Real-Time Motion Analysis with Linear-Programming *

Moshe Ben-Ezra Shmuel Peleg Michael Werman

Institute of Computer Science
The Hebrew University of Jerusalem
91904 Jerusalem, Israel
Email: {moshe, peleg, werman}@cs.huji.ac.il

Abstract

A method to compute motion models in real time from point-to-line correspondences using linear programming is presented. Point-to-line correspondences are the most reliable motion measurements given the aperture effect, and it is shown how they can approximate other motion measurements as well.

Using an L_1 error measure for image alignment based on point-to-line correspondences and minimizing this measure using linear programming, achieves results which are more robust than the commonly used L_2 metric. While estimators based on L_1 are not theoretically robust, experiments show that the proposed method is robust enough to allow accurate motion recovery in hundreds of consecutive frames. The entire computation is performed in real-time on a PC with no special hardware.

1 Introduction

Robust, real-time recovery of visual motion is essential for many vision based applications. Numerous methods have been developed for motion recovery from image sequences, among them are algorithms that compute the motion directly from the grey level or general local measures [7, 11, 9, 2]. A second class of algorithms use feature points to recover motion [4, 8]. A probabilistic error minimization algorithm [15] can be used to recover motion in the presence of outliers. Another class of algorithms use explicit probability distribution of the motion vectors to calculate motion models [13].

Most of the methods cited above have problems when computing high-order motion models (e.g. an affine motion model or a homography): either they are sensitive to outliers, or the execution speed is very slow. An algorithm is presented to recover such high-order motion models from point-to-line correspondences using linear-programming. Point-to-line correspondences are robust in the sense that they are largely insensitive to aperture effects and to T-junctions, unlike the common point-to-point correspondences. Point-to-line correspondences can also approximate other measurements as well, such as point-to-point correspondences, correspondences with uncertainty, and spatio-temporal constraints.

The L_1 metric ($\sum |a_i - b_i|$) can be used with the point-to-line correspondences, and is much more robust then the L_2 metric ($\sqrt{\sum (a_i - b_i)^2}$), for example, the median minimizes the L_1 metric, while the centroid (average) minimizes the L_2 metric. L_1 estimators are not considered truly robust [5, 14] as they can be sensitive to leverage points. However, our experiments show that linear-programming, minimizing an L_1 error measure is, robust enough to compute accurate motion of hundreds of frames, even with large moving objects in the scene. Moreover, this is done in real-time on a regular PC.

The linear programming solver does not need an initial guess, which is required for iterative re-weighted least-square algorithms (such as M-estimators). The re-weighting stage, which is similar to motion-segmentation, is sometimes as hard as the motion recovery itself. Comparisons between estimators based on the L_1 metric and the robust LMS (Least Median of Squares) estimators show that global motion analysis using the L_1 estimator is only slightly less robust than the LMS estimator, but the L_1 computation is much faster.

The alignment process has two steps. (i) Computing correspondences and representing them as point to line correspondences, is described in Sec. 2. (ii) Converting the alignment problem into a linear program using the point to line correspondences, and solving it, described in Sec. 3. Sec. 4 describes experimental results and comparisons with other methods. Sec. 5 gives concluding remarks. Appendix A

*This research was funded by DARPA through the U.S. Army Research Labs under grant DAAL01-97-R-9291, Supported by Espirit project 26247 - Vigor

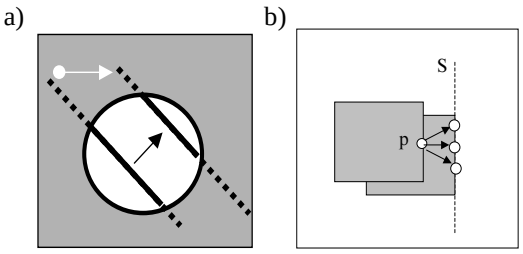

Figure 1. Aperture Effect. a) The white arrow represents the actual motion, while the black arrow represents the apparent motion. b) Point to line correspondence.

describes a possible explanation for the experimental insensitivity of L_1 motion estimators to leverage points.

2 Point to Line Correspondences

Point to line correspondence are used for their insensitivity to the aperture effect. This section describes the aperture effect, and the use of point to line correspondences to represent normal flow and uncertain correspondences.

2.1 Aperture Effect

Fig. 1.a describes the aperture effect: The apparent motion of a line when viewed through a small aperture is normal to the line. Therefore, the motion of point p in Fig. 1.b is defined only up to a line. The constraint on the displacement (u, v) such that point $p = (x, y)$ in the first image has moved to the straight line $S(x + u, y + v)$ in the second image and is defined by the line equation:

$$S(x+u, y+v) \equiv A(x+u) + B(y+v) + C = 0 \quad (1)$$

Without loss of generality we assume for the rest of the paper that $A^2 + B^2 = 1$ by normalization.

2.2 Normal Flow

A constraint on the optical flow in every pixel can be derived directly from image intensities using the gray level constancy assumption. This optical-flow constraint is given by [7, 11]:

$$uI_x + vI_y + I_t = 0 \quad (2)$$

where (u, v) is the displacement vector for the pixel, and I_x, I_y, I_t are the partial derivatives of the image at the pixel with respect to x, y and time.

Eq. (2) describes a line, which is the aperture effect line. When $I_x^2 + I_y^2$ are normalized to 1 the left hand side of Eq. (2) becomes the Euclidean distance of the point (x, y) from the line passing through $(x + u, y + v)$, which is also called the normal flow [1].

2.3 Fuzzy Correspondence

An optical flow vector between successive images in a sequence represents the displacement between a point in one image to the corresponding point in the second image. While it is difficult to determine this point to point correspondence accurately from the images automatically, a correspondence is usually assigned to the most likely point. For example, given a point in one image, the corresponding point in the second image will be the one maximizing some correlation measure. However, such correspondences are error prone, especially when other points in the second image have a local neighborhood similar to the real corresponding point.

A possible solution to this problem is to postpone the determination of a unique correspondence to a later stage, and to represent the uncertainty as given by the correlation values. In this case the correspondence will not be a unique point, but a fuzzy measure over a set of possible corresponding points.

The fuzzy correspondence of a point p can be represented as a matrix $M(p)$ (the fuzzy correspondence matrix). Each cell (i, j) of $M(p)$ corresponds to the probability that point p has a displacement of (i, j) [13]. In many cases the fuzzy correspondence matrix has a dominant compact shape: points on corners usually create a strong compact peak while edges form lines. A common case is an ellipse. While the fuzzy correspondence matrix contains all correspondence uncertainty, utilizing this information to its full extent is difficult.

To enable computation of global motion with linear programming, we propose an approximation of the fuzzy correspondence matrix by two point to line correspondences. This approximation is given by two lines: $S_i(x, y) \equiv (A_i x + B_i y + C_i) = 0 \quad (i = 1, 2)$ with two associated weights, W_i. Fig. 2.a-b show a distance map of each point from a line. The weighted sum of both distances forms an L_1 "cone". Each equidistant line on the cone is an L_1 "ellipse" with eccentricity proportional to the weights of the two lines, as shown in Fig. 2.c

This approximation can also be used to express point-to-point correspondence: $(x, y) \rightarrow (x', y')$ can be approximated by two point to line correspondences: between point (x, y) and the line $(x = x')$, and between point (x, y) and the line $(y = y')$, with weights $W_1 = W_2$.

Given a fuzzy correspondence matrix, the approximation by the sum of distances from two lines can be computed using a Hough transform as described in [3]. The following constraint can be used in the linear programming for the displacement for point (x, y):

$$W_1 S_1(x+u, y+v) + W_2 S_2(x+u, y+v) = 0 \quad (3)$$

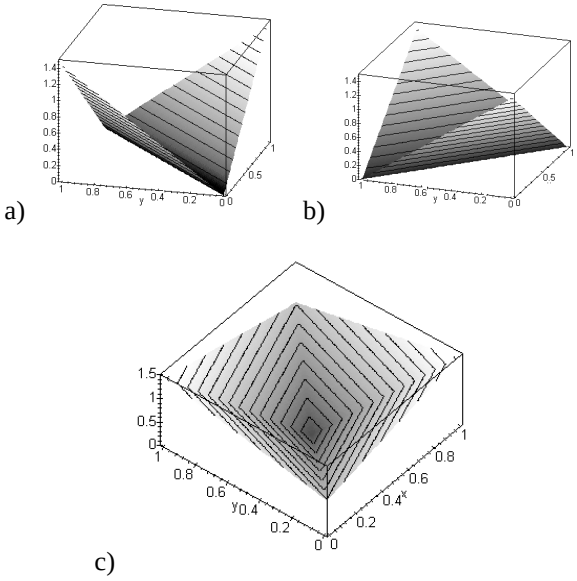

Figure 2. Approximation of an L_1 "ellipse" using two lines. a) Distance surface from one line. b) Distance surface from a second line. c) Weighted sum of distances from both lines, with weights of 2 from the first line and of 1 from the second line. Each equal-distance line is an L_1 "ellipse".

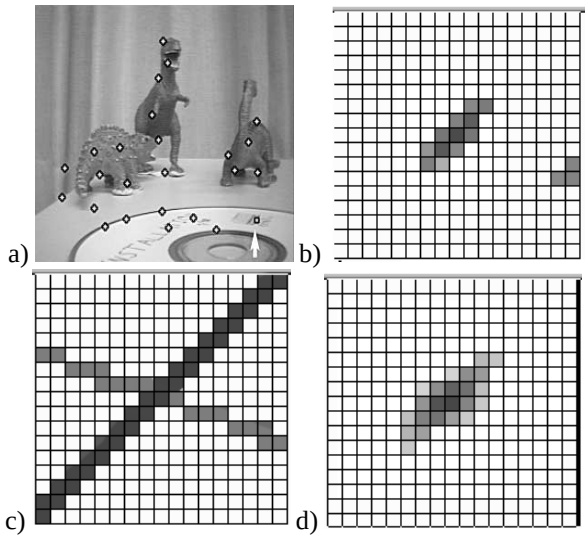

Figure 3. Approximations for fuzzy correspondences. b) The fuzzy correspondence matrix between successive images for the point that is marked with a white arrow in (a). c) The lines used to approximate the fuzzy correspondence matrix. Intensity corresponds to weight. d) Weighted sum of city-block distance from the two lines in (c) is used as the approximation of the fuzzy correspondence matrix in (b).

3 L_1 Alignment Using Linear Programming

The alignment process has two steps: (i) Computing correspondences and representing them as point-to-line correspondences. (ii) Converting the alignment problem into a linear-program using the point-to-line correspondences, and solving it. The first step was detailed in Sec. 2, and in this section the second step is described. In particular, we show how to compute an eight parameter 2D homography, which corresponds to the transformation between different views of a planar surface.

A homography H is represented by a 3×3 matrix, whose i'th row is designated H_i. A 2D point $p = (x, y, 1)^t$ (in homogeneous coordinates) located at image I_1 is mapped by the homography H into the 2D point p' in image I_2 as follows:

$$p' = \left(\frac{(H_1 \cdot p)}{(H_3 \cdot p)}, \frac{(H_2 \cdot p)}{(H_3 \cdot p)}, 1 \right)^t \quad (4)$$

The Euclidean distance of point p' from constraint line $S = (Ax + By + C)$, using Eq. (4), is given by the following equation which is zero when the alignment is perfect.

$$d(p', S) = \left(\frac{A(H_1 \cdot p)}{(H_3 \cdot p)} + \frac{B(H_2 \cdot p)}{(H_3 \cdot p)} + C \right) \quad (5)$$

Multiplying Eq. (5) by $(H_3 \cdot p)$ (which is non-zero for a finite size image) gives the following linear equation for the *residual error* of point p.

$$r(p', S) = d(p', S)(H_3 \cdot p) = A(H_1 \cdot p) + B(H_2 \cdot p) + C(H_3 \cdot p). \quad (6)$$

Since, in order to get a linear equation, we multiply the geometrical distance $d(p', S)$ by the (unknown) value $(H_3 \cdot p)$, the coordinates of p should be normalized to avoid bias [6]. Setting the residual error $r(p', S)$ to zero gives a linear constraint on the elements of the homography H that states that point p is mapped to point p' which is on the line S.

In order to recover the eight parameters of the 2D homography H, at least eight such point-to-line correspondences are required. Each point-to-line correspondence gives one point-to-line equation, and each point to two-lines correspondence (the linear approximation to fuzzy correspondence) gives two point-to-line equations. When more than eight point-to-line correspondences are given, H can be recovered using L_1 by solving the following minimization problem:

$$Minimize : \sum_{i=1}^{n} W_i |r(p'_i, S_i)|. \quad (7)$$

This error minimization problem is converted into the following linear program, where one constraint equation is written for each point-to-line correspondence:

$$
\begin{aligned}
&min: \ \textstyle\sum_{i=1}^{n} W_i(Z_i^+ + Z_i^-) \\
&\ s.t. \\
&A_i(H_1 \cdot p_i) + B_i(H_2 \cdot p_i) + C_i(H_3 \cdot p_i) + (Z_i^+ - Z_i^-) = 0 \\
&Z_i^+, Z_i^- \geq 0
\end{aligned}
$$

The expression $(Z_i^+ + Z_i^-)$ is the absolute value of the residual error, $r(p'_i, S_i)$. The is the result of the use in linear programming of only positive values [10]. Each residual error is represented by the difference of two non-negative variables: $r(p'_i, S_i) = (Z_i^+ - Z_i^-)$, one of which is always zero at the above minimum.

Notes:

1. When the constraints are of the form $Ax - b + Z = 0$, a basic feasible solution that satisfies the constraints is given by $x = 0, Z = b$. This enables the use of an efficient one-phase simplex algorithm to solve the problem.

2. This linear program can be used to minimize any linear equation: $Min(Ax - b)$. Parameter normalization or an additional constraint may be needed to avoid a zero root if $b = 0$.

3. If L is an $(M \times N)$ matrix, M is the number of constraint equations and N is the number of parameters $(M > N)$, then the linear program will have a total of $2(M + N)$ variables: $2N$ variables for the variable vector x, and $2M$ variables for the slack variable vector Z. The factor of two is needed since in linear programming each variable is represented by a difference of two non-negative variables.

4. The slack variable vector Z contains the error measures for each point and can be used for motion segmentation.

5. Additional linear constraints can be added to the recovered model. For example we can define a motion model that is more general than similarity but has bounded affine/projective distortions.

4 Experiments and Comparisons

To compare our model to existing point-to-point methods, we converted each point-to-point correspondence to two point-to-line correspondences according to Section 2.3. The panorama example used point-to-line correspondences computed from fuzzy correspondence matrices.

4.1 Mosaicing with Similarity Model

A panoramic image was created in real-time (10-12 frames / second) using a PC, as shown in Fig. 4.b. While the camera was scanning the scene, a large pendulum was swinging. The size of the pendulum was large enough to create about 15% outliers among the feature points. Since the stabilization algorithm used only frame to frame motion recovery, *any* error will cause the panorama to fail. Fig. 4 shows the pendulum (and its shadow) appearing/disappearing several times due to the swinging motion. However *all* frames were correctly aligned with a similarity model as can be seen by the objects that were not occluded by the pendulum.

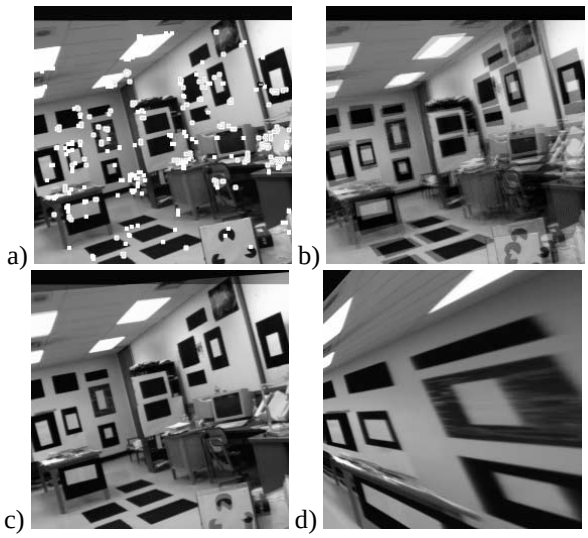

Figure 5. Computing homography using L_2 registration compared to L_1 registration. The same feature points were used in the L_1 minimization and the L_2 minimization. Both examples are single iteration output, no segmentation and no re-weighting were used. a) Selected feature points are marked on one images. b) The sum of the two original images to show their misalignment. c) The sum of the images aligned with the homography obtained using linear programming. d) Warping the second image towards the first image with the homography obtained using a least-square method.

4.2 Homographies: Comparison with L_2

This experiment compares the computation of a 2D homography using L_1 registration to the least-squares method for point-to-point registration. Given two images, the feature points were selected automatically from a bi-directional optical-flow field. Each selected point had a strong intensity gradient, and the optical flow from the first to the second image agreed with the optical flow from the second to the first image. Selected points are shown in Fig. 5.a.

The homography that minimizes the L_2 distance between the computed correspondences is shown in Fig. 5.d. It is completely useless due to outliers.

a)  b) 

Figure 4. Mosaicing examples. a) Point selected for motion computation. Four of the thirty points are located on the moving pendulum. b) Panoramic image that was created while a pendulum was swinging. The alignment was not affected by the outliers.

The L_1 alignment used the same data, but converted each point-to-point correspondence into two point-to-line correspondences (point (u, v) is converted to the two lines $(x = u)$ and $(y = v)$). Fig. 5.c shows the sum of the two images after alignment. The alignment is now very good, and can be compared to Fig. 5.b where the two images were added before alignment.

4.3 Computing Affine Motion from Normal Flow

An affine alignment between two images can be computed from normal flow by an iterative method. In this example 112 points residing on strong edges and spread evenly across the image were selected automatically. The iterations were as follows:

1. The normal flow is computed from spatio-temporal derivatives and represented by a line constraint as described in Sec. 2.2.
2. An affine motion model was computed using linear programming from the linear constraints.
3. The second image was warped toward the first image using the affine model.
4. Repeat steps 1-3 until convergence.

The iterations are necessary in this case since the accuracy of the normal flow depends on the accuracy of the spatio-temporal derivatives, which increases as the motion estimate becomes better. Fig. 6 shows the registration results for the L_1 registration by normal flow lines.

4.4 Efficiency Considerations

The linear programming approach is compared to the following well known probabilistic algorithm [15]:

Input: N matched pairs (either point to point or point to line).

Output: A linear motion model M^* of rank k that minimizes the sum of the absolute values of the residual errors.

Algorithm :

1. k pairs are selected randomly.
2. The motion model M is computed from the k selected pairs.
3. The sum of residual L_1 errors is computed from all pairs, using the recovered model.
4. The last three steps are repeated for t iterations.
5. Of all examined models, the model M^* having minimal error is selected.

If the probability of choosing an outlier is q, then in t iterations the probability P of having at least one perfect guess, with no outliers in all k selected points, is given by:

$$P = 1 - \left[1 - (1 - q)^k\right]^t. \tag{8}$$

Given the desired probability of success P, the number of necessary iterations t is given by

$$t = ln(1 - P)/ln(1 - (1 - q)^k). \tag{9}$$

The number of iterations required to reach a certain level of confidence is exponential in k and in q, and therefore this method is very expensive in these cases. The complexity of the linear program is polynomial in the number of constraints, which equals the number of correspondences. In most practical cases, however, the complexity is known to be nearly linear.

4.4.1 Synthetic Performance Experiment

In this test we tried to compare the actual performance of the two algorithms. The test consisted of the following synthetic data:
Number of matched pairs: 100 point to point correspondences.

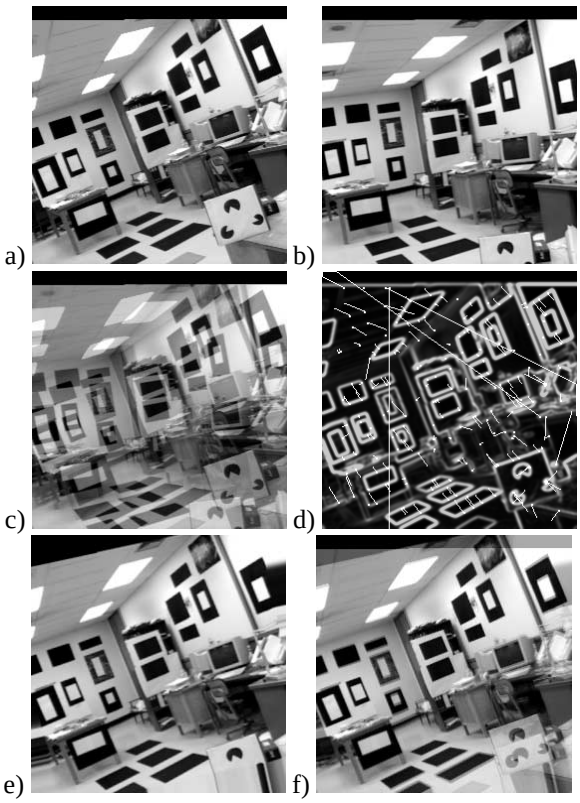

Figure 6. Normal-flow point-to-line L_1 registration. a) First image. b) Second image. c) Summation of (a) and (b) shows the displacement between the two images. d) Magnified Normal flow of the selected points at the last iteration, displayed over a gradient map. The outliers are easy to spot. e) (b) warped towards (a) using the affine model computed by L_1 alignment. f) Summation of (a) and (e) shows good alignment in most of the scene.

Rank of linear model: $k = 4$.
Outliers probability: $q = 0.4$ (three motion models, matched pairs are distributed as follows: 20, 60 ,20).
Added Noise: Normal distribution with zero mean and variance is 5% of the range. Even though the probabilistic algorithm was able to execute 7000 iterations during the time the single-iteration linear programming executed, the results obtained were inferior to linear programming as seen in Fig. 9.

4.4.2 Real-Time Performance

Programs for video mosaicing and for image stabilization were written based on fuzzy correspondences (Sec 2.3). Execution on a PC using windows NT with no special hardware an image sequence was directly processed from the camera at 10-12 frames per second. The panoramas in Fig. 4 were created in real-time using this program.

5 Concluding Remarks

This paper presented a new approach for real-time motion analysis by converting image measurements into point-to-line correspondences, and computing the motion model using linear programming. The presented approach was shown in many experiments to be resistant to outliers, efficient, and enables real-time performance on a regular PC.

Although other estimators, e.g. the LMS estimator, may be superior to the L_1 estimator, the L_1 estimator is recommended for many applications which have a large number of parameters. An analysis regarding the effects of leverage points on the L_1 metric in the image analysis domain is described in the Appendix, as well as a comparison between a least L_1 estimator and LMS.

References

[1] Y. Aloimonos and Z. Duric. Estimating the heading direction using normal flow. *IJCV*, 13(1):33–56, September 1994.
[2] P. Anandan. A computational framework and an algorithm for the measurement of visual motion. *Int. J. of Computer Vision*, 2:283–310, 1989.
[3] M. Ben-Ezra, S. Peleg, and M. Werman. Robust real-time motion analysis. In *DARPA98*, pages 207–210, 1998.
[4] O. Faugeras, F. Lustman, and G. Toscani. Motion and structure from motion from point and line matching. In *Int. Conf. on Computer Vision*, pages 25–34, 1987.
[5] R. P. Hampel F.R., Ronchetti E.M. and S. W.A. *Robust Statistics: The Approach based on influence Functions*. New York: John Wiley, 1986.
[6] R. Hartley. Minimizing algebraic error in geometric estimation problems. In *ICCV98*, pages 469–476, 1998.
[7] B. Horn and B. Schunck. Determining optical flow. In *AI*, volume 17, pages 185–203, 1981.
[8] T. Huang and A. Netravali. Motion and structure from feature correspondences: A review. *PIEEE*, 82(2):252–268, February 1994.
[9] M. Irani and P. Anandan. Robust multi-sensor image alignment. In *ICCV98*, pages 959–966, 1998.
[10] H. Karloff. *Linear Programming*. Birkhäuser Verlag, Basel, Switzerland, 1991.
[11] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *IJCAI81*, pages 674–679, 1981.
[12] P. Meer, D. Mintz, and A. Rosenfeld. Analysis of the least median of squares estimator for computer vision applications. In *CVPR92*, pages 621–623, 1992.
[13] Y. Rosenberg and M. Werman. Representing local motion as a probability distribution matrix and object tracking. In *DARPA Image Undersading Workshop*, pages 153–158, 1997.
[14] P. Rousseeuw. Least median of squares regression. *Journal of American Statistical Association*, 79:871–880, 1984.
[15] P. Torr and D. Murray. Outlier detection and motion segmentation. In *SPIE*, volume 2059, pages 432–443, 1993.

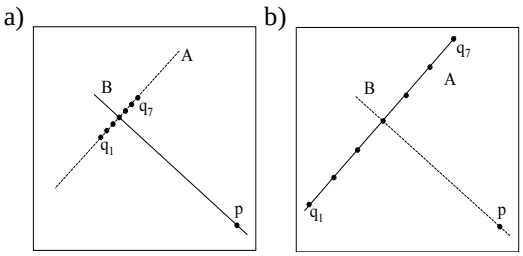

Figure 7. Leverage points for line regression.

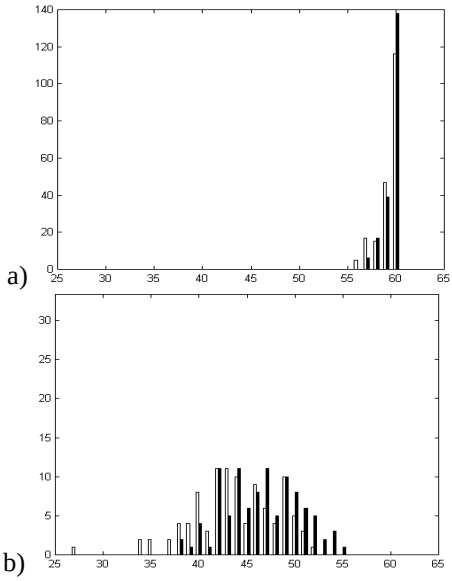

Figure 8. Monte-Carlo comparison between least L_1 and LMS. The histograms show the number of correct classifications. White bars corresponds to the L_1 recovered model. Black bars corresponds to the LMS recovered model. a) Low noise level. b) High noise level.

A L_1 Leverage Points in Motion Analysis

The L_1 metric has a breakpoint zero. This means that even a single outlier point can flip the recovered model very far from the real model. Such points are called leverage points. An example of a leverage point is shown in Fig. 7.a. Line A is the real model, points $q_1..q_7$ are located on line A. Point p satisfies: $L_1(p, A) > \sum_{i=1}^{n} L_1(q_i, B)$, where $L_1(a, b)$ represents the L_1 distance between a and b. This causes the model to flip into line B. Figure 7.b describes a very similar setup with points $q_1..q_7$ spread along the line A. This time there is no single point in the bounded-rectangle that qualifies as a leverage point. There is no "room" for a "lever" long enough to flip the model. In this particular case the breakpoint of the L_1 metric is larger than zero - we then refer to line A as the "dominant line".

In practice, the background of a scene usually forms a large model that is spread across the image, and thus is not subject to leverage points within the image boundaries. This behavior was confirmed by hours of testing in real-time rate (10-12 fps) and by synthetic Monte-Carlo tests presented in Sec. A.1.

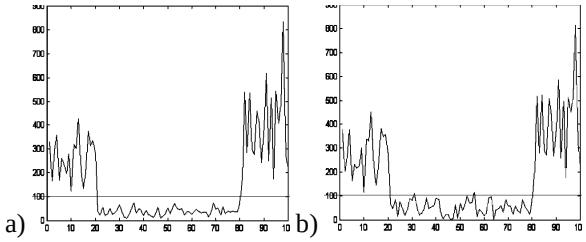

Figure 9. Comparing performance of linear-programming and a probabilistic algorithm running for the same time. 60 of the 100 points are in the desired model ($q = 0.4$). a) Error plot for the linear programming solution. clean separation is obtained. b) Error plot for the probabilistic algorithm solution. Separation between inliers and outliers is possible, but not for all points.

A.1 Comparing L_1 and LMS Estimators

The Least Median of Squares (LMS) estimator is a well known robust estimator [12, 14] which has a theoretical model breakpoint of 0.5 (i.e. it can tolerate up to 50% outliers). This is better than the theoretical breakpoint of the least L_1 estimator, which is zero.

Fig. 8 shows the results of comparing LMS with least L_1 on synthetic data using a Monte-Carlo method. Given two similar linear transformations (4 by 4 matrices), a single test consisted of randomly selecting 100 points (3D projective points), transforming 60 points with one model and 40 points with the second model, and then adding random noise to the transformed points. Given this data, the linear transformation was recovered one time using LMS and one time using a least L_1 estimator. The accuracy of a recovered transformation was computed by counting the number of correct classifications: The transformation was applied to the original points, and the points were sorted based on their error from the transformed points. In perfect conditions, all 60 points having lowest error should come from the first model. The "correct classifications" is the number of points actually coming from the first model among the 60 points having lowest errors. The test was performed 200 times, each time with a new set of points. The histogram in Fig. 8 shows for each number of correct classifications how many times it occurred among the 200 tests. as can be seen, the LMS has a shift to the right which means that it has more correct classifications. Therefore the LMS is better that the least L_1, but only slightly in the global motion analysis domain.