Numerical Stability of the 8-Point Algorithm

ECSE 6650: Computer Vision
Final Project
Submitted December 15 2005
by **Philip Lamoureux**

Introduction

In this project, we examine the 8-Point Algorithm for calculating the Fundamental Matrix for a stereo pair of cameras, with an eye towards numerical errors and stability in the presence of noisy data. We follow the methods outlined in Hartley [4], and present the theory and implementation of an improvement to the 8-Point algorithm that yields more stable results. Different implementations of the 8-Point algorithm are applied to severa datasets, including some that are generated artificially. The performance of the implementations are compared for differing numbers of points used and error present. The content and results of this report are similar to [4], but with a greater emphasis on the differences in performance between the Isotropic and Anisotropic scaling techniques.

In section 1, the theory of the Fundamental matrix is presented, as well as an overview of the regular 8-Point algorithm. We present an explanation of the susceptibility of the 8-Point Algorithm to numerical errors, and outline the improvements that help solve these shortcomings.

In section 2, the implementation of our programs is detailed, along with the experimental data that we gathered. An emphasis is placed on the various choices that we have in how we scale the data, and the relative contribution of the scaling and translation to the final stability of the results.

1. 8-Point Algorithm Theory

a. Problem Statement and Epipolar Geometry

The notion of Epipolar Geometry is a strong tool for us to use in Computer Vision. The essential idea is that, for a stereo pair of cameras, the projection of a 3D point on a camera screen will lie on a plane Π defined by the two camera centers and the 3D point. Furthermore, any such plane will always lie on a particular pair of points on both image screens, called the *epipoles*. This idea is shown in the figure below. This idea means that, given knowledge of the relative position and orientation of a pair of cameras and a given pixel in one image, we can constrain our search of the corresponding pixel in the other image to a single line, rather than the entire image. The time savings of such a tool should be readily apparent. In addition, we can use Epipolar Geometry arguments to solve the inverse problem: given correlated pairs of points, determine the relative position and orientation of two cameras.

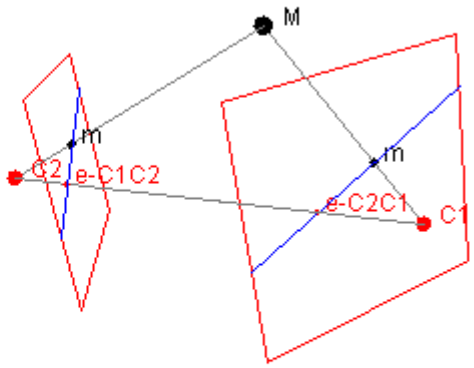

Figure 1: Conceptual epipolar system. The triangle lies on the plane P, which is defined by the two camera centers and the object point. Regardless of the position of point M, the epipoles e-C1C2 and e-C2C1 will not change. Figure courtesy of [1].

In this project, we seek to solve the inverse problem of determining the underlying geometry of the cameras in the form of the Fundamental Matrix based on matched points in two images. We will begin by explaining how the Fundamental Matrix arises as a result of Epipolar Geometry. The following derivation is adapted from [3].

Let us say for an arbitrary stereo camera system that we have two cameras, C_l and C_r. Which are separated by a translation $T = {}^{l:Cl\rightarrow Cr}_{\rho}$ and a rotation ${}^{r\rightarrow l}_{R}$ [1]. This means that for any point P, we can relate the position vector of P in the right and left camera frames by definition as:

1 The notation used here means that the translation is in the left coordinate frame of reference (l), and the translation is from the left camera center (C_l) to right right camera center (C_r). The rotation converts points from the right coordinate frame of reference to the left coordinate frame of reference. Though this notation may seem more bulky, it decreases the amount of possible ambiguity inherent in the discussion, and we argue that the increased complexity is justified by the added precision that this gives our discussion.

$$P_l = {}^{l:Cl\to P}\rho = R \times {}^{r\to l}\rho + {}^{r:Cr\to P}\rho + {}^{l:Cl\to Cr}\rho \quad (1.1)$$
$$P_r = {}^{r:Cr\to P}\rho = R\,({}^{l\to r}\rho + {}^{l:Cl\to P}\rho - {}^{l:Cl\to Cr}\rho) \quad (1.2)$$

From now on we will use the less precise, but more compact notation of T, P_l and P_r. Because the vectors T, P_l, and P_r are all coplanar, we know that if we take the cross product of two of these vectors, and then use the dot product on the result and the remaining vector, the result should be zero:

$$(T \times P_l)^T (P_l - T) = 0 \quad (1.3)$$

Combining (1.2) and (1.3) we get

$$(T \times P_l)^T R P_r = 0 \quad (1.4)$$

The cross product can in general be represented as

$$S = T \times P_l = \begin{bmatrix} 0 & -T_z & T_y \\ T_z & 0 & -T_x \\ -T_y & T_x & 0 \end{bmatrix} P_l \quad (1.5)$$

So we get the *Essential Matrix*, E, which is defined as:

$$E = S^T R \quad (1.6)$$

And we get the familiar result that

$$P_l^T E P_r = 0 \quad (1.7)$$

which establishes a very natural link between the relative positions of an object in the camera frames given the extrinsic vector.

We can extend this result to the actual projections of point P on the right and left camera frames by including the intrinsic camera matrices. In general, the intrinsic matrices for the right and left cameras will give us

$$\lambda U_l = W_l P_l \quad (1.8)$$
$$\lambda U_r = W_r P_r \quad (1.9)$$

Which we can substitute into (1.7) to obtain

$$U_l^T W_l^{-T} E W_r^{-1} U_r = 0 \quad (1.10)$$

Note that the scale factors disappear because in general they will not equal zero, so the other terms must result in zero. This will simplify to the familiar result of

$$F = W_l^{-T} E W_r^{-1} \quad (1.11)$$
$$U_l^T F U_r = 0 \quad (1.12)$$

b. Linear Solution Method and the 8-Point Algorithm

We now derive a linear method for extracting the Fundamental Matrix from a list of matched pairs of points. Much of this material was presented in a similar way in Project 2, but will be repeated here for completeness. This method can be found in [2], [3], and [4].

For any given matched pair of points $U_l = [c_l\ r_l]$ and $U_r = [c_r\ r_r]$, equation (1.12) may be

represented as:

$$A^{1\times 9} f^{9\times 1} = 0, \quad (1.13)$$

$$A = (c_l c_r \quad c_l r_r \quad c_l \quad r_l c_r \quad r_l r_r \quad r_l \quad c_r \quad r_r \quad 1) \quad (1.14)$$

$$f = (F_{11} \quad F_{12} \quad F_{13} \quad \ldots \quad F_{33})^T \quad (1.15)$$

If we have N correlated points, then we can build the linear set of equations

$$A^{N\times 9} f^{9\times 1} = 0, \quad (1.16)$$

$$A = \begin{pmatrix} c_{l1}c_{r1} & c_{l1}r_{r1} & c_{l1} & r_{l1}c_{r1} & r_{l1}r_{r1} & r_{l1} & c_{r1} & r_{r1} & 1 \\ c_{l2}c_{r2} & c_{l2}r_{r2} & c_{l2} & r_{l2}c_{r2} & r_{l2}r_{r2} & r_{l2} & c_{r2} & r_{r2} & 1 \\ & & & \cdot & & & & & \\ & & & \cdot & & & & & \\ & & & \cdot & & & & & \\ c_{lN}c_{rN} & c_{lN}r_{rN} & c_{lN} & r_{lN}c_{rN} & r_{lN}r_{rN} & r_{lN} & c_{rN} & r_{rN} & 1 \end{pmatrix} \quad (1.17)$$

$$f = (F_{11} \quad F_{12} \quad F_{13} \quad \ldots \quad F_{33})^T \quad (1.18)$$

Which can be solved for up to a scale factor if $N = 8$, and if N is greater than that, it will be solved uniquely in a way that minimizes equation 1.16. In general, to solve for N equations, the singular value decomposition (SVD) of A is taken so that

$$(U_A, D_A, V_A) = SVD(A) \quad (1.19)$$

$$s.t. \quad U_A D_A V_A^T = A \quad (1.20)$$

.

The solution for f that minimizes equation 1.16 is the smallest column of V. Call the value for F obtained here F'. An alternative method for determining this solution takes the eigenvector that corresponds to the smallest eigenvalue of A^tA. The solutions obtained by these two methods are identical.

As stated in the lecture notes, the matrix F is singular and of rank 2. In general, since we are calculating F based on discretized data with some finite amount of noise, these properties will not naturally be present in our answer. We will enforce these properties of F to refine the intermediate answer that we obtained above. For any singular matrix of rank two, the SVD will should result in a diagonal matrix D that has a zero value in the last element. If we take the SVD of F' and force the last diagonal element of D' to zero, we can recalculate a refined F.

$$(U_F, D_F, V_F) = SVD(F) \quad (1.21)$$

$$D_F' = \begin{pmatrix} D_{F11} & D_{F12} & D_{F13} \\ D_{F21} & D_{F22} & D_{F23} \\ D_{F31} & D_{F32} & 0 \end{pmatrix} \quad (1.22)$$

$$F = U_F D_F' V_F^T \quad (1.23)$$

Given the derivation above, the so-called 8-Point algorithm to determine the Fundamental matrix given at least eight matched points, presented originally by [5], is presented below.

Generic 8-Point Algorithm:

- For each matched point, add a new row to matrix *A* given in equation (1.17).
- Find *F'*, an intermediate value for *F* by taking the SVD of A. *F'* is the vector of *V* corresponding to the smallest diagonal element of *D*.
- Refine *F'* by taking SVD, setting the smallest diagonal value equal to zero, and recombining $U*D*V^T$.

d. Numerical Stability of 8-Point Algorithm

When implemented in the manner described above, the 8-point algorithm will often yield results that are only marginally useful. This is because the method above is highly sensitive to noise in the matched point data. We present two major explanations of this sensitivity to numerical error in this section, adopted from [4].

We begin our discussion on the stability of the 8-Point algorithm by examining the condition *k* of the matrix A^TA, which is a typical measure of the sensitivity of a matrix to noise. The condition *k* of the matrix A^TA is obtained by taking the SVD of that matrix, examining the ratio of the largest diagonal element to the smallest element. A large *k* will result in more numerical instability, a smaller *k* results in less instability.

It turns out that the *k* of A^TA will typically be very large. This is because a typical pixel coordinate will be on the order $U = (100\ 100\ 1)$ (for high-resolution pictures, this difference in scales becomes even greater). This means that every row of the *A* matrix will be on the order of $\boldsymbol{r}^T = (10^4\ 10^4\ 10^2\ 10^4\ 10^4\ 10^2\ 10^2\ 10^2\ 10^0)$. This means that the diagonal elements of matrix A^TA will be on the order of $(10^8\ 10^8\ 10^4\ 10^8\ 10^8\ 10^4\ 10^4\ 10^4\ 10^0)$.

The actual value for *k* can be bounded in the following way. Define X_r as the matrix that is made up of the last *r* rows and columns of A^TA, and $\lambda_i(X_r)$ is the i^{th} largest eigenvalue of that matrix. In this notation, $k = \lambda_1(X_9)/\lambda_8(X_9)$. Now, because of the *interlacing principle of matrices*, we know that $\lambda_8(X_9) \le \lambda_7(X_8) \le ... \le \lambda_1(X_2)$. We have a bound on $\lambda_1(X_2)$ because we know that the sum of the two eigenvectors $\lambda_1(X_2)$ and $\lambda_2(X_2)$ will sum to the diagonal elements of X_2, which is about 10^4. Therefore, $\lambda_8(X_9) \le 10^4$. Also because of the interlacing principle, $\lambda_1(X_9)$ is no smaller than the largest diagonal element of X_9, so $\lambda_1(X_9) \ge 10^8$. Therefore, $k \ge 10^4$, and in general will be significantly greater.

Our second argument for the numerical instability of the 8-Point algorithm is that, because of the typical pixel taking on the value $U = (100\ 100\ 1)$, the intermediate value for the fundamental matrix F' found in step two of the algorithm will typically result in a matrix with elements of magnitude

$$mag(F') \approx \begin{pmatrix} 10^{-4} & 10^{-4} & 10^{-2} \\ 10^{-4} & 10^{-4} & 10^{-2} \\ 10^{-2} & 10^{-2} & 10^{-1} \end{pmatrix} \quad (1.24) \ .$$

When we enforce the singularity constraint on *F'* in step three of the 8-point algorithm, we affect all elements of *F'* by approximately the same amount. Because of this, the relative effect of step three on the upper left elements is much greater than for the lower right elements. This is of importance to us because when determining the epipolar lines on a given image, we will be multiplying the smallest elements of *F* with the largest elements of the pixel coordinate *U*. Thus, if the upper left

elements of F' are perturbed by more than $O(10^4)$, we can expect large errors in the computed epipolar line. This is clearly undesirable.

The solution to the problems outlined above is to transform the original matched pixel coordinates so that the magnitudes of the elements of A^TA are approximately unity. This is addressed in the following section.

e. Improved 8-Point Algorithm

As shown above, it is greatly advantageous to the accuracy of the 8-point algorithm to have elements of A^TA are approximately unity. We must do this in a way that does not affect the properties of the fundamental matrix that is obtained. In this section we outline a method of translating and scaling the matched pixel coordinates in a way that we greatly reduce the numerical inaccuracies pointed out above, while still obtaining a Fundamental Matrix that relates the matched points to each other. The methods in this section are due to [4].

We first present an argument for the correctness of transforming the matched points. Suppose that we have a set of matches points for a left and right pair of images U_{li}, U_{ri}. If we multiply the points in the images by T_l and T_r, we obtain

$$\hat{U}_{li} = T_l U_{li} \quad (1.25) \quad .$$
$$\hat{U}_{lr} = T_r U_{ri}$$

If we substitute (1.25) into equation (1.12), we get

$$\hat{U}_l^T T_l^{-T} F T r^{-1} \hat{U}_r = 0 \quad (1.26) \quad .$$

If we define $\hat{F} = T_l^{-T} F T r^{-1}$, we can obtain $\hat{F}$ by running the 8-Point algorithm on $\hat{U}_l^T \hat{F} \hat{U}_r = 0$. Once we have obtained a value for $\hat{F}$, we can obtain the Fundamental matrix for the original data by:

$$F = T_l^T \hat{F} T_r \quad (1.27) \quad [2].$$

The question now becomes, how do we want to transform the matched points so that we can decrease the amount of numerical errors in the 8-Point algorithm. Our objective is to make the elements of the "average" point location $U_i = (c_i \; r_i \; 1)$ equal to unity. We can do this most effectively by scaling the points so that the average magnitude of the image coordinates is about unity. Additionally, if we first translate the points so that they are centered about the origin, the average difference in magnitude of any pair of points is minimized. An example of this transformation scheme is shown conceptually in the figure below.

---

2 [2] gives this equation as $F = T_l^{-T} \hat{F} T_r^{-1}$, which appears to be a typo.

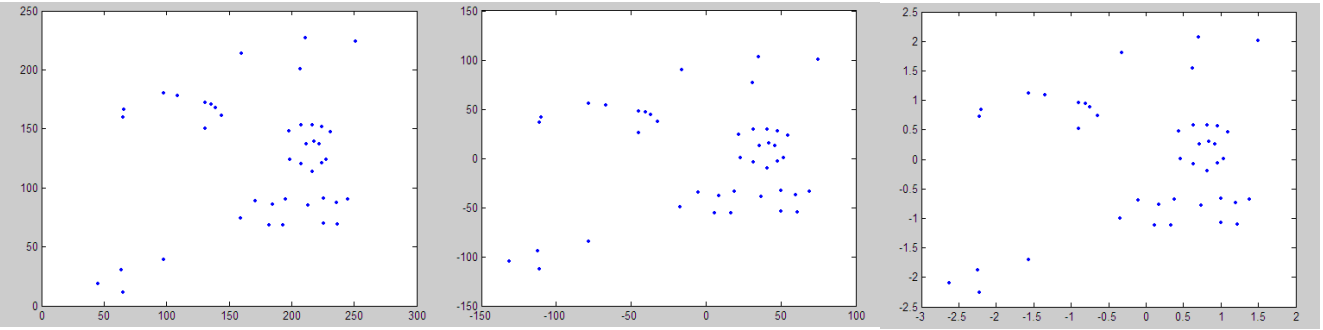

Figure 2: Translation and scaling of a set of points. From right to left: The original data, Translated data so that mean location is the origin, Scaled data so that average x and y distance to the origin is 1. Note the scale of the figures.

This transformation can be described more precisely as follows. For a given image, we want to translate the points such that:

$$\sum_{i=1}^{n} \frac{c_i}{n} = 0 \quad (1.28)$$

$$\sum_{i=1}^{n} \frac{r_i}{n} = 0 \quad (1.29)$$

and then scale them so that

$$\sum_{i=1}^{n} \frac{\sqrt{(c_i - \bar{c})^2 + (r_i - \bar{r})^2}}{n\sqrt{(2)}} = 1 \quad (1.30) \ .$$

This can be done by defining the following transformation matrix:

$$T = \begin{pmatrix} 1/d & 0 & -\bar{x}/d \\ 0 & 1/d & -\bar{y}/d \\ 0 & 0 & 1 \end{pmatrix} \quad (1.31) \ ,$$

where

$$\bar{x} = \sum_{i=1}^{n} \frac{c_i}{n} \quad (1.32)$$

$$\bar{y} = \sum_{i=1}^{n} \frac{r_i}{n} \quad (1.33) \ .$$

$$d = \sum_{i=1}^{n} \frac{\sqrt{(c_i - \bar{c})^2 + (r_i - \bar{r})^2}}{n\sqrt{(2)}} \quad (1.34)$$

The method presented above is called the "anisotropic scaling" of the data. That is, the x and y coordinates are scaled by the same factor. If the average x distance to the origin is greatly different than the average y distance (see figure below), then it may be advantageous to scale the data anisotropically.

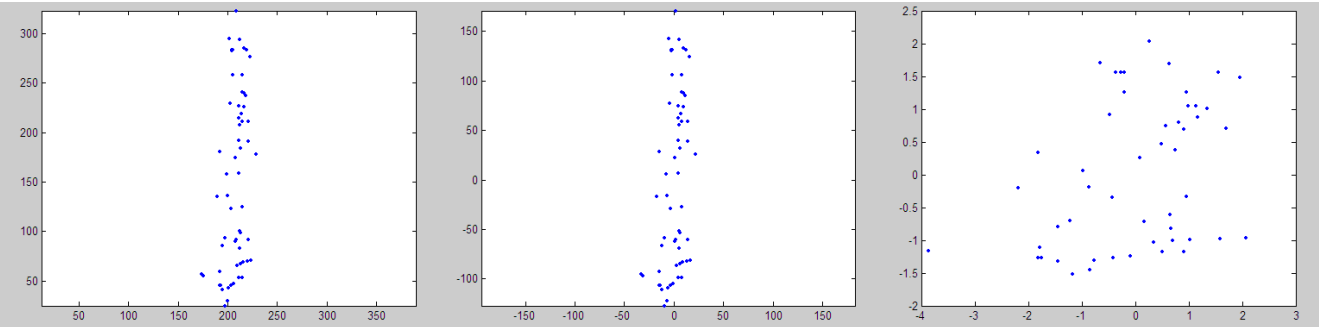

Figure 3: Translation and Anisotropic scaling of a set of points. From right to left: The original data, Translated data so that mean location is the origin, Scaled data so that average x and y distance to the origin is 1. Note the scale of the figures.

The transformation matrix for anisotropic scaling is:

$$T = \begin{pmatrix} 1/dx & 0 & -\bar{x}/dx \\ 0 & 1/dy & -\bar{y}/dy \\ 0 & 0 & 1 \end{pmatrix} \quad (1.35)$$

$$dx = \sum_{i=1}^{n} \frac{c_i - \bar{c}}{n} \quad (1.36)$$

$$dx = \sum_{i=1}^{n} \frac{r_i - \bar{r}}{n} \quad (1.37)$$

We now present the improved version of the 8-Point Algorithm, which is sometimes referred to at the "Normalized 8-Point Algorithm".

Normalized 8-Point Algorithm:

- Calculate Transformation matrices via equation 1.31 or 1.35. Transform all matched points.
- For each point, add a new row to matrix *A* given in equation (1.17).
- Find *F'*, an intermediate value for *F* by taking the SVD of A. *F'* is the vector of *V* corresponding to the smallest diagonal element of *D*.
- Refine *F'* by taking SVD, setting the smallest diagonal value equal to zero, and recombining $U*D*V^T$.
- Obtain the Fundamental matrix from the original untransformed data by taking $T_l^T F T_r$.

2. Implementation and Results

a. Implementation of Algorithms

The programs in this project were implemented in Matlab. There are three scripts that will each run a different implementation of the 8-Point Algorithm: reg_8_pt.m, normalized_isotropic_8_pt.m, and normalized_anisotropic_8_pt.m. Each of the three scripts will call the script add_lines.m, which will use the calculated F matrix to output an image of the system with the epipolar lines added. There are two more scripts that are used to generate the artificial images used in this project: generate_images.m and highlight_points.m.

The actual implementations of the various types of the 8-Point algorithm were fairly simple, after the math was understood. There was some problem with verifying that the Fundamental matrices that we were obtaining were correct or not, but this became much easier as we implemented visualization tools, such as the script that outputs the epipolar lines with lines connecting the epipolar lines with the features they belong to.

In the following experiments, we also implemented a version of the 8-Point algorithm with only the translation and another with only the scaling. This allows us to examine the relative contribution that the two steps give to the numerical stabilization of the results.

b. Datasets

Five pairs of images were used in this project: three real pairs of images and two that were created analytically. The real datasets consist of a pair of face images from a previous project, a pair of images of the Mars Rover from [6] and a pair of images of a milling machine from [7]. For these three pairs of images, the features were identified manually and stored in .txt files in the /data directory.

Figure 4: Left: Images of face. Middle: Images of Mars Rover. Right: Images of Milling Machine

In addition to these three datasets, two sets of images were created artificially using Matlab. The first of these pairs of images is of a number of rectangles that are roughly arranged along the three principle planes. The locations of the corners of the rectangles were perturbed slightly to ensure that we would not be able to pick eight coplanar points, though some combinations of points will be very nearly coplanar.

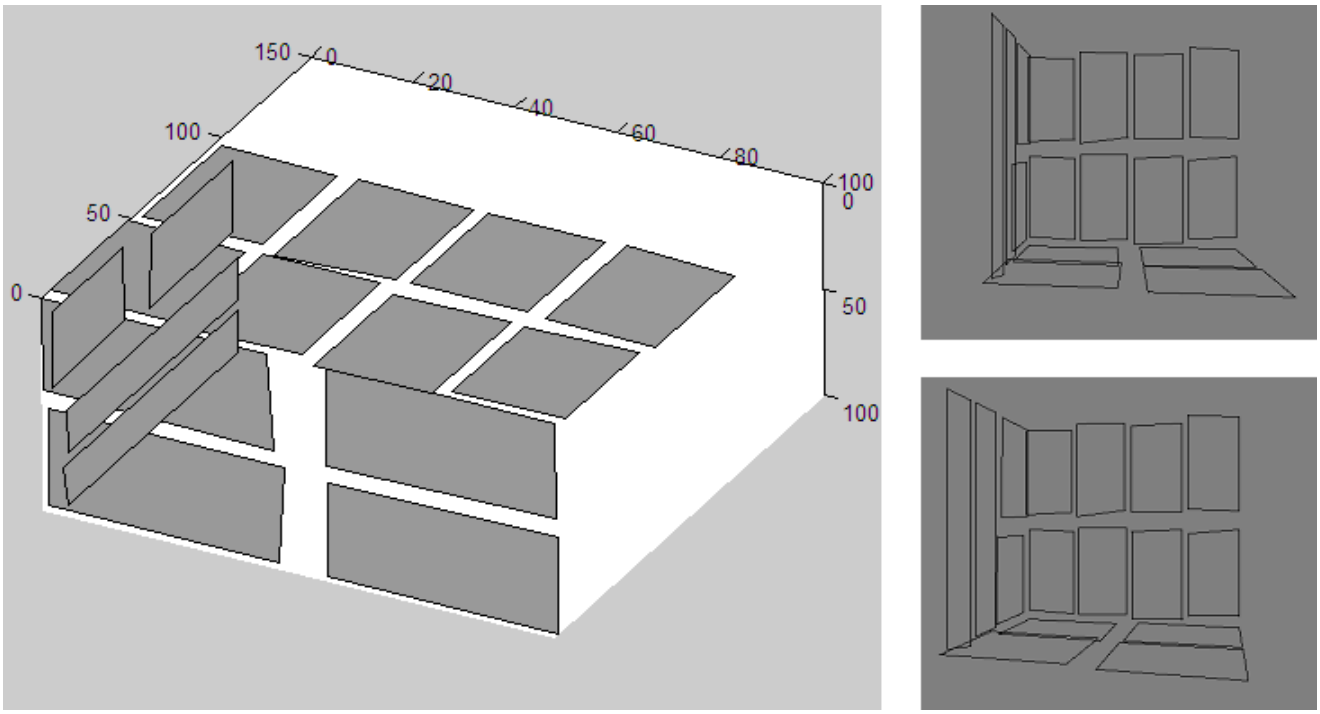

Figure 5: Left: Artificial environment made up of rectangles. Top right: left camera view. Bottom right: right camera view.

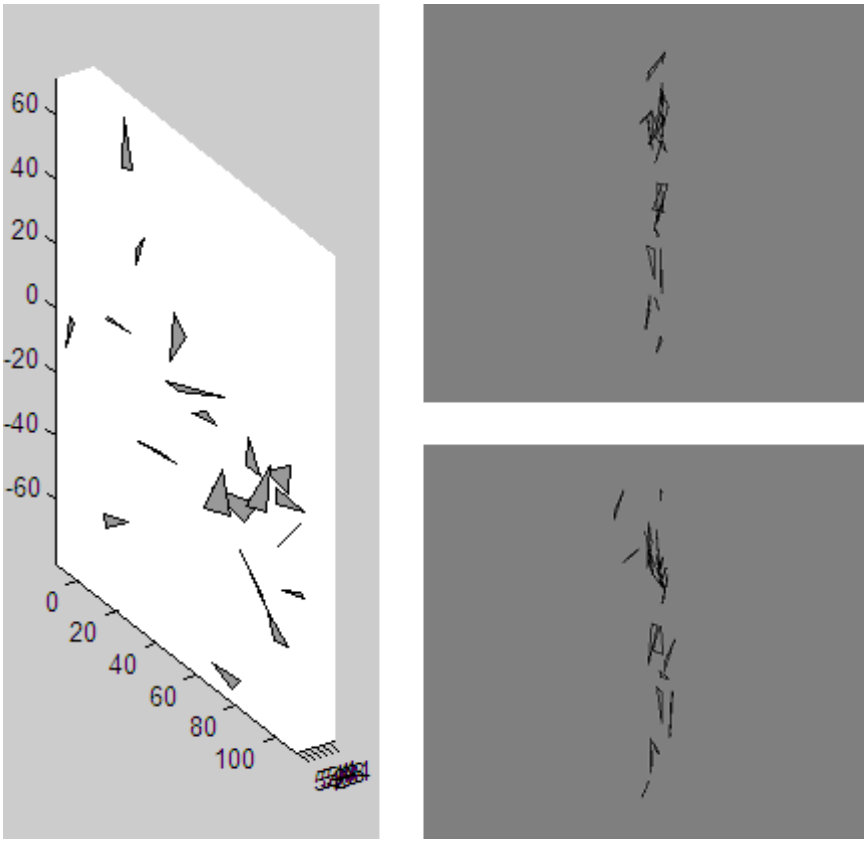

Figure 6: Left: Artificial environment made up of triangles arranged in a plane roughly in line with the cameras. Top right: left camera view. Bottom right: right camera view.

The second set of images is of a series of triangles that are randomly generated about a vertical plane

that is roughly in line with the cameras. This dataset was created specifically so that we could test the relative performance of the isotropic and anisotropic scaling versions of the improved 8-Point algorithm.

For both sets of artificial images, the projected locations of the vertices of the polygons are automatically written to a .txt file and the images saved to the same format as the real images in fig. 4.

c. Metrics

In analyzing the relative performance of the different implementations of the 8-Point algorithm, a diverse set of metrics were measured. These metrics are summarized in the table below.

Metric	***Description***
Time	Amount of time necessary to run the Algorithm. Smaller times are desirable.
Qualitative ability to construct epipolar lines	The epipolar lines for several feature points are overlaid on the image. The closer the feature points lie to the corresponding epipolar line, the better.
Average error	Distance between a feature point and the closest point on the corresponding epipolar line. As above, smaller error is good.
Condition of Martix A^TA	As stated in section ***1.d***, the condition of the matrix is a good indicator of the sensitivity to noise that is present in the system. Greater condition number corresponds to more sensitivity to noise.

Table 1: Metrics used in the evaluation of the various types of the 8-Point Algorithm

d. Experimental Results

Below we present data on the amount of time it takes for the various implementations of the 8-Point algorithm to execute. The following times were captured using Matlab's tic/toc command, and only the portion of the program that is described by the algorithms in section **1.c** and **1.e** are measured. That is, we do not measure the time it takes for the program to do things like read and write image files. The times below were captured for the "squares" dataset for various numbers of points to include in the algorithm. Note that there is a large amount of variability in the times it took for the programs to run, and the time does not seem to increase drastically for an increasing number of points. This is probably due to the computer allocating more or less processor capacity to the Matlab process for the different experiments.

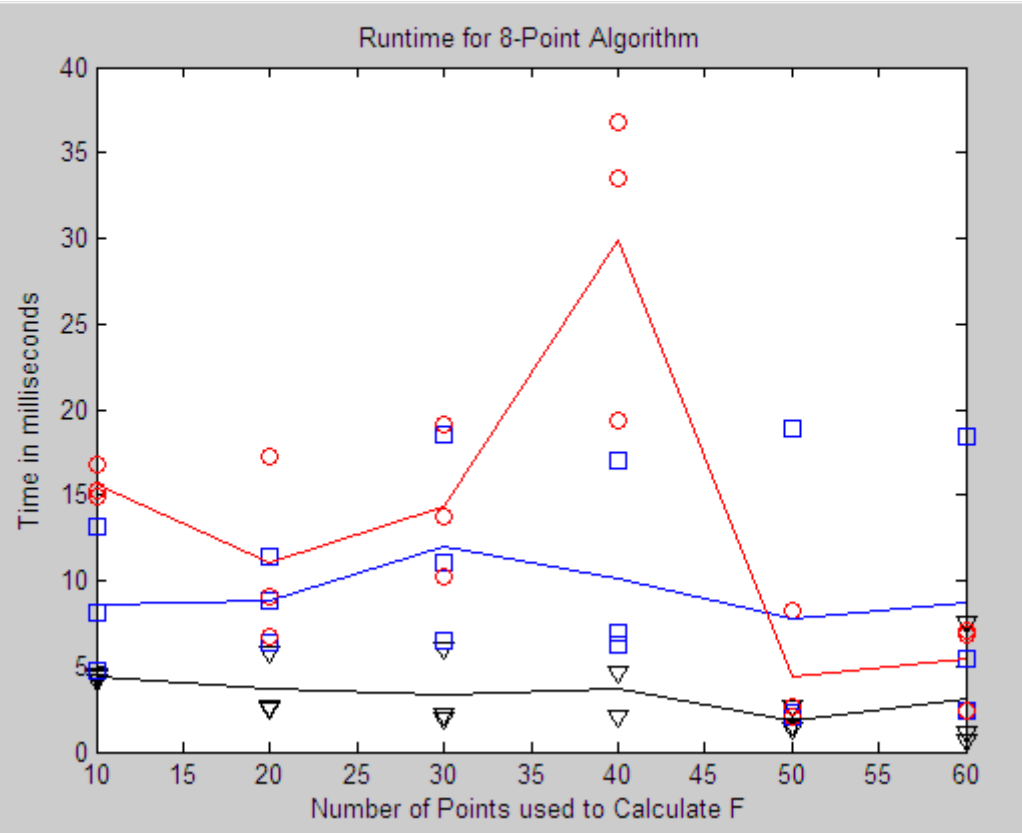

Figure 7: Time for the 8-Point algorithm to calculate F. Black triangles are for the regular implementation, Blue squares are for the isotropic implementation, Red circles are for the anisotropic implementation.

Algorithm	***Mean Time [ms]***	***Std Deviation of Time [ms]***
Regular	3.33	1.9
Isotropic	9.35	5.7
Anisotropic	13.44	9.7

Table 2: Data from Fig 7 Note that while the Anisotropic time average is larger than for the Isotropic algorithm, the difference is slight compared to the Standard Deviation.

Below we demonstrate qualitatively how well the different methods are able to generate epipolar lines for the datasets. In fig. 8, the regular implementation is used for the real-image datasets. In fig. 9, the anisotropically scaled 8-Point algorithm is used.

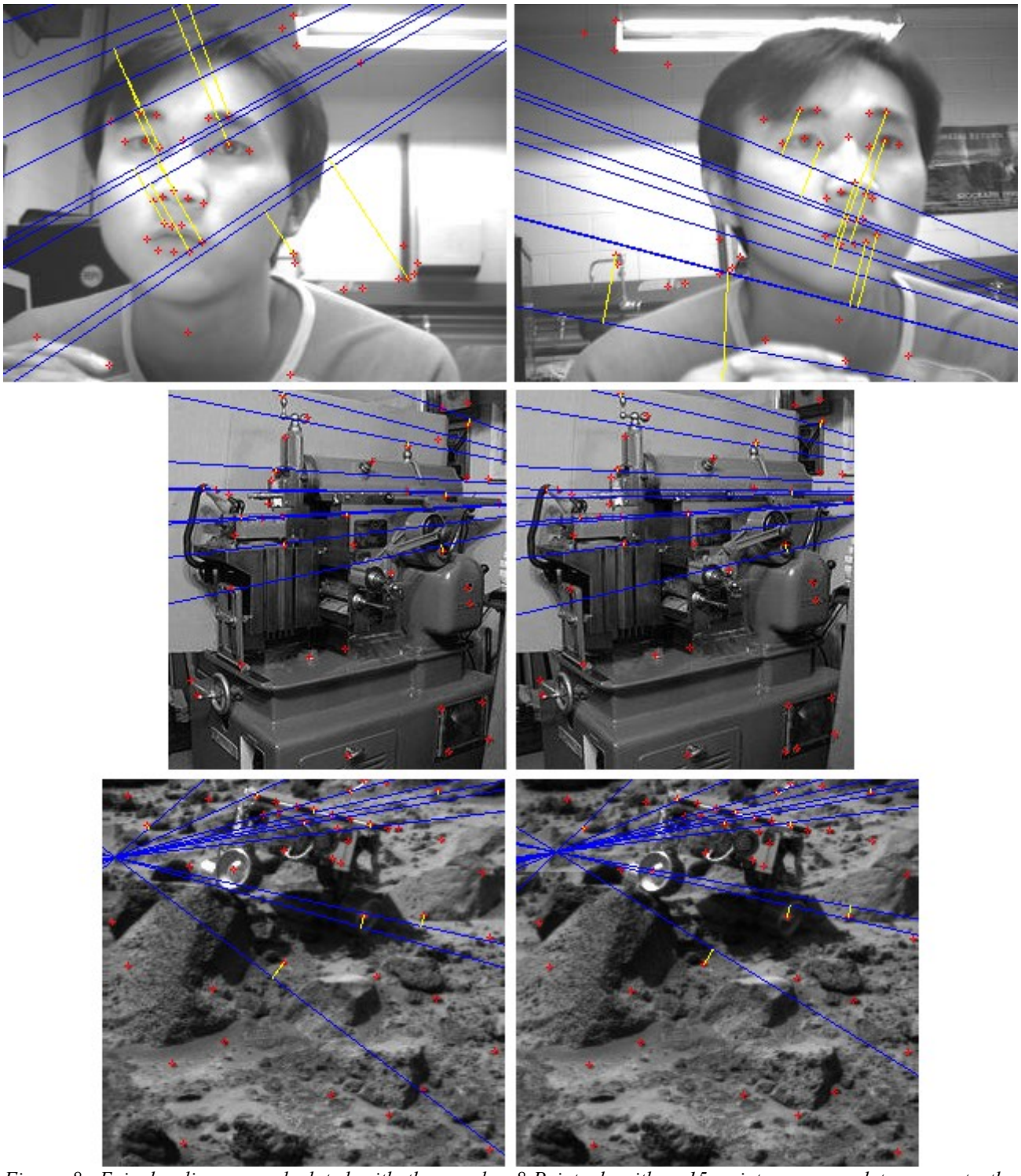

Figure 8: Epipolar lines as calculated with the regular 8-Point algorithm. 15 points were used to generate the Fundamental Matrix for each dataset. Note the gross amount of error in the top dataset. Yellow lines connect features with the closest point on the corresponding epipolar line.

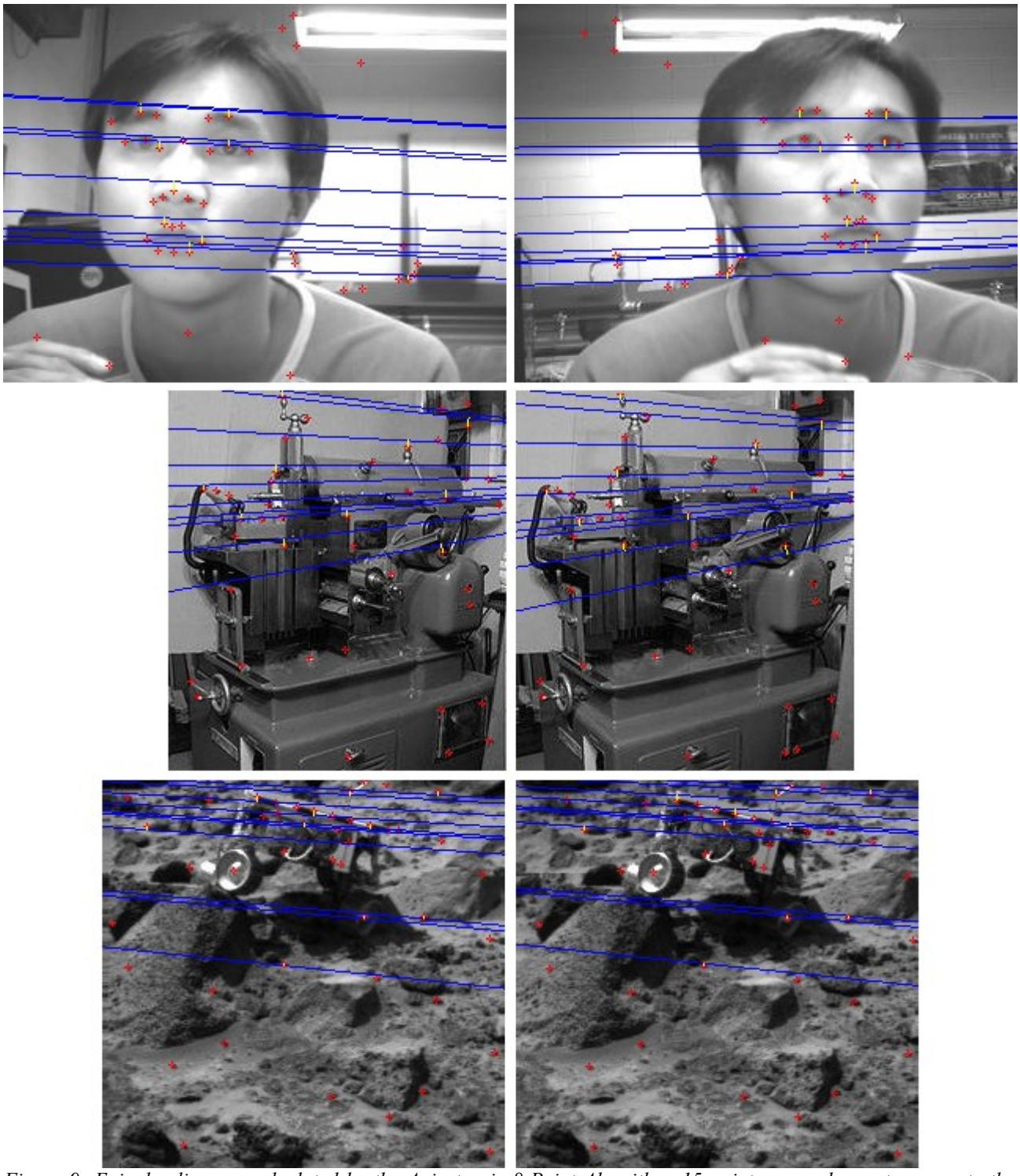

Figure 9: Epipolar lines as calculated by the Anisotropic 8-Point Algorithm. 15 points were chosen to generate the Fundamental matrix for each dataset. Note how the epipolar lines in the top figure are much improved over those in the previous figure.

Below we present data on the error between the features and the nearest point on the corresponding epipolar lines. In figure 10 below, we show the error plotted against the number of points used to calculate F. Data is taken for the regular implementation, Isotropic and Anisotropic normalization, as well as for just translation and just scaling. This gives us a good idea of the relative

contribution that scaling and translating give to the quality of the results. For a given number of points and implementation of the algorithm, we calculate a value for F and average the error over all points in the right and left frames.

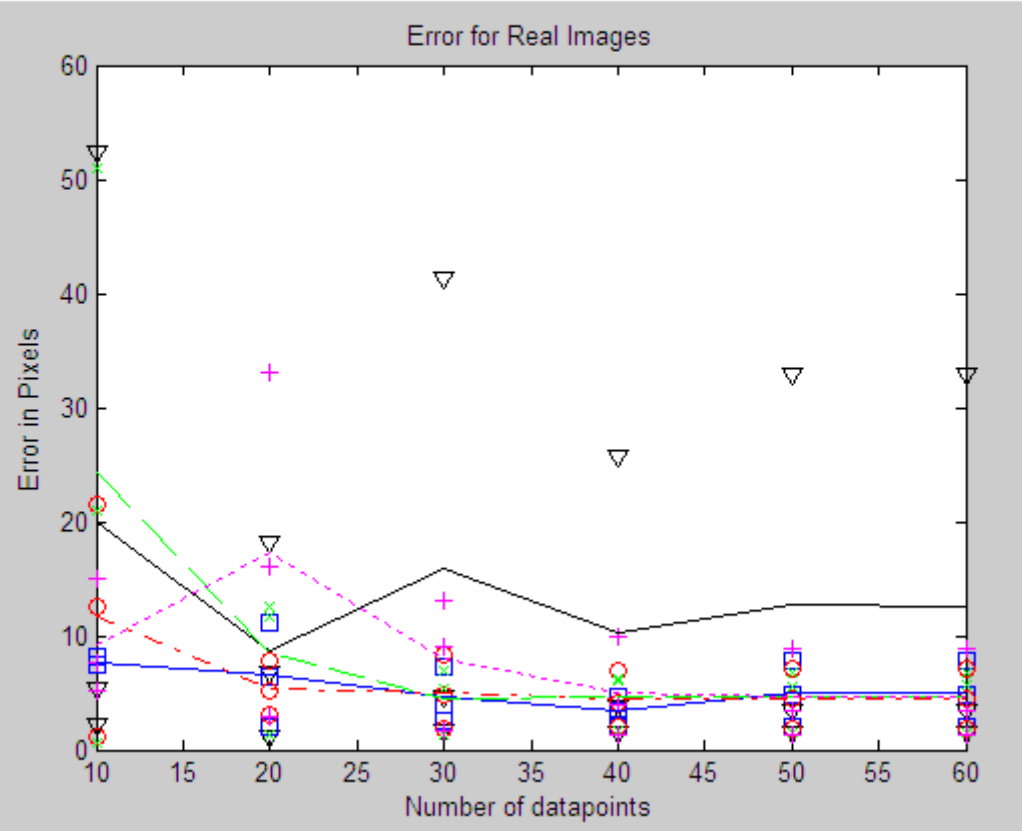

Figure 10: Error between feature and epipolar line. Upper solid line with Triangles is for the regular Algorithm. Dashed line with x's is for just translating the data points. Dotted line with crosses is for just scaling the data points. Dash-dotted line with circles is for Anisometric normalized algorithm. Lower solid line with squares is for Isometric normalized algorithm.

We also show the amount of error that occurs in the artificial data when Gaussian noise of varying magnitudes is added to the position of the features. We choose 40 points at random to calculate the F matrix, and then calculate the error between all points and their corresponding epipolar lines.

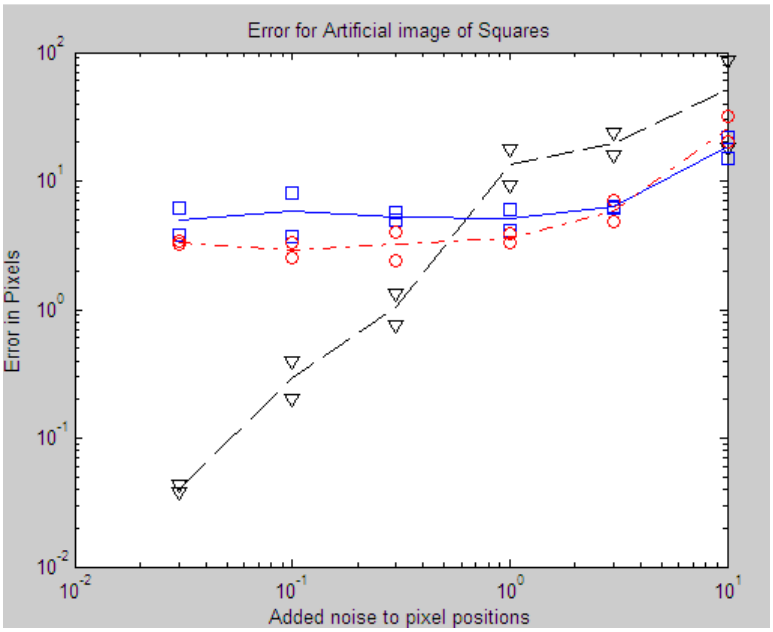

Figure 11: Error between feature and epipolar line for artificial Rectangle System. Dashed Line with triangles is for the regular implementation. Solid line with squares is for Isotropic normalized implementation. Dashed-dotted line with circles is for Anisotropic normalized implementation.

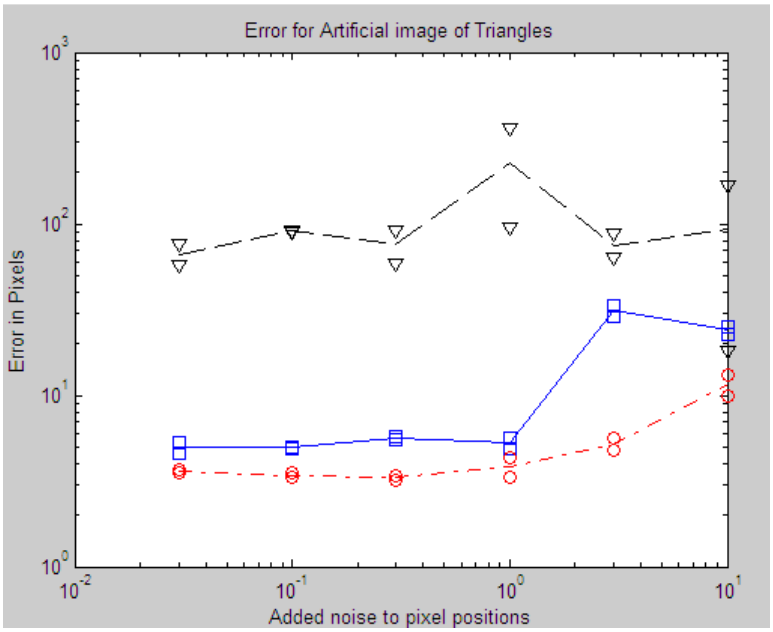

Figure 12: Error between feature and epipolar line for artificial Triangle System. Dashed Line with triangles is for the regular implementation. Solid line with squares is for Isotropic normalized implementation. Dashed-dotted line with circles is for Anisotropic normalized implementation.

Below we present data on the Condition number of the matrix A^TA for the different algorithms in various situations. The figure below shows the Condition number for the real images for different numbers of points used in the calculation. Data is taken for the regular implementation, Isotropic and Anisotropic normalization, as well as for just translation and just scaling.

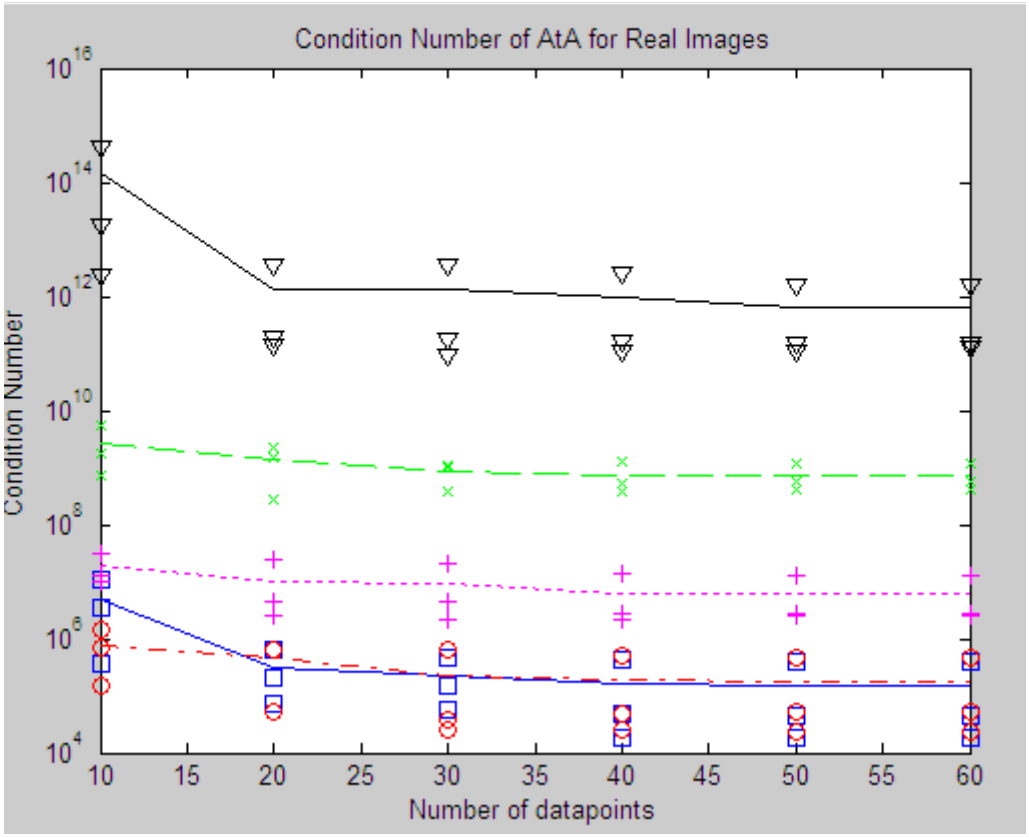

Figure 13: Condition number of AtA. Upper solid line with Triangles is for the regular Algorithm. Dashed line with x's is for just translating the data points. Dotted line with crosses is for just scaling the data points. Dash-dotted line with circles is for Anisometric normalized algorithm. Lower solid line with squares is for Isometric normalized algorithm.

e. Discussion of Experimental Results

The results of the experiments shown above verify much of what we would expect from the theory. The regular implementation of the 8-Point algorithm tends to have a very high condition number and tends to be prone to numerical errors. The performance of the 8-Point algorithm is in general improved by intelligently transforming the data before running the core algorithm, an improvement that is gained with negligible cost in computing time and programming complexity.

We were also able to show clearly that both scaling and translation contribute to the increase in the resistance to numerical errors. Implementing either scaling or translation will result in some improvement in the performance in the algorithm, but implementing both is the best.

Based on our data, we can suggest that at least 30 points be used for the computation of the Fundamental matrix, if the data is available. After 30 or 40 points, the increase in quality of results seems to become fairly minimal. However, we also see that the amount of time to calculate the Fundamental matrix is extremely small, so if we are not limited by computing power, in many cases we may include as many points as possible.

There are, however, some differences between our findings and those of [4]. Most notably is the observation in [4] that the performance of the Isotropic and Anisotropic scaling are virtually identical. In our experiments, it was found that while the Anisotropic scaling will cost a small amount of time more than Isotropic scaling, it outperforms the Isotropic scaling by at least a marginal amount

in all cases. Most importantly, for the case of the artificial images of the triangles, the anisotropic scaling resulted in significantly better performance. This leads us to believe that for any environments where the detected features are likely to be arranged in a line (such as points on the horizon), the Anisotropic implementation is clearly the best choice.

Lastly, there was a result that came as a complete surprise. That is that, for very small amounts of errors (sub-pixel), the regular implementation of the 8-Point algorithm can give more accurate results than the normalized algorithms. In practice, we will probably never see this and we are not advising the use of the regular 8-Point implementation, but this is of theoretical interest to us. It suggests that there are some small but inherent numerical errors associated with the scaling and translation that are not present in the regular implementation.

3. Conclusion

The 8-Point algorithm is a very fast and simple to implement way of determining the Fundamental Matrix of a stereo camera system given a set of matching points, but it can be very sensitive to noisy data. This sensitivity to noise is due primarily to the gross difference in magnitudes within the data being used: a datapoint will contain elements that differ by a few orders of magnitude. This sensitivity to noise can be greatly mitigated by intelligently transforming the data before running the core 8-Point algorithm on it.

The Anisotropic scaling is the preferred method of transforming the data, and will result in more stable results with a negligible increase in computing time and complexity of the program. In most cases the Anisotropic scaling will be only marginally better than the Isotropic scaling, but in some cases the improvement will be great.

References

1. Bougnoux, Sylvain: "Learning Epipolar Geometry". 2005. Webpage: http://www.ai.sri.com/~luong/research/Meta3DViewer/EpipolarGeo.html
2. Trucco, Emanuele and Verri, Alessandro: "Introductory Techniques for 3-D Computer Vision". 1998 Prentice Hall, Inc. Upper Saddle River, NJ.
3. Ji, Qiang: "3D Reconstruction". 2005. PDF Webpage: http://www.ecse.rpi.edu/Homepages/qji/CV/3dreconstruction.pdf
4. Hartley, Richard I.: "In Defense of the 8-Point Algorithm". PDF: 1995 GE-Corporate Research and Development, Schenectady , NY.
5. Longuet-Higgens, H.C.: "A Computer Algorithm for reconstructing a scene from two projections". Nature, 293:133-135, Sept 1981.
6. Riser, James: "An Easy Method for Making Stereo Pairs Using the SONY DSC-S70 Digital Camera". 2005. Webpage: http://www.jamesriser.com/StereoImages/MakingStereoImages.html
7. NASA: "Stereo Pairs Archive". 2005. Webpage: http://science.ksc.nasa.gov/mars/mpf/stereo-arc.html