiple robots is in many ways similar to the problem of path planning for single robot in a dynamic environment where there are unpredictable obstacles. One could view the other robots as dynamic obstacles. In an unknown environment, incremental heuristic search algorithms have been proposed like Lifelong planning A*[6], D*[7], D* lite[8], and others. D* searches all configuration space at the start and then updates only the affected parts when there are changes in environment. Lifelong planning A* searches the configuration space until the goal is found. The search space is less than D* algorithm in general, and Lifelong planning A* only updates the affected parts which have been searched. D* lite is a backward lifelong planning A*. The major difference between them is that lifelong planning A* determines the optimal path between the start point and the goal point repeatedly as the edge costs of a graph change, while D* lite determines the optimal path between current point of the robot and the goal point as the edge costs of a graph change while the robot moves towards the goal point. These online algorithms are suitable for dynamic environment with unknown static obstacles but for an environment with moving obstacles, they do not work efficiently. To improve efficiency, strategies such as abstraction and path refinement have been proposed. In grid-based map, high grid resolution is good for finding an optimal solution but this comes at increased computational cost. Abstraction is a technique to search a workable path in coarse resolution and path refinement can subsequently be used to refine the solution using a high resolution map. Unfortunately, abstraction and path refinement tend to suffer from sub-optimality. Koenig and Likhachev [9] proposes the Adaptive A* algorithm as a means to speed up the searching process instead of lowering the map resolution. Adaptive A* algorithm updates the heuristics according to the search result obtained from an initial application of the standard A* algorithm. In this way, Adaptive A* could make subsequent search operations expand less states and therefore run faster. Fiorini and Shiller [11] proposed using velocity obstacles for path planning in a dynamic environment with known moving obstacles. However, this approach may suffer from oscillations problems when used for path planning of multiple robots. More recently, Snape et al. [12] extended velocity obstacle to the hybrid reciprocal velocity obstacle for online path planning of multiple robots. It avoids oscillation by explicitly considering reciprocity where each robot assumes other robots are cooperating to avoid collision.

During past decades, many offline algorithms have also been proposed. Wang and Botea [10] proposed a modified A* algorithm for multi-agent path planning on grid maps. This approach searches alternative path for each vertex on map to avoid collisions between robots. The robot with the lower priority gives way to the one with a higher priority by moving into an alternative path to avoid collision and then returning back to its intended path. However, the grid map resolution used by this

approach is not flexible since it is inherent tied to the size of the moving robot and the computation of the alternative paths is costly. Silver [13] proposed the Cooperative A* (CA*) algorithm for cooperative pathfinding among multiple robots. It searches for a solution in a 3D space-time map. Two variants of CA* were proposed. He claimed that both the Hierarchical CA* and Windowed Hierarchical CA* can be used in a real time environment. Jansen and Sturevant [14] introduced a direction map for cooperative pathfinding. This direction map is used to change traversed path cost of each node. The direction map is changed using a learning formulation.

Unlike CA* [13], the proposed Spatial-Temporal (S-T) A* algorithm searches the path solutions in a 2D spatial map. Using a 2D map reduces the size of the search space and therefore reduces the time required to compute the solution. However, search space compression makes it harder to find viable solutions for all robots when their numbers are increased. We proposed additional mechanisms such as a wait time insertion strategy and adaptive re-ordering of the robot's priority to improve the performance of the S-T A* algorithm.

3 Problem Description

In a configuration space C, there are n homogeneous robots and m static obstacles. Each robot R_i has a unique initial state (x_i, y_i, θ_i) and end state (x'_i, y'_i, θ'_i). The values x and y represent the 2D location of the robot in a grid-based map, while the value θ represents the direction of the robot. All robots can rotate and move to their four directly connected neighbors with a constant velocity. The spatial coordinates and orientation of each robot at any time t is known to the central planner. All robots will move simultaneously toward their respective target point and do so without colliding into each other or known static obstacles. Let t_i represents the cooperative time cost of robot R_i, T_i is the individual time cost for robot R_i to reach its destination if there is no other robots. The goal is to find a viable solution that will allow all n robots to reach their respective target positions without incurring any collision and to achieve this based on the time-based objective function given by

$$T = \max(T_i) \quad \text{where } T = \operatorname{argmin}(\max(t_i)),\ i = 1,2\ldots n \tag{1}$$

In other words, the goal is to find the coordinated paths for all n robots such that the time (T) taken by the last robot to arrive at its target is minimized. For this reason, our decoupled path planning approach uses a priority strategy based on meeting this objective function. In our fixed priority (FP) scheme, the highest priority robot (whose path will be planned first) is the one that takes the longest time to reach its destination among the n robots. The time taken by the highest priority robot is essential the objective function T. We estimate this time by applying the standard A* algorithm on each robot to compute the fastest path while all other robots are at rest in their initial position. The next highest priority is given to the robot with the next longest path and so on.

The configuration space C is shown in Figure 3. It is a space-time configuration consisting of two spatial dimensions given by (x,y) and a time dimension given by t.

At each discrete time interval *dt*, a new map is generated since the moving robots (with constant velocity *v*) will move a unit grid node to a new location on the map as shown in Figure 3. Two equal-sized robots R_1 and R_2 are required to travel to their respective goal G_1 and G_2 in the fastest possible manner without colliding into each other or the static obstacle block *O* in the map.

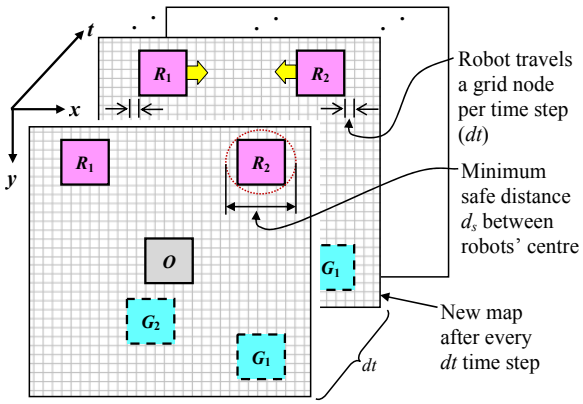


Fig. 3. The space-time configuration defined by two spatial dimensions *x* and *y*, and one time dimension *t*.

In our problem definition, the following assumptions were made:

- All objects in the environment (i.e. movable robots and static obstacle blocks) are square and of equal-sized width *w*;
- They can occupy more than one grid node and at any given time step, they are always centered about a grid node;
- The minimum safe distance between robots' centers d_s is given by $1.414w$ (see Figure 3). Since static obstacle blocks are the same size as the robot, the minimum safe distance between the robot's center and the center of the closest static obstacle block is computed in the same manner;
- Each robot is constraint to travel in any four directions along the grid lines with a constant velocity *v*. They travel by either going forward or reversing and therefore only required to make discrete rotations of 90 degrees to traverse all four directions.

4. Path Planning

The A* algorithm is a popular path planning algorithm. It has three cost values *g*, *h*, and *f*. The value *g* computes the path cost between the current and initial node. The value *h* estimates the path cost between current and target node. And *f* is the sum of *g* and *h*. The A* algorithm keeps a data structure called *openlist* for caching nodes to be expanded and another called *closelist* for storing the expanded nodes. The nodes in *closelist* should not be explored and added back into *openlist* again. At the beginning,

both *openlist* and *closelist* are empty and it puts the starting node into *openlist*. The starting node will then be expanded and the nodes around starting nodes will be explored and put into *openlist*. The expanded starting node will be deleted from *openlist* and put into the *closelist*. Then it will expand next node with the least *f* value in *openlist*. The node in *openlist* will be replaced if the same node with lower *g* value is added into *openlist*. This procedure is repeated until the target node is expanded, which in this case will cause the A* algorithm to return a successful search. If *openlist* eventually becomes empty, this signifies a failed search.

The A* algorithm is often use for planning the path of a single robot in a stationary environment. In its standard form, it is not suitable for path planning of multiple robots since other moving robots need to be considered when searching a collision-free path for each robot. We next introduce the 3D A* algorithm for solving this problem.

4.1 The 3D A* Algorithm

Given the configuration space *C* shown in Figure 3, the path search must now be performed within a three dimensional (3D) grid map consisting of two spatial dimensions and one time dimension, where each grid node is represented as (x,y,t). Assume the robots can translate a distance *dp* during one time interval *dt* (with no rotation). So for each time interval *dt*, the robot has five possible action to choose from: $(x+dp,y,t+dt)$, $(x-dp,y,t+dt)$, $(x,y+dp,t+dt)$, $(x,y-dp,t+dt)$ and $(x,y,t+dt)$. The path solution is a collision free path from $(x_0,y_0,0)$ to (x_d,y_d,t_d). The values x_0 and y_0 represent the coordinate of starting point, while the values x_d, y_d and t_d represent the coordinate of target point and the time taken to get to destination.

Cooperative A* (CA*) [13] is one variants of a 3D A* algorithm for multiple robots. In CA*, each robot searches its path in a 3D space-time map and takes into account the planned routes of other robots. These planned paths are marked into a reservation table and entries that are considered impassable are avoided in the searches of subsequent robots. A wait move is provided in the robot's action set to allow it to remain stationary in order to avoid collision. Due to the large search space generated by the 3D grid map, the 3D A* algorithm can be computationally intensive, especially if in the worst case, it searches all grid nodes in the 3D space. In order to address this limitation, we propose the Spatio-Temporal A* algorithm, which runs significantly faster.

4.2 Spatio-Temporal A*

In the Spatio-Temporal (S-T) A* algorithm, we adopt a 2D spatial grid map instead of 3D map. Collision detection with static obstacles is monitored using direct distance check. On the other hand, collision detection with other moving robots in the environment is done with the aid of a data structure called the *temporal occupancy table* (TOT), which stores the time-indexed planned path that has been computed for each of the *n* robots. The TOT has dimensions similar to the 2D spatial grid map, so every node in the grid map has a corresponding table entry in the TOT. The table

entries in the TOT are multi-layered, which means each of the n robot is able to input their own time stamp numbers from 0 to t indicating their respective planned path at any instance in time (incremented in discrete time step of dt). The robot moves along their respective ascending numbers as shown in Figure 4. When a robot R_i is currently searching its path and intents to move to the next node $p(x,y)$ at time stamp S_j, it queries all time stamp entries around the vicinity of table entry $p(x,y)$ in the TOT. It then checks to see if S_j matches any of the existing TOT entries populated by the higher priority robots R_{i-1}, R_{i-2} *to* R_0 earlier. Collision is detected if a time stamp match is found within the minimum safe distance vicinity d_s around $p(x,y)$.

The main difference between the S-T A* and the standard A* algorithm is that each explored node will query the TOT in the manner described earlier to check for possible collision with higher priority robots (whose path has already been planned) before it is added into openlist. Based on the stated objective function T, time is taken as the cost to update g values in S-T A* algorithm. The time cost is not just associated with the distance travelled but it also includes the time (trot) taken for the robot to perform a 90 degree rotation each time the travel direction changes. The heuristic function h used in our grid map environment is the Manhattan distance, which is converted to a time cost using a proportional distance-time relationship. So the S-T A* algorithm can be summarized as follows:

1. Initialize an empty *openlist* and *closelist*; and the g and h values of all nodes.
2. Put starting node into *openlist*.
3. Expand node v_i with the least f value in *openlist*. Where f is given by the sum of the two time-related cost values g and h.
4. Explore all neighboring nodes around v_i and consider adding newly explored nodes into *openlist* if they have no collisions detected in the TOT.
5. These explored nodes that are to be added to *openlist* will replace similar nodes already in *openlist* if their g values are lower. Otherwise, they will not be added into *openlist*.
6. Put expanded node v_i into *closelist*.
7. Goto step 3 unless target node is expanded or *openlist* is empty.

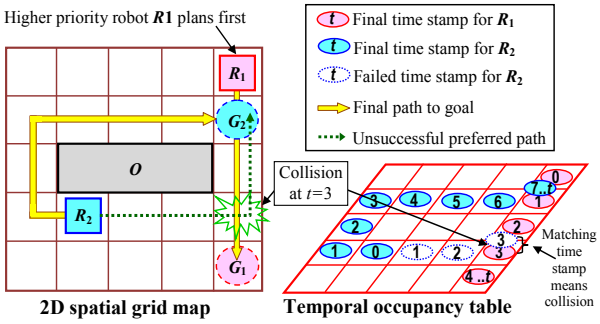


Fig. 4. The Spatio-Temporal A* algorithm makes use of the time stamp information in the temporal occupancy table (TOT) to detect potential collision.

The 2D spatial grid map in Figure 4 shows an example of the S-T A* algorithm in action. Robots R_1 and R_2 will move to their respective destinations G_1 and G_2. Robot

R_1 with a higher priority plans its path first, which means R_2 needs to plan its path taking into considering the path of R_1. The shortest path for R_2 to G_2 is denoted by dotted arrow. However, it detects a collision at t=3 and will then search other expanded nodes in openlist to find the next fastest alternative path to G_2 (solid arrow).

Though the S-T A* algorithm computes very efficiently, it quickly fails to find a viable path for all n robots when the number of robots in the environment increases. One way to improve this situation is to allow the robot to wait at an appropriate node to allow a colliding robot to move on before proceeding to the intended node. This is the strategy proposed in the next section.

4.3 S-T Wait-near-collision A*

In the basic S-T A* algorithm, nodes which are deemed to have collision are not added into *openlist*. This often results in a more time consuming longer path to the destination if the shorter path is blocked by a moving obstacle for a short period of time. If the robot could stop and wait for a short duration at an appropriate node, the shorter path could still be taken. The simplest way to add wait time to avoid collision is to delay the robot at the starting point before it begins moving, as adopted by Ji et al. [5]. This approach is problematic since the starting point may itself be a collision point if a higher priority robot has determined that this point is to be part of its fastest path while the lower priority robot is waiting. Moreover, the starting point wait delay inserted could be unduly long if we need to ensure every node in the current path is to be collision free. In our work, we adopted a more flexible wait time insertion strategy. Our goal is to wait as close to the node where collision has been detected. We call this the S-T Wait-near-collision (S-T-W) A* algorithm. We insert wait time at the closest possible antecedent node near the collision node. In this way, wait time insertion is not limited to only the starting node but any node in the current path that has already been planned. Preference is given to the node closest to where collision would have happened if the robot did not stop.

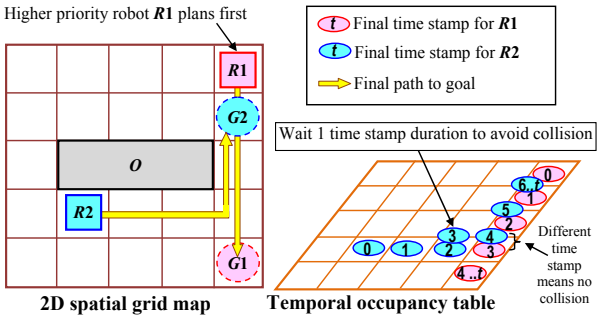


Fig. 5. The S-T A* Wait-near-collision algorithm makes the robot wait next to the node where collision has been detected.

S-T-W A* algorithm works in a similar way to the basic S-T A* algorithm except for the process of exploring nodes. The basic S-T A* algorithm ignores an explored

node if it has collision. However, the S-T-W A* algorithm will do a backward search to find the node's nearest antecedent node where the robot R_i can stop and wait for a period of time t_w necessary for the collision nodes to become passable. For instance, if an explored node v_i is found to have collision with a dynamic obstacle, then the backward path v_i, v_{i-1} .. v_0 (where v_i represent current explored node, v_{i-1} is v_i's parent and v_0 is the starting node) need to be checked sequentially until a node v_k (where $0 \leq k < i$) can be found where enough wait time can be added to make all nodes v_j (where $k < j \leq i$) collision-free. If v_k is found, then v_i can be added into openlist, or else v_i is ignored and we go on to explore or expand the next node. Figure 5 shows an example of the S-T-W A* algorithm in action. Robot R_2 takes a wait action at time stamp t=3 to let robot R_1 pass by without collision. R_2 then carries on moving again. Compared to the basic S-T A* algorithm in Figure 4, the path for robot R_2 in this case is faster by one unit time.

4.4 S-T A* with Adaptive Priority

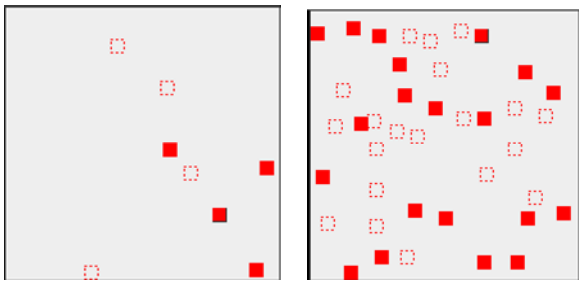
So far, all the variants of the decoupled path planning algorithms introduced will sequence the n robots using a fixed priority that is dependent on the individual time cost for each robot to reach their respective destination if no other robot is moving. The highest priority robot is the one that takes the longest time. In order to improve the probability of finding a viable solution for all n robots in a crowded environment, we introduce an adaptive priority re-assignment strategy. In adaptive priority re-assignment, the robot's priority is first given based on the fixed priority. If the current path searching robot R_i fails to find a viable path to its destination, its priority is raised by one level and is allowed to re-plan its path again. It continues to escalate its priority until it finds a viable path. This can at times leads to a priority adjustment racing problem, where a lower priority robot escalate its priority only to have the displaced higher priority robot regaining back its original priority when it cannot find a new path to its destination. A simple check is used to detect this cyclical priority re-assignment so that the path planning algorithm can terminate and return an unsuccessful result.

In prioritized planning an additional consideration needs to be catered for. A higher priority robot R_1 will not take into account the path planned by a lower priority robot R_2. As a result, a collision may happen when R_2 reaches its destination and remain at rest while $R1$ has to pass through R_2's destination point. So before a robot R_i reaches its destination p_d at time stamp S_j, it will query the temporal occupancy table (TOT) to ensure all existing TOT entries populated by the higher priority robots R_{i-1}, R_{i-2} to R_0 earlier around the vicinity of p_d is less than S_j. If not, robot R_i is made to wait at an appropriate node near p_d until it is safe to harbor at its target destination.

5. Experimental Results

In our simulation environment, a grid map with 100×100 nodes was used. Each robot is a square that occupies 5×5 grid nodes. Robots are limited to 4 directions of travel (with constant velocity) and 90 degrees for rotation. For convenience, no static

obstacles are used. All robots are expected to move to a new unique location from its unique starting point.



(a) (b)

Fig. 6. Two random initial scenarios. The robots' initial and destination states are shown in solid and dashed outline red squares respectively. (a) 4 robots and (b) 20 robots in a 100×100 grid map.

In our experiments, we compare the performance of various algorithms in 100 random initial scenarios for different number of robots (i.e. from 2 to 20). Two examples are shown in Figure 6. For a fair comparison, all algorithms used the same 100 initial scenarios. The simulations were carried out on an Intel(R) core™ 2 quad (2.83 GHz) with 3.25 GB of memory. This simulation program is written in the Java language on the Eclipse development environment.

5.1 Performance comparison of the 3D A* and variants of the S-T A* algorithms

We compare the performance of four algorithms, namely the 3D A* algorithm and three variants of the S-T A* algorithm. Fixed priority assignment was used when simulating the 3D A* (3D-FP A*), basic S-T A* (S-T-FP A*) and S-T A* with Wait-near-collision strategy (S-T-W-FP A*). The final S-T A* variant is the S-T-W A* algorithm with an adaptive priority re-assignment strategy (S-T-W-AP A*). In these set of simulations, the time taken to traverse one grid node is the basic unit time stamp dt and time to rotate 90 degrees was set at $9dt$. In these experiments, we compared two performance measures. The first is the *success rate* of finding viable paths for all n robots. Success rate is defined as the number of times in the 100 random initial scenarios all n robots found a path to their respective destinations without collision. The second is *computational time*. This is the average computational time taken to compute viable paths for all n robots. Simulation runs that had no solutions were discarded in computing these average times.

5.1.1 Comparison of success rate

Since S-T-FP A*, S-T-W-FP A* and 3D-FP A* algorithms use a fixed priority assignment. This means the robots with higher priorities may block the path of those with lower priorities, making it difficult for low priority robots to find a viable path to their destinations. The effectiveness of the proposed adaptive priority re-assignment strategy is evident in the success rate results shown in Figure 7, especially in a

crowded environment. S-T-W-AP A* continued to maintain a success rate of above 65% in a 20 robots simulation scenario (see Figure 6b). Failures with high robot counts were mainly due to deadlock situations similar to that shown in Figure 2.

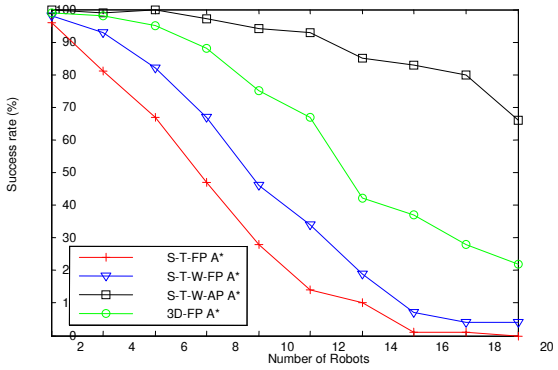


Fig. 7. Success rates of the four algorithms plotted against number of robots.

The strategy of inserting wait time also clearly improves the chances of finding viable solutions for all n robots, as can be seen from the better success rate of S-T-W-FP A* compare to the basic S-T-FP A* algorithm. The slower 3D-FP A* algorithm does perform better than the S-T-FP A* and S-T-W-FP A* algorithms, which suggest that searching within a 3D space-time grid map does provide a more extensive search than when it is done using a 2D spatial grid map. However, this comes at an unacceptably high computational cost (see Figure 8). A good compromise is to use the S-T-W-AP A* algorithm, which incorporates Wait-near-collision time insertion and adaptive priority re-assignment.

5.1.2 Comparison of computational time

As expected, the computational time results in Figure 8 clearly show that the slowest algorithm is the 3D-FP A* algorithm. Even with only two robots, due to the large 3D search space, the computational time remains high. The fastest algorithm is the basic S-T-FP A* algorithm. Its computational time increases linearly and gradually with increasing robot. This fast algorithm would be ideal for our application but its poor success rate may suggest that it would have to be used in conjunction with one of the other S-T variants as a 2-pass strategy (e.g. use basic S-T-FP A* first, if fail than use S-T-W-AP A*).

The computational time for S-T-W-FP A* algorithm was observed to increase much faster than the basic S-T-FP A* algorithm. This is because of the additional computational burden of performing wait time insertion when collisions are detected and more collisions are likely as the number of robots increase. Of the four algorithm, the one that is most influenced by the increasing number of robot is the S-T-W-AP A* algorithm that used adaptive priority re-assignment. The permutation of re-assignment priority increase significantly with increasing number of robots. The likelihood of

performing priority re-assignment also increases with more robots. This is evident in the steep slope in the computation time result of S-T-W-AP A* in Figure 8.

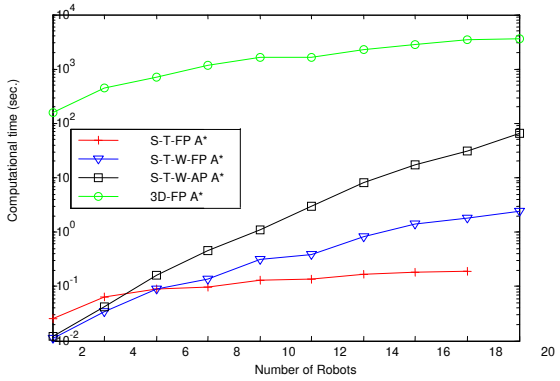


Fig. 8. Computational time of the four algorithms plotted against number of robots.

5.2 Meeting the Objective Function *T*

We studied the ability of the three different version of the S-T A* algorithms to meet the time-based objective function *T* defined in section 3. It is presented as the percentage of times the slowest robot to reach its destination actually did so within the objective function *T*.

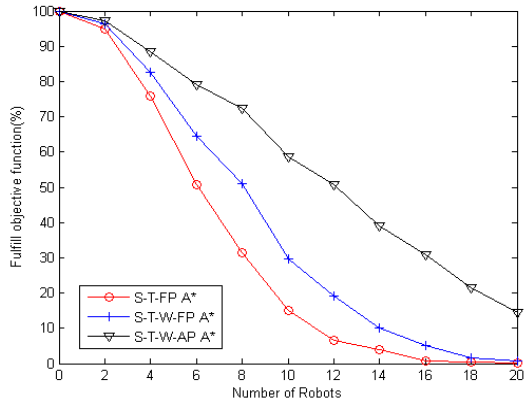


Fig. 9. The percentage of runs from all simulation runs that satisfy the objective function *T*, for all three algorithms**.**

Figure 9 show the percentages of runs from all simulations that satisfy the time-based objective function *T* defined in (1). Under fixed priority assignment, the S-T-W-FP A* algorithm met the objective function better than the S-T-FP A* algorithm. The wait time strategy not only increased success rate but also the number of

simulation runs that satisfies objective function *T*. Unfortunately, the fixed priority strategy falters as the number of robots increased. Under these circumstances, the novel adaptive priority strategy in the S-T-W-FAP A* algorithm is much better in producing higher percentage of runs that can meet the objective function *T*, besides producing more successful runs.

5.3 Rotation Speed of Robot (t_{rot})

We ran simulations to observe how the rotation speed of the robot (relative to its translation speed) affected the performance of the S-T A* algorithm. Simulations were done using the S-T-W-FP A* algorithm for rotation speeds of *dt, 3dt, 6dt, 9dt, 15dt and 30dt.* The time to translate from one node to another remains at the basic time stamp unit of *dt*.

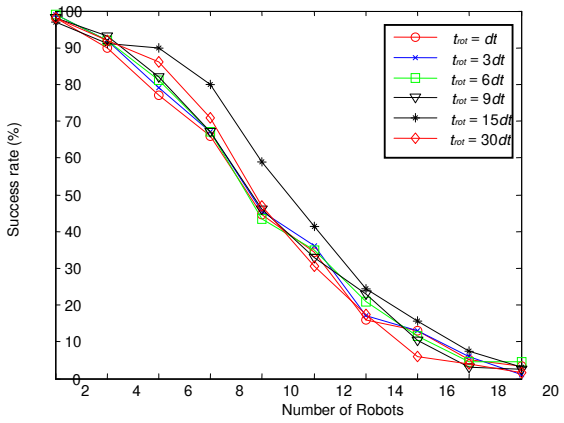


Fig. 10. Success rates of the S-T-W-FP A* algorithm with different rotation time t_{rot} plotted against number of robots.

The results in Figure 10 shows that the various rotation speeds of the robot does not significantly affect the success rate of the algorithm. Some improvement was observed at rotation speed of 15*dt* but it was not significant enough to have a meaningful interpretation. However, rotation speeds has a significant impact on the computational time of the S-T-W-FP A* algorithm as shown in Figure 11. This is because a longer rotation will increase the number of time stamps embedded within the overall path of each robot. This effectively increases the need for the S-T-W-FP A* algorithm to do more collision detection for each additional time stamp entry into the TOT, thus increasing the overall computational time. This result suggests that significant reduction in interactive response time for a path planning system (such as that shown in Figure 1) can be obtained if the physical robots can be designed to perform a rotation maneuver as quickly as possible

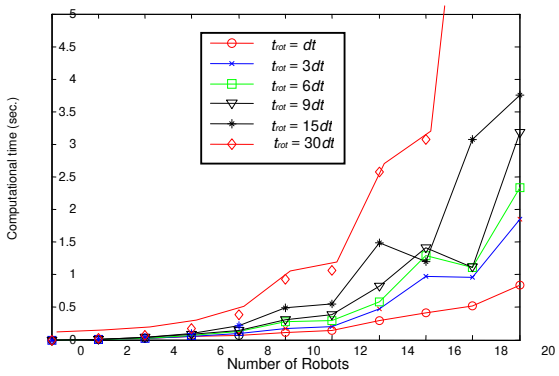


Fig. 11. Computational time of the S-T-W-FP A* algorithm with various rotation time t_{rot} plotted against robot count.

5.4 Different Wait Time Insertion Strategies

Section 4.3 discussed two possible wait time insertion strategies. The one adopted in [5] inserts wait time at the starting point and we propose inserting wait time as close to the collision node as possible. Using the S-T-W-AP A* algorithm, we simulated the performance of both these strategies from the perspectives of success rate and the computation time (as defined in sections 5.1.1 and 5.1.2 respectively). In these experiments, we set *dt* as the time for traversing one grid node and 3*dt* as the time to perform a 90 degree rotation.

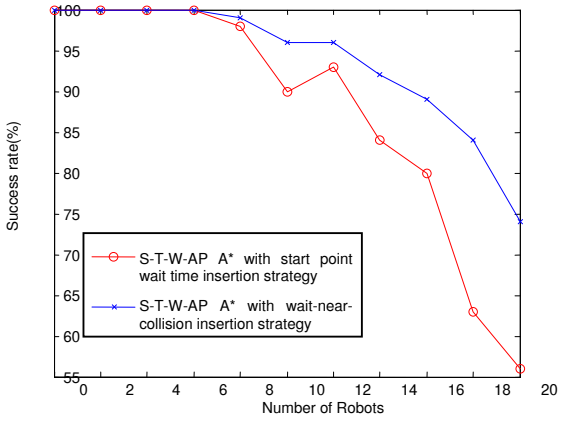


Fig. 12. Success rates of the S-T-W-AP A* algorithm with various wait time insertion strategies and robot count.

Figure 12 shows that the Wait-near-collision strategy has a higher probability of finding viable paths for all n robots as the number of robots in the environment increased. The better success rate over the starting point wait time insertion strategy became more evident with increasing number of robots. When comparing computational time (see Figure 13), the Wait-near-collision strategy also proved to perform better. The approach of inserting wait time at the starting point must ensure all nodes in the path are collision free and this takes more computational time to perform then the backward search technique of our Wait-near-collision approach.

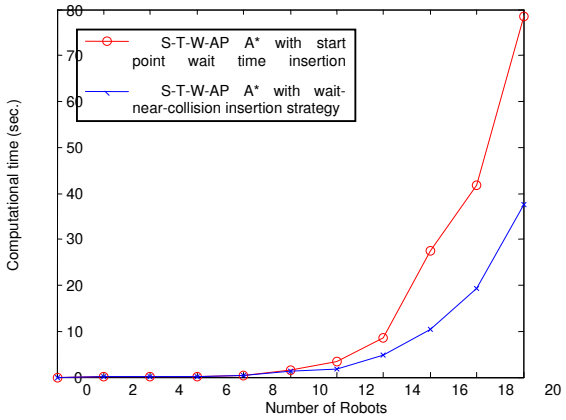


Fig. 13. Computational time of the S-T-W-AP A* algorithm with different wait time insertion strategies and robot count.

6 Conclusions and Future Work

In this paper, we proposed a Spatio-Temporal A* algorithm for path planning of multiple robots. It searches viable path solutions for robots in 2D spatial grid map and checks for collision with moving obstacles by using a temporal occupancy table. Our simulation results show that the basic S-T A* algorithm has poor success rate in crowded environments. However, with the addition of a strategy to insert wait time near collision nodes and the sequencing of path planning using an adaptive priority re-assignment scheme, we are able to obtain success rates that are superior to the more computationally demanding 3D A* algorithm. We are currently researching the coordination strategy between multiple robots to improve our ability to meet the time-based objective function T even in crowded scenarios. We are also looking into improving the computational time of the S-T-W-AP A* algorithm by adopting some abstraction and path refinement techniques for hierarchical planning.

Acknowledgements

This research is funded by the Singapore National Research Foundation IDM for Education grant NRF2008-IDM001-017.

References

1. Svestka, P., Overmars, M.: "Coordinated path planning for multiple robots. Robotics and Autonomous Systems, vol. 23, no. 3, 125--152 (1998)
2. Guo, Y., Parker, L.E.: A Distributed and Optimal Motion Planning Approach for Multiple Mobile Robots. Proceedings of the 2002 IEEE International Conference on Robotics & Automation, 2612 -- 2619 (2002)
3. Marchese, F.M.: Multiple Mobile Robots Path-Planning with MCA. International Conference on Autonomic and Autonomous Systems, 56--56 (2006)
4. Lee, B.H.: Collision-Free Motion Planning of Two Robots, IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS, vol.17, no.1, 21--32 (1987)
5. Ji, S.H., Choi, J.S., Lee, B.H.: A Computational Interactive Approach to Multi-agent Motion Planning, International Journal of Control, Automation, and Systems, vol.5,no.3, 295--306 (2007)
6. Koenig, S., Likhachev, M.: Incremental A* (2002)
7. Stentz, A.: Optimal and Efficient Path Planning for Partially-Known Environments", In Proceedings of IEEE International Conference on Robotics and Automation,vol.4, 3310--3317, May (1994)
8. Koenig, S.: Fast Replanning for Navigation in Unknown Terrain, IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION, vol. 21, no. 3, PP. 354--363 (2002)
9. Koenig, S., Likhachev, M.: Adaptive A*, In Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems, 1311--1312 (2005)
10. Wang, K.H., Botea, A.: Tractable Multi-Agent Path Planning on Grid Maps, Proceedings of the 21st International Joint conference on Artificial intelligence, 1870--1875 (2009)
11. Fiorini, P., Shiller, Z.: Motion planning in dynamic environment using velocity obstacles. International Journal of Robotics Research, vol. 17, no. 7, 760—772 July (1998)
12. Snape, J., van den Berg, J., Guy, S.J., Manocha, D.: Independent Navigation of Multiple Mobile Robots with Hybrid Reciprocal Velocity Obstacles. IEEE/RSJ International Conference on Intelligent Robots and Systems, 5917--5922 (2009)
13. Silver, D.: Cooperative Pathfinding. In *AIIDE*, 117--122 (2005)
14. Jansen, R., Sturtevant, N.: A New Approach to Cooperative Pathfinding. Proceedings of 7th International Conference on Autonomous Agents and Multiagent Systems, 1401--1404, (2008)

Designing the HRTeam Framework: Lessons Learned from a Rough-'N-Ready Human/Multi-Robot Team

Elizabeth Sklar[1,4], A. Tuna Ozgelen[1,4], J. Pablo Munoz[1], Joel Gonzalez[2], Mark Manashirov[1], Susan L. Epstein[3,4], and Simon Parsons[1,4]

[1] Brooklyn College, The City University of New York, USA
[2] City College, The City University of New York, USA
[3] Hunter College, The City University of New York, USA
[4] The Graduate Center, The City University of New York, USA
corresponding author: sklar@sci.brooklyn.cuny.edu

Abstract. In this workshop paper, we share the design and on-going implementation of our HRTeam framework, which is constructed to support multiple robots working with a human operator in a dynamic environment. The team is comprised of one human plus a heterogeneous set of inexpensive, limited-function robots. Although each individual robot has restricted mobility and sensing capabilities, together the team members constitute a multi-function, multi-robot facility. We describe low-level system architecture details and explain how we have integrated a popular robotic control and simulation environment into our framework to support application of multi-agent techniques in a hardware-based environment. We highlight lessons learned regarding the integration of multiple varying robot platforms into our system, from both hardware and software perspectives. Our aim is to generate discussion amongst multi-robot researchers concerning issues that are of particular interest and present particular difficulties to the Multi-Robot Systems community.

1 Introduction

This paper reports on the design and on-going implementation of a framework to support experimentation with mixed-initiative human/multi-robot teams. Our *HRTeam* framework is constructed to support multiple robots working with a human operator in a dynamic, real-time environment. The team is comprised of one human (the *operator*) plus a heterogeneous set of inexpensive, limited-function robots. Although each individual robot has restricted mobility and sensing capabilities, together the team members constitute a multi-function, multi-robot facility. The robots can be controlled directly by the human operator, or they can operate autonomously, without needing to wait for tele-operator input. Control of the robots is shared between the human operator and a software controller, and the locus of control can switch during run-time. The research questions we are investigating center around issues well-studied in the (virtual) *Multi-Agent Systems (MAS)* community: how to coordinate activity and allocate tasks to

team members in a real-time, dynamic environment; and how to integrate input from the human operator and find a balance between requiring too much direct control of many agents (robots), which may overwhelm the operator, and too little input from the operator, which may cause overall task completion to suffer. These issues present particular difficulties to the *Multi-Robot Systems (MRS)* community—and finding ways to address them is the focus of discussion here.

Our research is motivated by two related application areas: *urban search and rescue* [48, 67, 102] and *humanitarian de-mining* [38, 82]. In both instances, teams of robots are deployed to explore terrain that is potentially unsafe for humans and to locate targets of interest. In the first case, robots explore an enclosed space, such as a collapsed building, and search for human victims who may be physically trapped. The goal is to locate these victims and transmit their positions to human operators, so that human first responders can remove the victims to safety. In the second case, robots explore an open space, such as a field in a war zone, to search for anti-personnel mines that may be hidden from view. The goal is to locate these mines and transmit their positions to human operators, so that the mines can be disarmed and the area rendered safe for people to traverse.

Both application areas have a number of fundamental tasks in common. First, a robot must be able to explore a region (traverse and maneuver in the physical space) and *localize* (determine and track its position there). Second, a robot must be able to *recognize* objects of interest, using on-board sensors and possibly augmented intelligence to interpret sensor input. Third, a human operator must be able to communicate with the robots remotely and *strategize* so that the team can accomplish its overall task effectively. Ideally, in such a collaborative system, the human operator should not be overloaded with tasks, and the robots should not be idle. The team members should work together to accomplish the team's goal(s), taking advantage of members' individual abilities and strengths to complete tasks effectively and efficiently. Strategies to address these issues often stem from the MAS literature, where solutions have been implemented successfully in virtual environments—where agents can have perfect and often complete information. Unfortunately, in a multi-robot setting, most information is noisy, incomplete, and often out-of-date. So the challenge is to identify which MAS solutions can work in an MRS environment and adapt them accordingly.

As with any robotics research, a substantial effort must be made on the engineering side before any of these research questions can be investigated fully or satisfactorily. These efforts are more challenging in a multi-robot environment, simply because there are more hardware issues to contend with. Further, in a heterogeneous multi-robot environment, solving hardware problems for one (class of) robot does not necessarily solve the same problems for another (class of) robot; indeed, sometimes fixing one can break another. Finally, because we restrict our choice of hardware to inexpensive, limited-function robot platforms, additional constraints are presented. Note that this last aspect is not purely a function of budgetary realities, but rather part of our philosophy. There are always issues that arise when transferring results from a research environment to

a real-world setting. Often these issues are of a practical nature—e.g., network interference or uneven lighting conditions that did not occur in the lab suddenly confront a system deployed in a new venue—and can render elegant laboratory solutions useless outside the lab. By creating a less-than-ideal lab environment, we hope to be addressing some of these practical issues in our everyday setting.

In this workshop paper, we share the design of our HRTeam framework. We describe low-level system architecture details and explain how we have integrated a popular robotic control and simulation environment (Player/Stage [35, 97]) into our framework to support application of multi-agent techniques in a hardware-based environment. We highlight lessons learned regarding the integration of multiple varying robot platforms into our system, from both hardware and software perspectives. Our aim is to generate discussion amongst multi-robot researchers concerning issues that are of particular interest and present particular difficulties to the MRS community. Finally, we close with a brief summary and status report on our ongoing research investigations.

2 Related work

Research on Multi-Robot Systems, where more than one mobile robot is used, considers challenges faced by individual robots and how a robot team might help address these challenges. Areas of investigation include localization [23, 28, 71], mapping and exploration [19, 89], and strategies to manage wireless connectivity among robots [77]. With simultaneous localization and mapping (SLAM) [3, 39, 46, 94], additional information from several robots can simplify a problem and speed the solution that would have been provided by a single robot [28]; although multi-robot SLAM can also lead to inconsistency in position estimates [47, 58]. Other challenges for a multi-robot team are similar to those for one robot, complicated by the need to merge or expand single-robot solutions to incorporate other robots. Path planning [2, 11, 57, 93] is one well-studied example of this. Another example is the learning of controllers for teams of robots [69, 70], which is more complex than learning for individual robots.

The largest category of work on multi-robot systems, however, cannot be compared with work on single robots. Some tasks cannot be accomplished by one robot, such as the transport of an object too large for a single robot to move [24, 76, 91, 101]. Other issues, such as the dynamic allocation of tasks to robots [4, 5, 16, 60, 64, 87, 96], simply do not arise with a single robot. *Task allocation* is particularly challenging and has received substantial attention. The distribution of responsibilities among a group of individuals is a complex optimization problem. It is made more difficult because robot team requirements change over time [96], and because the abilities of individual robots to address particular tasks are conditioned on their changing locations. Heterogeneous robot teams, where each member has different capabilities, further complicates the optimization problem.

The task allocation literature for multi-robot teams includes a strong thread on the use of auctions [32, 54, 56, 83] and market-based mechanisms in general [20, 22, 33, 34, 103]. This work offers the various tasks for "sale" to robot team

members. Individual robots indicate how much they are willing to "pay" to obtain tasks, and tasks are allocated based on bids for them—typically to the robot that makes the best offer. For example, this approach has been used to organize robots for exploration tasks [50, 51, 104]. Areas to explore were offered "for sale," and robots bid based on their distance to the locations on offer. Allocation favored lower bids, and thereby tended to allocate areas closer to robots. The market was constructed, however, to ensure that robots did not remain idle when several robots were initially close to the same unexplored area. Another example is the use of simple auctions to allocate roles, and correspondingly, tasks associated with those roles, to robots in a multi-robot soccer team [30, 31]. Robots "bid" on roles based on their proximity to the ball and the goal. Roles changed in real time, as the game progressed. The ability both to consider individuals' changing abilities and to balance those against the performance of a team as a whole makes market-based approaches attractive.

Early work on multi-robot systems [14, 73] included *foraging*, a standard task that had robots systematically sweep an area as they searched for objects (e.g., [61, 59]). This has much in common with search and rescue, and with humanitarian demining—our target areas of application. Techniques have been developed to ensure that the entire boundary of a space is visited [99], that search finds a specific target [6, 7, 41, 42, 80, 84], that a mobile target is kept under constant observation [75, 68], and that a human-robot team can exchange search roles flexibly and appropriately [43].

Finally, given our focus on the deployment of many small robots, we should mention work on swarm robotics [88, 61]. Though recent work on swarms has looked at more focused task allocation to different robots [63] and on ensuring that the swarm spreads across different target sites [40], work in this area differs from ours in by being less deliberative, relying on numbers and randomness to get coverage rather than thoughtful deployment of resources.

Human-Robot Interaction (HRI) supports collaborative activities by humans and robots to achieve shared goals. Typical HRI research concentrates on the development of software and/or hardware to facilitate a wide range of tasks. These include robots maneuvering in physical spaces, both those designed for humans (e.g., [53]) or unfit for humans (e.g., [66]); people programming complex robots (e.g., [81]) or different types of simple robots (e.g., [8]); robots cooperating with human partners (e.g., [12, 25, 86, 98, 100]) and with other robots (e.g., [21, 55, 62, 92]); and user interfaces for communicating with robots (e.g., [49, 79]). Deployed HRI applications include cleaning [78], helping the elderly [95], assisting first responders in search and rescue tasks [17], demining in military settings [29], and teaching [52].

There are three main categories of control architectures for human-robot systems [37]: *fully autonomous*, where robots make decisions and control their actions on their own; *directly controlled*, where robots are driven by human operators; and *mixed-initiative* [15, 45], where robots share decision making with human users. Mixed-initiative systems reflect recent trends within the HRI community toward *socially intelligent* interfaces [9, 10, 26, 18] in which the aim is

for robots and humans to respond to each other naturally. We highlight several mixed initiative approaches here. *Adjustable autonomy* in a human-robot system permits dynamic transfer of control from human to robot and vice versa (e.g., [36, 85]). *Collaborative control* offers a dialog-based architecture in which decisions are "discussed" and made in real-time (e.g., [27]). Other examples of mixed-initiative systems include an affect-based architecture [1], and statistical techniques to infer missing information in human-robot communication [44].

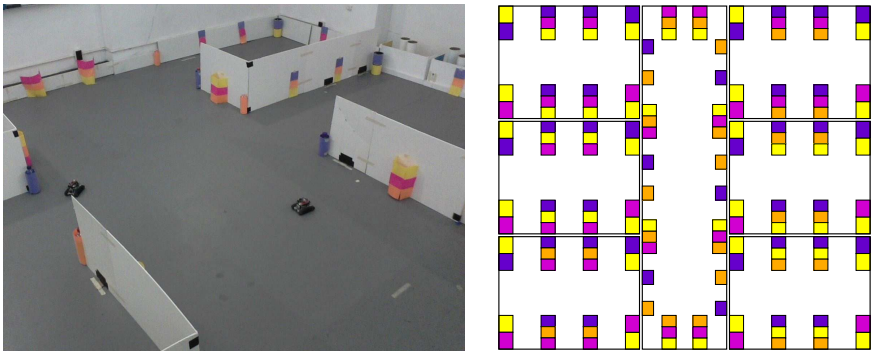
We see our work as fitting into the area of adjustable autonomy. Our main research goals are to establish how best to transition control of a robot from human to robot and, especially, vice-versa (with a large robot team the human operator must be sued sparingly to avoid overload), and to investigate how best to coordinate the robot team when it is operating autonomously. With regard to this latter aim, we plan to test a range of coordination techniques from the multiagent systems literature, taking techniques that have been polished theoretically and in simulation, and seeing how they perform in the rough and ready world of robotics.

3 Physical Environment

Our test arena, shown in Figure 1a, is a physical environment that is divided into seven regions: six rooms and a hallway. Each region contains color-coded landmarks to guide the robots using vision-based localization. Figure 1b contains a schematic of the landmarks[5]. These are composed of vertically-aligned markers with stacked bands of one, two, or three colors. The entire color palette consists of four colors: yellow, pink, orange, and purple. On the northeast corner of each of the six rooms, a "purple-over-yellow" landmark is placed. The northwest corner contains a "yellow-over-purple" landmark; the southwest corner contains "yellow-over-pink"; and the southeast corner contains "pink-over-yellow". Inside each room, a unique 3-color marker distinguishes that room from the others; each of the room markers include a purple band. In the hallway, a set of 3-color markers (without purple bands), using four unique color band permutations, mark the north, west, south and east walls of the hallway. Inside the hallway, the entrance to each room is marked with a single-colored purple landmark on the right side of the "doorway", and an orange landmark on the left.

The lighting conditions in the arena vary from one room to another. This means that it is not possible to have a single, non-overlapping color map with which to calibrate the colored landmarks; e.g., the orange and yellow color ranges tend to bleed together in some parts of the arena. The process of identifying landmarks involves first capturing images with robots' cameras and analyzing the images for "blobs" of color, then the color blobs are matched with landmarks

[5] Note that the landmarks are a proxy for more sophisticated vision processing that would allow us to recognise unique features of the test arena. Using the landmarks allows us to test other aspects of our environment as we develop this vision capability. The large number of landmarks are required because of the fixed cameras used by most of the robot platforms.

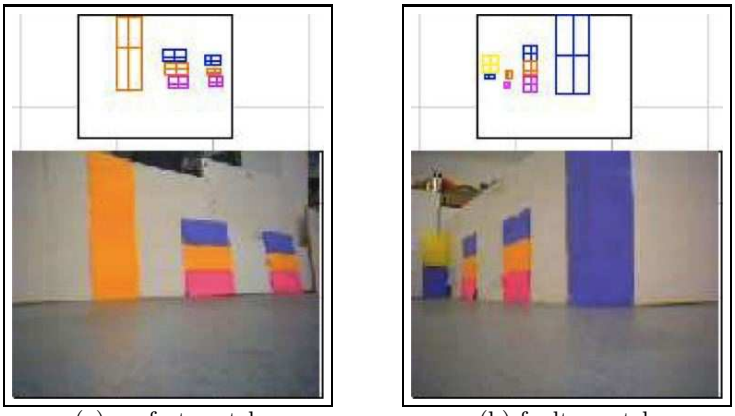


(a) view of test arena (b) schematic of arena landmarks

Fig. 1. Robots' physical environment

from a dictionary of known objects. An example is shown in Figure 2. The figure on the left shows a perfect match between a robot's image and the markers that were identified. The figure on the right, however, has missed one blob of color (a purple band at the top of the second marker from the left), which makes it a problem to identify that marker correctly. Some of our research involves applying machine learning techniques to a participatory human/robot process in which the system learns a reliability metric for the images with help from the human operator. While the system can recognize that problems exist without the help of a human, having a human in the loop can speed the learning process. In the example shown in Figure 2b, the system can quickly detect a problem with the image simply because there are no markers in its dictionary that consist of only an orange band on top of a pink band.

As mentioned earlier, the robots on our team are inexpensive, limited-function platforms. These are pictured in Figure 3. We have been experimenting with five different platforms, spanning a range of sensing and locomotion capabilities and communication technologies. Table 1 lists the hardware differences. Only the AIBO has a powerful enough on-board processor to function as a stand-alone platform. The Create is mounted with a Hokuyo URG-04LX Scanning Laser Rangefinder and a Dell laptop that communicates, via USB, to the robot and the laser device. The Fribbler and the SRV-1 have minimal on-board memory and so are controlled by off-board laptops with dedicated communication channels. The NXT has limited on-board memory and processing capabilities—more than the Fribbler and SRV-1, but substantially less than the AIBO. Currently, we operate the NXT in the same way as the Fribbler and SRV-1: via off-board laptop with dedicated communication channel. All of the devices listed as "wireless" in Table 1 use 802.11. The SRV-1 platform was originally built using an XBee radio device. Newer "Blackfin" models are now available with 802.11. We have found that the XBee radio suffers greatly from interference with the 802.11, particularly when the two types of communicating devices are in close proxim-



(a) perfect match (b) faulty match

Fig. 2. Landmark identification

ity with one another. We have also found that we must make judicious use of 802.11 communication, otherwise it is quite easy to flood our local network—for example, when multiple robots try to transmit high-frame-rate video feeds.

(a) AIBO (b) Create (c) Fribbler (d) NXT (e) SRV-1

Fig. 3. Robot gallery

The human team member—the *operator*—is positioned physically away from the test arena so that her only view of the space is via camera images sent to her by the robots. The operator' interface is shown in Figure 4. The right half of the window shows a bird's eye view that indicates the position of each robot in the arena. The system uses vision-based localization (albeit somewhat unreliable due to the landmark identification problems mentioned above) and a particle filter to estimate the (x, y) location and orientation of each robot in the arena. The origin $(0, 0)$ of the robot's environment is defined as the middle of the arena (in the middle of the hallway), with positive x moving north and positive y moving east. Orientation (θ) is measured in degrees, with 0° facing east, 90° facing north, 180° facing west and 270° facing east. Returning to the operator interface in Figure 4, the upper left region contains a "robot's eye view" of the environment. The lower left region contains manual controls that the human can use to drive one robot at a time. Depending on the experimental conditions,

platform	*sensing*	*locomotion*	*communication*
AIBO ERS-7 (`www.sonyaibo.net`)	camera	legged	wireless
Create (`www.irobot.com`) (with external laser device mounted on top)	laser	wheeled	wireless
"Fribbler" (= Scribbler: `www.parallax.com` + Fluke: `www.roboteducation.org`)	camera	wheeled	bluetooth
Mindstorms NXT (`mindstorms.lego.com`)	sonar	wheeled	bluetooth
SRV-1/ARM (`www.surveyor.com`)	camera	tracked	radio/wireless

Table 1. Robot platform capabilities

the other robots are either idle when the human operator is not driving them (primarily this mode is used for taking experimental control measurements), or they are operating autonomously (most of the time).

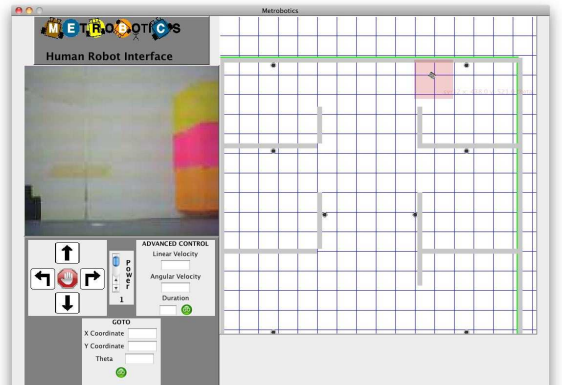


Fig. 4. Operator interface

4 Software Framework

Our software system employs a multi-layer architecture that combines multiple clients and multiple types of servers. A high-level overview of the system is shown in Figure 5. In the *agent layer*, the Central Server acts as the communication hub for all the components in the system, and is discussed separately, below. The Intelligence Engine supports system learning, task allocation and multi-robot coordination, as well as collaborative decision making with the human operator. This component is not discussed in detail here; for further description, see [90]. The Database Manager logs system activity. It collects experimental data and maintains a database of known objects and other shared data structures (e.g.,

a map). The Object Recognizer identifies objects in the environment, by using the Open Source Computer Vision Library (OpenCV) [72] to perform feature extraction on robot imagery. Colored "blobs" are segmented and Canny edge detection [13] is applied to outline object shapes. A Naïve Bayes classifier [65] matches input images with previously tagged images from our database. The Operator Interface comprises the *human layer*, and was described in the previous section. The *robot layer* is detailed below. Then we will return to discussion of the overall system architecture and focus on multi-server/multi-client aspects.

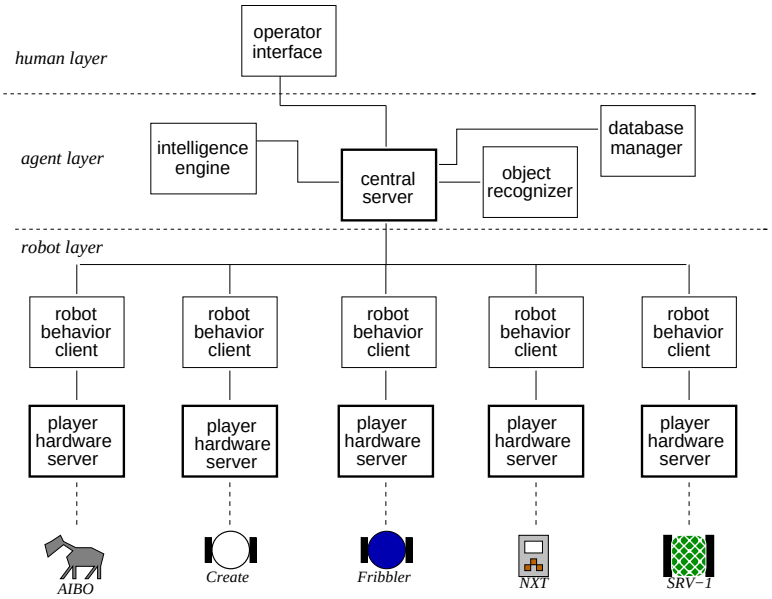


Fig. 5. The HRTeam system architecture. Each box is a process. The boxes outlined with thick borders are servers; the remaining boxes are clients.

4.1 Robot Layer

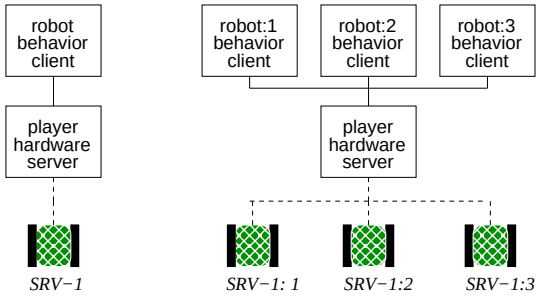
The robot layer is built on *Player/Stage*[6] [35, 97], a popular robot control and simulation environment. Player/Stage provides an open-source, modular client/-server framework for robotics programming that allows for unified control of multiple robot platforms. An abstract client class contains high-level robot control functionalities or behaviors (e.g., wall-following) and is extended to support the needs of a particular application. Hardware-specific drivers, implemented as servers, contain low-level sensor and actuator control functions (e.g., move forward, capture an image, etc.). In our framework, the client implements robot

[6] `http://playerstage.sourceforge.net/`

behaviors, such as perception, including some image processing, and low-level decision making for each robot. A platform-specific server, or driver, communicates directly with the robot hardware.

The advantage of using Player/Stage is that, for each hardware platform we initiate onto our team, we only need to write one driver for that platform; and for each set of robot behaviors, we only need to write one behavior client. We have adapted Player drivers for each of the five different robot platforms that are listed in Table 1. We have written one behavior client program that can control each of the robots in our system. A different behavior client process is instantiated for each robot, as explained below.

The use of Player/Stage presents an interesting system architecture question. It is possible to implement a system having a one-to-one correspondence between the robot behavior module, the hardware driver, and the physical robot (see Figure 6a). There may also be a one-to-many correspondence between the hardware driver and multiple physical robots (Figure 6b). In order to maintain individuality amongst robot team members, we always employ a one-to-one correspondence between robot behavior modules and physical robots.



(a) one robot, one hardware abstraction, one agent controller

(b) three robots one hardware abstraction, three agent controllers

Fig. 6. Player Framework

4.2 Central Server

An unusual aspect of our architecture is that there are multiple servers: the Central Server and a Player hardware server for each (class of) robot platform. The Central Server must be started up first, because it handles message passing and bookkeeping for the whole system. The Central Server keeps track of the instantiated components and of the robots that are connected to the system at any given time. All inter-process communication is handled asynchronously. All components have their own state machines; an example for the robot behavior

client is shown in Figure 7. The components are designed to handle unexpected messages, as well as normal operations. The Central Server is written in C++ and establishes a server socket that binds to a particular host name and port number, establishing a point of communication for the entire system; then it listens for clients to connect. All of the processes in the system are multi-threaded, in order to handle communication asynchronously, independent of the process's primary functionality. For example, the Central Server creates a thread for each new client that connects to it, to allow asynchronous processing of messages between the Central Server and each client.

Table 2 contains sample messages that are passed between the Central Server (CS) and a robot behavior client (RB). Table 3 contains sample messages that are passed between the Central Server (CS) and the operator interface (OI).

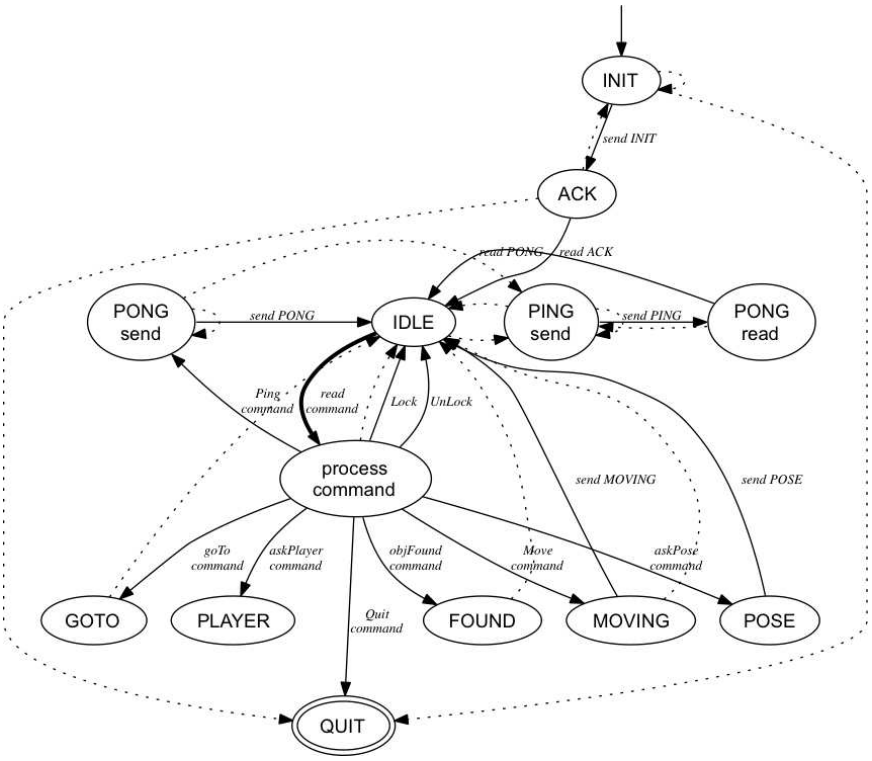


Fig. 7. State diagram for Robot Behavior client

RB $\rightarrow$ CS: **init** $< uds >$ An RB sends this command to CS when it first logs in. $< uds >$ stands for "unified data structure" that contains a string that identifies the type of robot (e.g., "aibo"); a string containing the name of the robot (e.g., "rosie"); a unique numeric identifier, which is treated like a session_id in the system and is determined by the server when a client first connects; and a list of the services that this robot provides, such as: "position2d", "camera", "distance", "contact".
RB $\leftarrow$ CS: **ack** $< id >$ Upon receiving the **ack** command, the RB will set the value of the id field in its local copy of the unified data structure. $< id >$ is a unique identifier (integer) that that CS sends to the RB to acknowledge its registration. It returns a unique ID number that the robot will need to use for all further communication, to identify itself in the system. This value is treated like a session_id.
RB $\leftarrow$ CS: **askpose** The CS sends an "askpose" message to the RB requesting information about its pose (location and heading).
RB $\rightarrow$ CS: **pose** $< x >< y >< \theta > [< \rho >]$ The RB sends back its (x, y) location and θ heading (degrees) within its environment. The last argument is confidence value, $0 \leq \rho \leq 1$, indicating the RB 's confidence in its location.
RB $\rightarrow$ CS: **broadcast found** $< color >$ The RB sends this message whenever it finds an object of interest. CS strips "broadcast" part of the message and passes "found" $< color >$ message to all connected clients, both robots and the GUI.
RB $\leftarrow$ CS:**move** $< id >< x - velocity >< y - velocity >< angular - velocity >$ The CS sends a "move" message to the robot requesting it to set its x, y and angular speeds to $< x - velocity >, < y - velocity >$ and $< angular - velocity >$. If the $< id >$ of the message does not match robots own id, the message is disregarded.
RB $\leftarrow$ CS: **goto** $< id >< map - x >< map - y >$ The CS sends a "goto" message to the robot requesting it to move to a particular location, $(< map - x >, < map - y >)$, on the field. If the $< id >$ of the message does not match robots own id, the message is disregarded.
RB $\rightarrow$ CS: **moving** The RB sends back an acknowledgment that it has received the "move" command and is executing the command. GUI does not need this confirmation, it will be used for data logging.

Table 2. Sample commands that flow between Central Server (CS) and Robot Behavior client (RB).

OI $\rightarrow$ CS **init** *uds* where *uds* is defined as in Table 2.
OI $\leftarrow$ CS **ack** $< id >$ where $< id >$ is defined as in Table 2.
OI $\rightarrow$ CS **askpose** $< id >$ The OI sends "askpose" to CS to retreive the (x, y) location and θ heading of a particular robot, by attaching its $< id >$. To retrieve pose information for all robots, $< id >$ is set to -1.
OI $\leftarrow$ CS **pose** $< num\_robots > [< robot\_pose\_info >]$ The CS sends back the number of robot pose information the message contains. Each pose information consists of robot's id, (x, y) location, θ heading (degrees) and confidence value, $0 \leq \rho \leq 1$, indicating the GUI's confidence in its location.
OI $\rightarrow$ CS **askplayer** $< id >$ The GUI requests for player CS information that a particular robot is using. This information is needed to communicate directly with player CS to receive camera feed of the robot.
OI $\leftarrow$ CS **player** $< id > < player - ip > < player - port >$ The CS sends back the player server information, $< player - ip >, < player - port >$ of the robot with id=$< id >$.
OI $\leftarrow$ CS **found** The GUI receives this message from the CS when a robot finds the object that the team is searching for. Currently it is used to stop the clock for the experiment.
OI $\rightarrow$ CS **move** $< id > < x - velocity > < y - velocity > < angular - velocity >$ The OI sends a "move" message to the CS to pass it to robot with id=$< id >$, requesting it to set its x, y and angular speeds to $< x - velocity >, < y - velocity >$ and $< angular - velocity >$.
OI $\rightarrow$ CS **goto** $< id > < x > < y >$ The GUI sends a "goto" message to the CS to pass it to robot with id=$< id >$, requesting it to move to a particular location, (x, y), on the field.
OI $\rightarrow$ CS **lock** $< id >$ The GUI sends a "lock" message to the GUI, requesting to take control of the robot with id=$< id >$.
OI $\rightarrow$ CS **unlock** $< id >$ The GUI sends an "unlock" message to the GUI, requesting to release control of the robot with id=$< id >$.

Table 3. Sample commands that flow between Central Server (CS) and Operator Interface (OI).

5 Lessons Learned

In this section, we describe some of the main lessons that we have learned from our work so far, largely in the form of problems we have had to struggle with.

The main problem that we have faced has been getting the robots to localize while engaged in their exploration tasks. As mentioned above, we are using vision-based localization. The underlying approach is a standard particle filter, and the particular implementation we are using is one we developed for our Aibo-based RoboCup soccer team [74]. The main difference, as far as vision is concerned, between the Aibo, the Surveyor and the Fribbler—the robots that we have been using most often in our experiments—is that the last two have fixed cameras. It turns out that this has a large effect on their ability to see landmarks. When the robots start up, and move to maximize the number of landmarks they see, they localize relatively quickly. However, when they are carrying out their assigned task, which typically involves navigating through the test arena to explore a designated room, they often go for several minutes without seeing more than a single landmark clearly enough to recognize it. As a result, they rapidly become unsure of their location and have to spend time specifically relocalizing. This is in contrast to the Aibo, which can track its position quite effectively even with many fewer landmarks in the environment.

A subsidiary problem has been the wireless control of the robots. Several of our robots do not have sufficient on-board processing to run a controller (as mentioned in the previous section). Rather, they are controlled over a wireless connection, either 802.11, radio or Bluetooth. The first issue with wireless was mentioned above: 802.11 and radio interfere, and so if we are using the two modes of communication, we have to keep the robots physically separate. This, of course, adds another layer of complexity to the control of the team. However, even if all the robots on the team use 802.11, there can still be issues. Even in the lab, where we have excellent wireless coverage, and little interference from other networks, it is easy to overload the bandwidth. With off-board processing, it is tempting to pull video off the robots at full-speed, but with more than a couple of robots, this floods the network. As a result we throttle the video feeds, though this naturally limits the use that both the robots and the human operator can make of the feeds. On the robot side, of course, this only makes the localization problem worse.

Finally, a more positive note. Despite the problems noted above, which are problems that would go away if we used robots with multiple camera angles and more on-board processing[7], we have found our experience of using sub-$1000 robots to be a positive one. With the Player drivers we have developed, it is possible to use such robots for serious research purposes, and their cost means that with even a modest budget, it is possible to deploy a fleet of robots.

[7] Our future work will explore building cheap custom robots with Gumstix or Arduino controllers and multiple cameras to explore this option.

6 Summary

We have described the design and on-going implementation of our HRTeam framework, which we have developed to support studies in human/robot teamwork. Our philosophy has been to deploy multiple low-cost, limited-function robots, to force the necessity of collaboration in order to complete tasks. Our rough-'n-ready laboratory environment offers special challenges, ranging from lighting variations and network interference to managing a suite of software components to control a heterogenous collection of hardware platforms. Several research activities are underway using the HRTeam framework. First, we are investigating ways to coordinate activity and allocate tasks to team members in a real-time, dynamic environment, concentrating on market-based mechanisms. Second, we are examining ways to incorporate real-time, dynamic input from the human operator into the multi-robot system. Finally, we are developing a participatory human/machine learning process to obtain reliability measures for the imaging data used in the localization process.

Acknowledgments

This work was supported by the National Science Foundation under #CNS-0851901 and #CNS-0520989, and by CUNY Collaborative Incentive Research Grant #1642.

References

1. Adams, J.A., Rani, P., Sarkar, N.: Mixed initiative interaction and robotic sys. In: Wkshp on Supervisory Control of Learning and Adaptive Sys, Tech Rept WS-04-10 (2004)
2. Alami, R., Robert, F., Ingrand, F., Suzuki, S.: Multi-robot cooperation through incremental plan-merging. In: Proc of the IEEE Conference on Robotics and Automation (1995)
3. Andrade-Cetto, J., Vidal-Calleja, T., Sanfeliu, A.: Multirobot C-SLAM: Simultaneous localization, control and mapping. In: Proc of the ICRA Workshop on Network Robot Systems (2005)
4. Atay, N., Bayazit, B.: Emergent task allocation for mobile robots. In: Proc of Robotics: Science and Systems Conference (2007)
5. Barlow, G., Henderson, T., Nelson, A., Grant, E.: Dynamic leadership protocol for S-Nets. In: Proc of the IEEE Intl Conference on Robotics and Automation (2004)
6. Bhattacharya, S., Candido, S., Hutchinson, S.: Motion strategies for surveillance. In: Proc of Robotics: Science and Systems Conference (2007)
7. Bhattacharya, S., Hutchinson, S.: Approximation schemes for two-player pursuit evasion games with visibility constraints. In: Proc of Robotics: Science and Systems Conference (2008)
8. Blank, D., Kumar, D., Meeden, L., Yanco, H.: Pyro: A python-based versatile programming environment for teaching robotics. ACM Journal on Educational Resources in Computing (JERIC) (2005)

9. Breazeal, C.: Toward sociable robotss. Robotics and Autonomous Systems 42 (2003)
10. Breazeal, C., Scassellati, B.: Robots that imitate humans. TRENDS in Cognitive Sciences 6(11) (2002)
11. Brumitt, B.L., Stentz, A.: Dynamic mission planning for multiple mobile robots. In: Proc of the IEEE Intl Conference on Robotics and Automation (1996)
12. Burke, J.L., Murphy, R.R.: Human-robot interaction in usar technical search: Two heads are better than one. In: Intl Workshop on Robot and Human Interactive Comm (2004)
13. Canny, J.: A computational approach to edge detection. IEEE Transactions on Pattern Analysis and Machine Intelligence 8 (1986)
14. Cao, Y.U., Fukunaga, A.S., Kahng, A.B.: Cooperative mobile robotics: antecedents and directions. Autonomous Robots 4(1) (1997)
15. Carbonell, J.R.: Mixed-initiative man-computer instructional dialogues. Tech. Rep. 1971, Bolt Beranek and Newman, Inc (1971)
16. Chaimowicz, L., Kumar, V., Campos, F.: A paradigm for dynamic coordination of multiple robots. Autonomous Robots 17(1) (2004)
17. Crasar: access oct 20. `crasar.org` (2010)
18. Dautenhahn, K.: A Paradigm Shift in Artificial Intelligence: Why Social Intelligence Matters in the Design and Development of Robots with Human-Like Intelligence. 50 Years of AI Journal LNCS 4850 (2007)
19. Dedeoglu, G., Sukhatme, G.S.: Landmark-based matching algorithm for cooperative mapping by autonomous robots. In: Distributed Autonomous Robotic Systems 4. Springer-Verlag (2000)
20. Dias, M.B., Zlot, R., Kalra, N., Stentz, A.: Market-based multirobot coordination: A survey and analysis. Tech. Rep. CMU-RI-TR-05-13, Carnegie Mellon University (2005)
21. Dias, M.B., Zlot, R., Zinck, M., Gonzalez, J.P., Stentz, A.: A versatile implementation of the traderbots approach for multirobot coordination. In: Proc of Intelligent Automated Systems (2004)
22. Dias, M.B., Zlot, R., Zinck, M., Stentz, A.: Robust multirobot coordination in dynamic environments. In: Proc of the IEEE Intl Conference on Robotics and Automation (2004)
23. Fenwick, J.W., Newman, P.M., Leonard, J.J.: Cooperative concurrent mapping and localization. In: Proc of the IEEE Conference on Robotics and Automation (2002)
24. Fink, J., Michael, N., Kumar, V.: Composition of vector fields for multi-robot manipulation via caging. In: Proc of Robotics: Science and Systems Conference (2007)
25. Finzi, A., Orlandini, A.: Human-Robot Interaction Through Mixed-Initiative Planning for Rescue and Search Rovers. Advances in Artificial Intelligence (AIIA) LNCS 3673 (2005)
26. Fong, T., Nourbakhsh, I., Dautenhahn, K.: A survey of socially interactive robots. Robotics and Autonomous Systems 42 (2003)
27. Fong, T., Thorpe, C., Baur, C.: Multi-Robot Remote Driving With Collaborative Control. IEEE Transactions on Industrial Electronics 50(4) (2003)
28. Fox, D., Burgard, W., Kruppa, H., Thrun, S.: A probabilistic approach to collaborative multi-robot localization. Autonomous Robots 8(3) (2000)
29. Freese, M., Matsuzawa, T., Oishi, Y., Debenest, P., Takita, K., Fukushima, E.F., Hirose, S.: Robotics-assisted demining with gryphon. Advanced Robotics 21(15) (2007)

30. Frias-Martinez, V., Sklar, E.I.: A framework for exploring role assignment in real-time, multiagent teams. In: The second European Workshop on Multi-Agent Systems (EUMAS) (2004)
31. Frias-Martinez, V., Sklar, E.I., Parsons, S.: Exploring auction mechanisms for role assignment in teams of autonomous robots. In: Proc of the Eighth RoboCup Intl Symposium (2004)
32. Gerkey, B.P., Mataríc, M.J.: Sold!: Auction methods for multi-robot control. IEEE Transactions on Robotics and Automation Special Issue on Multi-Robot Systems 18(5) (2002)
33. Gerkey, B.P., Mataríc, M.J.: Multi-robot task allocation: Analyzing the complexity and optimality of key architectures. In: Proc of the IEEE Intl Conference on Robotics and Automation (2003)
34. Gerkey, B.P., Mataríc, M.J.: A formal analysis and taxonomy c of task allocation in multi-robot systems. Intl Journal of Robotics Research 23(9) (2004)
35. Gerkey, B., Vaughan, R.T., Howard, A.: The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. In: Proc of the 11th Intl Conference on Advanced Robotics (ICAR) (2003)
36. Goodrich, M.A., Olsen, D.R., Crandall, J.W., Palmer, T.J.: Experiments in Adjustable Autonomy. In: IJCAI Workshop on Autonomy, Delegation, and Control: Interaction with Autonomous Agents (2001)
37. Goodrich, M.A., Schultz, A.C.: Human-robot interaction: a survey. Foundations and Trends in Human-Computer Interaction 1(3) (2007)
38. Habib, M.K.: Humanitarian Demining: Reality and the Challenge of Technology. Intl Journal of Advanced Robotic Systems 4(2) (2007)
39. Hajjdiab, H., Laganiere, R.: Vision-based multi-robot simultaneous localization and mapping. In: Procedings of the Canadian Conference on Computer and Robot Vision (2004)
40. Halász, A., Hsieh, M.A., Berman, S., Kumar, V.: Dynamic redistribution of a swarm of robots among multiple sites. In: IEEE/RSJ Intl Conference on Intelligent Robots and Systems (2005)
41. Hollinger, G., Singh, S.: Proofs and experiments in scalable, near-optimal search by multiple robots. In: Proc of Robotics: Science and Systems Conference (2008)
42. Hollinger, G., Singh, S., Djugash, J., Kehagias, A.: Efficient multi-robot search for a moving target. Intl Journal of Robotics Research 28(2) (2009)
43. Hollinger, G., Singh, S., Kehagias, A.: Efficient, guaranteed search with multi-agent teams. In: Proc of Robotics: Science and Systems Conference (2009)
44. Hong, J.H., Song, Y.S., Cho, S.B.: Mixed-initiative human-robot interaction using hierarchical bayesian networks. IEEE Transactions on Systems, Man and Cybernetics, Part A 37(6) (2007)
45. Horvitz, E.: Principles of mixed-initiative user interfaces. In: Proc of the Computer-Human Interaction Conference (CHI) (1999)
46. Howard, A.: Multi-robot simultaneous localization and mapping using particle filters. Journal of Robotics Research 25(12) (2006)
47. Huang, G.P., Trawny, N., Mourikis, A.I., Roumeliotis, S.I.: On the consistency of multi-robot cooperative localization. In: Proc of Robotics: Science and Systems Conference (2009)
48. Jacoff, A., Messina, E., Evans, J.: A standard test course for urban search and rescue robots. In: Proc of the Performance Metrics for Intelligent Systems Workshop (PerMIS) (2000)

49. Kaber, D.B., Wright, M.C., Sheik-Nainar, M.A.: Multimodal interface design for adaptive automation of a human-robot system. Intl Journal of Human-Computer Studies 64 (2006)
50. Kalra, N.: A market-based framework for tightly-coupled planned coordination in multirobot teams. Ph.D. thesis, The Robotics Institute, Carnegie Mellon University (2007)
51. Kalra, N., Ferguson, D., Stentz, A.: Hoplites: A market-based framework for complex tight coordination in multi-robot teams. In: Proc of the IEEE Intl Conference on Robotics and Automation (2005)
52. Kanda, T., Hirano, T., Eaton, D.: Interactive robots as social partners and peer tutors for children: A field trial. Human-Computer Interaction 19 (2004)
53. Kang, S., Lee, W., Kim, M., Shin, K.: ROBHAZ-rescue: rough-terrain negotiable teleoperated mobile robot for rescue mission. In: IEEE Intl Workshop on Safety, Security and Rescue Robotics (2005)
54. Koenig, S., Keskinocak, P., Tovey, C.: Progress on agent coordination with cooperative auctions. In: Proc of the AAAI Conference on Artificial Intelligence (2010)
55. Lagoudakis, M., Berhault, M., Koenig, S., Keskinocak, P., Kelywegt, A.: Simple auctions with performance guarantees for multi-robot task allocation. In: Proc of Int'l Conference on Intelligent Robotics and Systems (IROS) (2004)
56. Lagoudakis, M., Markakis, V., Kempe, D., Keskinocak, P., Koenig, S., Kleywegt, A., Tovey, C., Meyerson, A., Jain, S.: Auction-based multi-robot routing. In: Proc of Robotics: Science and Systems Conference (2005)
57. LaValle, S.M., Hutchinson, S.A.: Optimal motion planning for multiple robots having independent goals. IEEE Transactions on Robotics and Automation 14(6) (1998)
58. Lázaro, M.T., Castellanos, J.A.: Localization of probabilistic robot formations in SLAM. In: Proc of the IEEE Intl Conference on Robotics and Automation (2010)
59. Lerman, K., Galstyan, A.: Mathematical model of foraging in a group of robots: Effect of interference. Autonomous Robots 13(2) (2002)
60. Lerman, K., Jones, C.V., Galstyan, A., Matarıć, M.J.: Analysis of dynamic task allocation in multi-robot systems. Intl Journal of Robotics Research 25(3) (2006)
61. Liu, W., Winfield, A.F.T., Sa, J., Chen, J., Dou, L.: Towards energy optimization: Emergent task allocation in a swarm of foraging robots. Adaptive Behavior 15(3) (2007)
62. Mataric, M., Sukhatme, G., Ostergaard, E.: Multi-robot task allocation in uncertain environments. Autonomous Robots (2003)
63. McLurkin, J., Yamins, D.: Dynamic task assignment in robot swarms. In: Proc of Robotics: Science and Systems Conference (2005)
64. Michael, N., Zavlanos, M.M., Kumar, V., Pappas, G.J.: Distributed multi-robot task assignment and formation control. In: Proc of the IEEE Intl Conference on Robotics and Automation (2008)
65. Mitchell, T.M.: Machine Learning. McGraw Hill (2005)
66. Murphy, R.R.: Marsupial and shape-shifting robots for urban search and rescue. IEEE Intelligent Systems 15(2) (2000)
67. Murphy, R.R., Casper, J., Micire, M.: Potential tasks and research issues for mobile robots in robocup rescue. In: Robot Soccer World Cup IV, LNAI 2019. Springer Verlag (2001)
68. Murrieta-Cid, R., Muppirala, T., Sarmiento, A., Bhattacharya, S., Hutchinson, S.: Surveillance strategies for a pursuer with finite sensor range. Intl Journal of Robotics Research (2007)

69. Nelson, A., Grant, E., Barlow, G., Henderson, T.: A colony of robots using vision sensing and evolved neural controllers. In: IEEE/RSJ Intl Conference on Intelligent Robots and Systems (2003)
70. Nelson, A., Grant, E., Henderson, T.: Evolution of neural controllers for competitive game playing with teams of mobile robots. Robotics and Autonomous Systems 46 (2004)
71. Nerurkar, E.D., Roumeliotis, S.I., Martinelli, A.: Distributed maximum a posteriori estimation for multi-robot cooperative localization. In: Proc of the IEEE Intl Conference on Robotics and Automation (2009)
72. Open Source Computer Vision Library (OpenCV). `http://sourceforge.net/projects/opencvlibrary/`
73. Ota, J.: Multi-agent robot systems as distributed autonomous systems. Advanced Engineering Informatics 20 (2006)
74. Ozgelen, A.T., Kammet, J., Marcinkiewicz, M., Parsons, S., **Elizabeth Sklar**: The 2007 MetroBots Four-legged League Team Description Paper. In: RoboCup 2007: Robot Soccer World Cup XI (2007)
75. Parker, L.E.: Cooperative robotics for multi-target observation. Intelligent Automation and Soft Computing 5(1) (1999)
76. Pereira, G.A.S., Campos, M.F.M., Kumar, V.: Decentralized algorithms for multi-robot manipulation via caging. Intl Journal of Robotics Research (2004)
77. Rooker, M.N., Birk, A.: Multi-robot exploration under the constraints of wireless networking. Control Engineering Practice 15 (2007)
78. Roomba: access oct 20. `www.irobot.com` (2010)
79. Rooy, D., Ritter, F., St Amant, R.: Using a simulated user to explore human-robot interfaces. In: ACT-R Workshop (2002)
80. Royset, J., Sato, H.: Route optimization for multiple searchers. Tech. rep., Naval Postgraduate School, Monterey, CA (2009), `http://faculty.nps.edu/joroyset/docs/RoysetSato_MultiSearcher.pdf`
81. Sandini, G., Metta, G., Vernon, D.: The iCub Cognitive Humanoid Robot: An Open-System Research Platform for Enactive Cognition. 50 Years of AI Journal LNCS 4850 (2007)
82. Santana, P.F., Barata, J., Correia, L.: Sustainable Robots for Humanitarian Demining. Intl Journal of Advanced Robotic Systems 4(2) (2007)
83. Sariel, S., Balch, T.: Efficient bids on task allocation for multi-robot exploration. In: Proc of the Nineteenth Intl Florida Artificial Intelligence Research Society Conference (2006)
84. Sarmiento, A., Murrieta-Cid, R., Hutchinson, S.: A multi-robot strategy for rapidly searching a polygonal environment. In: Proc of the 9th Ibero-American Conference on Artificial Intelligence (2004)
85. Scerri, P., Pynadath, D.V., Tambe, M.: Why the elf acted autonomously: Towards a theory of adjustable autonomy. In: Proc of the Intl Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS) (2002)
86. Severinson-Eklundh, K., Green, A., Hüttenrauch, H.: Social and collaborative aspects of interaction with a service robot. Tech. Rep. IPLab-208, Royal Institute of Technology, Stockholm (January 2003)
87. Shah, K., Meng, Y.: Communication-efficient dynamic task scheduling for heterogeneous multi-robot systems. In: Proc of the IEEE Intl Symposium on Computational Intelligence in Robotics and Automation (2007)
88. de Silva, V., Ghrist, R., Muhammad, A.: Blind swarms for coverage in 2-d. In: Proc of Robotics: Science and Systems Conference (2005)

89. Simmons, R., Apfelbaum, D., Burgard, W., Fox, D., Moor, M., Thrun, S., Younes, H.: Coordination for multi-robot exploration and mapping. In: Proc of the 17th National Conference on Artificial Intelligence (2000)
90. Sklar, E.I., Epstein, S.L., Parsons, S., Ozgelen, A.T., Munoz, J.P.: A framework in which robots and humans help each other. In: Proc of the AAAI Symposium Help Me Help You: Bridging the Gaps in Human-Agent Collaboration (2011)
91. Spletzer, J., Das, A.K., Fierro, R., Taylor, C.J., Kumar, V., Ostrowski, J.P.: Cooperative localization and control for multi-robot manipulation. In: Proc of the IEEE/RSJ Intl Conference on Intelligent Robots (2001)
92. Stone, P., Veloso, M.: Communication in domains with unreliable, single-channel, low-bandwidth communication. In: Proc of the Intl Joint Conference on Artificial Intelligence (IJCAI) (1998)
93. Svestka, P., Overmars, M.H.: Coordinated path planning for multiple robots. Robotics and Autonomous Systems 23 (1998)
94. Thrun, S., Liu, Y., Koller, D., Ng, A.Y., Ghahramani, Z., Durrant-Whyte, H.: Simultaneous localization and mapping with sparse extended information filters. Journal of Robotics Research (2004)
95. Tyrer, H., Alwan, M., Demiris, G., He, Z., Keller, J., Skubic, M., Rantz, M.: Technology for successful aging. In: Proc of Engineering in Medicine and Biology Society (2006)
96. Vail, D., Veloso, M.: Dynamic multi-robot coordination. In: Schultz, A.C., Parker, L.E., Schneider, F.E. (eds.) Multi-Robot Systems: From Swarms to Intelligent Automata. Kluwer (2003)
97. Vaughan, R.T., Gerkey, B.: Really Reusable Robot Code and the Player/Stage Project. In: Brugali, D. (ed.) Software Engineering for Experimental Robotics. Springer (2007)
98. Wegner, R., Anderson, J.: Agent-based support for balancing teleoperation and autonomy in urban search and rescue. Intl Journal of Robotics and Automation 21(2) (2006)
99. Williams, K., Burdick, J.: Multi-robot boundary coverage with plan revision. In: Proc of the IEEE Conference on Robotics and Automation (2006)
100. Woods, D., Tittle, J., Feil, M., Roesler, A.: Envisioning human-robot coordination in future operations. IEEE Transactions on Systems, Man and Cybernetics, Part C 34(2) (2004)
101. Yamashita, A., Arai, T., Ota, J., Asama, H.: motion planning of multiple mobile robots for cooperative manipulation and transportation. IEEE Transactions on Robotics and Automation 19(2) (2003)
102. Yanco, H., Baker, M., Casey, R., Keyes, B., Thoren, P., Drury, J.L., Few, D., Nielsen, C., Bruemmer, D.: Analysis of Human-Robot Interaction for Urban Search and Rescue. In: Proc of the IEEE Intl Workshop on Safety, Security and Rescue Robotics (2006)
103. Zheng, X., Koenig, S.: K-swaps: Cooperative negotiation for solving task-allocation problems. In: Proc of the Intl Joint Conference on Artificial Intelligence (2009)
104. Zlot, R., Stentz, A., Dias, M.B., Thayer, S.: Multi-robot exploration controlled by a market economy. In: Proc of the IEEE Conference on Robotics and Automation (2002)

Bounded Optimal Constrained Team Coordination with Delay Penalties and Location Choice

G. Ayorkor Korsah*, Anthony Stentz, and M. Bernardine Dias

Robotics Institute, Carnegie Mellon University,
Pittsburgh, PA 15213, USA
ayorkor@alumni.cmu.edu, {axs,mbdias}@ri.cmu.edu

Abstract. Many domains, such as emergency assistance, agriculture and construction, will increasingly require effective coordination of teams of mobile robots and humans to accomplish a collection of spatially distributed heterogeneous tasks. Although some tasks are independent, others may be related by constraints arising due to the complementary capabilities of different types of agents, which require them to cooperate to achieve certain goals. This paper addresses the problem of optimally assigning spatially distributed tasks to a team of heterogeneous mobile agents in domains where tasks may be related by precedence or simultaneity constraints and have a choice of locations at which they can be performed. Both the manner in which constraints are satisfied and the locations chosen for tasks impact the overall team utility. We present a novel mathematical model of the problem, describe how it can be solved optimally in a branch-and-price framework, and analyze the impact of problem features on the efficiency of the solution process.

Keywords: Multi-agent coordination, branch-and-price

1 Introduction

Multi-agent coordination problems range from those that require loosely coordinated teams in which agents independently perform their assigned tasks, to those that require tightly coordinated teams. Between the two extremes are many scenarios in which agents in a team must perform tasks, some of which are independent, and others of which are related by constraints such as precedence or simultaneity constraints. The efficiency of the team is related both to how efficiently each agent performs its own tasks and to how well sub-teams of agents coordinate with each other when performing tasks related by constraints. For example, a delay by one agent in performing its assigned task can negatively impact the efficiency of the team as a whole by causing a chain of delays for other agents, due to precedence or simultaneity constraints between the tasks. Furthermore, when there is a choice of locations at which a task may be performed,

* Also published as G. Ayorkor Mills-Tettey

the team's efficiency can be improved by choosing a location that minimizes the travel time for the team. We tackle the problem of planning for a team of heterogenous mobile agents, such as robots, humans, and vehicles, in such domains. This problem is significantly harder than a typical task allocation and routing problem because of the existence of *cross-schedule dependencies* [8] in the form of inter-task constraints and delay penalties. These cross-schedule dependencies result in a joint interdependent schedule optimization problem that must be solved simultaneously with task allocation.

While it is sometimes acceptable to simply find a feasible solution to the coordination problem, in many domains we seek high quality, or even optimal solutions. For example, in commercial applications such as agriculture, efficiency translates into higher profits and so it is useful to strive for optimality, or to be able to bound the suboptimality of a given solution. In other domains, striving for optimality may be motivated by the potentially high cost of suboptimal solutions. For example, in emergency response, inefficient solutions may translate into the loss of human life or damage to property. Certainly, the time it takes to compute the optimal solution is an important consideration, since waiting too long to find the optimal solution may cost more than executing a suboptimal solution. In situations when it is necessary to execute a suboptimal solution, it is useful to have a bound on the suboptimality of the chosen solution. In this work, we focus on computing bounded optimal solutions to the constrained team coordination problem we have described.

Consider a problem in which individuals with special needs must be evacuated or sheltered in an emergency. These individuals may have special transportation or sheltering needs that must be taken into account during emergency planning. Considering available transportation options (e.g. vans, ambulances, helicopters), support teams, and shelters, an emergency assistance plan for these individuals will determine which vehicle will pick up each individual and when. It will also schedule any support teams (e.g. medical personnel) which need to be available before, at the time of, or after pickup or drop-off of an individual. The evacuation plan must also determine which shelter each individual will be taken to, considering the individual's particular requirements. With appropriate information about the individuals needing service, an optimal evacuation plan can be created ahead of time, and this optimal plan can become a seed plan that is adjusted as needed in the event of an actual emergency [9]. For illustrative purposes in this paper, we consider a simplified version of this problem, in which there are a number of clients that must be visited by a medical agent before they are then moved by a transportation agent to an appropriate shelter. Given the starting location of all clients as well as the locations of the shelters, we wish to compute an optimal plan for the team of medical and transportation agents.

This paper presents a novel set-partitioning mathematical model for the problem of allocating, scheduling and choosing locations for tasks in such domains. We present a branch-and-price approach to solving this problem. Our approach enables finding a bounded optimal solution, considering the value of tasks completed, travel costs, as well as delay costs due to satisfying temporal constraints.

We use simulation experiments to characterize the performance of the algorithm as a function of key problem features, namely delay penalties and location choice.

2 Problem Definition

We consider a problem in which a set of mobile agents, K, is available to perform a collection of tasks. Each compound task, which may involve the collaboration of multiple agents, can be decomposed into a number of simpler single-agent tasks related by precedence and/or simultaneity constraints. We designate the set of single-agent tasks as J. Each single-agent task $j \in J$ in turn consists of one or more spatially distributed primitive tasks or *subtasks* that must be performed in a given order. For example, the compound task of attending to a client in our example scenario consists of the two single-agent tasks of a medical visit and a transportation service. A medical visit is a task with a single subtask, whereas transporting a customer comprises two subtasks: a pickup at one location and a drop-off at another. Different single-agent tasks are suited to different types of agents in the system, based on available capabilities and resources. In our example problem, medical tasks cannot be performed by transportation agents and vice-versa. Each subtask, $i \in I$ may have a fixed location or a choice of a small set of locations L_i at which it may be performed. For example, a client may be dropped off at one of a small number of shelters. Subtasks might have time windows constraining their start time, and in the case of transporting items from one location to another, subtasks might use up a finite capacity available on the assigned agent. Pairs of subtasks in the problem might be related by precedence or simultaneity constraints, thus creating constraints between different agents' schedules. For example, the medical visit task, comprising a single subtask, must be performed before the pickup subtask of the transportation service. In summary, the problem features that need to be considered in assigning agents to tasks are capability constraints, location choice, time window constraints, agent capacity constraints, precedence/simultaneity constraints, and routing and delay costs. The time and precedence/simultaneity constraints may result in delays in the agents' schedules, which may increase the cost, or conversely, reduce the total value of the solution.

Table 1. Nomenclature for problem definition

Symbol	**Definition**
K	Set of agents
J	Set of tasks
I	Set of subtasks
L_i	Set of possible locations for subtask $i \in I$

3 Related Work

Various forms of the problem of allocating and scheduling spatially distributed tasks are the subject of large bodies of work in multi-robot systems and operations research. However, existing approaches do not adequately address cross-schedule dependencies and those approaches that have begun to incorporate cross-schedule precedence and synchronization constraints have done so for fairly simple problems with homogenous agents and single-step tasks.

Market-based task allocation strategies have been proven efficient in many multi-robot task allocation problems [5, 6]. Based on the principles of markets and auctions, agents are designed as self-interested agents that operate in a virtual economy by bidding on tasks. While highly efficient, market-based approaches in general do not provide optimality bounds or guarantees, which is the focus of this work. Some mathematical programming approaches have been applied to multi-robot coordination. For example, Koes [7] discusses allocating joint tasks to a team of robots, where tasks have associated rewards that decay linearly, and the system can be constrained through capability constraints. While an interesting and related problem, the model does not capture our desired problem features of delay penalties, multi-step tasks, and location choice.

Vehicle routing problems (VRPs) address the transportation of passengers or the distribution of goods between depots and final users. VRPs can be expressed as mixed integer programming problems (MIP), defined on a graph in which the nodes correspond to locations of tasks to be performed, and edges correspond to travel segments between these locations. Proposed mathematical models can be broadly categorized as 3-index models and 2-index (or set-partitioning) models. For example, Cordeau [4] defines, for the dial-a-ride (DARP) problem (a variant of the VRP), a 3-index binary variable x_{ij}^k which is equal to 1 if vehicle k travels from node i to node j in the final solution. In contrast, Savelsbergh and Sol [12] propose a set-partitioning model for the DARP in which Ω_k is the set of feasible routes for vehicle k, and the 2-index variable x_r^k is a binary decision variable that takes on the value 1 if route $r \in \Omega_k$ is performed by vehicle k and 0 otherwise. Each route in Ω_k is a path through a subset of nodes, and is feasible in that all capacity and time constraints are satisfied along the route. A branch-and-price process is used to find a solution.

Recent work in the vehicle routing literature has begun to consider precedence constraints and simultaneity constraints. In particular, Bredstrom and Ronnqvist present two different approaches. In one case [3], they create a three-index formulation of a vehicle routing problem, taking into consideration timing/synchronization constraints between individual tasks. In another case [2], they present a set-partitioning formulation that takes into consideration precedence constraints. Larsen et al [10] and Rasmussen et al [11] similarly address vehicle routing problems with precedence and synchronization constraints. None of this work, however, addresses cost-related cross-schedule dependencies such as delay penalties, nor do they address heterogeneous agents and tasks, multi-step tasks, agent capacity constraints and location choice.

This paper presents a set-partitioning model which addresses location choice and precedence (and/or simultaneity) constraints, while also being able to penalize delay time as needed. A significant contribution of this work is a model of an important problem for which no model currently exists. Another important contribution of this work is the analysis of the impact of these problem features, particularly delay penalties, on the performance of the solution approach.

4 Mathematical Model

We present a set-partitioning model with side constraints for this problem. The set-partitioning model, while representing complete feasible routes with single variables, also exposes time variables in the master problem formulation, thus allowing delays to be penalized by putting delay time variables in the objective function. We adopt the terminology of the vehicle routing literature and use the term *route* to represent a single agent's plan – that is, a sequence of subtasks that the agent will perform at given locations according to the computed schedule.

In a set-partitioning approach, feasible routes for agents are represented by columns in the mixed integer linear program. In particular, a binary variable x_r^k indicates whether an agent k performs a route r chosen from among all feasible routes R_k for agent k. In our problem, a feasible route is an ordered set of subtasks to be performed at chosen locations, such that all subtasks corresponding to the same single-agent task occur on the same route and agent capacity constraints are not violated. A typical set-partitioning formulation would consist of these variables alone, with constraints specifying that each agent must perform only one route, and each task must appear on only one route.

In our formulation, however, we include additional time variables that appear in side constraints enforcing the precedence constraints between subtasks that may appear on different routes. The real-valued variable d_i^k (the execution-delay variable) represents the amount of time that agent k, having arrived at the chosen location for subtask i, has to wait before it can begin execution of subtask i. This delay might be due to precedence constraints involving other subtasks being performed by other agents, or it might be because subtask i has a specific time window during which it must be performed. d_i^k is 0 if agent k is not assigned to subtask i, or if there is no execution delay. The real-valued variable t_i represents the time that execution begins on subtask i. If subtask i is not executed in the optimal solution, t_i is 0. In addition to the domain variables x_r^k, d_i^k, and t_i, the model includes helper variables $a_{i'i}$ representing the indirect delay in the arrival time for subtask i due to the execution-delay time for subtask i' occurring earlier on the same route. If subtasks i' and i are not on the same route in the chosen solution, $a_{i'i}$ is 0. This is also the case if i' and i are on the same route, but there is no arrival delay. These helper variables are needed to ensure a linear formulation; without these variables, the model would need to be non-linear, containing product terms of the form $d_i^k x_r^k$.

Table 2 summarizes the variables and defined quantities appearing in the mathematical model. The quantity v_j represents the value or reward of completing a single-agent task j, which may of course comprise more than one subtask.

Table 2. Defined variables and constants

Variable	**Definition**	**Type**
x_r^k	Whether agent k performs route r	Binary
d_i^k	Delay time of agent k for subtask i	Real
t_i	Execution start time for subtask i	Real
$a_{i'i}$	Arrival delay for subtask i caused by subtask i' (helper variable)	Real
Term	**Definition**	**Type**
R_k	Set of feasible routes for agent k	Set
P	Set of precedence constraints	Set
v_j	Value of completing task j	Real
c_{1r}^k	Travel cost for route $r \in R_k$	Real
c_2^k	Wait cost per unit time for agent k	Real
π_{jr}^k	Whether task j is on route $r \in R_k$	Binary
γ_{ilr}^k	Whether subtask i occurs at location l on route $r \in R_k$	Binary
$\delta_{i_1 i_2 r}^k$	Whether subtask i_1 occurs before subtask i_2 on route $r \in R_k$	Binary
τ_{ilr}^k	No-wait start time of subtask i at location l on route $r \in R_k$	Real
τ_∞	End of planning horizon	Real
D_i	Maximum allowed delay for subtask i	Real
$\epsilon_{i_1 i_2}^P$	Minimum desired time gap between service completion on subtask i_1 and service commencement on subtask i_2 for $(i_1, i_2) \in P$	Real
$[\alpha_{il}, \beta_{il}]$	Valid time window within which to start subtask i at location l	Real
λ_{il}^k	Service time for subtask i performed by agent k at location l	Real
λ_i	Service time for subtask i in chosen solution $= \sum_{k \in K} \sum_{r \in R_k} \sum_{l \in L_i} \lambda_{il}^k \gamma_{ilr}^k x_r^k$	Real
y_i	Whether subtask i is performed in chosen solution $= \sum_{k \in K} \sum_{r \in R_k} \sum_{l \in L_i} \gamma_{ilr}^k x_r^k$	Binary

The value c_{1r}^k represents the total travel cost of the route $r \in R_k$, and c_{2r}^k represents the delay penalty per unit time for agent k. The indicator π_{jr}^k is 1 if task j occurs on route $r \in R_k$ and 0 otherwise. Similarly, γ_{ilr}^k is 1 if subtask i occurs at location l on route $r \in R_k$ and 0 otherwise, and $\delta_{i'ir}^k$ is 1 if subtask i' occurs before subtask i on route $r \in R_k$ and 0 otherwise. The value τ_{ilr}^k represents the time that subtask i would be started on route $r \in R_k$ assuming no delay time was necessary. It is computed during route-planning from the travel time and the execution time for all earlier tasks on the route. τ_∞ represents the end of the planning horizon. The value D_i represents the maximum allowed execution delay time for subtask i; α_{il} and β_{il} represent the earliest and latest times respectively that service can begin on subtask i when it is performed at location l. λ_{il}^k represents the service time for subtask i when it is performed at location l by agent k. In the model, we use λ_i to represent the service time of subtask i in the chosen solution (0 if i is not performed), and y_i to indicate whether or not subtask i is performed in the selected solution. That is, λ_i and y_i are

placeholders for the following expressions, respectively:

$$\lambda_i = \sum_{k \in K} \sum_{r \in R_k} \sum_{l \in L_i} \lambda_{il}^k \gamma_{ilr}^k x_r^k$$

$$y_i = \sum_{k \in K} \sum_{r \in R_k} \sum_{l \in L_i} \gamma_{ilr}^k x_r^k \equiv \sum_{k \in K} \sum_{r \in R_k} \pi_{jr}^k x_r^k$$

(where j is the task to which subtask i belongs)

Finally, P represents the set of precedence constraints in the problem. Each precedence constraint $p = (i', i) \in P$ indicates that execution of subtask i' must end at least $\epsilon_{i'i}^P$ time units before service begins on subtask i.

In the model, the objective function (1) strives to maximize the difference between total value or reward and overall travel cost and delay penalty. (C1) specifies that each agent is assigned to exactly one route (which may be an empty route, allowing an agent not to be used). (C2) specifies that each task is assigned to at most one agent, and can in fact be rejected by assigning it to no agent. These two are variations on the standard set-partitioning constraints.

Constraints (C3) through (C7b) are side constraints resulting from including time variables in the master problem formulation and from enforcing precedence constraints. (C3) computes the start time for a subtask and (C4) represents a bound on the delay times. (C5a-C5c) represent constraints on the arrival delay helper variables, which enable the start time of each subtask to be computed correctly, taking into consideration the delay time of all prior subtasks on the selected route. These constraints (C5a-C5c) together represent a linearization of the nonlinear equation $a_{i'i} = \sum_{k \in K} \sum_{r \in R_k} \delta_{i'ir}^k d_{i'}^k x_r^k$. Constraints (C6a) and (C6b) represent the time window bounds for the subtask. Finally, (C7a) and (C7b) capture the precedence constraints of the problem: (C7a) indicates that the second task i in the precedence constraint $(i', i) \in P$ is performed only if the first task i' is performed; (C7b) ensures that the start times of the task satisfy the precedence constraints. Similar constraints to (C7a) and (C7b) can represent simultaneity constraints, by changing the inequalities to equalities, and removing the $\lambda_{i'}$ and $\tau_\infty(y_i - y_{i'})$ terms from (C7b).

It should be noted that capacity constraints do not appear directly in this model for the master problem but are dealt with when generating feasible routes. Route generation also performs location choice by fixing the location of each subtask on the route. The solution of the master set-partitioning problem then selects between all generated feasible routes for an agent, thus finalizing the location choice for each subtask. It also fixes the time for each task by setting delay times as needed to ensure that cross-schedule precedence constraints are satisfied while still respecting the travel and execution times.

Maximize:

$$\sum_{j\in J}\sum_{k\in K}\sum_{r\in R_k} v_j \pi^k_{jr} x^k_r - \sum_{k\in K}\sum_{r\in R_k} c^k_{1r} x^k_r - \sum_{i\in I}\sum_{k\in K} c^k_2 d^k_i \tag{1}$$

Subject to:

$$\sum_{r\in R_k} x^k_r \le 1 \qquad \forall k \in K \tag{C1}$$

$$\sum_{k\in K}\sum_{r\in R_k} \pi^k_{jr} x^k_r \le 1 \qquad \forall j \in J \tag{C2}$$

$$t_i - \sum_{l\in L_i}\sum_{k\in K}\sum_{r\in R_k} \tau^k_{ilr}\gamma^k_{ilr} x^k_r - \sum_{i'\in I} a_{i'i} - \sum_{k\in K} d^k_i = 0 \qquad \forall i \in I \tag{C3}$$

$$d^k_i - D_i \sum_{l\in L_i}\sum_{r\in R_k} \gamma^k_{ilr} x^k_r \le 0 \qquad \forall i \in I, k \in K \tag{C4}$$

$$\sum_{k\in K} d^k_{i'} - a_{i'i} + D_{i'} \sum_{k\in K}\sum_{r\in R_k} (\delta^k_{i'ir} x^k_r) \le D_{i'} \qquad \forall i' \in I, i \in I \tag{C5a}$$

$$a_{i'i} - D_{i'} \sum_{k\in K}\sum_{r\in R_k} \delta^k_{i'ir} x^k_r \le 0 \qquad \forall i' \in I, i \in I \tag{C5b}$$

$$a_{i'i} - \sum_{k\in K} d^k_{i'} \le 0 \qquad \forall i' \in I, i \in I \tag{C5c}$$

$$-t_i + \sum_{l\in L_i} \alpha_{il} \sum_{k\in K}\sum_{r\in R_k} \gamma^k_{ilr} x^k_r \le 0 \qquad \forall i \in I \tag{C6a}$$

$$t_i - \sum_{l\in L_i} \beta_{il} \sum_{k\in K}\sum_{r\in R_k} \gamma^k_{ilr} x^k_r \le 0 \qquad \forall i \in I \tag{C6b}$$

$$y_i - y_{i'} \le 0 \qquad \forall (i', i) \in P \tag{C7a}$$

$$t_{i'} - t_i + \lambda_{i'} + \tau_\infty (y_i - y_{i'}) + \epsilon^P_{i'i} y_{i'} \le 0 \qquad \forall (i', i) \in P \tag{C7b}$$

5 Branch-and-Price Algorithm

Mixed integer programming problems are generally solved in a branch-and-bound framework. To begin, a bound on the solution is computed by relaxing the integrality constraints and solving the resulting linear program. This bound is an upper bound for a maximization problem and a lower bound for a minimization problem. Subsequently, branching decisions are made on fractional variables that should be integer, and the solution process is repeated at each node of the branch-and-bound tree, until a solution is found that satisfies the integer constraints and whose objective function is at least as good as the best bound on the nodes in the tree.

In a set-partitioning model like ours, it is not possible to enumerate all possible variables/columns in the integer program up front, and this is where a *column generation* process is useful. The algorithm starts out by considering only a subset of columns (in our case, feasible routes), and new columns are added as needed. The columns to be added are determined by solving a problem called the *pricing subproblem*, derived from the dual variables of the master problem. A branch-and-price algorithm is a branch-and-bound algorithm in which column generation occurs at each node of the branch-and-bound tree. A detailed explanation of branch-and-price, such as how the pricing problem is derived, is outside the scope of this paper. Barnhart et al [1] provide a useful introduction and theoretical discussion.

We develop a custom branch-and-price algorithm to solve our mathematical model. We discuss two aspects of our solution process, namely, how we perform column generation, and how we make branching decisions.

5.1 Pricing and Column Generation

Designating the dual variables corresponding to constraints (C1) to (C7b) in our mathematical model as u^1 to u^{7b} respectively, we derive the pricing subproblem for our model to be the problem of finding feasible routes r for agent k for which the following quantity is positive:

$$
\begin{aligned}
p_r^k = & -(u_k^1 + c_{1r}^k) \\
& + \sum_{i \in I} \sum_{l \in L_i} (D_i u_{ik}^4 - \alpha_{il} u_i^{6a} + \beta_{il} u_i^{6b}) \gamma_{ilr}^k \\
& + \sum_{j \in J} \sum_{l \in L_{i_o^j}} (v_j - u_j^2) \gamma_{i_o^j lr}^k \\
& + \sum_{(i'i) \in P} \sum_{l \in L_{i'}} (u_{i'i}^{7a} + (\tau_\infty - \epsilon_{i'i}^P - \lambda_{i'l}^k) u_{i'i}^{7b}) \gamma_{i'lr}^k - \sum_{(i'i) \in P} \sum_{l \in L_i} (u_{i'i}^{7a} + \tau_\infty u_{i'i}^{7b}) \gamma_{ilr}^k \\
& + \sum_{i \in I} \sum_{l \in L_i} \tau_{ilr}^k u_i^3 \gamma_{ilr}^k + \sum_{i' \in I} \sum_{i \in I} (D_{i'} u_{i'i}^{5b} - D_{i'} u_{i'i}^{5a}) \delta_{i'ir}^k
\end{aligned}
\tag{2}
$$

Such routes, if they exist, could potentially increase the objective function of the solution. At each node of the branch-and-bound tree, we find such "profitable" routes, add one or more of them to the master problem, re-solve the relaxed master problem, and continue the column generation process. When no such route is found, column generation ends at that node, and the branch-and-bound process continues.

To gain a better understanding of the pricing subproblem represented by equation 2, note that for a given instance of the subproblem, the dual variables u are constants. Also recall that γ_{ilr}^k indicates whether subtask i is performed at location l on route $r \in R_k$. Thus, all terms in the equation that are multiplied by γ_{ilr}^k represent costs for {subtask, location} pairs that are visited along the

route. Some of these terms, such as the third line, are included for all {subtask, location} pairs along the route. Others are included only if the {subtask, location} pair satisfies certain properties. For example, the third line has terms that are included only for subtasks representing the first step of their corresponding tasks and the fourth lines has terms that are included only for subtasks that are involved in precedence constraints. All the terms described thus far are constant values that can be computed independently for each subtask along the route. However, the first term on the fifth line is a value that is linear in the arrival time at the subtask, assuming no delays. Finally, the last term depends on the relative order of every pair of subtasks along the route, since $\delta^k_{i'ir}$ indicates whether subtask i' occurs before subtask i on route $r \in R_k$.

We can think of the pricing problem as the problem of searching for a route through a graph in which nodes represent {subtask, location} pairs, edges indicate that an agent can perform one subtask after another, and transition costs in the graph are determined by equation 2. A feasible route is one that satisfies agent capacity constraints as well as the branching constraints at the current branch-and-bound node.

To enable the search process to compute the overall price of a route as expressed by Equation 2, we decompose the equation into a value for each node visited and each edge traversed along the route. Whereas in a typical route-planning problem, the transition cost from from one node to another would depend only on the two nodes in question, the last two terms of Equation 2 complicate the cost structure. As a result, the transition cost to a node from another in the graph depends not only on these two nodes, but also on the time spent traveling from the beginning of the partial route up to these nodes, and on what subtasks have been performed earlier on that partial route.

To solve this pricing subproblem, we have developed a route-planning algorithm that performs a search through a multi-dimensional state space, to find a path from a start node to a goal node while satisfying the necessary constraints. Each state in the space being searched is identified by the graph node n representing a given {subtask, location} pair, the no-wait arrival time t_a of the agent at the node along the route, and the unordered set S_p of subtasks that have been previously completed along the route to that state: $state := \{n, t_a, S_p\}$. The route-planning algorithm can use either a depth-first search or a best-first search focused with a heuristic. For the experiments in this paper, we use the depth-first search mode. Our branch-and-price process can use either the commercial solver CPLEX or the open-source solver LPSolve to solve the relaxed master problem. For the experiments in this paper, we use CPLEX.

5.2 Branching

A simple branching decision for our problem would be to set a fractional x^k_r variable to either 0 or 1. However, it is necessary to make higher-level branching decisions, both for efficiency and in order not to complicate the column generation procedure described earlier. We adopt the following branching decisions, in the priority order listed below.

- *Branching on task pairs 'together'*: When there are fractional routing variables such that two tasks occur together on some route but not on another, we branch by forcing the two tasks to be on the same route ("together") in one branch or on different routes ("not together") in the other branch.
- *Branching on subtask pair order*: When the fractional routing variables include two routes with the same subtasks performed in different orders, we branch by constraining the subtasks to occur in a specific order in one branch and in the opposite order in the other branch
- *Branching on subtask location*: When the the fractional routing variables include two routes with the same subtasks such that a subtask is performed at two different locations on each route, we branch by forcing the subtask to be performed at one location in one branch and not at that location in the other branch.
- *Branching on task agent*: Lastly, when the fractional routing variables represent the same route performed by two different agents, we branch by forcing a task on that route to be performed by a given agent in one branch, and not by that agent in the other branch.

6 Experiments and Results

The problem addressed in this paper is a complex combinatorial optimization problem which is strongly NP-hard. There are several features that affect the computed solutions, as well as the performance of the solution process. In this section, we focus on two of these features, namely delay penalty and location choice. We first describe these problem features. We then present a sample problem and illustrate the impact of these features on the computed solutions. Finally, by presenting performance metrics on several randomized experiments, we characterize how these key problem features impact the performance of the branch-and-price algorithm.

6.1 Problem Features of Interest

Delay Penalty: The delay penalty is the cost associated with the execution delay time for an agent and subtask. We express delay penalty per unit time as a fraction of the travel cost per unit time for that agent. For example, a delay penalty of 0.5 means that it costs an agent half as much to be idle for a given amount of time as it does to travel for that same amount of time. In a given domain, travel costs might capture fuel, personnel costs and vehicle wear-and-tear when the vehicle is moving, whereas delay penalties might cover personnel costs and a fraction of fuel and vehicle wear-and-tear costs for when the vehicle is stationary.

Location Choice: The number of location choices for a given subtask is the number of possible locations at which the subtask can be performed. For example, in the evacuation scenario, if there are two possible shelters to which a given client can be transported, the drop-off subtask of the transportation task has two location choices.

6.2 Example Problem and Solution

We illustrate the approach with an evacuation problem with 6 clients, 1 medical agent, 2 transportation agents and 2 shelter locations (Figure 1). The agents and tasks are illustrated in a simulated 10km x 10km environment. The agents travel at a speed of 60 km/hr or 1 km/minute. The transportation agents have a capacity of 3, meaning that they can carry 3 clients at a time. To serve more than 3 clients, one or more drop-offs would be needed before picking up additional clients. The compound task of serving a client requires two single-agent tasks, the first comprising 1 subtask (a medical visit) and the second comprising 2 subtasks (a pickup subtask followed by a drop-off subtask). There is a single precedence constraint between the medical visit and the pickup subtask. No explicit precedence constraint is needed between the pickup subtask and the drop-off subtask because subtasks of a single task are defined to be strictly ordered. As such, there are a total of 18 subtasks to be allocated by the system, with 6 pairwise precedence constraints to satisfy.

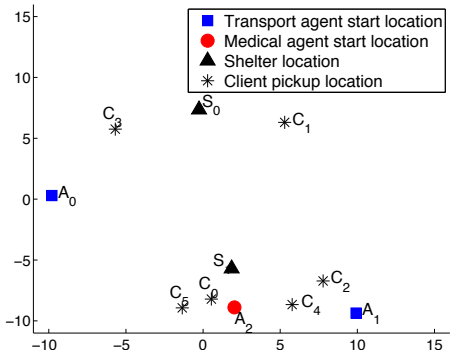


Fig. 1. Example problem with 6 clients, 2 transportation agents, 1 medical agent, and 2 shelters.

Figure 2(a) shows the optimal solution when there is no delay penalty and each client must be transported to its closest shelter (that is, there is only 1 drop-off location choice per client). The left illustration shows the computed routes, while the right one shows the agent schedule, coded by travel time, delay time, and service time. Service times are annotated with the subtask type (V for "visit", P for "pickup" and D for "dropoff") and client IDs. For drop-off subtasks, they are further annotated with the shelter ID. Because there is no delay penalty in this first example, the algorithm computes the routes that minimize the total travel time for all agents, with no consideration of whether a given client is ready to be picked up at the time the transportation agent arrives at the client's pickup location. This results in significant delays for the transportation agents

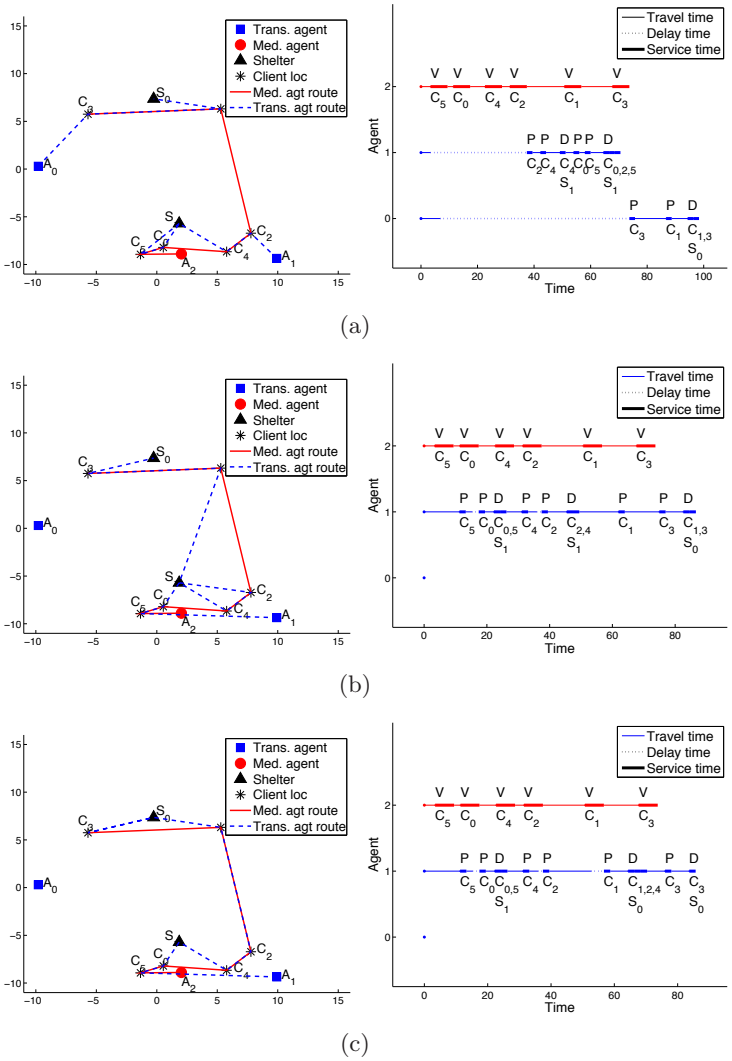


Fig. 2. Solution routes (left) and schedules (right) to example problem, with (a) no delay penalty and 1 location choice, (b) delay penalty of 0.5 and 1 location choice, (c) delay penalty of 0.5 and 2 location choices.

when they arrive at a client's location before the client has been seen by the medical agent.

When we introduce a delay penalty of 0.5, the optimal solution computed by the algorithm changes significantly, as illustrated in Figure 2(b). Because medical visits are the bottle-neck in the problem and it is now costly to have a transportation agent wait for a medical agent, the optimal solution makes use

of only 1 transportation agent. In this way, it is able to reduce the overall delay, at the expense of increased total travel time for the team.

Figure 2(c) shows the impact on the solution when there is a delay penalty of 0.5 and each client is not constrained to be transported to its closest shelter, but may be transported to either shelter; that is, there are 2 drop-off location choices per client. This flexibility in the drop-off location enables the algorithm to come up with a better solution with reduced travel time.

Table 3 summarizes the optimal solution to this example problem as a function of delay penalties and the number of location choices. It can be noted that when there is no delay penalty for this problem, the solution is the same for 1 or 2 drop-off location choices – that is, the optimal drop-off location for each client is its closest shelter. When there is a delay penalty, it is beneficial to have a choice of locations at which the clients can be dropped off.

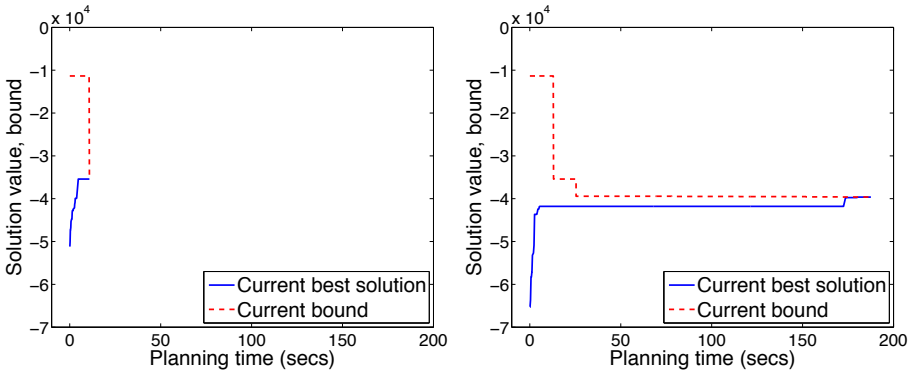
Table 3. Optimal solution as a function of delay penalty and location choices

Delay Penalty (dp)	**Location Choices**	**Total travel time** t_t (mins)	**Total delay time** t_d (mins)	**Total team cost** $t_t + dp * t_d$
0.0	1	81.66	100.93	81.66
0.0	2	81.66	100.93	81.66
0.5	1	96.68	3.62	98.49
0.5	2	91.76	7.62	95.57

Figure 3 shows the best solution and best bound over time for the example above, with 1 drop-off location choice and a delay penalty of 0.0 (left) and 0.5 (right). In both cases, the algorithm finds good solutions early, demonstrating its usefulness as an anytime solution approach. However, a non-zero delay penalty has a significant impact on the time it takes to find and prove the optimal solution. This is because the algorithm must essentially evaluate the trade-off between travel time and delay time in potential solutions it encounters during the solution process.

6.3 Simulation Experiments

The simulation experiments characterize the behavior of the solution process as a function of problem features. We used the same evacuation problem with one medical agent, two transportation agents with a capacity of 3, and two shelter locations. We varied the number of clients from 2 to 10, resulting in a range of 6 to 30 subtasks, since each client requires 3 subtasks. We create 5 instances of each problem configuration with random client, agent, and shelter locations. We considered delay penalties of 0 or 0.5, and 1 or 2 drop-off location choices per client, resulting in 4 combinations of problem features. The experiments were run on a 2.67 GHz Intel processor.



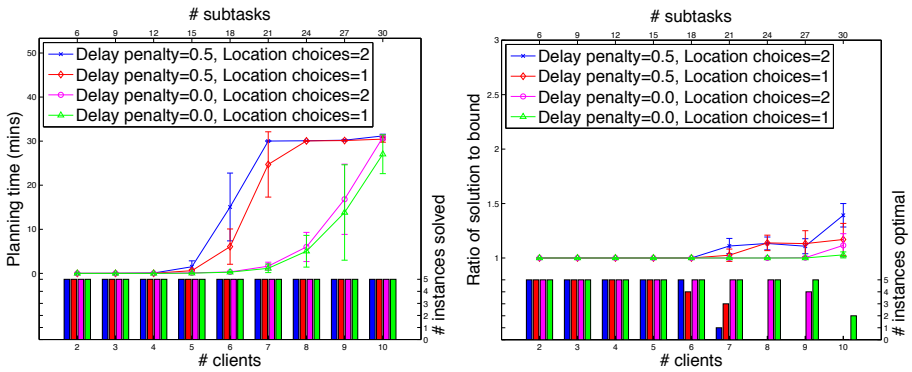
(a) Delay penalty=0.0 (b) Delay penalty=0.5

Fig. 3. Example solution profiles (current best solution and best bound over time).

Figure 4(a) shows the time to find and prove the optimal solution, averaged over 5 random instances of each problem configuration. The horizontal axis represents the number of clients, which is proportional to the number of subtasks, since each client requires a total of 3 subtasks: a medical visit, a pickup and a drop-off. The error bars on the time plots represent one standard deviation from the mean for each problem configuration. Under the plots of solution time, there is a bar chart indicating, for each problem configuration, how many of the 5 random instances were solved successfully. For these experiments, solution time to compute and prove the optimal solution was capped at 30 minutes. The combinatorial nature of the problem is apparent in the rapid increase in the time needed to prove solution optimality as the problem size increases. Figure 4(b) shows the average ratio of solution to bound at termination of the algorithm for each problem configuration. A ratio of 1 indicates an optimal solution. Under the graph of bound ratios, there is a bar chart indicating in how many cases the algorithm found a provably optimal solution.

Figures 4(a) and 4(b) illustrates a significant distinction between the solution complexity of problems with and without delay penalties. The steep increase in planning time begins after 7 clients for problems *without* delay penalties, and after only 5 clients for problems *with* delay penalties. The increased complexity of problems with delay penalties in this scenario is also illustrated in the solution bounds in Figure 4(b). For problems with no delay penalties, we can find optimal or effectively optimal solutions for problems with up to 9 clients, and the terminating bound ratios for problems with 10 clients are small. With a delay penalty of 0.5, however, most of the solutions are provably optimal for only up to 6 clients. For 10 clients, the average ratio of the solution to bound at termination was larger than the case with no delay penalties. The figures also illustrate that problems with 2 location choices are slightly more difficult than those with

only 1 location choice. The gap between these cases is more significant with a non-zero delay penalty than without.



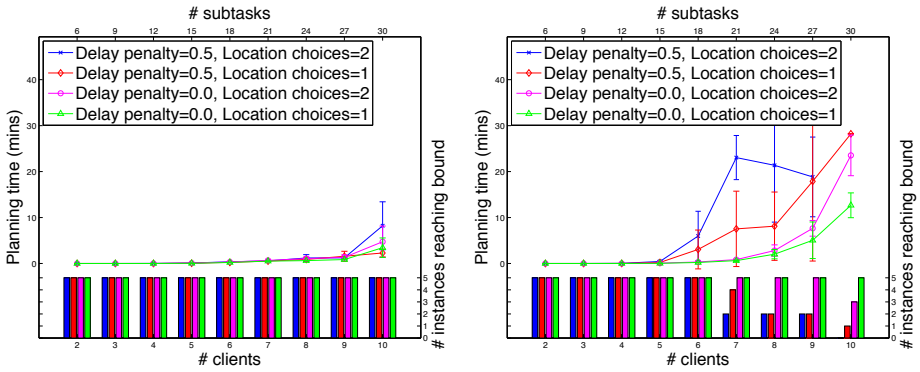
(a) Solution time (capped at 30 mins) (b) Solution bound after 30 mins

Fig. 4. Overall planning time and bound

.

The above results illustrate that, as expected from the strongly NP-hard nature of the problem under consideration, overall planning time increases significantly as the problem gets larger. As illustrated in the previous section, an advantage of the branch-and-bound framework is that the algorithm may find several feasible solutions before it finds and proves the optimality of the final solution. It can thus be used as an "anytime" algorithm which gives progressively better solutions as it proceeds, and can be terminated early with a possibly suboptimal solution. Furthermore, the algorithm provides a bound on how suboptimal the provided solution might be.

Figures 5(a) and 5(b) further illustrate the "anytime" nature of the algorithm by showing the time required to find a solution within a factor of 2 of the bound and within a factor of 1.1 of the bound, respectively. Underneath the time plots, the bar graph indicates the number of instances of each problem configuration for which the algorithm successfully found a solution within the specified bound sometime within the maximum allotted time. This is the number of instances over which the planning time is averaged for each configuration. Within the maximum allotted planning time of 30 minutes, we were able to find solutions within a factor of 2 of the bound for problems with 9 or fewer clients within a minute and a half, and for problems with 10 clients in less than 10 minutes (Figure 5(a)). In the maximum allotted planning time of 30 minutes, we could find solutions that were within a factor of 1.1 of the bound for most problems with up to 9 or 10 clients and without delay penalties. With delay penalties, however, there were much fewer problems with more than 6 clients for which we were able to find solutions within a factor of 1.1 of the bound (Figure 5(b)).



(a) Time to solve to within a factor of 2 of bound

(b) Time to solve to within a factor of 1.1 of bound

Fig. 5. Planning time to specific bound ratios

.

7 Discussion

This paper focuses on optimal planning for task allocation, scheduling and routing with cross-schedule dependencies. Since the solution approach finds and proves the optimal solution, it is advantageous and most suitable for pre-planning in domains where information about the tasks and key features of the environment are known ahead of time. The approach enables the team to begin execution with a plan that is guaranteed to be of high quality. During plan execution, real-world timing variations may necessitate minor adjustments in the computed plan. In other work [8], we outline and demonstrate on a set of indoor robots, a flexible strategy for executing such pre-computed optimal plans to ensure that cross-schedule constraints are satisfied, even in the presence of timing variations during execution. The strategy also enables graceful degradation of the plan if tasks fail.

Due to the highly combinatorial nature of the problem under consideration, the optimal planning approach is not suitable for real-time re-planning in dynamic domains in which new tasks come in over time, or failed tasks must be reallocated. However, the initial optimal plan can be used as a seed plan for more heuristic re-planning approaches. Such an approach combining optimal pre-planning and market-based dynamic task allocation has been demonstrated for problems without cross-schedule dependencies [9].

8 Conclusions

We have presented a novel mathematical formulation and a branch-and-price approach to task allocation and scheduling for a team of heterogenous mobile agents addressing a set of spatially distributed tasks related by precedence

and/or simultaneity constraints. The approach computes the optimal solution, taking into consideration delay penalties and reasoning about location choice for task execution. We characterize the performance of the algorithm as a function of the existence of delay penalties and location choice. Ongoing work extends the model and solution approach to address additional inter-task constraints. Furthermore, we will explore heuristic solution approaches to complement the presented bounded optimal solution approach.

Acknowledgments. This work was made possible by the support of NPRP grant #1-7-7-5 from the Qatar National Research Fund. The statements made herein are solely the responsibility of the authors.

References

1. C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46:316–329, 1998.
2. D. Bredström and M. Rönnqvist. A branch and price algorithm for the combimed vehicle routing and scheduling problem with synchronization constraints. Norwegian School of Economics and Business Administration, Department of Finance and Management Science, Discussion Paper Number FOR7 2007, 2007.
3. D. Bredström and M. Rönnqvist. Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. *European Journal of Operations Research*, 191:19–31, 2008.
4. J.-F. Cordeau. A Branch-and-Cut Algorithm for the Dial-a-Ride Problem. *Operations Research*, 54(3):573–586, 2006.
5. M. B. Dias. *TraderBots: A New Paradigm for Robust and Efficient Multirobot Coordination in Dynamic Environments*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, January 2004.
6. B. P. Gerkey and M. J. Matarić. Sold!: auction methods for multirobot coordination. *Robotics and Automation, IEEE Transactions on*, 18(5):758–768, Oct 2002.
7. M. Koes, I. Nourbakhsh, and K. Sycara. Heterogeneous multirobot coordination with spatial and temporal constraints. In *Proceedings of the National Conference on Artificial Intelligence (AAAI), 2005*, 2005.
8. G. A. Korsah. *Exploring bounded optimal coordination for heterogeneous teams with cross-schedule dependencies*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, February 2011.
9. G. A. Korsah, B. Kannan, I. A. Fanaswala, and M. B. Dias. Enhancing market-based task allocation with optimal initial schedules. In *Intelligent Autonomous Systems 11 - IAS-11*, pages 249 – 258, August 2010.
10. J. Larsen, A. Dohn, and M. S. Rasmussen. The vehicle routing problem with time windows and temporal dependencies. Technical Report 1.2009, DTU Management Engineering, April 2009.
11. M. S. Rasmussen, T. Justesen, A. Dohn, and J. Larsen. The home care crew scheduling problem: Preference-based visit clustering and temporal dependencies. Technical Report 11.2010, DTU Management Engineering, May 2010.
12. M. Savelsbergh and M. Sol. Drive: Dynamic routing of independent vehicles. *Operations Research*, 46(4):474–490, 1998.

Flood Disaster Mitigation: A Real-world Challenge Problem for Multi-Agent Unmanned Surface Vehicles

Paul Scerri, Balajee Kannan, Pras Velagapudi, Kate Macarthur, Peter Stone, Matt Taylor, John Dolan, Alessandro Farinelli, Archie Chapman, Bernadine Dias, and George Kantor

Abstract. As we advance the state of technology for robotic systems, there is a need for defining complex real-world challenge problems for the multi-agent/robot community to address. A well-defined challenge problem can motivate researchers to aggressively address and overcome core domain challenges that might otherwise take years to solve. As the focus of multi-agent research shifts from the mature domains of UGV and UAVs to USVs, there is a need for outlining well-defined and realistic challenge problems. In this position paper, we define one such problem, *flood disaster mitigation*. The ability to respond quickly and effectively to disasters is essential to saving lives and limiting the scope of damage. The nature of floods dictates the need for a fleet of low-cost and small autonomous boats that can provide situational awareness (SA), damage assessment and deliver supplies before more traditional emergency response assets can access an affected area. In addition to addressing an essential need, the outlined application provides an interesting challenge problem for advancing fundamental research in multi-agent systems (MAS) specific to the USV domain. In this paper, we define a technical statement of this MAS challenge problem based and outline MAS specific technical constraints based on the associated real-world constraints. Core MAS sub-problems that must be solved for this application include coordination, control, human interaction, autonomy, task allocation, and communication. This problem provides a concrete and real-world MAS application that will bring together researchers with a diverse range of expertise to develop and implement the necessary algorithms and mechanisms.

Keywords: multi-agent systems, challenge, communication, autonomy, path-planning, coordination, task-allocation

1 Introduction

Robotics challenge problems like FIRST, DARPA Challenges, BotBall, MAGIC, etc [17, 24, 23, 7] have shown to be an effective motivational tool for invigorating robotics researchers at all levels, from high-school to experienced professionals, while solving real-world problems. Such challenges offer an incredible opportunity to shorten the time-cycle required to advance the state of the art in autonomous vehicle technology. The success of the DARPA challenges, Grand and

Urban, are a testamant to this. Robots such as Stanley and Boss have become part of the robotics lore, while the developed technological solutions have become the backbone for translating commercially developed autonomous vehicles on our roads from a dream to reality. Furthermore, as we look to translate the developed technology and success over different applications, there is a need to define real-world challenges in alternate domains like USVs.

According to the United Nations, annual flooding currently impacts in excess of 500 million people, costs the world up to $60 billion USD and the number of casualties exceed 20,000 in Asia alone [22]. Changes to the environment, such as mining, deforestation, and general industrialization are likely to worsen the problem worse over time. Unfortunately, flooding disasters disproportionately effect people in under-developed countries due to lack of early warning systems, flood control and emergency response infrastructure. The scope and application of the problem have far-ranging implications. Currently, despite large scale flooding disasters world over, in the immediate aftermath, victims are largely left to fend for themselves. The lack of relief aid is in part due to a limited knowledge of the affected areas and specific needs of the victims. Thus, any reliable solution, regardless of efficiency, will have immediate real-world benefits, while further research and development can increase the value of the system over time. In most cases, floods occur over large areas and over relatively long time-scales. Often their occurrence can be cyclical in nature and can be predicted well in advance, e.g., monsoonal or hurricane flooding can be expected annually during a well-defined season. We believe that the inherent properties and the scale of impact of a flooding disaster make it an ideal problem to be addressed with robot teams with multi-agent technology playing a central role. Finally, the cyclical nature of flooding means that it is likely that solutions can be iteratively evaluated and improved in real disaster environments over time. We believe that flood mitigation might be the *seminal* challenge for MAS because it is an important real-world problem for which MAS appears to be an ideal and essential technology.

In order to effectively address the problem, small, autonomous watercrafts are ideal for flood mitigation and response. Relative to other types of vehicles, watercraft are simple, robust and reliable. By keeping the vehicles small, most safety issues can be avoided simply by ensuring that if there is a collision, it can cause at most very minor damage. Influenced in part by field experiences of Murphy et al. [18], we believe unmanned surface vehicles (USV's), such as airboats, rather than unmanned underwater vehicles (UUVs), are better suited to this operational domain. Airboats are flat-bottomed boats that use an above-water fan to propel themselves forward safely and effectively through shallow or debris-filled water.

The challenge, then is to be able to construct and deploy small and capable airboats at a low cost. The low cost is particularly important for feasibly deploying sizable teams capable of covering large areas. In the immediate future, three or four cooperative boats might be deployed to provide some situational awareness over a small area, but as algorithms become more scalable, hundreds or thousands of vehicles could be used to provide detailed situational awareness

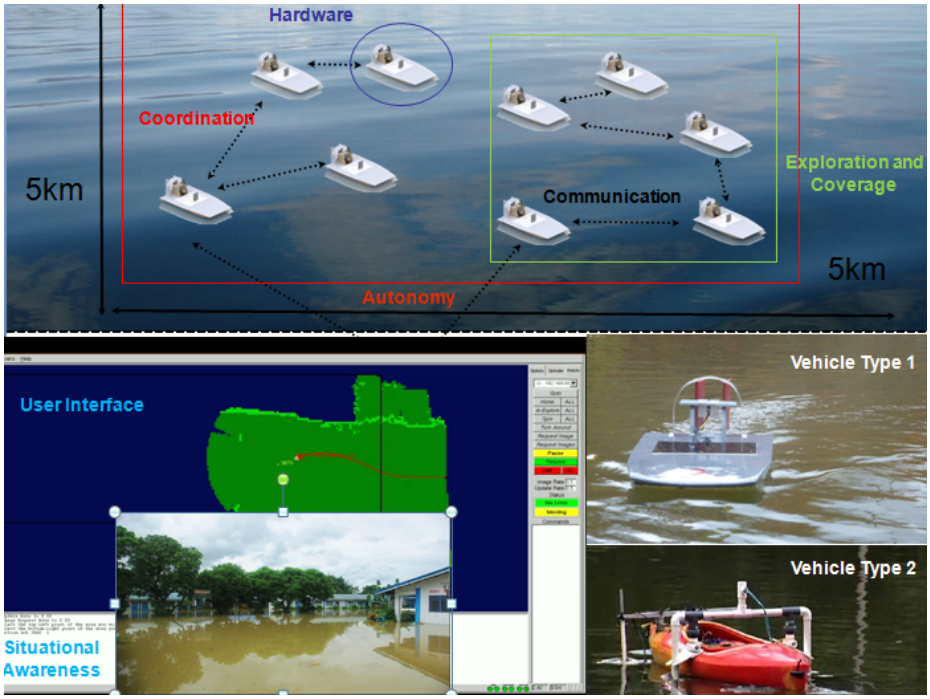


Fig. 1. Flood Disaster Mitigation Challenge

over dramatically larger areas. Longer term, a combination of autonomous aircraft and surface boats might be able to quickly cover a large area, being robust to obstacles and debris and able to safely navigate in places where large numbers of civilians are moving around.

2 Problem Definition

The contribution of this paper is not a description of algorithms for controlling boats for flood mitigation, but rather a technical description of the problem from the perspective of multi-agent systems. We present constraints descriptions for sub-areas of MAS so as to provide target problem descriptions for algorithm developers to overcome. Specifically, we pose the challenge as a sustained distributed situational monitoring problem in flood affected areas over a relative large area (25 sq km) using a large team (25-50) of autonomous watercrafts with minimal human oversight. This primary system objective leads naturally to a number of specific technical challenges that must be overcome to successfully complete the challenge (see Figure 1). Our challenges to the research community are:

Vehicle Design: *How to develop a robotic watercraft platform specific to the demands and requirements of flood disaster mitigation activities?*

Vehicle Intelligence: *How to robustly control a single boat and perform fundamental tasks of way-point following, collision avoidance, information gathering, energy management, payload management, etc?*

Team Intelligence: *How to develop efficient techniques for coordinating a team of airboats, identifying the scale of autonomy, situational awareness, establishing and maintaining communication network, path planning and task allocation?*

We need to bring together mature technologies from different areas including *autonomy, robotic watercraft, coordination, networking, fault-tolerance, coverage, exploration and human-robot interaction* towards building an integrated, large scale, autonomous system capable of monitoring and payload delivery in dynamic environments over an extended period of time.

3 Vehicle Design

For the principal task of providing situational awareness, identifying potential victims, and augmenting current first responder capabilities, any prototype design should address essential components pertaining to *cost*, *sensing*, *payload*, *power*. Relative to other types of vehicles, surface boats or *airboats* are simple, robust and reliable. Cognizant to the operating domain, we argue that the the overall size and weight of the boats should be relatively small. By keeping the vehicles small, most safety issues can be avoided simply by ensuring that if there is a collision, then the relatively low operational speeds cause only minor damage. Specifically, the following issues must be considered:

- Sensors are a major component of the cost of the boat and are most likely to fail. Hence, there is significant advantage to be able to use simple and cheap sensors. However, this typically makes control more difficult. For effective operation, the vehicles should have four essential capabilities: *adequate range*, *communications*, *navigation*, and *environmental sensing*.
- The surface boats must be inexpensive and reliable so that we can build, maintain, and deploy a large numbers of them at a fraction of the cost of commercial alternates.The challenge is to find the right balance between using low-cost simple sensors and making the boat useful.
- Using large numbers of boats over extended periods will inevitably result in individual failures. The large number of hours that boats will be in the water in this project will provide data that quantifies the type, nature and rate of failures, providing key input to future development and for evaluating the effectiveness of the overall solution. Hence, the boats must be easy to construct, maintain and to repair.
- Power will be a limiting factor in boat performance, having a direct impact on attributes such as range and speed. The boats must be capable of traversing a set distance within a given amount of time in order to preserve the quality of the acquired information. Based on initial analysis of the problem domain, we estimate the operational range for exploration to be in the range of 5 km

over the course of several hours. This translates to average operating speeds of $2 - 3$ km/hr, which confirms to operational safety standards.

- The boats must have some payload capability for dropping off essential supplies as well as picking up water samples for contamination analysis.

Vehicle navigation in the intended environment involves two basic challenges pertaining to obstacle avoidance, negotiating surface debris, and adapting to "waterscape" changes as flood waters rise or recede. Development of novel individual and joint control schemes to handle these requirements is therefore a key area of research. Control in an aquatic environment has been studied widely in larger surface vehicles, primarily at oceanic scales, and complex active control problems in semi-autonomous vehicles. In the former, a "follow-the-carrot" style approach is often sufficient, as the larger vehicle size coupled with high confidence measures in vehicle and obstacle position makes trajectory errors small. However, there are several aspects of the flood mitigation domain that warrant further research. The small size of the boat and unstructured nature of the environment mean that a-priori planning is limited and necessarily uncertain. It is impractical to precisely map and localize the boat with sufficient precision to avoid all obstacles, making reactive control a necessity for short-range obstacle avoidance. Second, while complete avoidance is necessary for safety at higher speeds, it is primarily a matter of efficiency at lower speeds. Low-speed collisions with obstacles often do not lead to damage or entanglement, and it is possible for boats to bump objects occasionally at low speeds. Finally, the high speed of the boats and water currents and eddies, relative to boat size, make vehicle dynamics significantly less predictable. Large discrepancies in trajectory may be effected by small changes in ambient current and vehicle hydrodynamics. Thus, building accurate motion models through physics alone may not be practical or possible.

In developing control strategies, it is important to note that given the nature of these vehicles, the objectives of the strategy are to minimize power consumption and travel time under the constraint of safe traversal of the environment. The uncertainty and complexity of realistic aquatic motion models make learning-based approaches particularly attractive. Methods such as reinforcement learning offer a number of advantages, including the ability to adapt to changing environmental conditions such as water currents, changes in payload, and the possibility of transferring learning between vehicles. While the environment is unstructured at a fine scale, it is evident that particular classes exist by virtue of human development. Areas such as towns provided a semi-structured but dense lattice where potential for human interaction is high, while rural areas or existing waterways will be less dense in terms of static obstacles, but potentially rich in manned vehicle traffic. In order to move efficiently in these and other classes of environment, it may be necessary to implement hybrid or layered control approaches that explicitly model the different areas.

4 Vehicle–Level Intelligence

Flood environments are uniquely cluttered, and for practical purposes, unknown a-priori. Situation awareness constitutes the collection of data that enable operators to better characterize the state of flooded areas for the prioritization of emergencies, the allocation of resources, and the establishment of further relief infrastructure. Information such as imagery of affected areas, traversability roadmaps or obstacle maps, and sampling of environmental factors such as water quality and temperature can play a role in creating this representation. The on-board sensor package must ensure vehicle safety, and, moreover, provide useful situation awareness to human operators to facilitate rescue and response activities. Safe traversal of the environment requires ego-motion estimation and obstacle detection. The former problem is made challenging by the small scale of the vehicles and obstacles, and the chaotic nature of water currents. Unlike ground robots, aquatic surface vehicles drift with prevailing currents that are hard to measure and model, resulting in motion that cannot easily be predicted. Absolute positioning using GPS can significantly assist in correcting for this, but commercial receivers at the price point of the vehicle cannot resolve to the require accuracy. In addition, bearing cannot be directly estimated from these measurements, and is also subject to drift. Obstacle detection requires sensors to look out over the surface of the water for potential hazards, both above and at the surface. The distance at which this can be done safely is a critical factor in determining the maximum rate of travel of the vehicle.

The sensory constraints can be overcome by novel estimation and filtering methods that will enable useful streams of information to address these three tasks while incorporating only data provided from relatively low-cost sensors. Such sensors include many of the staples of ground robotics, including MEMS accelerometers, gyros, and magnetometers, physical contact switches, local optic flow sensors, in-air IR and sonar range finders, and stereo or monocular vision. Interestingly, many of these sensors are already present in modern mobile devices such as smart phones, making these embedded platforms even more attractive as a low-cost, integrated solution to consider.

A key area of study for the MAS community lies in using vehicles jointly to improve perception of local and environmental features. While boats cannot control many aspects of the environment, they can (a) exchange information that allows other boats to reduce uncertainty or correlate features between vehicles and (b) use other boats for relative localization through direct or indirect relative tracking (i.e. boat-mounted fiducials). Since large teams of these vehicles are expected, this may be a very powerful alternative to more expensive local sensing strategies. Successful joint sensing strategies can bring down the cost of the vehicles by requiring less of individual sensors.

5 Team–Level Intelligence

This section outlines the key sub-problems that arise at the level of the team – the boats and the human controllers. We break the challenge problem into

six key areas, each of which is an important problems in its own right and has already received significant attention from the MAS research community. These are:

1. *Autonomy and human interface*,
2. *Situational awareness*,
3. *Communications & networking*,
4. *Path planning*,
5. *Team planning*,
6. *Task allocation*.

The Airboats Challenge brings with it the domain specific constraints of operating in the midst of a flood disaster and the added complexity of integrating the sub-problems to produce a coherent system. This is a key reason why Airboats Challenge is so appealing as a challenge problem: although big, the global MAS problem is relatively modular, and does not need to be treated as a whole. Instead, each of the sub-problems we have identified can be tackled by separately, and this paper is written in the hope that many separate research groups with different interest and expertise will all be able to contribute.

5.1 Autonomy and Human Interface

One of the most interesting research challenges for the Airboats Challenge is to design a system that will be able to provide the appropriate *level* of autonomy for the agents. A set of exemplar tasks follow to demonstrate the necessary types of autonomy, as well as identifying the underlying research challenges imposed by these behaviors.

Task 1: Autonomous exploration. Prior knowledge about the terrain might be useful for identifying potentially traversable areas. However, realistically in such a fluid environment, many areas may become (un)traversable by the boats, necessitating alternate agent behavior. Exploration will be essential towards updating the local and, subject to communication constraints, the global maps. As a consequence the team, on an individual and sub-team levels, must be capable of performing autonomous exploration of its surrounding environments. Interestingly the autonomy level for exploration could be varied from fully-autonomous to tele-operated on a case-to-case basis.

Task 2: Human interaction. The agents must be capable of detecting and interacting with civilians towards providing accurate situational awareness to the responders. To this end, agents should be able to integrate information coming from the on-board sensors (e.g. cameras and microphones) for detecting survivors. Civilians should then be approached to provide information to the base station about their position, photographs of the area, voice transmissions, etc. This throws up very interesting challenges in identifying the most effective manner of interaction with humans. Moreover, the airboats must be able to autonomously identify adversarial behaviors. Potential countermeasures include sending a "SOS" signal, identifying potentially hostile people by taking pictures for later identification, etc.

Task 3: Self awareness. The unpredictability of communication range dictates that the agents be capable of autonomously returning to the base station or to dynamically determine an alternate rendezvous point. The ability to return to the base station would be necessary for enabling agents to return with collected information in the absence of communication connectivity. Direct communication to base station may be infeasible due to limited network structure and the agents will likely need to coordinate to construct a network infrastructure. This might require autonomous task-switching on a sub-team level. Furthermore, in Airboats Challenge , operations will be carried out over a long time scale; as a consequence the agents should be self aware and capable of switching states to operate for days, up to approximately two weeks. Agents will need to return to the base station for recharging, resupply, repair, etc.

Task 4: Situational awareness. The agents should be able to recognize dangerous situations and activate specific behavior. For example, the agents should be able to recognize when the agent is going to hit an obstacle, or when the agent's localization has failed. Actions to address such situation may include sending broadcasting an alarm signal, quickly changing direction, or activating a search routine. It is important to note that "recognizing a situation" here refers to the agents ability to reason about abstract concepts such as "the agent is in trouble" and "the agent is having difficulty navigating to point X". Moreover the agent must be able to identify these crucial situations quickly, reliably and using inexpensive sensors.

While the above autonomous behaviors have been broadly addressed in the autonomous agents and robotics community, we believe that deploying a systems which is able to perform this kind of autonomous tasks in Airboats Challenge poses many interesting research challenges. Among the many issues which prevents direct application of off-the-shelf solutions in this domain, the most important address are: (a) algorithms must work in real time with low-power devices and will be unlikely to find guaranteed optimal solutions; (b) humans operators not experts in controlling nor repairing the agents – any proposed solution must be very simple and reliable; (c) agents should use adjustable autonomy: when human operators can provide help, the agents should try to take advantage of their expertise; (d) agents must work in a broad range of non-optimal conditions: for instance, if the weather or lighting conditions change, the agents should continue to work, even if it reduces their efficacy.

5.2 Situational Awareness

Providing situational awareness (SA) for human operators is a primary goal of the system. The task involves collecting information about the environment and getting it to the operators to allow them to understand the disaster that they are dealing with. SA also has a role to play in informing the networking, path planning and task allocation problems of the Airboats Challenge . Specifically, SA is is used to put constraints on the set of feasible paths, and consequently network configurations, that the agents can take, and also to assign levels of importance to different tasks. From a MAS perspective, we are only interested

in the task of collecting information and communicating it back to the human operators and among the agents members; We are not interested in the important human factors issues related to its presentation.

The rate of change of different parts of the environment will be very different, with some requiring new information be collected and transmitted regularly and others only requiring an occasional visit for new information. Primarily, we anticipate that cameras will provide the majority of the data for getting situational awareness. Often still imagery will be sufficient, however under certain situations video data might be essential to to do situational awareness, for e.g. the rate of water movement. However, other sensors such as microphones or wind-gauges might provide useful information. In flood disasters where water sits for long periods of time, it may be necessary to collect water samples to allow for checking for diseases. This would necessitate boats bringing samples all the way back to operators for analysis.

Two additional factors make the SA problem more complex from the MAS perspective. First, not all areas are as important as others. For example, areas that are likely to have high population density or will be critical for moving humans around the environment are more important than open areas where humans are not expected. The relative importance of different areas maps to preferences on locations to visit. Second, incoming data might be ambiguous or unclear and humans might request clarification in the form of additional information about an area. Both of these factors link SA to task allocation, in we expect that SA information will be used to identify tasks of high importance or value.

The problem can be formalized as follows. Consider the world to be made up of a set of locations, $L = \{l_1, \ldots, l_n\}$. For each $l_i \in L$, a cost function $C_{l_i}(t) \rightarrow \mathcal{R}$ defines the value of not getting information on that location for a length of time t. Each time the location is visited, the function resets and might change. For example, areas found to have nothing of interest, will reset to a function that increases very slowly over time, while areas with a lot of interest will reset to functions that increase very rapidly over time. The system will not know in advance how the function will reset after it is visited, but we assume it will know as soon as it is visited. Another function, $V_{l_i}(t) \rightarrow \mathcal{R}$ gives the relative value of that location over time. The overall optimization is to minimize the cost of not seeing locations multiplied by the value of the location over time. That is,

$$\min \sum_{t=t_s}^{t=t_e} \sum_{l_i \in L} C_{l_i}(t) V_{l_i}(t)$$

where we assume time is discretized and t_s and t_e represent the start and end of the mission respectively.

5.3 Communications & Networking

The networking aspect of the flood mitigation problem is working out how to configure the boats to form an ad hoc network, in addition to making use of

any available infrastructure such as cell phone networks, to allow communication among the team and human operators. This component of the Airboats Challenge is of fundamental importance, since without a functioning communication network, the other team-level sub-problems – the situational awareness, path planning and task allocation capacities – of the system will be severely curtailed. Furthermore, in addition to running the algorithms that address these problems, maintaining a communications network itself places hard constraints on the solutions to the path planning and task allocation problems.

Regarding the SA goals of the system, we anticipate that the boats will be collecting a lot of potentially useful information, and will benefit from tight coordination with other boats when possible. However, this is likely to lead to there being far more data than communication bandwidth. The physical locations of the boats will create the physical network, hence the networking challenge is fundamentally to work out how the boats should move get the "best" network structure. Clearly, the positions of the boats cannot be dictated solely by the requirements of the networking, since this will impede their ability to do their primary task. However, it may be possible or necessary to dedicate some boats solely to the task of being network routers. Low-level issues of how to efficiently communicate data or to create more powerful transmitters are considered beyond the scope of the multi-agent problem. Similarly, it is anticipated to be the case that energy use for receiving and sending data is negligible versus energy costs of moving the boats around.

One network concept sometimes used in environments without wireless infrastructure is the idea of *delayed communication*, where robots will hold onto information and actively plan to get back to a location to transmit that information at some later time. Delayed communication is likely to be a useful mechanism in the flood mitigation problem, especially since delays on the order of minutes are unlikely to be important.

The wireless network connecting the boats and ground stations is required for sending three types of messages. First, messages are required to get information from the robot sensing the information to the boat or human who can utilize that information. Second, messages are required to facilitate coordination between the boats. Third, messages are required for human override of autonomous action, e.g. tele-operation. Appropriately designed coordination algorithms should mean that no particular message is absolutely critical to overall operation, instead each message will have some value to the team. Messages should only be delivered once and may pass through intermediate nodes to get to their destination. Naturally, there will be some time before which a message has no value and often a time after which a message has no value, e.g. information has become stale or opportunity for coordination has passed.

The movement of the boats around the environment and the availability of infrastructure, e.g. mobile phone towers, induces a network that changes over time. Because small vehicles moving in a complex environment, carefully placed mobile phone towers and human operators will have dramatically different communications equipment, it is not reasonable to assume that links are symmetric.

There will be constraints on edges in the network which restrict traffic on that edge. For example, a boat may have links to four other boats, but it cannot communicate with them at the same time, since the same wireless medium is being used for each link. More complex models might include constraints that capture interference between links degrading capacity, e.g. two different boats cannot broadcast on the same channel at the same time, but we believe these details are practically unimportant for this domain.

The network aspect of this problem is focused on providing the infrastructure to allow message delivery, other parts of an overall system will actually determine which messages are delivered. Thus, we have to think about the problem of optimizing the network structure as one of optimizing the potential for message delivery. This optimization must include the possibility that messages fail to be delivered and that the coordination is inefficient.

5.4 Path Planning

Path planning sits at the interface of vehicle- and team-level intelligence. For example, some path plans can be generated independent of other agents, such as return routes to a base station, while others require tight coordination of the actions of several agents, as when network connectivity requirements are paramount. Furthermore, solutions to a path planning problem may be constrained by environmental conditions (garnered from SA), network considerations, task requirements and vehicle power constraints.

The path planning component for a boat will be impacted by all other parts of the system, e.g. the networking component will tell it how it must move to maintain an appropriate network and the task allocation component will tell it what it must achieve in the environment. In an ideal solution, feedback from a path planner would impact other parts of the system, e.g. by indicating that it is expected to take the boat a long time to perform a particular task, hence it is better allocated elsewhere.

To generate even an independent path, the boat must deal with partial observability, because the environment is not perfectly known and action uncertainty since movement through the environment is inherently uncertain. Given multiple tasks, e.g. places to take observations or deliver supplies, the robot must appropriately order its tasks for best overall performance. It must also carefully balance risks, e.g. taking unknown but potentially more direct routes or moving at higher speed, time to complete time-sensitive tasks and the need to keep the boat intact for future efforts. The environment will not be completely static, making it necessary for the path planner to reason intelligently about the impact of any possible obstacles in advance and planning around them when they occur. Planning will need to occur over significant amounts of time, since boats may travel to tasks that take on the order of hours to reach. Therefore, in its most general formulation the single vehicle path planning problems can be considered as a Partially Observable Markov Decision Process (POMDP).

The path planning will be mostly individual but cooperation could dramatically improve overall performance. For example, if it is not known whether a

particular route is traversable, it may be optimal for one boat to first go down that street while others wait or take longer, safer routes. This type of exploit versus explore tradeoff is often studied in the literature, but not in the context of such complex individual planning. Cooperation will also be required to avoid hindering progress of other boats, e.g. impeding progress down a narrow alley, A natural and general framework for the multi-agent path planning problem is that of Decentralized POMDP. However, Dec-POMDPs are known to be intractable in general settings [1]. Therefore, a main research issue here is to find alternative formalizations or approximate techniques that can provide good solutions while meeting the real time constraints of the application.

Finally, while we anticipate that the primary focus of the path planner will be coming up with a path that achieves all the objectives of the boat at a minimal cost, some attention will need to be paid to actually being able to move the boat around the environment. While we consider issues of control outside of the scope of the MAS problem, environment features such as currents in the water, winds and narrow passageways will significantly effect what the boat can achieve and should be considered as a part of the path planning process. For example, in an area expected to have significant currents, it is not reasonable to plan or expect a fast path directly across or against the current.

5.5 Team Planning

Disaster response domains, like the one discussed in this paper, typically involve multiple sub-teams of agents working together towards achieving a common goal, saving lives and disaster mitigation. Each team-member has specific capabilities particularly suited to certain tasks. While some tasks are independent of each other, other tasks may be related by different constraints. As agents move about the environment, they have a direct influence on other team members from tightly-coupled scenarios [2, 8, 19] where multiple agents are required to complete a task, to loosely-coupled ones where the action of one agent might block the movement of others [6]. Team planning addresses the problem of decomposing a high-level set of goals into smaller independent, primitive tasks.

5.6 Task Allocation

Task allocation impacts the performance efficiency of teams in significant ways. Allocating vehicles to different tasks in an efficient and effective way is a crucial issue for the Airboats Challenge . More than any other sub-problem, task allocation connects together the components of the team-level intelligence of the system: The set of tasks may represent both SA and networking goals; The cost and benefits of completing tasks are computed using outputs from path planning and SA problems, and may be constrained by network considerations; and, we expect there to be human oversight of the weights attributed to tasks.

Task allocation is a very well known and widely studied problem in MAS, and many solutions have now been proposed, however, in the Airboats Challenge scenario, the task allocation problem is particularly challenging as the system

is composed of a large number of vehicles that will be equipped with cheap and low power devices and will have to coordinate in a highly dynamic and partially unknown environment.

Task allocation is usually formalized considering a set of tasks $\mathcal{T} = \{T_1, \cdots, T_m\}$, a set of agents $\mathcal{A} = \{A_1, \cdots, A_n\}$ and a reward matrix $R = \{r_{ij}\}$ where r_{ij} indicates the reward achieved by the system when agent A_i execute task T_j. An allocation matrix $A = \{a_{ij}\}$ defines the allocation of agents to task with $a_{ij} \in \{0, 1\}$ and $a_{ij} = 1$ if agent A_i is allocated to task T_j. The goal of the system is then to find

$$\arg\max_{A} \sum_{i=1}^{|\mathcal{A}|} \sum_{j=1}^{|\mathcal{T}|} r_{ij} a_{ij}$$

Moreover, a set of constraints $\mathcal{C}$ usually describes valid allocations of agents to tasks, for example, one task could be executed at most by one agent or exactly k agents, or completing a task could be outright infeasible because of constraints on the actions of an agent. Therefore the above optimization must be performed subject to $\mathcal{C}$.

A first important challenge for the task allocation approach is to deal with a dynamic environment, where tasks appear, disappear and the reward to execute them may change during the mission execution: in the Airboats Challenge domain, vehicles will deal with tasks such as searching for civilian in a predefined area, approaching a group of detected civilians, collaborating with a set of other vehicles to relay information to the base station and so forth. These tasks are not known before hand and will be discovered during the mission; in addition, failures of vehicles should be taken into account: vehicles could be potentially stolen or the communication infrastructure could experience temporary break down. Hence, the above problem formulation must take time into account and one way to express this is to have that agents, tasks, reward matrix and consequently allocation matrix dependent on time and then find a series of allocation, one for each time step, such that the sum of reward over time is maximized:

$$\arg \max_{\{A^{t_s}, \cdots, A^{t_e}\}} \sum_{t=t_s}^{t_e} \sum_{i=1}^{|\mathcal{A}^t|} \sum_{j=1}^{|\mathcal{T}^t|} r_{ij}^t a_{ij}^t$$

Therefore, the solution algorithm should be capable of continuously monitoring the environment and adapt the task allocation solutions to unexpected changes.

Second, in the Airboats Challenge vehicles should be able to take decision on their own, without necessarily relaying on information, or directives, from the base station; moreover, such decentralized task allocation approach must be designed to run on low power, cheap devices (such as smart phones). The low cost devices combined with the large scale operational domain, eliminates the use of intense computation and communication resources, typical of complete

algorithms[1], as their coordination overhead (computation time, message number and size), would be simply unacceptable in this scenario.

Third, since vehicles act in the real world without a complete knowledge of the environment, the benefit that the whole system would acquire for a given allocation of tasks is very hard to predict: vehicles are uncertain of their action outcome (e.g, a boat might be stalled while traveling towards an interesting area) and, more important, even if a task is completed successfully the reward for the team might be different than what is expected (e.g. it could be very hard to decide which group of civilians is more in need of help without having accurate information about their situation).

Finally, vehicles might need to form coalitions to execute tasks. Consider the example where agents might need to form sub-teams to approach a group of civilians while maintaining connectivity with the base station, or to search a given area of the environment where there is a high chance of discovering civilians. In our formulation coalition effects can be expressed by representing rewards as a set of functions instead of as a matrix: $R^t = \{r_j^t(a_{1j}^t, \cdots, a_{|\mathcal{A}|^t j}^t)\}$ and considering the following objective function[2]:

$$\arg \max_{\{A^{t_s}, \cdots, A^{t_e}\}} \sum_{t=t_s}^{t_e} \sum_{j=1}^{|\mathcal{T}^t|} r_j^t(a_{1j}^t, \cdots, a_{|\mathcal{A}|^t j}^t)$$

Coalition formation is known to be a very hard problem to solve and current solutions can find optimal coalitions only for relatively small number of agents (in the order of 30) [20], so there is a clear need for approximate solutions in this context.

As mentioned above, there exists many potential approaches to address our task allocation problem, that range from approximate DCOP solution techniques [21, 5, 11, 4], to decomposing the problem as mixed integer linear programming problems [12, 15], market based approaches [9, 16], hybrid approaches [14, 13], etc and that have been used in similar application domains. Despite the fairly rich suite of algorithms for addressing team planning, the dynamic and complex environments, continuous configuration and observation spaces, and relative large team sizes coupled with limited computing and sensing far exceed the complexity handled by many existing approaches. Deciding how to represent the problem and determining which classes of algorithms are effective remains an open area of research.

6 Discussion

In order for the problem to be accepted in the MAS community as an open challenge problem, a case-study and subsequent feasibility analysis of the various de-

[1] With complete algorithms we are guaranteed to find the optimal solution

[2] If we aim to solve this problem using linear programming techniques we need to represent the reward for each possible coalition, this results in a combinatorial element in the complexity of the problem.

scribed components including vehicle design, intelligence and team-intelligence for the outlined problem must be performed. The feasibility analysis would allow us to identify system bias and weight associated with individual components as it affects overall system performance. This subsequently would allow us to formulate the challenge as a mathematical problem that can then be modeled for a simulator or real-world system. Genuine practical success may require that a modular open source architecture is developed into which various algorithms can be inserted. The development of such an architecture would also separate the hardware development from the software development and allow for researchers to collaborate and focus on specific domain. As part of the development framework, we are working on building a realistic simulator for the project as well as developing a prototype vehicle model.The simulator is intended as a open source resource that will allow the community to test and evaluate individual component algorithms as well as a full-system model on a common platform.

Furthermore, the feasibility study will also address an important component of any multi-robot system, *evaluation metrics.*The mission critical nature of the operating domain dictates the need for a high operating efficiency for the Airboats Challenge . In order to objectively evaluate operational performance, there is a need to have a well-defined and detailed set of metrics. Based on observations from earlier work in developing metrics for multi-robot teams [10]. we believe that for the challenge problem the success metric should be a combination of qualitative and quantitative measures that can be used to analyze, evaluate, and subsequently improve performance of a team of airboats towards the overall goal of mitigating disasters during flooding. The goal therefore is to identify a set of flexible tools for researchers to use for in-depth system analysis. In addition, it is important to identify evaluation criteria that can help determine the quality of a metric in terms of the domain specific constraints, comprehensive understanding, construct validity, statistical efficiency, and measurement technique efficiency [3]. The idea of identifying generalizable classes allows researchers to independently evaluate specific sub-problems that constitute the challenge.

Finally, the unfortunate prevalence of floods will give many opportunities for solutions to be field tested, requirements to be updated and new designs to be explored. Beyond constrained environment testing, real world evaluation in places like the Philippines are essential for extended evaluation.

7 Conclusions

In this position paper, we present a challenge problem of using cooperative airboats to perform flood disaster mitigation. Floods are the natural disaster with the biggest annual impact and dis-proportionally affect the economically backward. We have outlined the key technical challenges and argued that the research from the MAS community is well suited to tackle many of the technologies that are necessary to develop a low-cost, high-impact solution. We are currently developing prototype simulators and robots to work on this problem and anticipate initial testing to occur in the near future. We plan to make the

simulation environment open for anyone in the community to test and contribute algorithms. It is also planned to make it possible for anyone in the community to provide code for key MAS functions on the robots themselves. This will provide both a realistic and important test for the algorithms and allow the MAS community to make a genuine contribution to the world.

References

1. Daniel S. Bernstein, Shlomo Zilberstein, and Neil Immerman. The complexity of decentralized control of markov decision processes. In *Proc. of UAI-2000*, pages 32–37, 2000.
2. M Bernardine Dias, R. Zlot, N. Kalra, and A. Stentz. Market-based multirobot coordination: A survey and analysis. *Proceedings of the IEEE*, 94(7):1257 –1270, July. 2006.
3. B. Donmez, P. Pina, and M. L. Cummings. Evaluation criteria for human-automation performance metrics. In *Proceedings of Performance Metrics for Intelligent Systems Workshop*, 2008.
4. A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *Seventh International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-08)*, pages 639–646, May 2008.
5. S. Fitzpatrick and L. Meetrens. *Distributed Sensor Networks A multiagent perspective*, chapter Distributed Coordination through Anarchic Optimization, pages 257–293. Kluwer Academic, 2003.
6. Owen Holland and Chris Melhuish. Stigmergy, self-organization, and sorting in collective robotics. *Artif. Life*, 5(2):173–202, 1999.
7. Geoffrey Hollinger and Sanjiv Singh. Towards experimental analysis of challenge scenarios in robotics. In *12th International Symposium on Experimental Robotics*, December 2010.
8. Dapeng Jiang, Yongjie Pang, and Zaibai Qin. Coordinated control of multiple autonomous underwater vehicle system. *Intelligent Control and Automation (WCICA), 2010 8th World Congress on*, pages 4901 –4906, July. 2010.
9. E. Jones, M Bernardine Dias, and A. Stentz. Learning-enhanced market-based task allocation for oversubscribed domains. In *International Conference on Intelligent Robots and Systems, 2007. IROS 2007.*, November 2007.
10. B. Kannan and L. E. Parker. Metrics for quantifying system performance in intelligent, fault-tolerant multi-robot teams. *International Conference on Intelligent Robotics and Systems*, November 2007.
11. C. Kiekintveld, Z. Yin, A. Kumar, and M. Tambe. Asynchronous algorithms for approximate distributed constraint optimization with quality bounds. In *Proceedings of the Ninth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS–10)*, pages 133–140, Toronto, ON, Canada, 2010.
12. Mary Koes, Illah Nourbakhsh, and Katia Sycara. Heterogeneous multirobot coordination with spatial and temporal constraints. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI)*, pages 1292–1297. AAAI Press, June 2005.
13. Mary Koes, Katia Sycara, and Illah Nourbakhsh. A constraint optimization framework for fractured robot teams. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 491–493, New York, NY, USA, 2006. ACM.

14. G. Ayorkor Korsah, Balajee Kannan, Imran Fanaswala, and M Bernardine Dias. Improving market-based task allocation with optimal seed scheduling. *Intelligent Autonomous Systems 11 (IAS-11)*, 0:249 – 259, August 2010.
15. G. Ayorkor Korsah, A. Stentz, M Bernardine Dias, and Imran Fanaswala. Optimal vehicle routing and scheduling with precedence constraints and location choice. In *ICRA 2010 Workshop on Intelligent Transportation Systems*, May 2010.
16. M. Lagoudakis, E. Markakis, D. Kempe, P. Keskinocak, A. Kleywegt, S. Koenig, C. Tovey, A. Meyerson, and S. Jain. Auction-based multi-robot routing. In *Proceedings of the International Conference on Robotics: Science and Systems*, pages 343–350, 2005.
17. David P. Miller, Charles Winton, and Jerry Weinberg. Beyond botball. In *AAAI Spring Symposium, Robots and Robot Venues: Resources for AI Education*, 2007.
18. Robin R. Murphy, Eric Steimle, Chandler Griffin, Charlie Cullins, Mike Hall, and Kevin Pratt. Cooperative use of unmanned sea surface and micro aerial vehicles at hurricane wilma. *Journal of Field Robotics*, 25(3):164–180, 2008.
19. Lynne E. Parker and Andrew Howard. Experiments with a large heterogeneous mobile robot team: Exploration, mapping, deployment and detection. *International Journal of Robotics Research*, 25:431–447, 2006.
20. T. Rahwan, S. Ramchurn, N. Jennings, and A. Giovannucci. An anytime algorithm for optimal coalition structure generation. *Journal of Artificial Intelligence Research (JAIR)*, 34:521–567, April 2009.
21. Paul Scerri, Alessandro Farinelli, Steven Okamoto, and Milind Tambe. Allocating tasks in extreme teams. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 727–734, 2005.
22. United Nations University. *Two Billion People Vulnerable to Floods by 2050; Number Expected to Double or More in Two Generations Due to Climate Change, Deforestation, Rising Seas, Population Growth*. United Nations University, 2004.
23. Christopher Urmson, Joshua Anhalt, Hong Bae, J. Andrew (Drew) Bagnell, Christopher Baker, Robert E. Bittner, Thomas Brown, M. N. Clark, Michael Darms, Daniel Demitrish, John M. Dolan, David Duggins, David Ferguson, Tugrul Galatali, Christopher M. Geyer, Michele Gittleman, Sam Harbaugh, Martial Hebert, Thomas Howard, Sascha Kolski, Maxim Likhachev, Bakhtiar Litkouhi, Alonzo Kelly, Matthew McNaughton, Nick Miller, Jim Nickolaou, Kevin Peterson, Brian Pilnick, Raj Rajkumar, Paul Rybski, Varsha Sadekar, Bryan Salesky, Young-Woo Seo, Sanjiv Singh, Jarrod M. Snider, Joshua C. Struble, Anthony (Tony) Stentz, Michael Taylor, William (Red) L. Whittaker, Ziv Wolkowicki, Wende Zhang, and Jason Ziglar. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics Special Issue on the 2007 DARPA Urban Challenge, Part I*, 25(1):425–466, June 2008.
24. Christopher Urmson, Joshua Anhalt, Daniel Bartz, Michael Clark, Tugrul Galatali, Alexander Gutierrez, Sam Harbaugh, Joshua Johnston, Hiroki Kato, Phillip L. Koon, William Messner, Nick Miller, Aaron Mosher, Kevin Peterson, Charlie Ragusa, David Ray, Bryon K. Smith, Jarrod M. Snider, Spencer Spiker, Joshua C. Struble, Jason Ziglar, and William (Red) L. Whittaker. A robust approach to high-speed navigation for unrehearsed desert terrain. *Journal of Field Robotics*, 23(1):467–508, August 2006.