Model-Based Vision for Space Applications

Karen Chaconas
Marilyn Nashman
Ronald Lumia

National Institute of Standards and Technology
Robot Systems Division
Gaithersburg, MD 20899

Abstract

This paper describes a method for tracking moving image features by combining spatial and temporal edge information with model based feature information. The algorithm updates the two-dimensional position of object features by correlating predicted model features with current image data. The results of the correlation process are used to compute an updated model. The algorithm makes use of a high temporal sampling rate with respect to spatial changes of the image features and operates in a real-time multi-processing environment. Preliminary results demonstrate successful tracking for image feature velocities between 1.1 and 4.5 pixels every image frame. This work has applications for docking, assembly, retrieval of floating objects and a host of other space-related tasks.

1. Introduction

The ability to visually track an object during arbitrary motion is an important part of interacting with the environment. Humans adeptly recover three-dimensional structure of an object from its rigid-body motion in order to accomplish manipulation, locomotion, and object recognition [6]. Motion and edge information are known to be important cues to the recovery of object structure. Understanding the relative motion between an object and an observer aids not only in the recovery of object structure but also provides useful information required to interact with the objects. In order to visually track an object, the object's orientation and position must be rapidly updated. In six degree-of-freedom robotic applications, real-time camera images can provide a dense stream of data from which to extract object features and recover rigid body motion.

An object's three-dimensional structure can be reconstructed from the projection of its motion onto a sequence of two-dimensional images [22]. Two methods for measuring visual motion are detection of spatio-temporal intensity changes and feature tracking [18]. The measurement of spatio-temporal intensity changes can be accomplished using correlation models, energy filters [1] [4], or gradient techniques [2] [13].These algorithms provide a description of two-dimensional image motion as a result of changes in intensity in the image. They cannot differentiate between intensity differences due to changes in viewer motion and intensity changes produced by object motion with respect to a light source. Thus, they are not suited to the task of tracking a moving object. The class of algorithms that measure visual motion by tracking features provides a means to directly measure physical motion in the world. These algorithms, however, have the disadvantage of requiring feature correspondence between images.

By combining the feature tracking approach with model-driven techniques, the feature tracking process is constrained and the correlation of features between frames is simplified. This results in a computationally inexpensive and accurate system. This paper describes an approach designed to achieve these goals and the implementation of this approach in the Intelligent Controls Group (ICG) laboratory at the National Institute of Standards and Technology. The next section discusses model-based feature tracking methods and, in particular, the two-dimensional tracking method we use. Section 3 details the implementation of the algorithm in our lab. The fourth section quantifies the accuracy and speed of this algorithm in preliminary experiments by tracking a planar target, and the final section discusses the implications of these results.

2. Model-based Feature Tracking

Model-based feature tracking correlates image features with object model features to take advantage of model information. Correlation between extracted features and an object model can be performed in either a two-dimensional [9] or three-dimensional [12] [19] frame-of-reference. A useful survey of work done using these methods can be found in [20]. Three-dimensional tracking involves comparing a set of two dimensional features extracted from an image to a three-dimensional model. In the most general case, this means solving the three-dimensional recognition problem for each successive image frame. The three-dimensional position and orientation of the feature are computed by analyzing images taken at different positions. Correspondence of features between image frames is determined, and the three-dimensional feature position is computed and matched to the model. This process is time consuming computationally expensive.

A more efficient way of using a three-dimensional frame of reference for model matching involves computing only the changes in structure position and orientation between image

frames [20]. The set of extracted two-dimensional features that must be matched with a three-dimensional model is thus reduced. By projecting the model into image frame coordinates, the search space is further reduced since the approximate location of each model feature in the two-dimensional image frame is known. The information extracted and used to update the model is usually quite accurate, and since all surfaces of the object model are available, problems of changing viewpoints or occlusion are handled. However, the computational complexity of these algorithms prevent their use in real-time applications.

The model-based feature tracking approach where the matching occurs in two-dimensions is a less complex alternative to the tracking problem. It assumes the ability to process a continuous sequence of two-dimensional images in real-time [9]. Zero or one-dimensional features such as vertices, centroids, or edge segments are extracted and matched to a two-dimensional projection of the model. Analogous to the three-dimensional case, the process is simplified by computing motion features which represent the changes in position and orientation between image frames. Since the temporal sampling rate is high, there is little change in position and orientation between successive frames and the correlation between observation and model is simplified. The two-dimensional model is continually updated based upon the most recent observation. Tracking in two dimensions continues exclusive of additional information as long as the object motion is continuous. A model update obtained from three-dimensional information is required if there is occlusion or a change of direction causing loss of two-dimensional information. In general, two-dimensional feature tracking is an inexpensive method of correlating observations with model information and is well-suited for real-time applications [20].

The approach in our lab is based on model-based feature tracking in two dimensions. There are two phases to the method that are used: initialization and tracking. During initialization, an operator defines the position of the object features in the image at the point where tracking is to begin. In our preliminary experiment, a planar object is attached to a pendulum that is constrained to two-dimensional motion and tracked while it is swinging. A sequence of camera images that demonstrates this motion is digitized and stored. This sequence is used as input to our algorithm and provides us with repeatable motion for our experiments. Since the path is repeatable, the operator can select the feature points, which are the object corners, from any one of the images in the sequence. Tracking begins when all features selected by the operator are within an acceptable distance from the extracted image corners. Successful tracking continues while the extracted image features remain within a predefined distance from the predicted corners. Tracking is lost when this distance is exceeded.

Figure 1 depicts an overview of the algorithm during the tracking phase. In this figure, $I(x,y)_t$ refers to the intensity function at a pixel located at position (x,y) at time t. Motion and edge features from a sequence of images are correlated with model information. Model-based tracking involves segmentation and correlation of observed data with model data and the prediction of the model position at the next time interval.

During the segmentation phase, optical flow and edge orientation are extracted from an incoming sequence of images. This information represents the temporal and spatial edge information respectively. The image flow results from changes in intensity between frames and a temporal differencing algorithm is used to measure these changes. Incoming images are smoothed using a Gaussian convolution, G*, to diminish the effects of spurious noise in the image. Two temporally-consecutive, smoothed images are subtracted from each other in order to detect any change in intensity due to motion between the frames (Equation 1).

$$\frac{\partial}{\partial t} I(x,y) = G^*(x,y)_t - G^*(x,y)_{t+1} \qquad [1]$$

All non-moving features in the image disappear in this difference image since the grayscale value of a pixel in the second frame is being subtracted from the identical grayscale value in the first frame. The resulting optical flow image is thresholded to produce a binary image. This operation results in a segmented scene reflecting changing intensity values between successive images. However, motion segmentation occurs whether the changing intensity is due to relative motion between the camera and the object or between the object and a light source and can therefore be an ambiguous basis for segmentation.

By requiring object features in an image to adhere to consistent spatial as well as temporal properties, the segmentation of features is more robust. Edges are also extracted during the segmentation process to provide additional information. Spatial orientations of edge points are computed directly from the sequence of input images. This information is used to determine the spatial orientations of the image motion points. Since the positions of the edge points on the object change between frames, a dense set of edge points is extracted from the sum of two temporally-consecutive, smoothed images. A two-dimensional spatial gradient operator is applied to all points (x,y) in the image. The actual direction, θ, of each point in the image is defined to be perpendicular to the direction of the gradient of the intensity function f(x,y) at that point:

$$\theta = \operatorname{atan}\frac{(\nabla_y f)(x,y)}{(\nabla_x f)(x,y)} + \frac{\pi}{2} \qquad [2]$$

Since the edge extraction and the motion extraction operations are performed in parallel on the same input images, this provides the advantage of having edge orientation information for most motion pixels.

The next step, correlation of the extracted motion points with the model edges, makes use of the segmentation information. Each motion pixel is either labelled or discarded depending on its similarity to the model. The labelling process is based on two criteria. The first criterion is the two-dimensional spatial proximity of a motion point to the model lines.The second criterion is the similarity of direction of the edge at the location of this motion with the angular direction of the model line.

The description of each line in the model includes the slope-intercept form of the line, the coordinates of the endpoints of each line segment, and the angular direction of the line. The first step in the correlation process compares the angular direction of the model line with the edge direction of the candidate motion point to determine if the angular disparity is within an acceptable range:

$$|\theta_{model} - \theta_{data}| < \delta \qquad [3]$$

If this condition is satisfied, the distance d is computed between the point at image coordinate (x_i, y_i) and each of k lines used to define the object model using equation [4]:

The minimum distance between the image motion point and each of k model lines is used to determine if that point is less than a distance threshold, ζ, from the line. The point is labelled as belonging to the model line segment it best matches when both the spatial proximity and orientation conditions are satisfied.

$$d = \frac{A_k x_i + B_k y_i + C_k}{(A_k^2 + B_k^2)^{1/2}} \qquad [4]$$

where $(A_k x + B_k y + C_k)$ is the general form for the k^{th} line in the model.

After the motion points are segmented, a line-fitting technique is used to update the two-dimensional position of each model line. The line-fitting technique uses a least-squares linear regression which minimizes the squared error in either the x or y direction. The equations that compute slope, m, and the y intercept, b, of the best fitting line through the n points (x_i, y_i) when minimizing the x direction error are:

$$m = \frac{n(\sum_i (x_i y_i)) - (\sum_i x_i)(\sum_i y_i)}{n\sum_i (x_i^2) - (\sum_i x_i)^2} \qquad [5]$$

$$b = \frac{(\sum_i y_i)(\sum_i x_i^2) - (\sum_i x_i)\sum_i x_i y_i}{n(\sum_i x_i^2) - (\sum_i x_i)^2} \qquad [6]$$

Minimizing the least square error in the x direction presents a problem as the line being fit approaches vertical. As this happens, the denominator in equation [5] approaches zero, and the fit becomes less accurate. A more accurate fit takes this into account and minimizes errors in both x and y. Comparison of the standard deviation of the x coordinates to that of the y coordinates gives a measure as to whether the line is more horizontal or more vertical. A larger standard deviation in x means the line tends toward the horizontal. When the standard deviation of the y coordinates is larger, a linear regression of x on y is used since the line is more vertical. In this case, the equation for the slope and intercept of the fitted line is given by

$$m = \frac{n(\sum_i y_i^2) - \sum_i (y_i)^2}{n(\sum_i x_i y_i) - (\sum_i x_i)(\sum_i y_i)} \qquad [7]$$

$$b = \frac{-((\sum_i x_i)(\sum_i y_i^2) - (\sum_i y_i)(\sum_i x_i y_i))}{(n(\sum_i x_i y_i) - (\sum_i x_i)(\sum_i y_i))} \qquad [8]$$

If the standard deviation of x coordinates is less than a predefined threshold value, the line is considered to be vertical, and therefore the slope is undefined.

The computed lines are intersected to determine the two-dimensional corner positions of the object model. Each corner point (x_c, y_c) is computed by solving for the intersection of the two fitted lines which contain the corner as an endpoint. The distance between the computed corners and the model corners is used to determine the proximity of the results with the prediction. If the resulting distance is within an acceptable limit, ε, then a successful match has been detected (Equation [9]):

$$\sqrt{(x_c - x_{model\,k})^2 + (y_c - y_{model\,k})^2} < \varepsilon \qquad [9]$$

When the distance exceeds ε, tracking is lost; distances less than ε indicate tracking. When tracking is successful, the corners are filtered using an exponential smoothing filter [17] to predict the corner positions at the next time interval. Each corner is filtered independently of the others since the object motion isn't necessarily parallel to the image plane and the corners will move at different rates in the image plane. The filtering of each corner is done using a weighted average of the current and all past positions of that corner with exponentially decreasing weights (Equation 10).

$$(x, y)_t' = \alpha (x, y)_t + (1 - \alpha)(x, y)_t' \qquad [10]$$

In this equation, $(x, y)_t'$ is the filtered corner position and $(x, y)_t$ is the unfiltered corner position. The value of $(x, y)_t'$ before any previous iterations is set to the position of the corner when tracking is initially begun. The smoothing constant, α, is in the range $0 < \alpha \leq 1$ and, in our case, is chosen to be 0.2. The corner is filtered again using equation 11.

$$(x, y)_t'' = \alpha (x, y)_t' + (1 - \alpha)(x, y)_t'' \qquad [11]$$

The value of $(x, y)_t''$ before any previous iterations is also set to the position of the corner when tracking is initially begun. After the filtered values are determined, the predicted corner position $(x,y)_{t+1}$ is computed using equation [12].

$$(x, y)_{t+1} = (2 + \frac{\alpha}{1 - \alpha})(x, y)_t'$$

$$(-(1+\frac{\alpha}{1-\alpha})\,(x,y)_t^{\cdot}) \quad [12]$$

The predicted model information is used to segment the extracted image flow information at time t+1. Predictions are computed each execution cycle regardless of whether updated observed corners are available. This allows predicted positions to be sent to the correlation process continually, though the spacing between successive positions decreases while no new data are being observed. Figure 2 shows the relative frequency with which predicted data are generated as compared to data extracted from incoming images over a period of about 1 second. Since the results of the tracking algorithm are used as feedback in a real-time system, the implementation of this algorithm must be fast enough to be stable. Section 3 describes the approach we use.

3. Implementation

Processing in the integrated vision testbed in the ICG laboratory is accomplished using a programmable real-time image processor, the Pipelined Image Processing Engine (PIPE)[1] [5] and a multiprocessor system as shown in Figure 3. In this figure, the large grey rectangles represent how the software processes are distributed on this hardware. The incoming images from a CCD camera are digitized by PIPE to provide 8-bit grayscale images that are 242x256 pixels in size. The images are processed by lookup tables, neighborhood operators, and arithmetic logic units that are defined in the PIPE application program. Smoothing, temporal integration, and edge and motion detection are performed on the grayscale images as described in equations [1] and [2]. The Iconic-to-Symbolic Mapper (ISMAP) stage of PIPE converts information from an image format to a symbolic list and is used to store the binary motion image as a list of pixel positions. In addition, the corresponding edge direction values are stored in the ISMAP iconic buffer where they are mapped onto the memory of one of the microprocessors via a specialized PIPE-VME interface board. Figure 3 displays these pipelined processes as black parallelograms.

The remaining software processes operate in real-time in the multiprocessing environment. They are implemented within the hierarchical sensory-interactive robot control system in our lab [7] [10] [15] [16] [21] that is defined in the NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM) [3]. The control system is composed of three parallel systems that cooperate to perform telerobot control (Figure 4). The task decomposition system breaks down objectives into simpler subtasks to control physical devices. The world model supplies information and analyzes data using support modules. It also maintains an internal model of the state of the environment in the global data system. The sensory processing system monitors and analyzes sensory information from multiple sources in order to recognize objects, detect events and filter and integrate information. The world model uses this information to maintain the system's best estimate of the past, current, and possible future states of the world. The processes are labeled according to their functional role in the NASREM architecture as sensory processing (SP), world modeling (WM), or task decomposition (TD) modules. Each device or sensor of the telerobot has a support process in each of the three columns of the control system. For example, the task decomposition functions associated with planning the actions for processing camera data reside in the task decomposition hierarchy; the world modeling functions for supporting those plans reside in the world model hierarchy, and the image processing techniques required for executing those plans reside in the sensory processing hierarchy. The world modeling support modules communicate asynchronously with the task decomposition and sensory processing systems. Data flows bidirectionally between adjacent levels within any given hierarchy. The interfaces to the sensory processing system allow it to operate in a combination of bottom-up (data driven) and a top-down (model driven) modes. Bottom-up processing involves the extraction of knowledge from sensory data, and top-down processing is used to correlate predicted information from the world model with extracted information from the environment. The interfaces between the sensory processing system and the world model allow updated information to be sent to the world model and predicted information or sensory processing parameters to be sent to the sensory processing system.

The implementation is based on the concept of cyclically executing modules which serve as the computational units for the NASREM architecture [11]. After initialization, all computations are performed by cyclically executing processes that communicate via global read-write interfaces. Each unit acts as a process which reads inputs, performs computations, and then writes output. Such a process always reads and executes on the most current data; it does not wait for new data to arrive since reliable cyclic execution requires that a module be able to read or write data with minimal delay. Reading and writing involve the transfer of data between local buffers and buffers in global memory. System software has been written to prevent data corruption during these transfers.

Three cyclically-executing software processes execute the model-based feature tracking algorithm. These reside on two of the three microprocessor boards. The remaining software process performs communication with PIPE. The PIPE communication process is a cyclically executing process which polls PIPE status, reads the ISMAP output produced by PIPE, and writes it to the appropriate common memory locations that it shares with the segmentation process. Though the amount of data transferred is large, on the order of hundreds to thousands of pixels every cycle of 60 Hz, the direct memory accessing provides a high rate of accessibility to the symbolic data. The execution time for this process takes an average of 90 ms on a 68020 processor for about 1400 motion pixels, an average sample for our tests described in the following sections. The correlation steps described in equations [3] - [4] and the line-fitting described in equations [5] - [8] are computationally intensive. The processing times for these operations depend on the number of motion points. This process requires

1. Commercial equipment and materials are identified in this paper in order to adequately specify the experimental procedure. Such identification does not imply recommendation or endorsement by NIST, nor does it imply that the materials or equipment identified are necessarily the best for the purpose.

high bandwidth, and for this reason, the correlation and line-fitting process executes on a dedicated 68030 microprocessor at an average rate of 110 ms per execution cycle. Since the execution time is greater than that of the PIPE communications process, it always has new data at the beginning of its execution cycle. The resulting lines are written to a common memory buffer shared with the process which computes the corners of the observed object. This process computes all of the object corners in 2.1 ms and then updates the buffer shared with the filtering and prediction process. The filtering and prediction in equations [10] - [12] are used to obtain all of the corners in the predicted model, and this process executes in 3.1 ms. These processes are combined on one board, a 68020 processor since their total execution time is small compared to the other processes.

4. Results

A preliminary set of experiments to determine how accurately this model-based feature tracking algorithm can track an object was run for the case of simple translational velocity. In these experiments, a planar, rectangular object was mounted on a pendulum as shown in Figure 5. The pendulum is released at different heights to provide image motion at differing velocities. The only a priori knowledge about the object is the image positions of the four corners at an arbitrary point during the path of the pendulum that are used as a starting point for tracking as described in section 2. These points establish a crude model and are necessary to establish when the object has initially been matched. Once the observed data matches the initial model, the object is tracked by the single-camera vision system in the manner previously described.

In order to quantify the accuracy with which the model predicts the position of the observed data, the distance between the model corners and the actual corners is computed. This difference is used to determine the accuracy of the fit between the predicted and the observed data at varying object velocities. The accuracy of the tracking algorithm is also affected by the threshold parameters used. The threshold used in equation [4] controls the labelling of motion pixels and determines how closely the data can be correlated to the model. The threshold used to match the model corners to the observed corners in equation [9] determines how closely the model must match the observed data before tracking is lost. Our experiments consisted of three cases of varying object velocity. Velocity is measured in the image plane as the distance that an object feature moves between camera frames. The velocities tested are 1.1 pixels per frame, 3.9 pixels per frame, and 4.5 pixels per frame. For each velocity, the correlation threshold is tested at four values, 3 pixels, 5 pixels, 10 pixels and 15 pixels. In addition, the corner matching threshold is tested at 32 pixels, 26 pixels and 20 pixels. In all, twelve sets of data were collected for each velocity. Each set of data consists of 200 error measurements.

The threshold parameters were varied for three different object velocities measured in the image plane. Table 1 summarizes the results of experiments for the three velocities at varying correlation thresholds. The corner threshold remains constant at 20 pixels. Tables 2 and 3 are similarly organized except that the corner thresholds are set to 26 pixels and 32 pixels, respectively. Continuous tracking was performed successfully for all cases over a period of 200 iterations. By comparing the tables, it can be seen that the threshold used to determine successful tracking described in equation [9] does not play a significant role. Table 1 shows that for each object velocity, the mean error increases as the correlation threshold increases. This is a result of the fact that as the distance threshold is relaxed, there is a greater chance of misclassifying motion pixels. This effect is noticeable at faster velocities since there are more motion pixels available for processing. Also it can be seen that at distance thresholds of 3 and 5 pixels, the tracking error decreases with increasing velocity. This can be attributed to the value chosen for the smoothing constant α described in equations [10] - [12] which provides predictions more closely matching the observed data at a velocity of 4.5 pixels per 60 Hz. It is not clear that this trend would continue for higher velocities using the same smoothing constant. At distance thresholds of 10 and 15 pixels the tracking error increases as the velocity increases. This is caused by a greater number of motion pixels being present at higher velocities compounding the effect of the relaxed threshold. Similar conclusions can be drawn by analyzing Tables 2 and 3.

5. Conclusions

The method described in this paper successfully tracks moving image features by correlating a combination of extracted spatial and temporal edge information with an object model. The use of spatial information in the form of edge point orientations constrains the correlation process since motion points whose orientation is outside the orientation tolerance are discarded. This provides the advantage of being able to track an object in an unconstrained environment. Since a feature point has to be moving and in the correct orientation and position to be matched to the object model, other features can be in the field of view without being considered as part of the object. Another advantage of this algorithm is that the predicted model information can vary from the extracted image features up to 15 pixels and still track successfully. This is an improvement over algorithms that base correlation only on local properties. Preliminary results demonstrate successful tracking for image feature velocities between 1.1 and 4.5 pixels between image frames.

In the future, we plan to expand the modelling capability of our system to handle the appearance and disappearance of object features. By using a three-dimensional model we will be able to predict the most stable set of features to track for a given object pose. Modelling the motion of the object will enable us to provide pose predictions in the absence of sensory data. We also plan to continue the experiments on model-based feature tracking and to extend the scope of our algorithms to include processing on a stereo set of cameras [8]. Knowledge of the two-dimensional position of the same feature as viewed from two cameras will enable us to determine the position and orientation of the object in world-space using range from triangulation. This capability will allow us to supply feedback information to a manipulator system to aid in tasks involving tracking or grasping a moving part.

6. References

[1] Adelson, E. H., J. R. Bergen, "Spatio-temporal Energy Models for the Perception of Motion," *Journal of the Optical society of America A*, Vol. 2, No. 2, February, 1985, pp. 284-299.

[2] Albus, James S., Tsai-Hong Hong, "Motion, Depth, and Image Flow," *IEEE Robotics and Automation Conference*, Cincinnati, OH, May 13-18, 1990.

[3] Albus, J. S., H. G. McCain, R. Lumia.,"NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM)," NIST Technical Note 1235, Gaithersburg, MD, July, 1987.

[4] Allen, Peter K., "Real-time Motion Tracking Using Spatio-Temporal Filters," *Proceedings of the DARPA Image Understanding Workshop*, Palo Alto, TX, May 23-26, 1989.

[5] Aspex, Inc., "PIPE--An Introduction to the PIPE System," New York, 1987.

[6] Bolles, Robert C., H. Harlyn Baker, David H. Marimont, "Epipolar-Plane Image Analysis: An Approach to Determining Structure from Motion," *International Journal of Computer Vision*, Vol. 1, 1987, pp. 7-55.

[7] Chaconas, K., M. Nashman, "Visual Perception Processing in a Hierarchical Control System", NIST Technical Note 1260, Gaithersburg, MD, March, 1989.

[8] Chaconas, K., "Range from Triangulation Using An Inverse Perspective Method to Determine Relative Camera Pose," NIST Internal Report 4385, Gaithersburg, MD, August, 1990.

[9] Crowley, James L., Patrick Stelmaszyk, Christopher Discours, "Measuring Image Flow by Tracking Edge-Lines," *Proceedings of the 2nd International Conference on Computer Vision*, 1988, pp. 658-664.

[10] Fiala, J., "Manipulator Servo Level Task Decomposition," NIST Technical Note 1255, NIST, Gaithersburg, MD, October, 1988.

[11] Fiala, J. "Note on NASREM Implementation," NIST Internal Report 89-4215, Gaithersburg, MD, December, 1989.

[12] Gennery, Donald B., "Tracking Known Three-Dimensional Objects," *Proceedings of the National Conference on Artificial Intelligence*, Pittsburgh, PA, August 18-20, 1982, pp. 13-17.

[13] Horn, B. K. P., B. Schunk, "Determining Optical Flow," *Artificial Intelligence*, Vol. 17, 1983, pp. 185-203.

[14] Jenkin, M., J. K. Tsotsos, "Applying Temporal Constraints to the Dynamic Stereo Problem," *CVGIP*, 33, 1986, pp. 16-32.

[15] Kelmar, L. "Manipulator Servo Level World Modeling," NIST Technical Note 1258, NIST, Gaithersburg, MD, March, 1989.

[16] Lumia, R., Fiala, J., Wavering, A., "The NASREM Robot Control System and Testbed," 2nd Intl. Symp. on Robotics & Automated Manufacturing, Albuquerque, NM, November, 1988.

[17] Montgomery, D. C., L. A. Johnson, and J. S. Gardiner, "Forecasting & Time Series Analysis," Second Edition, McGraw-Hill, New York, 1990.

[18] Spetsakis, Minas E., John Aloimonos, "Closed Form Solution to the Structure from Motion Problem from Line Correspondences," *Proceedings of the National Conference on Artificial Intelligence*, 1987, pp 738-743.

[19] Thompson, D. W., J. L. Mundy, "Model-based Motion Analysis - Motion from Motion," *Robotics Research: The Fourth International Symposium*, R. C. Bolles and B. Roth, eds., The MIT Press, Cambridge, MA, 1988, pp. 229 - 235.

[20] Verghese, Gilbert, Charles R. Dyer, "Real-time Model-Based Tracking of Three-Dimensional Objects," Computer Sciences Technical Report #806, University of Wisconsin - Madison, November, 1988.

[21] Wavering, A., "Manipulator Primitive Level Task Decomposition," NIST Technical Note 1256, NIST, Gaithersburg, MD, October, 1988.

[22] Waxman, Allen M., Kwangyoen Wohn, "Image Flow Theory: A Framework for 3-D Inference from Time-Varying Imagery," LSR-TR-1, Boston University, January, 1986.

[23] Waxman, Allen M., Jian Wu, F. Bergholm, "Convected Activation Profiles: Receptive Fields for Real-Time Measurement of Short-Range Visual Motion," *International Conference on Computer Vision*, April, 1988.

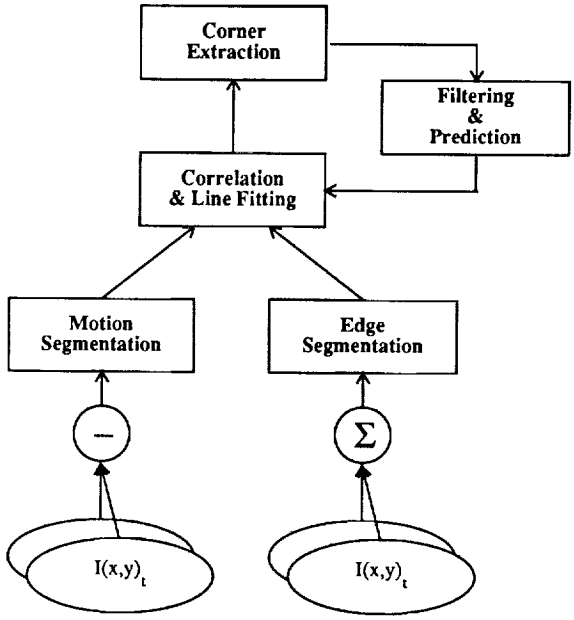

Figure 1. Model-based Feature Tracking Algorithm

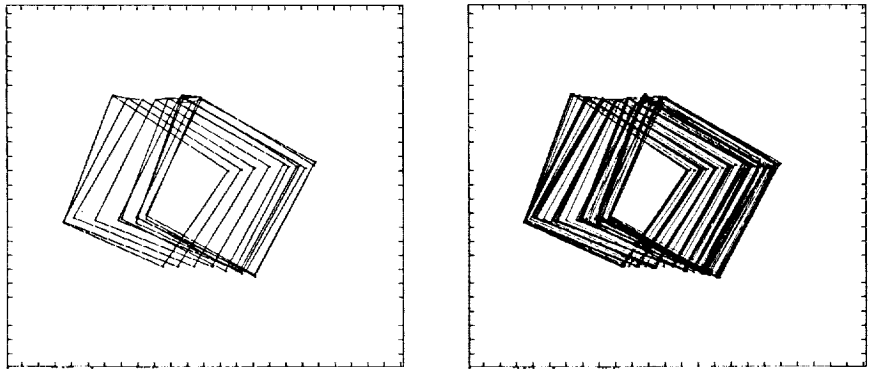

Figure 2. (a) Observed Object Positions (b) Predicted Object Positions

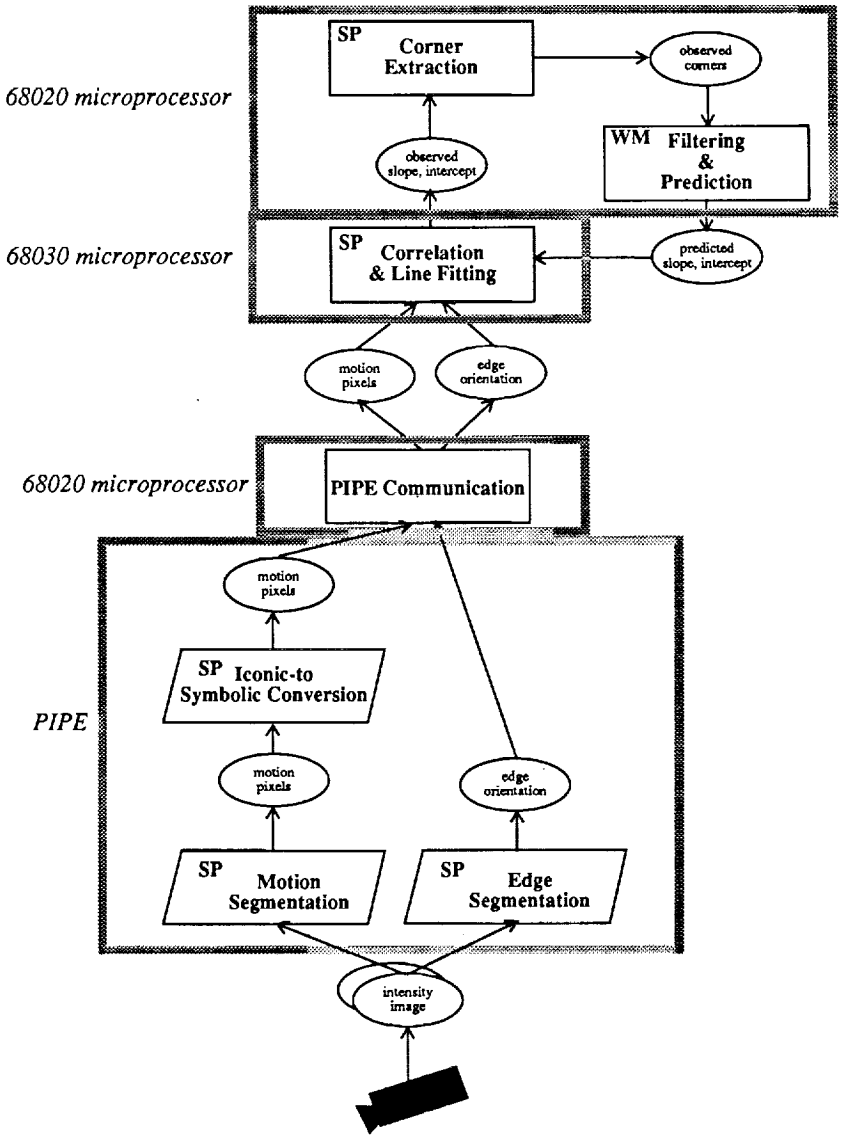

Figure 3. Implementation of Model-Based Feature Tracking Algorithm

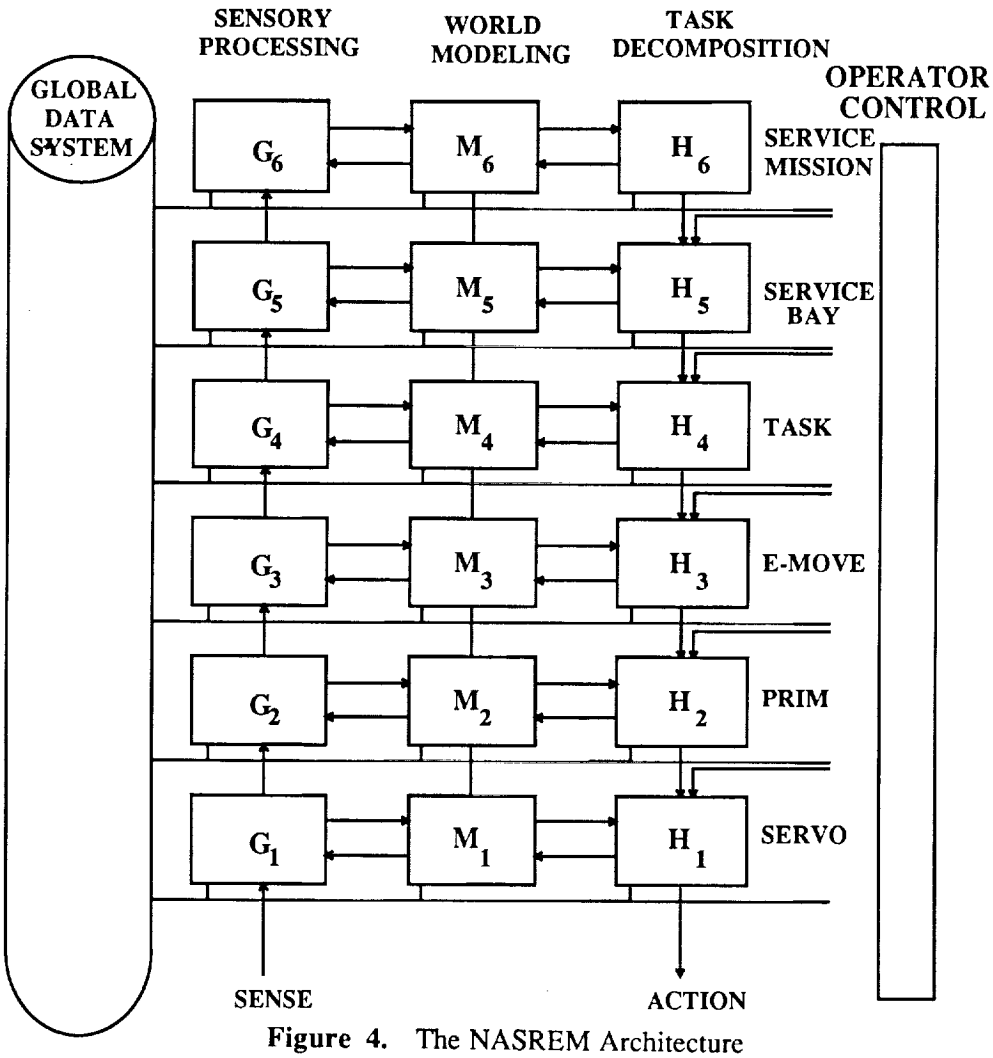

Figure 4. The NASREM Architecture

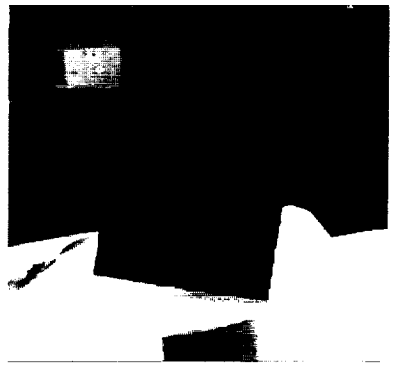

Figure 5. Experimental Scenario

	Distance Threshold for Correlation			
Velocity	**$\zeta = 3.0$**	**$\zeta = 5.0$**	**$\zeta = 10.0$**	**$\zeta = 15.0$**
1.1	0.302	0.338	0.335	0.407
3.9	0.045	0.096	0.096	1.151
4.5	0.009	0.024	1.174	1.368

Table 1. Mean Data Error Using Corner Threshold $\varepsilon = 20$

	Distance Threshold for Correlation			
Velocity	**$\zeta = 3.0$**	**$\zeta = 5.0$**	**$\zeta = 10.0$**	**$\zeta = 15.0$**
1.1	0.301	0.313	0.357	0.407
3.9	0.110	0.108	0.052	0.052
4.5	0.009	0.005	1.177	1.368

Table 2. Mean Data Error Using Corner Threshold $\varepsilon = 26$

	Distance Threshold for Correlation			
Velocity	**$\zeta = 3.0$**	**$\zeta = 5.0$**	**$\zeta = 10.0$**	**$\zeta = 15.0$**
1.1	0.302	0.313	0.348	0.407
3.9	0.031	0.109	0.097	1.160
4.5	0.001	0.003	1.174	1.273

Table 3. Mean Data Error Using Corner Threshold $\varepsilon = 32$