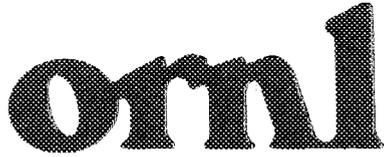


MARTIN MARIETTA ENERGY SYSTEMS LIBRARIES



3 4456 0331123 5

ORNL/TM-11718
CESAR-91/03



OAK RIDGE
NATIONAL
LABORATORY



Robot Self-Location in
Unknown Environments

Elizabeth R. Stuck

OAK RIDGE NATIONAL LABORATORY
 CENTRAL RESEARCH LIBRARY
 CIRCULATION SECTION
 ROOM ROOM 175
LIBRARY LOAN COPY
 DO NOT TRANSFER TO ANOTHER PERSON
 If you wish someone else to see this
 report, send its name with report and
 the library will arrange a loan.

MANAGED BY
MARTIN MARIETTA ENERGY SYSTEMS, INC.
FOR THE UNITED STATES
DEPARTMENT OF ENERGY

This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from the Office of Scientific and Technical Information, P.O. Box 62, Oak Ridge, TN 37831, prices available from (615) 576-8401, FTS 626-8401

Available to the public from the National Technical Information Service, U.S. Department of Commerce, 5285 Port Royal Rd., Springfield, VA 22161.

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

ORNL/TM-11718
CESAR-91/03

Engineering Physics and Mathematics Division

ROBOT SELF-LOCATION IN UNKNOWN ENVIRONMENTS

Elizabeth R. Stuck

DATE PUBLISHED — February 1991

Office of Engineering Research Program
Basic Energy Sciences
U.S. Department of Energy

Prepared by the
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, Tennessee 37831
managed by
MARTIN MARIETTA ENERGY SYSTEMS, INC.
for the
U.S. DEPARTMENT OF ENERGY
under contract DE-AC05-84OR21400



3 4456 0331123 5

CONTENTS

ABSTRACT	vii
1. INTRODUCTION	1
1.1 MOTIVATION	1
1.1.1 Error in Odometry	1
1.1.2 Error in Triangulation	2
1.2 OBJECTIVE	3
1.3 RESEARCH QUESTIONS	3
1.4 FOCUS	4
1.5 BACKGROUND	5
2. GENERAL SOLUTION STRATEGY	7
2.1 ALGORITHM	7
3. IMPLEMENTATION	9
3.1 VISION	9
3.2 MATCHING	10
3.3 TRIANGULATION	12
3.3.1 Obtaining Feature Angles	12
3.3.2 Robot Triangulation	13
3.3.3 Feature Triangulation	14
3.3.3.1 Calculating Range to an Unknown Feature	14
3.3.3.2 Calculating the Robot's Current Position	15
3.4 SELF-LOCATION	16
4. EXPERIMENTAL DATA	17
5. SUMMARY	21
5.1 ROBOT VS. FEATURE TRIANGULATION	21
6. RECOMMENDATIONS	23
7. ACKNOWLEDGMENTS	25
REFERENCES	27
APPENDIX	29
PROCEDURE FOR USING PROGRAMS	29
DIAGRAM OF PROGRAM DATA FLOW	31
DESCRIPTION OF PROGRAMS	32
PROGRAM INTERFACES	36
FILE FORMATS	37
ACCESSING PROGRAMS	40

LIST OF FIGURES

<u>Fig.</u>		<u>Page</u>
1	Odometry error	2
2	Triangulation error	3
3	One implementation of the general strategy	8
4	Sample image	9
5	Sample image after blurring and Laplacian	10
6	Matched feature points	11
7	Sequences of feature matches	11
8	Angles used in triangulation	12
9	Robot triangulation	13
10	Calculating feature range	14
11	Calculating robot position	16
12	First image of sequence	17
13	Last image of sequence	18
14	Diagram of program data flow	31

LIST OF TABLES

<u>Table</u>		<u>Page</u>
1	Experimental data	18
2	Inputs and outputs of programs	36

ABSTRACT

It is often necessary for robots to navigate in environments which are not known in advance. In this context, self-location is the problem of determining how far and in what direction motion has occurred. Because of wheel slippage and other errors, odometry cannot be depended upon to provide precise or accurate positional information. Triangulating from visual features can help make position estimation more accurate. This paper describes work that was done during a three-month student research internship at the Center for Engineering Systems Advanced Research (CESAR) of the Oak Ridge National Laboratory, exploring the problem of robot self-location in unknown environments. This work included the development and integration of a set of programs which present a partial solution to this self-location problem. These programs use a sequence of images which are acquired as the camera moves between positions with a motion which is known approximately. Visual features are extracted from the images and matched through time. Triangulation using these features then provides a rough estimate of the range of these features from the camera. Kalman filtering (not implemented) can then be used to integrate the information from odometry and vision to provide a better estimate of the position of the robot.

1. INTRODUCTION

We define the problem as follows. The robot moves in a plane (termed the ground plane or plane of motion), so its position is described by the triple (x, y, θ) . x and y determine the robot's locality, and θ its heading. The robot's initial position is set to the origin, $(0,0,0)$. The question to be answered is, after n moves, what is the new position (x', y', θ') ?

This problem has several important characteristics. The robot is traveling through an unknown environment. This means that there is no *a priori* knowledge of any features which could be used by the robot to self-locate. However, there are two other sources of information which can be used for this purpose. The robot can observe perceptual features, although any measurements it makes will incorporate error. Wheel odometry is another source of information. The direction and distance the wheels turn may be measured. However, this measurement, too, will include error. Using these two sources of information, triangulation may be used to self-locate. Since we are dealing with digital computers, there will be round-off error, which will lead to imprecision in the results. The goal then is to use perceptual information coupled with odometry to refine the estimate of the robot's position.

1.1 MOTIVATION

Much of the work that has been done addressing mobile robot navigation assumes that the world is known in advance. However, this is not a reasonable assumption to make in most contexts. Many environments change rapidly over time. Many environments are not knowable in advance with the precision needed to solve detailed navigation problems. Other environments may be known in advance, but not with complete accuracy.

The problem being addressed here does not face many of the difficulties introduced when knowledge of the environment is required. Moving objects will, in most cases, not disturb the self-location process. (Object motion may be a problem when it occurs at a slow enough rate that it is not detected quickly.)

1.1.1 Error in Odometry

Figure 1 illustrates how positional error accumulates when using odometry. R_0 is the robot's initial position. The robot is then instructed to move a certain distance at a given heading to position R_1 . However, when a robot executes a command to move a certain distance at a given heading, it will not move exactly as commanded, due to wheel slippage and mechanical imprecision in the wheel drivers and encoders. The ellipse around R_1 represents the uncertainty associated with the actual position of the robot after executing the motion command. This uncertainty is made up of error that is bounded (mechanical imprecision), and unbounded error (wheel slippage). Thus, the position of the robot is most likely to be somewhere in this ellipse around R_1 . (It is possible, but much less likely, that the robot's position is outside this ellipse, since it is very unlikely that wheel slippage is great enough to cause this much error.)

The uncertainty is in the shape of an ellipse because there is commonly more error associated with a robot changing heading than there is in moving a given distance. At the next step, when the robot moves to R_2 , the error accumulates, and there is consequently a larger uncertainty associated with this position of the robot.

2 INTRODUCTION

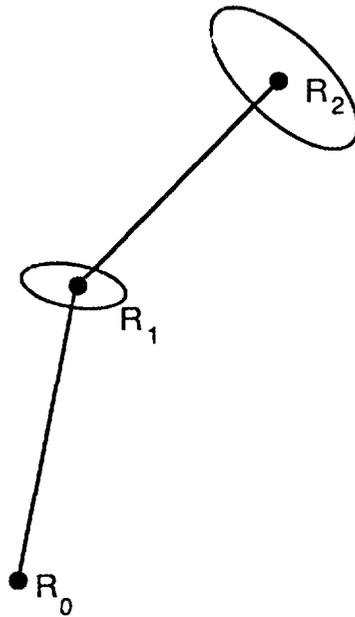


Fig. 1. Odometry error.

1.1.2 Error in Triangulation

Figure 2 illustrates the uncertainty which occurs when triangulating the position of a feature. The robot first sees the feature, F , when it is at position R_{i-1} . Due to noise in the imaging process, there is some angular uncertainty in where the robot detects the feature. Thus, the robot only knows that the feature is somewhere between the two lines indicated. When the robot moves to R_i , it again detects the angle of the feature, with some uncertainty. The resulting area formed between the intersecting lines represents the uncertainty on the estimated location of the feature.

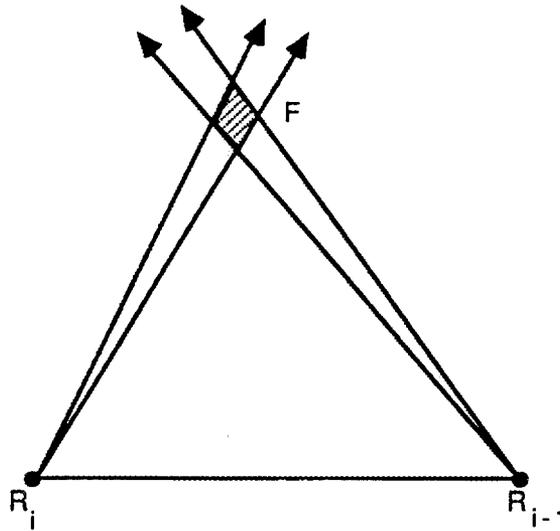


Fig. 2. Triangulation error.

1.2 OBJECTIVE

The objective of this project was to implement methods which have been recently presented in the literature and discover the limitations of these methods. There are two useful consequences of this work. First, this project makes available a set of programs implementing visual feature detection and matching which can be used in self-location. This software will be useful for future research on related topics in vision and navigation. Second, the results of this project suggest possible areas for future research.

1.3 RESEARCH QUESTIONS

There are numerous potentially interesting research questions which can be explored. For example, one can investigate the use of parallelization. Can this procedure be parallelized? Would parallelization result in a significant amount of speed-up?

A second research issue involves the conflict between the needs of matching and those of triangulation. In order to match more quickly and more reliably, it is desirable to minimize the change in the imaged positions of features. The closer the imaged position of a feature is in one image to its imaged position in the next image, the better. However, triangulation requires a large change in the relative angles between the robot and features. How can both requirements be met? If we build long sequences of features, we need to address the problems associated with using features over long periods of time. Features can become occluded, move, or become confused with other features.

Another question has to do with how complex features should be. Complex features may reflect characteristics of the world more reliably. *Perceptually significant groupings*,¹ are similar across varying viewpoints. They are based on characteristics of the world rather than the scene, so they are less likely to change under different situations. *Topographic structures*,² such as ridges and valleys in image characteristics, are somewhat complex, and are thus both less

4 INTRODUCTION

common and more easily distinguishable. Similarly, surfaces are characteristics of the world. However, there may not be enough of such complex features to meet the requirements of the procedure. In addition, they may be expensive to compute. The process of detecting and matching features will become more computationally expensive. On the other hand, if simple features are used, there may be enough of them to meet the requirements of self-location, but they may be difficult to distinguish and identify uniquely. There are many more of them, which makes the matching problem more difficult. Thus, incorrect matches may be more common, introducing another source of error in the data. However, they are much less computationally expensive to detect and match. Once the type of features has been chosen, we must also consider how matching should be done. Given the features in one image, should they be searched for in the next image based on motion information?

A fourth question deals with the reliability and limitations of using real (vs. synthetic) data. Digital sensors provide imprecise data, which is due to discretization, and most incorporate noise. In addition, the interpretation of sensor data is still a difficult problem. How much information can be extracted from experimental data and how reliable is this information? What kinds of limitations does the use of experimental data place on computation? The use of experimental data also introduces the problem of how to eliminate "bad data" in the form of incorrect matches. Methods must be developed to detect incorrect matches and remove them from further computations. It may be possible to use characteristics of features which can increase the number of correct matches. Or there may be ways of filtering out incorrect data at later stages of processing.

Another issue is what kind of sensing to use. Since we want to measure the angles of features with adequate precision, sonar is clearly not an appropriate type of sensor. There are several forms of visual sensing which can be used, such as passive (CCD) or active (laser-range finder). If CCD cameras are used, one must also choose between monocular or stereo (or even trinocular) vision. Once the sensor has been chosen, questions arise as to the kinds of limitations placed on computation by the form of sensing. There are other types of sensing as well, having to do with measuring motion of the robot. This raises the issue of whether or not odometry can be used reliably. Does it provide data that can be used, or is the noise simply too great? If odometry measurements are noisy, can the noise be modeled? In addition, we need to determine how well the motion parameters need to be estimated for this approach to work, as well as how changing these parameters affects how much the approach reduces positional uncertainty.

1.4 FOCUS

Clearly, only one or two of the above issues can be explored in three months. Therefore, I needed to limit the scope of my work. I chose to focus on using real CCD monocular imagery to do triangulation. The reason for this choice was that I was interested in exploring the limitations and requirements that using real imagery, especially CCD, monocular imagery, place on a solution of the self-location problem. Will this method work when monocular vision is used? I also wanted to explore the requirements of each part of a system that solves this problem.

In order to meet these goals, I chose to use programs which had already been written at the University of Minnesota (by my thesis advisor, Dr. William Thompson). These programs detect and match features. The algorithms which these programs implement are of interest here only with respect to their limitations when used to solve the self-location problem.

1.5 BACKGROUND

This problem has been addressed by several other researchers. There are two bodies of work which are closely related to the work presented here. The first set of research efforts deals with self-location on an abstract level, and so will only be mentioned briefly. Smith and Cheeseman³ present a method for estimating relative locations of objects and the corresponding expected error using Kalman filtering. Wang⁴ and Watanabe and Yuta⁵ both discuss methods for robot self-location based on odometry.

The second relevant body of work includes research which presents solutions to the self-location problem which use real imagery. The goal of Kriegman, Triendl and Binford⁶ is to instantiate a generic world model. They use stereo in a hallway to detect two-dimensional features (vertical edges). Stereo and motion uncertainty is represented using a normal distribution. Their solution uses odometry to estimate the robot's position. Matches are based on similar intensities and neighborhood consistency (intervals between neighboring matches). They use the extended Kalman filter to reduce uncertainty on both feature and robot position. The state model is composed of the odometry transform, the correspondence points and the covariances. Camera calibration is done to obtain the baseline and to find the relationship between the two cameras' intensities and epipolars. The system appears to work reasonably quickly. Since only features at the horizon are used, it is not clear if there may be situations in which this approach may have problems. In addition, the cameras may become misaligned through time and it is not obvious how well the approach will deal with this.

Ayache and Faugeras⁷ address the problem of building and updating a three-dimensional representation of the environment. They use trinocular vision for detecting 3-D lines in both synthetic and real imagery (a cluttered room). Error is modeled using a Gaussian distribution. Robot position is estimated from odometry. Features are matched based on displacement and geometric parameters, and the Mahalanobis distance is used to reject bad matches and outliers. They use the extended Kalman filter to reduce uncertainty on both feature and robot position. It is not clear how cameras are calibrated and how well this approach works if they become misaligned. This approach also seems computationally expensive. No information is given about how long this solution takes to compute.

Matthies and Shafer⁸ and Matthies and Kanade⁹ address the problem of how best to model error in triangulation. They use stereo vision on real imagery (a room) and use three-dimensional features (obtained with Moravec's interest operator). Their solution estimates robot rotation and translation from feature correspondences. Three-dimensional pruning handles bad matches. This approach uses the fact that under rigid motion, the distance between three-dimensional points does not change over time. They use the extended Kalman filter to reduce uncertainty on both feature positions and robot position. The state model includes the location of feature points. In the results presented, only forward motion along the optical axis was performed (no rotation or sideways translation), so it is not clear how well this approach would work for more complex motions. They do not discuss how the cameras are calibrated or how this approach would deal with changes in camera alignment.

All of these research efforts use stereo or trinocular vision. This simplifies the matching and triangulation problems, since the baseline between cameras may be assumed to be known in advance and not to change. The use of non-monocular vision also allows the approximate localization of features using images obtained

6 INTRODUCTION

from one position of the robot. Two of the three approaches employ edges as features. Since edges are more complex features, more processing is required to obtain them, but there are also fewer features, which simplifies the matching process. All of the solutions use Kalman filtering to integrate the range and position information and reduce robot position uncertainty. Common assumptions include using a Gaussian distribution for modeling error in vision and motion measurements. They assume that the error covariances of vision and motion measurements is known, which is a parameter needed in Kalman filtering. They also assume that triangulation can be linearized (which simplifies the Kalman filtering process) without rendering the information useless. Finally, some of the solutions deal with the initial calibration of the cameras, but none of them deals explicitly with what happens when the cameras become misaligned during motion, as they almost surely will.

2. GENERAL SOLUTION STRATEGY

The general solution strategy is made up of four stages. In the first image processing stage, images are acquired from different positions. They are filtered and features are generated from them. In the second feature matching stage, feature matches are found and filtering is done to eliminate matches of low probability. These matches are linked into longer sequences that span more than two images. These sequences are also filtered. In the third triangulation stage, these feature sequences are used to estimate the range of the features. Filtering can also be done at this step to eliminate inconsistent data. In the fourth and final stage of self-location, the range of features is used to refine the position of the robot.

Triangulation using visual features places conflicting demands on a computational system. In order to find matches, images must be taken from positions which are close to each other. However, the best results from triangulation are obtained the closer the differences between angles get to 90° . In order to meet both of these requirements, matches are found between successive images of a sequence. These pairwise matches are then integrated into lists which track points across the entire sequence of images.

Monocular passive vision (using a CCD camera) is used to acquire a sequence of images. Features in the form of local maxima and minima points were used and matching was done on the basis of proximity and similar intensities. These choices were made because of the software that was available and because of the short duration of the project. Error was not modeled. The final step of robot position refinement was not dealt with, but a brief discussion of techniques which could be useful is presented at the end of this paper.

2.1 ALGORITHM

Figure 3 illustrates one algorithm which implements the general strategy outlined above. Note that in this version, triangulation and filtering are both included in the move-perceive loop. However, it might make sense to move both of these computations outside of the loop, for two reasons. First, the robot's position might be precise enough through several iterations, and it might not be necessary to refine the position this frequently. Second, it might not be possible to triangulate this frequently, if the angular position of features does not change rapidly enough.

8 GENERAL SOLUTION STRATEGY

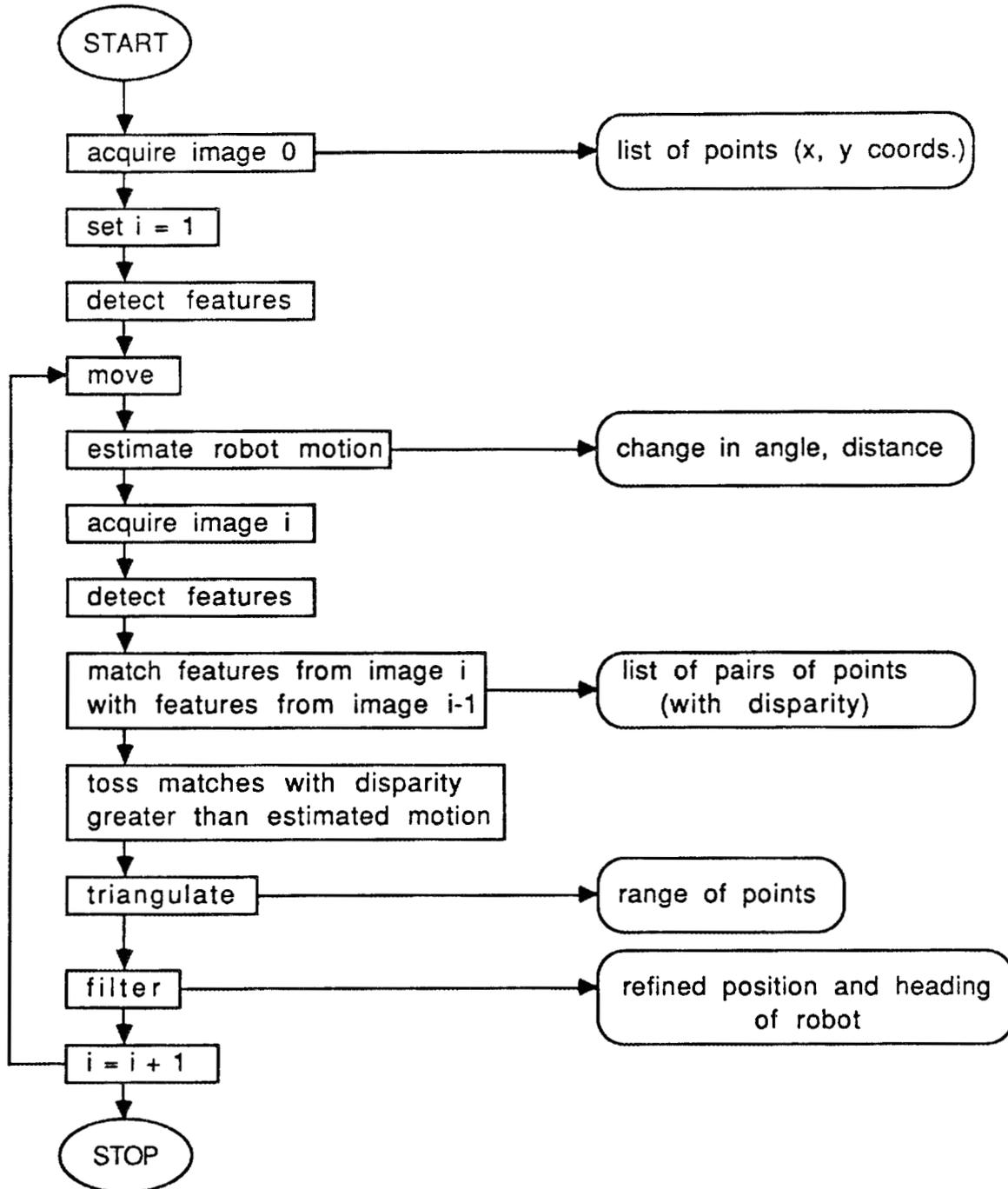


Fig. 3. One implementation of the general strategy.

3. IMPLEMENTATION

All but the programs used to display the results run on both a VME-based system and a UNIX system. The display programs run only on the VME system. This system has a Motorola 68020 CPU, a Datacube digitizer, and a CCD video camera.

3.1 VISION

Images of size 256×256 were used. Here is a sample image, taken in the CESAR lab:



Fig. 4. Sample image.

In the image processing stage, the $\nabla^2 * G$ operator is applied to the images. The images are blurred by two Gaussians with distributions determined by the characteristics of the images. A Laplacian operator is then applied to detect zero crossings. Finally, features are detected by locating local minima and maxima in the resulting Laplacian image.

Figure 5 shows the resulting image after the Laplacian is applied. Note that there is some noise in this processed image; zero crossings have been found in areas in which, upon examination of the original image, they would not be expected to be found.

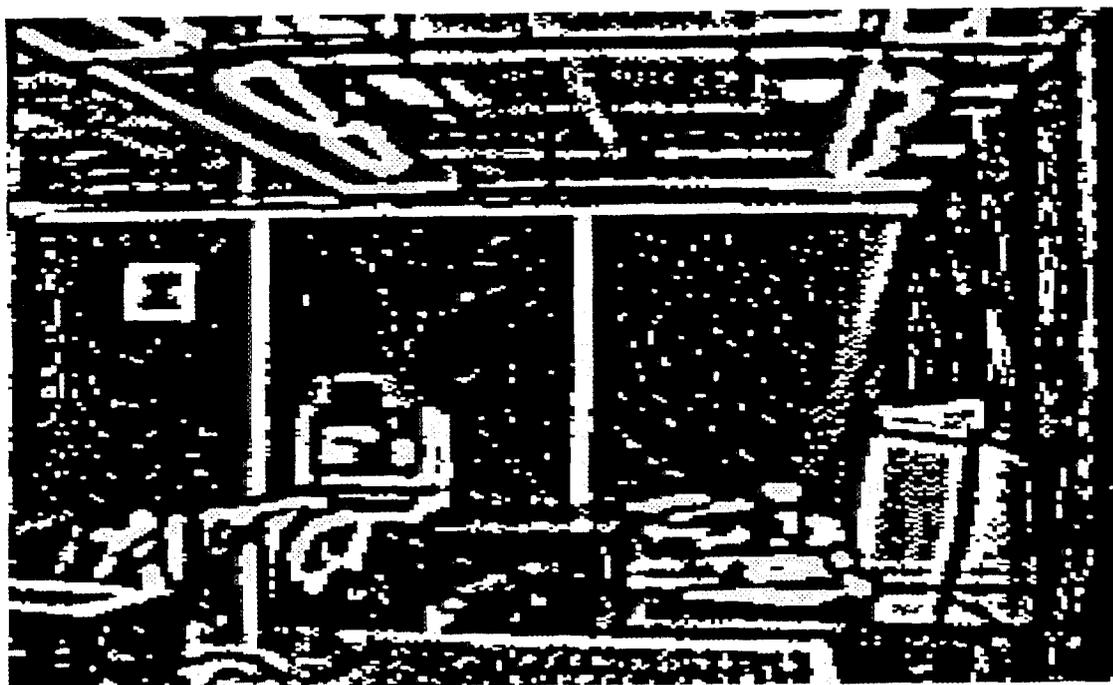


Fig. 5. Sample image after blurring and Laplacian.

3.2 MATCHING

In the feature matching stage, initial matches are found on the basis of proximity of the imaged positions of features in two images. Then, better matches are found iteratively based on the intensity of these minima and maxima, using a local neighborhood consistency measure. In Fig. 6, matched features (in white) are overlaid on the original image. Note that again the results are noisy. Several features have been found in unexpected places.

In the last step of matching, pairs of matches are linked to form longer sequences of matches. Sequences are shown in Fig. 7 overlaid on the original image. The position of the feature in the first image is shown in white. The black line plots the displacement of the feature from the original image to the image in which the feature was last found. Note that the length of the sequence does not necessarily indicate that the sequence spans many images. It indicates only that the displacement between the first and last position of the feature is large.



Fig. 6. Matched feature points.



Fig. 7. Sequences of feature matches.

12 IMPLEMENTATION

Not surprisingly, several sequences have been found which are not consistent with the other sequences. The camera is translating towards a point (the focus of expansion) which is approximately one-fourth of the way down from the top of the image and one-fourth from the right side. Thus we would expect that all of the segments representing sequences, when extended, would intersect at or near that point. Clearly, however, some of the segments representing sequences are pointing the wrong way or have the wrong orientation.

3.3 TRIANGULATION

3.3.1 Obtaining Feature Angles

Before discussing these approaches in more detail, we first describe how to obtain the angle of a feature with respect to the robot. In Fig. 8, the feature of interest, F , lies in a plane which is parallel to the image plane. The robot, R , lies in the plane of motion, which is perpendicular to the first plane. There are two angles which describe the position of the feature, a and b . a is the angle between the optical axis and the projection of the feature on the plane of motion. b is the angle between the optical axis and the projection of the feature on a third plane which is orthogonal to the first two planes. I have chosen to use only one angle for triangulation calculations in order to simplify the calculations. I use a as this angle because when the camera is facing the direction of motion, a has more degrees of freedom than b . b could be used in addition, but in most circumstances would not provide much extra information. However, if the camera were pointed at the ceiling, for example, b would provide more information than a and should be used instead. The range that is found for the feature, r , refers to the distance between the robot and the projection of the feature onto the plane of motion. Note that in the discussion which follows, robot headings and the angular positions of features are given with respect to the robot's original heading.

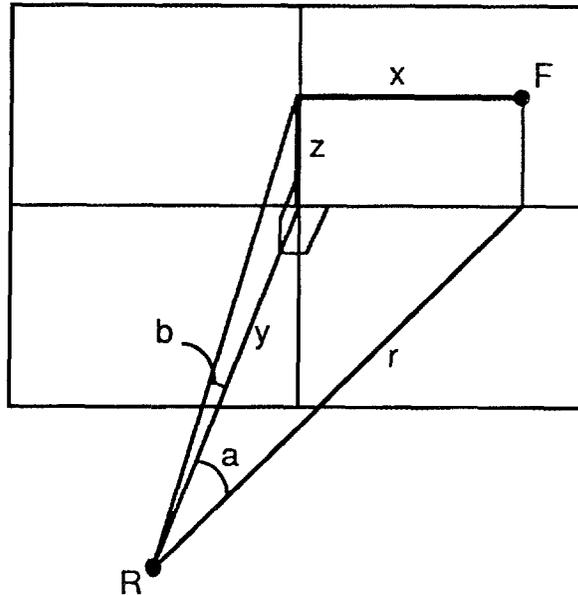


Fig. 8. Angles used in triangulation.

3.3.2 Robot Triangulation

Visual measurements provide the angle from the robot, R , to a feature, F . The position of the feature's projection is given by:

$$x = X + r \sin(a)$$

$$y = Y + r \cos(a)$$

where (X, Y) is the robot's position, (x, y) is the feature's position, a is the angular position of the feature as discussed in the previous subsection, and r is the range from the robot to the feature.¹⁰

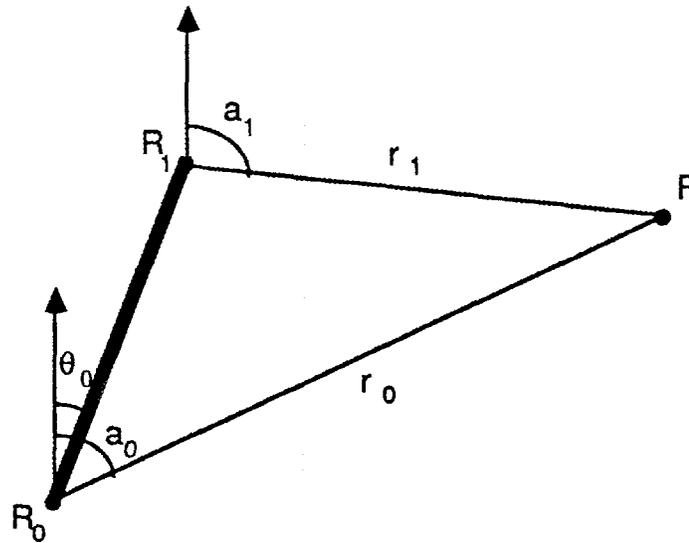


Fig. 9. Robot triangulation.

Given two measurements of a feature's angular position taken at two positions of the robot (the position of the robot is indicated by subscripts):

$$x = X_0 + r_0 \sin(a_0) \quad x = X_1 + r_1 \sin(a_1)$$

$$y = Y_0 + r_0 \cos(a_0) \quad y = Y_1 + r_1 \cos(a_1)$$

we can calculate the range to the feature from either position (in this case, from (X_1, Y_1)):

$$r_1 = \frac{(X_1 - X_0) \cos(a_0) - (Y_1 - Y_0) \sin(a_0)}{\sin(a_0 - a_1)}$$

When the difference between the two angles is small, the resulting uncertainty in the calculated position of the feature will be large. This uncertainty can be reduced in two ways. We can ignore the results of any calculations in which the difference between the two angles is below a certain threshold. We can also use the absolute value of $\sin(a_0 - a_1)$ as a weighting function when using the calculated range.

3.3.3 Feature Triangulation

There are two computations involved in feature triangulation. The first, which calculates the range to (or position of) a feature, must be done when the feature is first encountered. The second computation uses the known positions of features to calculate the current position of the robot.

3.3.3.1 Calculating Range to an Unknown Feature

Calculating the range to a new feature requires the angular position of the feature from both positions, a_{i-1} and a_i , the previous position of the robot, (X_{i-1}, Y_{i-1}) , plus the distance, d_{i-1} , and the heading, θ_{i-1} , that the robot traveled between the previous and current positions.

We can calculate the range to the feature from both of the robot's positions by using the Law of Sines:

$$\frac{r_{i-1}}{\sin(a_i - \theta_{i-1})} = \frac{r_i}{\sin(a_{i-1} - \theta_{i-1})} = \frac{d_{i-1}}{\sin(a_i - a_{i-1})} .$$

Therefore,

$$r_{i-1} = \frac{d_{i-1} \sin(a_i - \theta_{i-1})}{\sin(a_i - a_{i-1})}$$

and

$$r_i = \frac{d_{i-1} \sin(a_i - \theta_{i-1})}{\sin(a_i - a_{i-1})} .$$

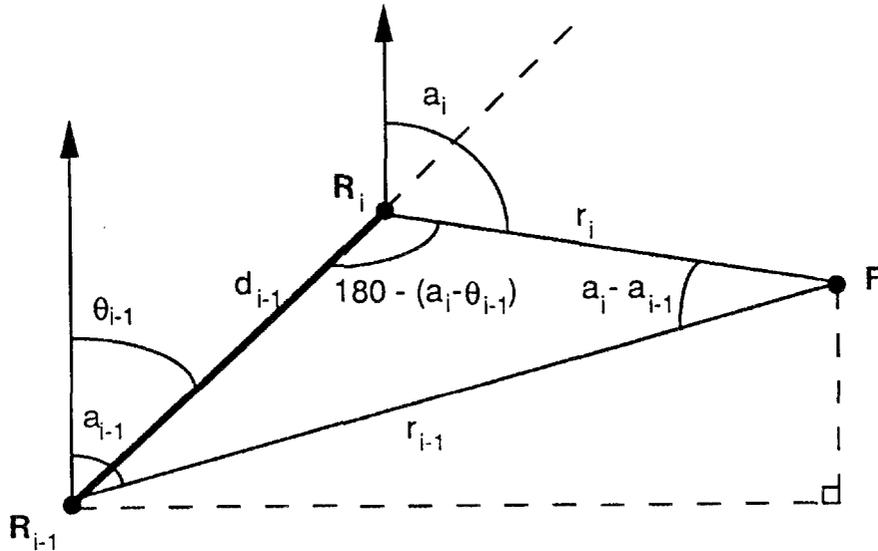


Fig. 10. Calculating feature range.

We can also calculate the position of the feature. The robot's current position is given from the distance and heading traveled (obtained from odometry):

$$\begin{aligned} X_i &= X_{i-1} + d_{i-1} \sin(\theta_{i-1}) \\ Y_i &= Y_{i-1} + d_{i-1} \cos(\theta_{i-1}) \end{aligned} .$$

We know the position of the feature given the robot's previous position and the feature's range and angular position from the robot's previous position:

$$\begin{aligned} x &= X_{i-1} + r_{i-1} \sin(a_{i-1}) \\ y &= Y_{i-1} + r_{i-1} \cos(a_{i-1}) \end{aligned} .$$

Since we can express the range of the feature in terms of the distance and heading traveled between robot positions and the angular positions of the feature from both positions, we can eliminate range from the previous equations to obtain the feature's position:

$$\begin{aligned} x &= X_{i-1} + \frac{d_{i-1} \sin(a_i - \theta_{i-1}) \sin(a_{i-1})}{\sin(a_i - a_{i-1})} \\ y &= Y_{i-1} + \frac{d_{i-1} \sin(a_{i-1} - \theta_{i-1}) \cos(a_{i-1})}{\sin(a_i - a_{i-1})} \end{aligned} .$$

3.3.3.2 Calculating the Robot's Current Position

The second computation involved in feature triangulation uses the ranges from the robot's previous position to two features and the angular positions of these two features at the previous and current positions to calculate the current position of the robot. This requires four measurements.

Measurements of feature n from both robot positions (feature number is indicated by superscripts):

$$\begin{aligned} x^n &= X_{i-1} + r_{i-1}^n \sin(a_{i-1}^n) & x^n &= X_i + r_i^n \sin(a_i^n) \\ y^n &= Y_{i-1} + r_{i-1}^n \cos(a_{i-1}^n) & y^n &= Y_i + r_i^n \cos(a_i^n) \end{aligned}$$

Measurements of feature m from both positions:

$$\begin{aligned} x^m &= X_{i-1} + r_{i-1}^m \sin(a_{i-1}^m) & x^m &= X_i + r_i^m \sin(a_i^m) \\ y^m &= Y_{i-1} + r_{i-1}^m \cos(a_{i-1}^m) & y^m &= Y_i + r_i^m \cos(a_i^m) \end{aligned}$$

These measurements can be combined to yield the range to either of the features from the current position:

$$r_i^n = \frac{r_{i-1}^m \sin(a_i^m - a_{i-1}^m) + r_{i-1}^n \sin(a_{i-1}^n - a_i^m)}{\sin(a_i^n - a_i^m)} .$$

The new position of the robot can be calculated by:

$$\begin{aligned} X_i &= X_{i-1} + r_{i-1}^n \sin(a_{i-1}^n) - r_i^n \sin(a_i^n) \\ Y_i &= Y_{i-1} + r_{i-1}^n \cos(a_{i-1}^n) + r_i^n (-\cos(a_i^n)) \end{aligned} .$$

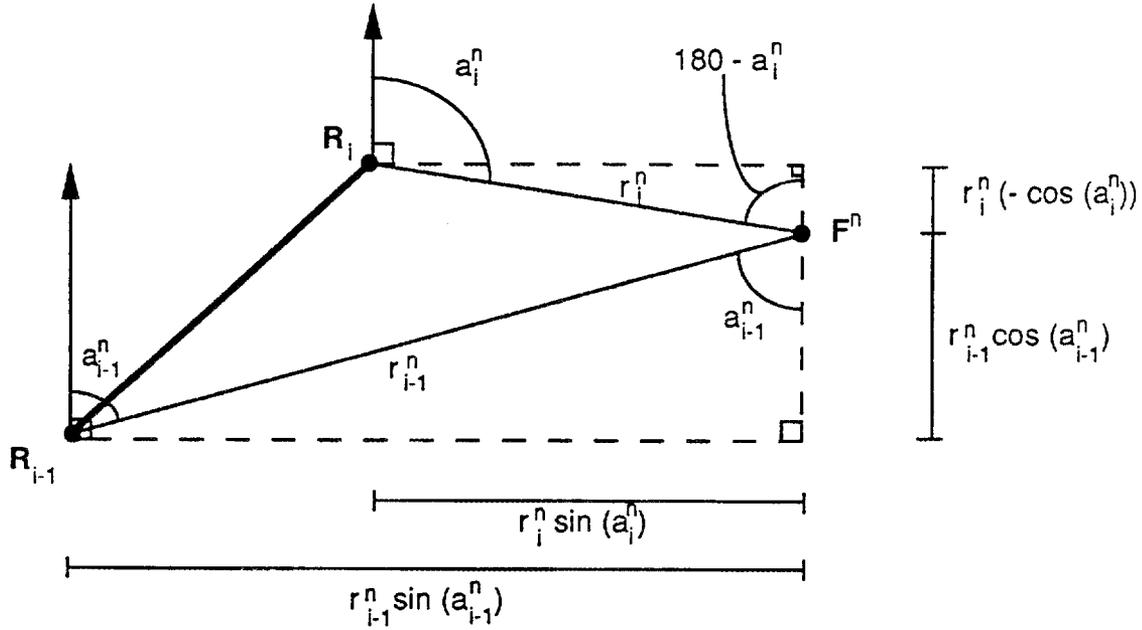


Fig. 11. Calculating robot position.

Therefore, we can calculate the robot's current position:

$$X_i = X_{i-1} + r_{i-1}^n \sin(a_{i-1}^n) - \frac{\sin(a_i^n)[r_{i-1}^m \sin(a_i^m - a_{i-1}^m) + r_{i-1}^n \sin(a_{i-1}^n - a_i^m)]}{\sin(a_i^n - a_i^m)}$$

$$Y_i = Y_{i-1} + r_{i-1}^n \cos(a_{i-1}^n) - \frac{\cos(a_i^n)[r_{i-1}^m \sin(a_i^m - a_{i-1}^m) + r_{i-1}^n \sin(a_{i-1}^n - a_i^m)]}{\sin(a_i^n - a_i^m)}$$

3.4 SELF-LOCATION

There are two ways to use the feature range information. The calculations may be made pairwise and then combined, or a technique known as Kalman filtering may be used.

Kalman filtering is a recursive form of least squares estimation. It is useful when the parameter of interest, in this case the robot's position and the range of features to it, is changing. The information that is required to use this method include models of the system and of the measurement process. The system model includes a description of the transition as well as its noise and the covariance of the noise. In other words, equations which describe the motion of the robot and incorporate the noise involved in this process must be generated. The measurement model is made up of a description of how measurements may be predicted, as well as the noise and covariance of measurements.

4. EXPERIMENTAL DATA

One experiment was run in which twelve images were acquired. The camera was moved forward five inches between each image's acquisition. Figure 12 shows the original image in the sequence.

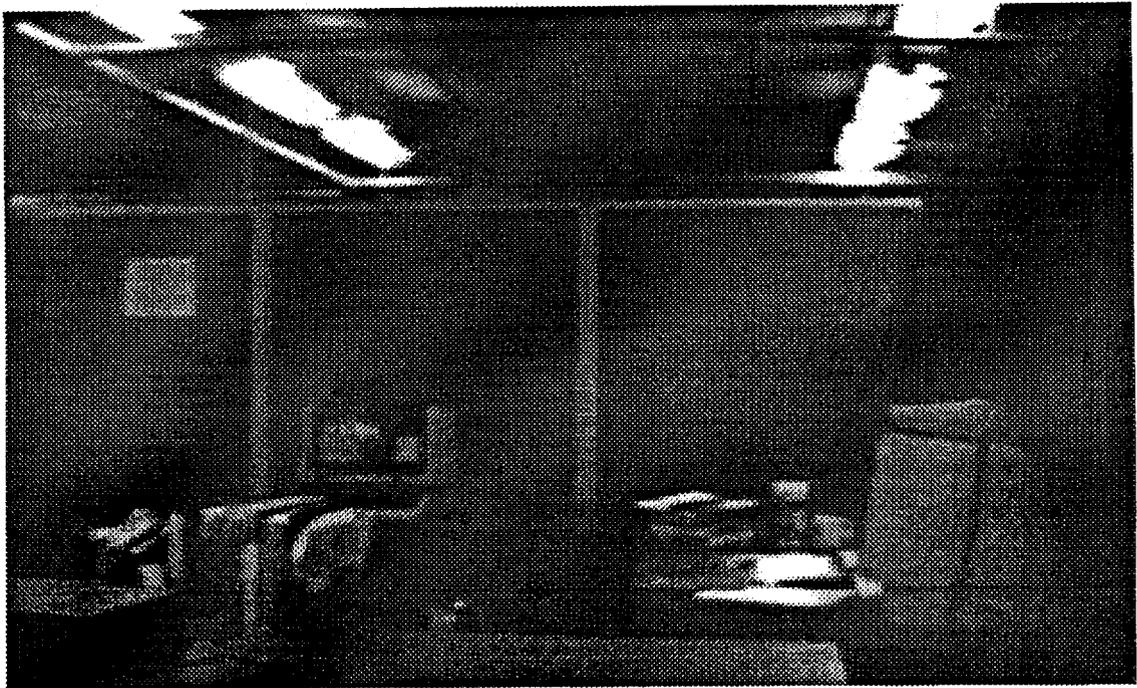


Fig. 12. First image of sequence.

In Table 1, the first column (image) refers to the position of the image in the sequence. The second column (features) gives the number of features found in that image. The third column (pairs) gives the number of matched features found between consecutive images. The fourth column (sequences) gives the total number of sequences of all lengths which originate in the corresponding image. The subsequent columns give the number of sequences of length two through nine which originate in each image.

There was found to be relatively large error in the preliminary data obtained from triangulation (not shown). This can be explained by the occurrence of incorrect matches, as well as the short length of sequences that were found (see Table 1). The judicious filtering of short sequences may eliminate much of the useless data.

5. SUMMARY

Preliminary results demonstrate that using experimental (vs. synthetic) data to calculate the positions of features and the robot poses some difficult problems. However, even this relatively simple approach has provided some useful information. Clearly more work needs to be done to explore these problems further.

5.1 ROBOT VS. FEATURE TRIANGULATION

One of the advantages of robot triangulation is that it uses all of the available information. However, it requires a model of the error in the odometry, which may not be known with much precision, if at all. This method is most useful in environments in which features appear and disappear frequently, and cannot be relied upon to remain.

An advantage of feature triangulation is that it uses odometry data sparingly. When it does use odometry, it requires only the distance traveled by the robot from its previous position to its current one, rather than total distance traveled, which has accumulated error. In addition, this information is required only for new features, when their positions need to be computed. However, feature triangulation requires that old features remain available for more than two time steps, so that they may be used to calculate the position of the robot. This method may initially appear to be more desirable, but in environments in which features appear and disappear frequently, this method will not have any advantage over robot triangulation, since odometry will have to be used just as frequently.

The usefulness of these methods also depend on the type of feature used. More complex features are less likely than simple features to appear and disappear frequently, and so may be coupled successfully with the feature triangulation method. Simple features would be best used with robot triangulation.

6. RECOMMENDATIONS

There is clearly a great deal of research which could build upon the work reported here. As was discussed in the background section, Kalman filtering can be used to integrate the information obtained from both vision (feature angles) and odometry. It would be informative to implement Kalman filtering to see how well this method would work to reduce the uncertainty of the robot's position.

Another aspect of this work that could be pursued is better features, in terms of complexity and reliability. More complex features, such as edges, could be used in place of points. More reliable features, obtained by additional filtering, would provide better data for triangulation. Both of these characteristics should be implemented. This would address the question of how many features need to be used to obtain good results in triangulation.

Currently, matching is done simply by examining a neighborhood around the imaged position of a feature. One way to increase the number of correct feature matches is to predict the new imaged position of a feature by using knowledge about the motion of the robot and the range of the feature. Another way to cut down on the number of incorrect matches is by filtering sequences which have been found. Local filtering could be done by examining the local neighborhood of a sequence to see if the length and orientation of the sequence is consistent with other nearby sequences. More global filtering could be done by determining the focus of expansion and eliminating all inconsistent sequences.

The only experimentation which was done was simple translation. In theory, the approach presented here should work for rotation as well, but this clearly needs to be tested. In addition, an assumption was made that the camera pointed in the same direction as the robot's motion. If this is relaxed, it might be possible to obtain better data for triangulation by, for example, looking to the side while moving forward.

The two triangulation approaches could be further studied and compared. Are they indeed complementary? Is there some way that they could be combined and provide better results? An apparent drawback of feature triangulation is that features may appear and disappear so frequently in most environments that odometry has to be used at every time step. It would be useful to see whether this is indeed true.

7. ACKNOWLEDGMENTS

I thank Dr. François Pin for his support and advice concerning strategies in general research as well as for providing specific ideas to investigate. I also thank Dr. David Reister for his technical assistance and willingness to act as a sounding board for new ideas.

REFERENCES

1. D. G. Lowe, *Perceptual Organization and Visual Recognition*, Kluwer Academic Publishers, 1985.
2. T.-C. Pong, "Matching Topographic Structures in Stereo Vision," Technical Report TR 87-2, University of Minnesota, Minneapolis, Minnesota, 1987.
3. R. C. Smith and P. Cheeseman, "On the Representation and Estimation of Spatial Uncertainty," *The International Journal of Robotics Research* 5(4), 56-68 (Winter 1986).
4. C. M. Wang, "Location Estimation and Uncertainty Analysis for Mobile Robots," *Proceedings of the International Conference on Robotics and Automation* 2, pp. 1230-1235 (1988).
5. Y. Watanabe and S. Yuta, "Estimation of Position and Its Uncertainty in the Dead Reckoning System for the Wheeled Mobile Robot," *Proceedings of the 20th ISIR*, pp. 205-212 (1989).
6. D. J. Kriegman, E. Triendl, and T. O. Binford, "Stereo Vision and Navigation in Buildings for Mobile Robots," *IEEE Transactions on Robotics and Automation* 5(6), pp. 792-803 (December 1989).
7. N. Ayache and O. D. Faugeras, "Maintaining Representations of the Environment of a Mobile Robot," *IEEE Transactions on Robotics and Automation* 5(6), pp. 804-819 (December 1989).
8. L. Matthies and S. A. Shafer, "Error Modeling in Stereo Navigation," *IEEE Journal of Robotics and Automation*, RA-3(3), pp. 239-248 (June 1987).
9. L. Matthies and T. Kanade, "The Cycle of Uncertainty and Constraint in Robot Perception," *Proceedings of the International Symposium on Robotics Research*, MIT Press (1987).
10. D. B. Reister, "The Self-Location Problem for a Robot," Personal Communication (1989).

APPENDIX

PROCEDURE FOR USING PROGRAMS

The pseudo-code on the following page outlines the procedure to be used when running the programs to solve that part of robot self-location which has been implemented as of September 1990. (Those programs with names which begin with capital letters are so marked to indicate that there are two slightly different versions on the VME and UNIX systems.) Figure 13 presents a diagram summarizing this procedure.

The following programs are used:

- Delsqg
- lminmax
- featurelist
- Flink
- Match__init
- Match

This procedure requires eight arguments: then number of files to be processed, **files**, the basename of the files, **base**, the diameter of the Gaussian kernels, **dia**, the threshold level, **thr**, the disparity window, **disp**, the neighborhood, **nbr**, the number of iterations in the relaxation process, **iters**, and the threshold to use when filtering matches, **ft_r__thr**. Recommended values are **dia = 3**, **thr = 225**, **disp = 7**, **nbr = 11**, **iters = 10**, and **ft_r__thr = 0.5**.

Output filenames are created by each program by appending the appropriate number or letter to the basename of the file. See the source code of particular programs for more detail.

Arguments which are passed to this procedure are boldfaced; internal variables are italicized.

```

filenum = 1
prev = concat (base, ".", filenum)
next = concat (base, ".", filenum + 1)
comb = concat (base, ".", filenum, filenum + 1)

Delsqg prev dia
lminmax prev_dia_thr
featurelist prev_dia_thr.3_

for (i = 0; i < files; i++) {
  /* process images and detect features */
  Delsqg next dia
  lminmax next_dia_thr
  featurelist next_dia_thr.3_

  /* match features */
  Flink prev_dia_thr.3__ next_dia_thr.3__ \
    comb_dia_thr.3_disp_disp
  Match_init -i comb_dia_thr.3_disp_ prev next 5 1 \
    comb_dia_thr.3_disp_d_
  Flink -neighbors prev_dia_thr.3__ \
    comb_dia_thr.3_disp_d_n_nbr
  Match comb_dia_thr.3_disp_d_ comb_dia_thr.3_disp_d_n_ \
    default iters comb_dia_thr.3_disp_d_n_m

  /* filter matches */
  mv comb_dia_thr.3_disp_d_n_m combm
  matchfilter combm ftr_thr

  filenum++;
  prev = concat (base, ".", filenum)
  next = concat (base, ".", filenum + 1)
  comb = concat (base, ".", filenum, filenum + 1)
}

/* generate feature sequences for all but last two images */
/* (second to last image only has sequences of length */
/* two, which have already been generated in match file) */
for (i = 1; i < files; i++) {
  track_points -i files base
}

/* triangulate */
for (i = 1; i <= files; i++) {
  filename = concat (base, ".", i, "mfs")
  triang filename triang.motion
}

```

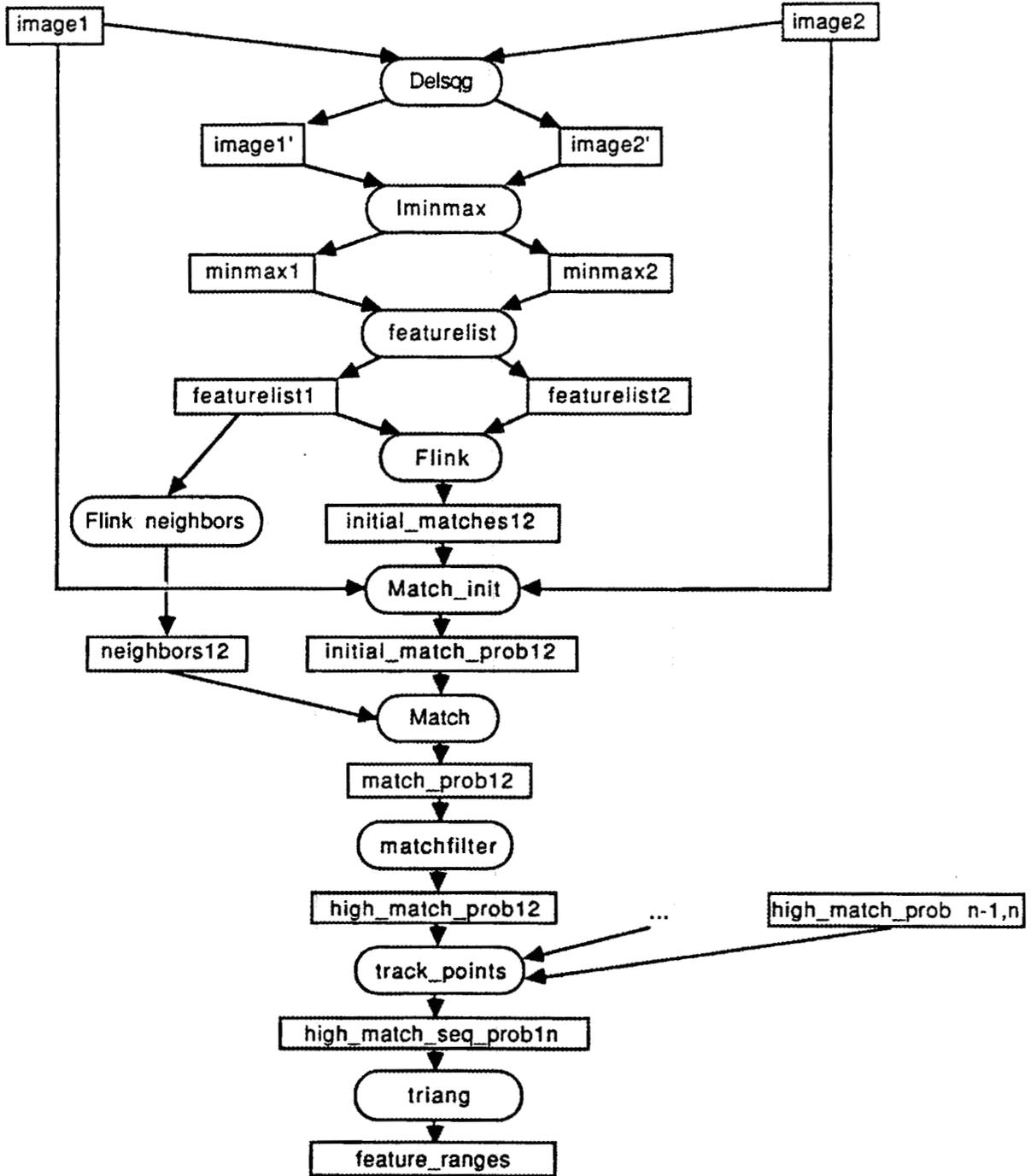


Fig. 14. Diagram of program data flow.

DESCRIPTION OF PROGRAMS

To get feature points:

```
Delsqg image1 laplacian1 diameter  
Delsqg image2 laplacian2 diameter
```

Applies $\nabla^2 * G$ operator to original image. Output is image.

```
set_thresh laplacian_file level
```

Sets a threshold for use by the lminmax program.

```
lminmax laplacian1 minmax [threshold [neighborhood]]  
lminmax laplacian2 minmax [threshold [neighborhood]]
```

Computes local maxima and minima of $\nabla^2 * G$ files. Input is image file. Output file is list of feature points.

To match feature points:

```
diff_stats image1 image2 [rows [cols]]
```

Computes difference statistics between two image files. (Useful in setting parameters for the matching programs.) Images have default size of 256×256 .

```
featurelist minmax feature_list_1  
featurelist minmax feature_list_2
```

Converts images of feature points to text files listing coordinates and type of feature points.

```
Flink [-nolabel] feature_list_1 feature_list_2 matchlist  
window_size
```

Merges text files indicating feature points to create single file indicating matches. -nolabel flag causes feature types to be ignored.

Match_init [-il] matchlist image1 image2 window inc [noise maxdiff] init_prob

Adds initial likelihoods to each possible match.

Flink -neighbors feature_list_1 neighbors window_size

Makes a list of neighbors for first frame features.

Match [-options] init_prob neighbors parm_file iterations output

Does actual matching. Output is a text file with disparity and likelihood values for each first frame feature point.

matchfilter input_file probability_threshold

Removes all matches from input_file with probabilities below given threshold.

track_points [-]current_file_num numfiles basename

Links together sequences of feature points connected across multiple frames. (File names are assumed to be in the format basename.num, where num indicates the position of the image in the sequence. The beginning value of num is assumed to be 1 with the final value numfiles.)

triang input_file motion_file

Calculates range of features from robot. Input is either a disparity file or a feature point trace. Output is a file listing features and their ranges.

To Display Images:

```
display frame [image [rows [columns]]]
```

Reads in an image file and displays it. Specified frame is displayed if no image is given. Optional arguments give size of image; default is 256×256 .

```
display0 frame image [origin_row [origin_column]]
```

Reads in an image file and displays it. Specified frame is displayed if no image is given. Optional arguments give placement of origin of image in memory; default is (50,50). The image is assumed to be 256×256 .

To Display Matches:

```
displaym frame image [origin_row [origin_column]]
```

Reads in a disparity file and displays the features for which matches have been found. Data already in memory is overwritten only where features are present. Optional arguments give placement of origin of image in memory; default is (50,50). The image is assumed to be 256×256 .

To Display Match Sequences:

```
displaytc frame image [origin_row [origin_column]]
```

Reads in a feature point trace file, displays the feature point at its originating position and plots a line from this position to its final position. Data already in memory is overwritten only where features are present. Optional arguments give placement of origin of image in memory; default is (50,50). The image is assumed to be 256×256 . (A feature point trace file contains a list of matches tracked over multiple, consecutive frame pairs. It is produced by track-points).

Shell Files to Automate Processing (On Iris Only):

`goip number basename diameter threshold_level`

Sets up feature points to be matched. Uses features defined by extrema in the Laplacian of the smoothed images. **number** is the number of image files to be processed. **basename** is the base name of the image files. (File names are assumed to be in the format **basename.num**, where **num** indicates the position of the image in the sequence. The beginning value of **num** is assumed to be 1 with the final value **number**.) **diameter** is the size for the $\nabla^2 * G$ filter. **threshold_level** is the intensity value to be used for thresholding features.

`gomatch number basename max_disparity max_neighbor iterations`

Establishes the initial list of matches and then runs the relaxation matching program. **number** is the number of image files to be processed. **basename** is the base name of the image files. (File names are as in `goip`.) **max_disparity** is the maximum possible disparity expected between frames. (Maximum vertical and horizontal disparities are currently assumed to be the same.) **max_neighbor** defines the neighborhood over which the relaxation updating looks for support. Make this at least two times larger than **max_disparity**. **iterations** is the number of passes through all of the possible matches that are made. If more than two image files are specified, each consecutive pair will be matched.

`gotrack number basename`

Generates sequences of feature matches. **number** is the number of image files to be processed. **basename** is the base name of the image files. (File names are as in `goip`.) **number** - 1 iterations are performed, since running `track_points` on the last match file is redundant (the last match file contains sequences of length two).

PROGRAM INTERFACES

Table 2. Inputs and outputs of programs.

Name	Input	Output
Delsqg	image	$\nabla^2 * G$ image
lminmax	$\nabla^2 * G$ image	overlay image with local minima and maxima
featurelist	overlay image	text file list of feature points and their coordinates
Flink	two feature lists	list of match pairs (points and the match disparities)
Flink	one feature list	list of neighbors
Match_init	two images and the initial match pairs	list of matches and initial probabilities
Match	list of initial matches and list of neighbors	list of matches and final probabilities
matchfilter	list of matches	list of matches with probabilities over given threshold
track_points	multiple lists of match pairs	multiple lists of match sequences

FILE FORMATS**Format of Feature List File:**

<feature list>	text string indicating type of file
<...misc info...>	
nmline nmsamp	number of lines, samples in image file
n	number of feature points
i1 j1 label	line and sample coordinates, label
i2 j2 label	
i3 j3 label	
...	
...	
in jn label	

(Points are sorted in non-decreasing order of *i*, and within *i* in non-decreasing order of *j*.)

Format of Match List:

<match list>	text identifier
<match list window size = n>	
nmline nmsamp	number of lines, samples in original file
total	number of frame one feature points
line samp	line, sample index of first feature point
nmatch1	number of matches for this point
dline1 dsamp1	line and sample disparities
:	
dlinen dsampn	
:	
:	
nmatchm	number of matches for last point
dline1 dsamp1	
:	
dlinen dsampn	

Format of Initial Match File:

```

<initialized list>
<match list window size = n>
<match_init: window = n, inc = n>
nmline nmsamp      number of lines, samples in original image
total             number of frame one feature points
line samp         line, sample index of first feature point
nmatch1           number of matches for this point
dline1 dsamp1 prob1  line and sample disparities, initial probability
:
dlinen dsampn probn
p_unmatch         probability of unmatchable
:
:
nmatch            number of matches for last point
dline1 dsamp1 prob1
:
dlinen dsampn probn
p_unmatch

```

Format of Neighbor File:

```

<neighbor list>
<neighbor list window size = n>
nmline nmsamp      number of lines, samples in original image
total             number of frame one feature points
line samp         line, sample index of first feature point
n_neighbor_1      number of neighbors for this point
line1 samp1       line and sample indices of neighbor
:
linen sampn
:
:
n_neighbor_m      number of neighbors for last point
line1 samp1
:
linen sampn

```

Format of Parameter File:

```

parm_1_name = parm_1_value (Parameters can be in any order,
:                          but all should be specified.)
:
parm_n_name = parm_n_value

```

Format of Match Files:

```

<disparity file>
<...misc info...>
nmline nmsamp          number of lines, samples in original image
total                 number of frame one feature points
line_1 samp_1         line, sample index of first feature point
dline1 dsamp1 prob1   line and sample disparities, likelihood
:
:
line_n samp_n
dlinen dsampn probn

```

(The disparities in the file correspond to the match labels with the largest likelihood. The unmatchable label is coded implicitly. A small value for the likelihood indicates that "unmatched" is the result of the matching. A likelihood of 0 means that the disparity values in the file should be ignored, and the point is definitely "unmatchable.")

Format of Feature Point Trace:

```

<feature point trace>
<...misc info...>
nmline nmsamp          number of lines, samples in original image
nmpoints              number of feature points
nmmatches              maximum number of future feature points
line1 samp1           first feature point
nmfuture1              number of matches
line1f1 samp1f1 prob1f1 line, sample of nearest new match
:
line1fn samp1fn prob1fn line, sample of farthest new match
:
linem sampm           last feature point
nmfuturem              number of matches
linemf1 sampmf1 probmf1 line, sample of nearest new match
:
linemfn sampmf1n probmf1n line, sample of farthest new match

```

Format of Triangulation Files:

```

<triangulation file>
<...misc info...>      information records
nmline nmsamp           size of image
features               number of features
nmmatches              maximum number of matching points
line_1a samp_1a angle_1a first point in sequence
total_1                number of matches of 1st feature
ln1b sm1b an1b pr1b rn1b corresponding point in 2nd image
      :
      :
ln1z sm1z an1z pr1z rn1z last corresponding point
      :
      :
line_na samp_na angle_na point in next to last image
total_n                number of matches of last feature
lnnb smnb annb prnb rnnb corresponding point in 2nd image
      :
      :
lnnz smnz annz prnz rnnz last corresponding point

```

Format of the Motion File:

```

<motion file>
total                  number of steps
dist_1 heading_1      distance and heading for first step
      :
      :
dist_n heading_n      distance and heading for last step

```

ACCESSING PROGRAMS

These programs work on both the VME-based system, vislab, and on the UNIX system, IRIS. Display capabilities exist only on the VME system. On vislab, the programs are currently located in /hu/usr/liz/BIN. The source and on-line documentation is in /hu/usr/liz/SRC, and the relocatable files are in /hu/usr/liz/RELOC. Image sequences are in /hu/usr/liz/256IMAGES. On IRIS, the programs are currently located in /usr/people/liz/bin, and the source, documentation and relocatable files are in /usr/people/liz/src. Image sequences are in /usr/people/liz/256 images.

INTERNAL DISTRIBUTION

- | | |
|-------------------|---|
| 1. J. E. Baker | 23. J. C. Schryver |
| 2. A. L. Bangs | 24. P. F. Spelt |
| 3. M. Beckerman | 25. F. J. Sweeney |
| 4. R. J. Carter | 26. M. A. Unseren |
| 5. J. R. Einstein | 27. R. C. Ward |
| 6. K. Fujimura | 28-29. Laboratory Records
Department |
| 7. C. W. Glover | 30. Laboratory Records,
ORNL-RC |
| 8. J. P. Jones | 31. Document Reference
Section |
| 9. H. E. Knee | 32. Central Research Library |
| 10. G. Liepins | 33. ORNL Patent Section |
| 11. E. M. Oblow | |
| 12-16. R. C. Mann | |
| 17-21. F. G. Pin | |
| 22. D. B. Reister | |

EXTERNAL DISTRIBUTION

34. Dr. Peter Allen, Dept. of Computer Science, 450 Computer Science, Columbia University, New York, NY 10027
35. Dr. Wayne Book, Department of Mechanical Engineering, J. S. Coon Building, Room 306, Georgia Institute of Technology, Atlanta, GA 30332
36. Prof. John J. Dorning, Department of Nuclear Engineering and Physics, Thornton Hall, McCormick Rd., University of Virginia, Charlottesville, VA 22901
37. Dr. Steven Dubowsky, Massachusetts Institute of Technology, Building 3, Room 469A, 77 Massachusetts Ave., Cambridge, MA 02139
38. Dr. Avi Kak, Dept. of Electrical Engineering, Purdue University, Northwestern Ave., Engineering Mall, Lafayette, IN 47907
39. Dr. James E. Leiss, 13013 Chestnut Oak Dr., Gaithersburg, MD 20878
40. Prof. Neville Moray, Dept. of Mechanical and Industrial Engineering, University of Illinois, 1206 West Green St., Urbana, IL 61801
41. Prof. Mary F. Wheeler, Dept. of Mathematical Sciences, Rice University, P.O. Box 1892, Houston, TX 77251
42. Dr. Wes Snyder, Center for Communications and Signal Processing, North Carolina State University, P.O. Box 7914, Raleigh, NC 27695-7914
- 43-47. Elizabeth R. Stuck, Computer Science Dept., 4-192 EE/CS Building, University of Minnesota, 200 Union St., Minneapolis, MN 55455
48. Dr. William B. Thompson, Computer Science Dept., 4-192 EE/CS Building, University of Minnesota, 200 Union St., Minneapolis, MN 55455
- 49-58. Technical Information Center, P.O. Box 62, Oak Ridge, TN 37831
59. Assistant Manager, Energy Research and Development, DOE/ORO.