N-04

146077

P-29

NASA Technical Memorandum 103999

A Parallel Implementation of a Multisensor Feature-Based Range-Estimation Method

Raymond E. Suorsa and Banavar Sridhar

(NASA-TM-103999) A PARALLEL IMPLEMENTATION OF A MULTISENSOR FEATURE-BASED RANGE-ESTIMATION METHOD (NASA) 29 p

N93-18883

Unclas

G3/04 0146077

January 1993

National Aeronautics and Space Administration

NASA Technical Memorandum 103999

A Parallel Implementation of a Multisensor Feature-Based Range-Estimation Method

Raymond E. Suorsa and Banavar Sridhar, Ames Research Center, Moffett Field, California

January 1993

National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, California 94035-1000

1 Summary

There are many proposed vision based methods to perform obstacle detection and avoidance for autonomous or semi-autonomous vehicles. All methods, however, will require very high processing rates to achieve real time performance. A system capable of supporting autonomous helicopter navigation will need to extract obstacle information from imagery at rates varying from ten frames per second to thirty or more frames per second depending on the vehicle speed. Such a system will need to sustain billions of operations per second. To reach such high processing rates using current technology, a parallel implementation of the obstacle detection/ranging method is required. This paper describes an efficient and flexible parallel implementation of a multisensor feature-based range-estimation algorithm, targeted for helicopter flight, realized on both a distributed-memory and shared-memory parallel computer.

2 Introduction

The design of intelligent low-altitude guidance systems for helicopters requires information about objects in the vicinity of the flightpath of the vehicle. The sensor system on the helicopter must be able to detect objects such as buildings, trees, poles and wires during flight. A complete obstacle-detection system may consist of an active ranging sensor and passive ranging using electro-optical sensors. A comprehensive overview of this problem can be found in references [1, 2, 3].

Several techniques have been proposed for range determination using electro-optical sensors [4, 5, 6]. These techniques use optical flow resulting from the relative motion between the sensor and objects on the ground together with the helicopter state from an inertial navigation system to compute range to various objects in a scene. One algorithm of interest can detect, track, and estimate range to image features (i.e., patches of an image with common statistics or spatial structure) over time from a multisensor system mounted on a vehicle moving with arbitrary six degrees of freedom [7].

The estimation of range using electro-optical sensors involves large volumes of data, for example, 15 MB/sec for 8-bit grey scale stereo images at 30 frames per second, and requires processing power in the range of a few billion operations per second. Today, there is no single off-the-shelf microprocessor or digital signal processor which can meet this demand. The large amount of computation required to solve problems in computer vision is well-recognized, and parallel processing presents an approach to achieve the speed necessary for real-time implementation [8]. However, parallel processing does not provide a linear increase in speed and the actual increase depends on the computer architecture and the application [9]. The selection of a parallel processing architecture for the range-estimation problem has to examine the trade-offs between several architectures in terms of their effect on overall speed increase, processor utilization, programmability, and physical constraints. A promising system must be adaptable to changes in the vision algorithm, exhibit good scalability, and must be installable, at some point, on board a helicopter. The constraints of high speed, algorithm flexibility, and system scalability favor a general-purpose parallel RISC-based sys-

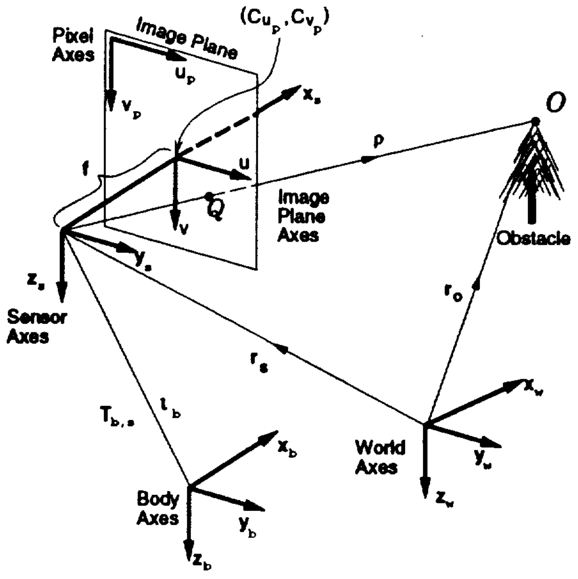


Figure 1: Obstacle geometry.

tem over traditional pipelined image processors. This does not prevent a viable system from using a traditional image processor as a "front end" to the parallel computer. Simply due to the algorithm's complexity, a traditional "image processing" approach will be lacking in flexibility [10].

This paper presents a multiple-sensor range-estimation algorithm along with a discussion of an efficient and flexible method of parallelization which is necessary to realize real-time operation on a distributed-memory or shared-memory parallel computer. The paper is organized as follows: Section 2 gives a quick overview of the mathematical background of optical flow. Section 3 describes the extended-Kalman-filter-based range-estimation algorithm, extension of the procedure to multiple sensors, initialization procedure and an introduction to the multirate Kalman filter [11]. Section 4 discusses the feature tracking algorithm and section 5 describes virtual processing regions, a software abstraction used for parallelization. Section 6 describes three load balancing schemes based on virtual processing regions. Section 7 presents some initial results using a distributed-memory parallel computer composed of a network of workstations and a modern shared-memory multiprocessor. Section 8 completes the paper with some concluding remarks and a discussion of future work.

The authors would like to thank Silicon Graphics Inc. for access to a IRIS 4D/480 for timing measurements.

3 Optical Flow

Consider a rotorcraft-mounted sensor rotated with respect to the body axis by the orthonormal rotation matrix $T_{b,s}$ and offset from the vehicle's body axis by l_b. The sensor, in motion with respect to an inertial Earth-fixed world axis system, observes an obstacle O whose location is fixed in the Earth frame as shown in Fig. 1. We wish to determine the relative position of the obstacle O with respect to the sensor

$$\rho = r_o - r_s \tag{1}$$

The imaging sensor maps the world object O whose location in sensor axes is $\rho_s = [\rho_{sx}, \rho_{sy}, \rho_{sz}]^T$ onto the image plane at Q by perspective projection according to the equation

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} f\rho_{sy}/\rho_{sx} \\ f\rho_{sz}/\rho_{sx} \end{bmatrix} \tag{2}$$

where f is the focal length of the sensor. As the sensor moves, ρ_s changes and so the image location corresponding to O changes as follows:

$$\begin{bmatrix} \dot{u} \\ \dot{v} \end{bmatrix} = \frac{1}{\rho_{sx}} \begin{bmatrix} f\dot{\rho}_{sy} - u\dot{\rho}_{sx} \\ f\dot{\rho}_{sz} - v\dot{\rho}_{sx} \end{bmatrix} \tag{3}$$

The sensor frame is moving so the derivative of ρ_s is determined using the Coriolis equation

$$\dot{\rho}_w = \dot{\rho}_s + \omega \times \rho \tag{4}$$

where $\dot{\rho}_w$ is the derivative of ρ in world axes and is equal to the negative of the sensor velocity in world axes, $\dot{\rho}_s$ is the derivative of ρ in sensor axes, and ω is the rotation of the sensor axes relative to the world axes. Let $V_s = [V_{sx}, V_{sy}, V_{sz}]^T$ and $\omega_s = [\omega_{sx}, \omega_{sy}, \omega_{sz}]^T$ be the linear and angular velocity of the sensor with respect to the world frame and resolved in the sensor axes. Then noting that O is fixed in world axes and using equations (1) and (4), we obtain the relation

$$\dot{\rho}_s = -V_s - \omega_s \times \rho_s \tag{5}$$

The motion of the image point corresponding to O can now be written in terms of the sensor motion using equations (3) and (5), giving the result

$$\begin{aligned}
\dot{u} &= \dot{u}_T + \dot{u}_R \\
\dot{v} &= \dot{v}_T + \dot{v}_R \\
\dot{u}_T &= \left(-fV_{sy} + uV_{sx}\right)/\rho_{sx} \\
\dot{v}_T &= \left(-fV_{sz} + vV_{sx}\right)/\rho_{sx} \\
\dot{u}_R &= \tfrac{uv}{f}\omega_{sy} - f\left(1 + \tfrac{u^2}{f^2}\right)\omega_{sz} + v\omega_{sx} \\
\dot{v}_R &= f\left(1 + \tfrac{v^2}{f^2}\right)\omega_{sy} - \tfrac{uv}{f}\omega_{sz} - u\omega_{sx}
\end{aligned} \tag{6}$$

The motion of the image point corresponding to O due to sensor motion is known as optical flow. Here the optical flow has been decomposed into components due to translational and rotational motion of the sensor, denoted by the subscripts T and R, respectively. V_s and ω_s can be derived from the rotorcraft's inertial navigation system. With knowledge of the sensor motion, the focal length, and the optical flow $[\dot{u}, \dot{v}]$ obtained from a feature tracking algorithm, the range, ρ_{sx}, of an object O at the image location $[u, v]$ can be determined from the optical flow equations (6). The full vector ρ_s can then be recovered with the perspective projection equation (2). Knowledge of the dynamics of a sensor in an Earth-fixed inertial frame is an essential element in this range-estimation algorithm.

4 Kalman Filter

There have been many published methods of extracting range from motion imagery [12, 13, 14, 15, 16]. A simple method would be to measure the optical flow between every consecutive frame, and using equations (2) and (6), as described earlier, extract the object location ρ_s. The trouble with such a method is the unreliability and low signal-to-noise ratio of a single measurement of $[\dot{u}, \dot{v}]$, due mainly to pixel quantization and inaccuracies of subpixel localization. To greatly improve the range estimate of an object, an extended Kalman filter (EKF) is used to recursively estimate ρ_s given multiple measurements of $[\dot{u}, \dot{v}]$.

Several Kalman filter implementations were studied by Sridhar and Phatak [4], who obtained the best results by selecting the state vector $X = \rho_s$ and the measurement vector $Z = [u, v]$. With these definitions, equation (5) becomes the state equation and the perspective projection equation (2) become the measurement equation. The state and measurement equations can be written as follows

$$\begin{aligned} \dot{X} &= -[\omega_s]X - V_s \\ Z &= h(X) = [f\rho_{sy}/\rho_{sx}, f\rho_{sz}/\rho_{sx}]^T \end{aligned} \tag{7}$$

where

$$[\omega_s] = \begin{bmatrix} 0 & -\omega_{sz} & \omega_{sy} \\ \omega_{sz} & 0 & -\omega_{sx} \\ -\omega_{sy} & \omega_{sx} & 0 \end{bmatrix} \tag{8}$$

The state equation is a time varying linear system that depends on the camera's translational and rotational velocities. The measurement equation is a nonlinear function of the state.

The continuous time state and measurement equations can be converted to their discrete time equivalents assuming that V_s and ω_s are constant during the sampling interval ΔT. The discrete time system equations are

$$X(k+1) = \Phi(k)X(k) + \Gamma(k)U(k) + \Gamma_d(k)\zeta_X(k) \tag{9}$$

$$Z(k) = h[X(k)] + \zeta_z(k) \tag{10}$$

where $\Phi(k)$ is the state transition matrix, $\Gamma(k)$ is the input distribution matrix, $U(k) = -V_s(k)$ is the control vector, $\Gamma_d(k)$ is the disturbance distribution matrix, and $\zeta_X(k)$ and $\zeta_z(k)$ model the process noise and measurement noise, respectively. Zero mean gaussian white noise is assumed such that $R(k) \equiv cov(\zeta_z(k))$ and $Q(k) \equiv cov(\zeta_X(k))$. The state transition matrix and the control distribution matrices derived by Sridhar and Phatak can be found in [4]. The measurement equation is linearized about the current estimate of X giving

$$Z(k) = H(k)X(k) \tag{11}$$

$$H(k) = \partial h(X)/\partial X = f \begin{bmatrix} -\rho_{sy}/\rho_{sx}^2 & 1/\rho_{sx} & 0 \\ -\rho_{sz}/\rho_{sx}^2 & 0 & 1/\rho_{sx} \end{bmatrix} \Bigg|_{\hat{X}(k)} \tag{12}$$

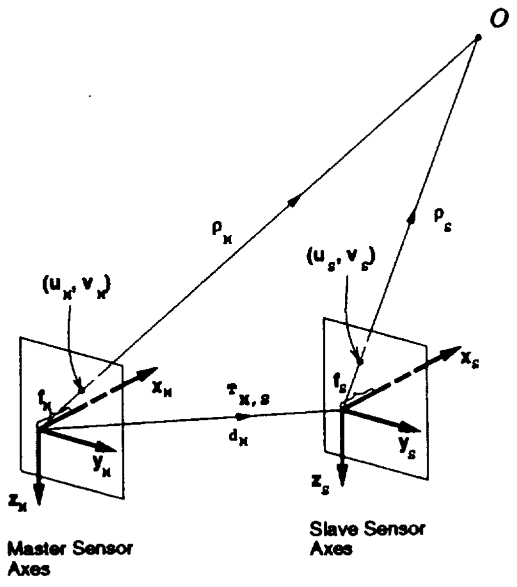


Figure 2: Two-sensor initialization.

The discrete time state equation (9) and the linearized measurement equation (11) can be used in a standard Kalman filter to recursively estimate the state vector X and the state covariance matrix P.

The Kalman filter consists of two parts: the measurement update which improves the state estimate given a new measurement, and the time update which propagates the state forward in time according the system dynamics. Before each iteration of the Kalman filter, we have estimates of $X(k)$, $P(k)$, $Q(k)$, and $R(k)$. The measurement update is then performed according to the following equations:

$$\begin{aligned} \tilde{X}(k) &= \hat{X}(k) + K(k)[Z(k) - h(\hat{X}(k))] \\ \tilde{P}(k) &= [I - K(k)H(k)]\hat{P}(k) \end{aligned} \tag{13}$$

where $H(k)$ is computed from $\hat{X}(k)$ as described above and the Kalman filter gain $K(k)$ is computed using the equation

$$K(k) = \hat{P}(k)H(k)^T[H(k)\hat{P}(k)H(k)^T + R(k)]^{-1} \tag{14}$$

The time update equations are

$$\begin{aligned} \hat{X}(k+1) &= \Phi(k)\tilde{X}(k) + \Gamma(k)U(k) \\ \hat{P}(k+1) &= \Phi(k)\tilde{P}(k)\Phi(k)^T + \Gamma_d(k)Q(k)\Gamma_d(k)^T \end{aligned} \tag{15}$$

4.1 Initialization

As noted above, the Kalman filter requires initial estimates for X and P. The initial estimate for X can be derived either of two ways. The first method is based on measurements of a feature's location within two images ΔT seconds apart from a single moving camera. The second method, applicable in multisensor configurations, uses measurements from two sensors displaced in space only. The initial estimate of the state covariance matrix P is chosen a priori.

The single sensor initialization method uses the optic flow equations, the perspective projection equations, and the camera's translational and rotational velocities which are assumed constant during the interval ΔT between images. First, the optic flow equations (6) are solved for x_s, where $[u, v]$ is the feature location in the sensor image plane. The optic flow equations actually comprise an overdetermined system of two equations in the one unknown x_s, so a single quadratic equation in x_s is formed by summing the squares of the two optic flow equations. Once x_s is found, y_s and z_s can be determined from the perspective projection equations (2).

A two-sensor stereo analysis is used to generate the initial estimate for a multisensor image sequence. In Fig. 2 two sensors are shown from an n-sensor configuration. One of the n sensors is designated as the *master* sensor and any of the others may be chosen as the *slave* sensor. An object O in the field of view (FOV) of the master and slave sensors will be imaged by both sensors. If occlusion effects are ignored then the image plane locations of O in the master and slave sensors are $[u_M, v_M]$ and $[u_S, v_S]$ defined by the perspective projection equation (2) and vectors ρ_M and ρ_S, respectively. The slave sensor is located at a position d with respect to the master sensor, and the transformation from the master sensor to the slave sensor is given by an orthonormal rotation matrix $T_{M,S}$. The master and slave sensors and the object O are related by the following equation:

$$\rho_S = T_{M,S}(\rho_M - d_M) \tag{16}$$

Equation (16) can be expressed term by term

$$\begin{bmatrix} \rho_{Sx} \\ \rho_{Sy} \\ \rho_{Sz} \end{bmatrix} = \begin{bmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & t_{33} \end{bmatrix} \begin{bmatrix} \rho_{Mx} - d_{Mx} \\ \rho_{My} - d_{My} \\ \rho_{Mz} - d_{Mz} \end{bmatrix} \tag{17}$$

If we assume $\rho_{Mx} \sim \rho_{Sx} = \rho_x$, (i.e., O is approximately the same distance from the center of each sensor's axis system) then equation (17) gives rise to two equations which may be solved for ρ_x. Equation (17) suggests that the master and slave sensors may be arbitrarily placed. It is numerically desirable though to place the master and slave sensors such that the roll and pitch between the sensors are minimized and d_M has a major component along either y_M or z_M. This is usually the case in standard two-sensor stereo setups. Equation (17) simply gives the full relations when two sensors do not have their scanlines registered. In our setup (as in most setups), the displacement d_M between the master and slave sensors will be dominated by the d_{My} component; therefore, we have chosen to solve equation (17) for ρ_x using the equation for ρ_{Sy}

$$\rho_x = \frac{f_M(t_{21}d_{Mx} + t_{22}d_{My} + t_{23}d_{Mz})}{t_{21}f_M + t_{22}u_M + t_{23}v_M - \frac{f_M}{f_S}u_S} \tag{18}$$

Equations (18) and (2) can then be solved for ρ_M. This will be used as the initial state for the EKF in the master sensor's axis system.

4.2 Multirate EKF

An image feature belonging to a far-away object or a feature near the FOE may have an interimage motion smaller than can be resolved by the measurement process [7, 17]. The effective signal-to-noise ratio of shift measurements can be increased by lengthening the time interval between images. The increased time interval can be effected by pausing one or more frames before a new measurement is made. This method essentially increases the motion baseline for the optical flow measurements. Each feature will have an optimal measurement delay m such that its measured shift between frame $k-m$ and frame k is greater than some constant value

$$dr < \sqrt{(u(k)-u(k-m))^2+(v(k)-v(k-m))^2} \tag{19}$$

where $[u(k), v(k)]$ is the measurement at time k. The choice of dr is based on the a priori estimate of the measurement noise ζ_z. The measurement noise is affected by the pixel quantization noise and the accuracy of the subpixel interpolation scheme used in subpixel correlation measurements, which is discussed in a following section. It should be noted that equation (19) is meaningful only if the sensor locations from which the two measurements have come have not rotated with respect to each other for at least $m\Delta T$ seconds. The reason for this, based on the optical flow equations (6), is that the rotational components of the flow $[\dot{u}_R, \dot{v}_R]$ do not contain any information of an object's range. It would be unwise to use this method, based on equation (19), to increase the signal-to-noise ratio of distant features without first removing the feature's rotational component from the shift measurement. Since the helicopter will always be rotating somewhat over time, one of the candidate measurements needs to be rotated into the sensor frame of the other measurement before comparing them using equation (19). To up-rotate a measurement from the coordinate system at time $k-m$ to the one at time k, the two equations for $[\dot{u}_R, \dot{v}_R]$ from (6) are used in their discrete form. These two equations can be written as a nonlinear vector function $g(\cdot)$ of the angular rates and image location $[u, v]$ of the feature at time i

$$g(i) \equiv \begin{bmatrix} \dot{u}_R(i) \\ \dot{v}_R(i) \end{bmatrix} \tag{20}$$

Equation (20) is used to calculate the image velocity induced by rotation of the sensor axis. In order to up-rotate a feature location $[u(i-1), v(i-1)]$ to time i, the rotational velocity $g(i-1)$ is multiplied by the sampling time ΔT and is added to the image plane location of the feature at time $i-1$. This will give the up-rotated image plane location at time i. This process is performed m times to find the location of $[u(k-m), v(k-m)]$ at time k. The following is the iterative equation to up-rotate features

$$\begin{bmatrix} u(i) \\ v(i) \end{bmatrix} = \begin{bmatrix} u(i-1) \\ v(i-1) \end{bmatrix} + g(i-1)\Delta T, \qquad i = k-m+1, \cdots, k-1 \tag{21}$$

The Kalman filter time updating is still performed at a constant rate which is equal to or faster than the smallest measurement delay possible. If a feature has an optimal measurement delay m, then every mth image a new measurement update is performed according to equations (13) and (14); otherwise, a trivial measurement update is performed according to

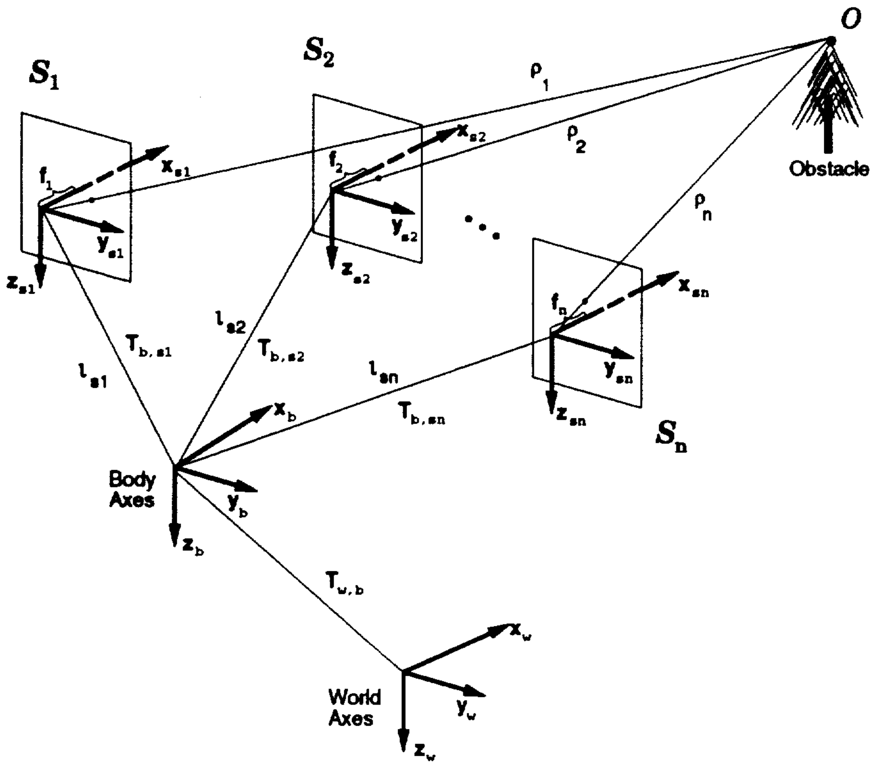


Figure 3: Multisensor geometry.

the following equations:

$$\begin{aligned} \tilde{X}(k) &= \hat{X}(k) \\ \tilde{P}(k) &= \hat{P}(k) \end{aligned}$$

It should be noted that as a distant object approaches the sensor array its interimage shift will increase. Therefore, the measurement delay m for a feature will decrease over time until $m = 1$, indicating that a measurement is made during each frame.

5 Feature Tracking Algorithm

The most difficult aspect of feature-based passive range estimation is the accurate measurement of the optical flow (interimage shift) for each feature. Results of earlier research indicate that at least two passive sensors should be used for range estimation to eliminate problems associated with subpixel motion near the FOE [18]. The combination of stereo and motion processing has been found to produce a more robust range map than motion or stereo alone. The geometry for multisensor feature tracking is illustrated in Fig. 3. In the figure an object O is imaged by n sensors $(S_1, \ldots, S_n)$. If a feature belonging to O is visually consistent among the sensors, then a measurement may be made of the feature's location in each sensor's image plane. Each measurement can be treated as an independent measurement of a feature's optic flow as seen by different sensors.

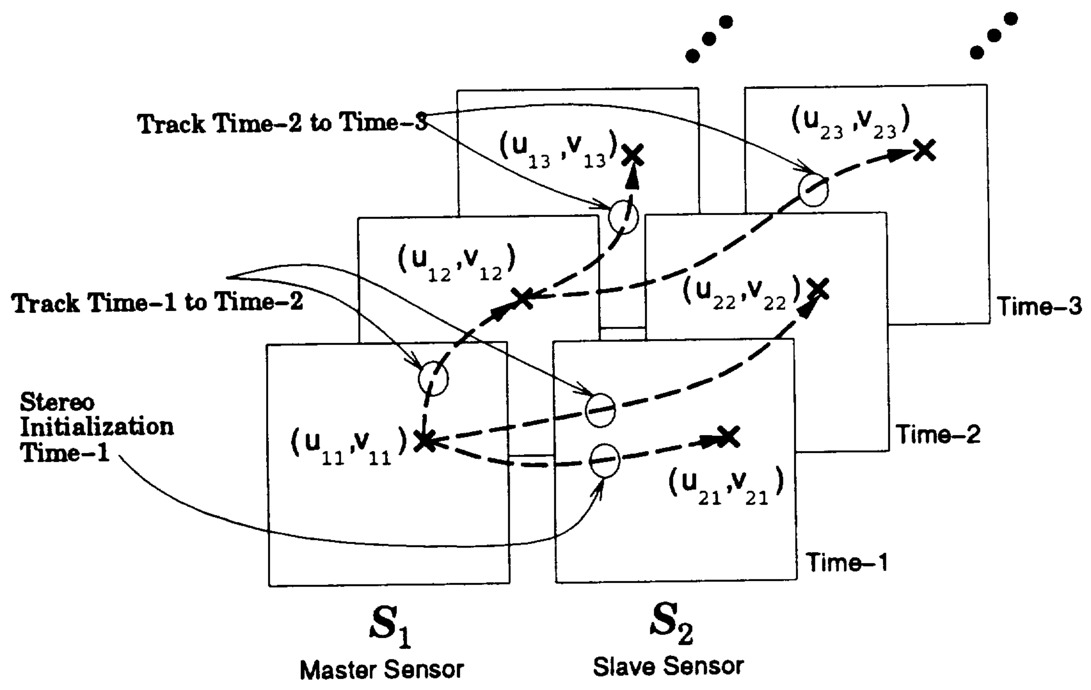


Figure 4: Multisensor feature tracking.

An extended Kalman filter is used to estimate the range for a feature as well as to fuse measurements from several sensors. Measurement fusion is effected by linearizing the measurement update equation of the filter for the appropriate sensor.[1] The Kalman filter time update is used to generate a prediction of a feature's location in one of the sensor axes at the next image sampling time. Time update, coupled with perspective projection, gives a prediction of a feature's location in each sensor's image plane at the next sampling time [7].

The feature tracking algorithm is currently limited to tracking features in imagery where the sensor geometry has been fixed. This limitation is only a practical matter, not a theoretical restriction. A fixed sensor geometry means that during the tracking of features the relation (position and orientation) of each sensor to all the other sensors and the vehicle is fixed, thus disallowing pan/tilt sensor mounts.

5.1 Autonomous Tracking Units

Low-level feature tracking is parallel in nature. Each feature, once it is detected, is assigned an autonomous tracking unit (ATU). The ATU can be implemented in software as a separate process or thread. The tracking unit can track a feature in a single sensor over time. The tracking unit can also obtain multiple measurements by using more sensors. Fig. 4 depicts the strategy used to track a single feature in a two-sensor stereo image sequence. A feature is initially detected at image location $[u_{11}, v_{11}]$ in the master sensor. The ATU tracks the feature over time within the master sensor, while a match to a slave sensor is used to add

[1]Since each sensor will have different dynamics due to helicopter motion, the Kalman filter will need to be linearized for each sensor during measurement update.

stereoscopic information to the range estimation process. The range is initialized using a pure stereo measurement. The tracking strategy of Fig. 4 can accommodate any number of like sensors.

5.1.1 Feature detection

The feature tracking mechanism begins with the process of feature selection by partitioning the master image using a *cell grid*. Each cell is a square pixel area with an odd number of pixels, $2n+1$, to a side. Each grid cell is scanned to see if an image feature is present. Features can therefore be detected only with the spatial accuracy (within the image plane) of the grid resolution. Given an $N \times N$ image the grid would have $N/(2n+1) \times N/(2n+1)$ cells. The cell grid greatly speeds up feature extraction because only $N^2/(4n^2+4n+1)$ locations need to be searched instead of N^2 locations which many traditional token-based matching schemes require [19]. A cell size of 11×11 pixels gives good overall performance, balancing matching accuracy (discussed later) versus spatial resolution. A 512×512 pixel image would therefore be divided into 46×46 cells with 6 pixels remaining along two of the edges.

For this implementation, feature selection is based on intensity variance within a grid cell. The following equation shows the intensity variance calculation at a grid cell centered at $[u, v]$ of size $(2n+1) \times (2n+1)$ pixels ($n = 5$ for a 11×11 cell size, and let N=(2n+1)):

$$\mu_I(u,v) = \frac{1}{N^2} \sum_{i=-n}^{n} \sum_{j=-n}^{n} I(u+i, v+j) \tag{22}$$

$$\sigma_I^2(u,v) = \frac{1}{N^2-1} \sum_{i=-n}^{n} \sum_{j=-n}^{n} (I(u+i, v+j) - \mu_I(u,v))^2 \tag{23}$$

If σ_I^2 is greater than a constant threshold value σ_C^2, then the image location $[u, v]$ is said to be a feature.

5.1.2 Search window

Once a feature has been detected in a grid cell, a correspondence is generated between it and an identically sized pixel area in another image. This provides measurement of the feature's optical flow. The target image may be taken from another sensor at the same time (stereo) or from the same sensor at a different time (motion) or a combination of both (motion/stereo). Search windows are generated by projecting the estimated three-dimensional feature location onto the target sensor image plane.

If a feature is new (i.e., has no range estimate) then a worst case guess is made based on a priori near and far clipping planes of a range volume in front of the helicopter. Fig. 5 illustrates the search window generation procedure for a newly detected feature in the motion/stereo approach. An object O gives rise to a feature in sensor $\mathcal{S}_1$ (the master sensor) where ρ_{s1} intersects the image plane. Only the feature's basis vector $e_{\rho_{s1}}$ is known, since only a single sensor can be used to detect new features. The minimum, nominal, and maximum lengths for ρ_{s1} can be computed using $e_{\rho_{s1}}$ and the near and far clipping planes, R_{min} and

R_{max}. The nominal value of ρ_{s1} is calculated using $R_{nom} = (R_{max} + R_{min})/2$. The three vectors $\rho_{s1_{min}}$, $\rho_{s1_{nom}}$, and $\rho_{s1_{max}}$ are generated using the following equation with R equal to appropriate clipping value:

$$\begin{bmatrix} \rho_{s1x} \\ \rho_{s1y} \\ \rho_{s1z} \end{bmatrix}_R = \begin{bmatrix} R \\ R\frac{u}{f_1} \\ R\frac{v}{f_1} \end{bmatrix} \tag{24}$$

The three vectors $\rho_{s1_{min}}$, $\rho_{s1_{nom}}$ and $\rho_{s1_{max}}$ can then be transformed from the master sensor axis at time $(k-1)$ to the appropriate target sensor at time (k) ($\mathcal{S}_n$ in Fig. 5). The three vectors, now resolved into the $\mathcal{S}_n$ sensor axis at time (k), can be projected onto the appropriate target image plane using perspective projection. The result of projecting these three vectors is an image plane boundary where the feature should lie if the object O was within the minimum and maximum range clipping planes along the vector $e_{\rho_{s1}}$. Fig. 5 also shows the search window generated by the image plane boundary. The location $[u_{min}, v_{min}]$ corresponds to the projected vector $\rho_{sn_{min}}$; likewise for $[u_{nom}, v_{nom}]$ and $[u_{max}, v_{max}]$. The search window is a diamond shape with the width of the diamond equal to twice the distance from $[u_{nom}, v_{nom}]$ to $[u_{max}, v_{max}]$. This shape approximates a three-dimensional error ellipsoid projected onto the image plane.

Once a feature has an initial range estimate, the near and far clipping planes are derived from the state covariance matrix generated by the Kalman filter. Therefore, as the Kalman filter for a particular feature converges, the clipping planes used for search window generation collapse around the correct range, decreasing the size of the search window and reducing computation. A minimum search window size is enforced to prevent the search window from shrinking too much during convergence.

5.1.3 Correlation

The tracking unit uses the search window computed from a feature's range estimate to find all the pixels in the target sensor image where the feature may be located. A correlation calculation is then made between the cell in the master sensor which bounds the feature and cell sized regions in the target image centered at each pixel within the search window.

The correlation operates on the original pixel intensities rather than image-derived measurements to use as much of the actual image information as possible. The result of the correlation is a value in the interval $[0, 1]$, where 1 indicates a perfect correlation and 0 an uncorrelated match. The algorithm can use any normalized two-dimensional correlation method. We have achieved good performance using normalized correlation which is presented below:[2]

$$\eta_c = \frac{(2n+1)^2 \mu_{AB} - \mu_A \mu_B}{\sqrt{(2n+1)^2 \sigma_A - \mu_A^2}\sqrt{(2n+1)^2 \sigma_B - \mu_B^2}} \tag{25}$$

[2]It should be noted that standard normalized correlation produces a correlation value on the interval $[-1, 1]$. Our implementation maps all negative correlations onto zero.

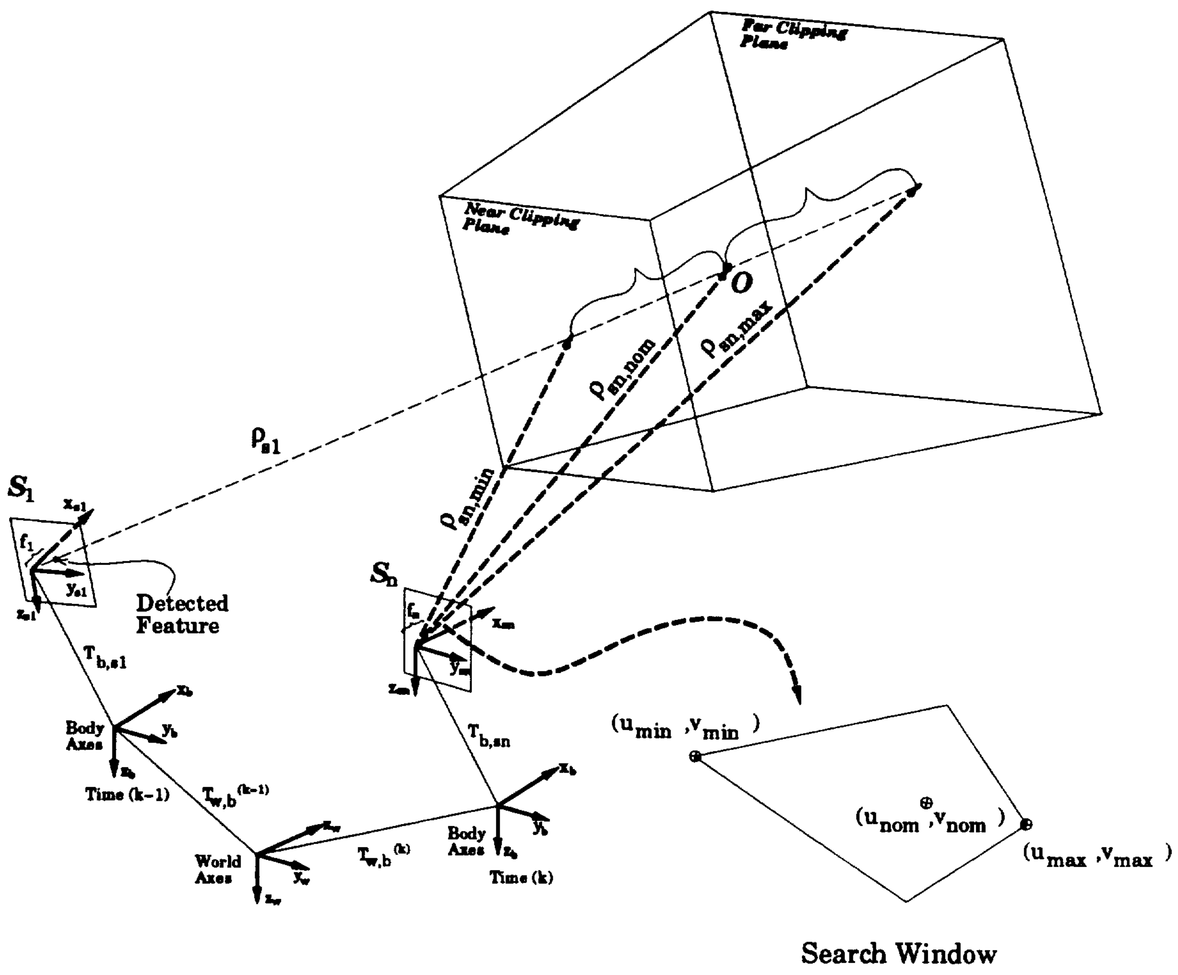


Figure 5: Search window generation.

where

$$\mu_A = [\sum_{i=-n}^{n} \sum_{j=-n}^{n} I_A(u-i, v-j)] \tag{26}$$

$$\mu_B = [\sum_{i=-n}^{n} \sum_{j=-n}^{n} I_B(p-i, q-j)]$$

$$\mu_{AB} = [\sum_{i=-n}^{n} \sum_{j=-n}^{n} I_A(u-i, v-j) I_B(p-i, q-j)]$$

$$\sigma_A = [\sum_{i=-n}^{n} \sum_{j=-n}^{n} I_A^2(u-i, v-j)] \tag{27}$$

$$\sigma_B = [\sum_{i=-n}^{n} \sum_{j=-n}^{n} I_B^2(p-i, q-j)]$$

The feature is detected at $[u, v]$ in image A and is correlated with image B at location $[p, q]$ as given by the search window. The result of all the correlations is a correlation surface the size of the search window. The image plane location of the peak value of the surface is the "best match" between the cell in the master image and search area in the target image. Quadratic interpolation of the correlation function separately along u and v is used to refine the estimate of the peak to subpixel accuracy. When the next frame is acquired the nearest pixel to the subpixel location is chosen as the location of the feature. The error between the subpixel feature location and the closest integer pixel is then added into the subpixel interpolation for the next frame. Thus features are correlated on integer pixel boundaries but tracking is maintained at subpixel accuracy over multiple frames.

6 Virtual Processing Regions

The autonomous tracking units described in the previous section are task-parallel in nature. Once a feature is detected within the cell grid, an ATU is spawned to track the feature. If a feature leaves the image plane or otherwise becomes untrackable then the ATU dies. As motion imagery evolves, ATUs will track the optical flow within the image. Thus an ATU will generally flow from the center of the image toward an edge (assuming forward motion). If each ATU spawned by a particular grid cell is assigned to a processor then the data requirements (image data) for that processor must be the union of the data requirements of each ATU spawned by that cell. This is not a problem when the features are young and close to the originating grid cell. Over time, though, the ATUs will spread out from the originating cell and could potentially cover much of the image. Therefore, the autonomous nature of ATUs will lead to data requirements which will evolve to be nonlocal within the image space.

Nonlocal data requirements for each task can lead to poor performance in a multiprocessor system. This is due mainly to the communication overhead of either sending large portions of the data space to each processor in a distributed-memory system, or memory

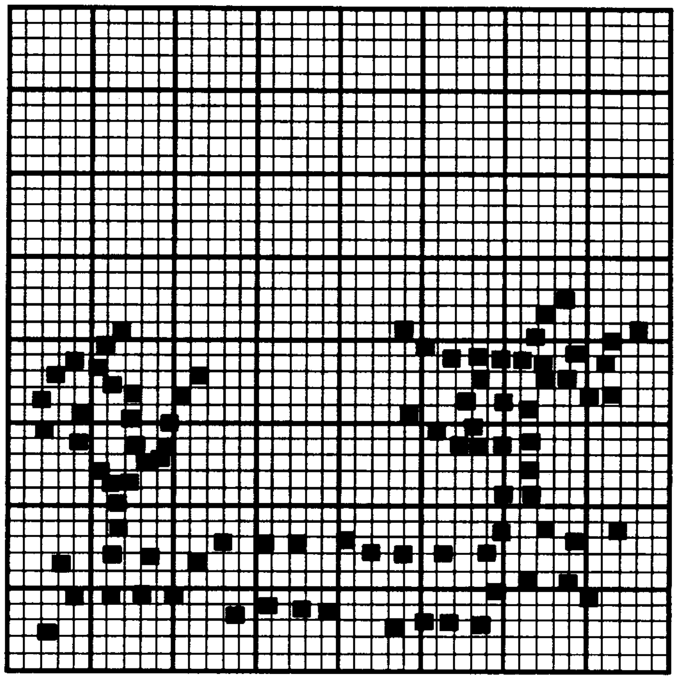


Figure 6: Image plane partitioning.

network contention in a shared-memory system [20]. The implication of this is that task parallelism alone is not sufficient to efficiently map the algorithm onto simple parallel hardware. To overcome the data locality requirements, a higher level abstraction is introduced above the level of the ATU. This abstraction, the *virtual processor region* (VPR), adds spatial locality restrictions to each ATU within the image space.

Fig. 6 illustrates the idea behind virtual processor regions. The textured squares represent the location of ATUs within a master sensor image plane. The ATUs are arranged to simulate the tracking of two trees and several ground features. The image is divided into 8×8 VPRs (heavy lines). Each VPR is responsible for maintaining a rectangular arrangement of grid cells. In this example each VPR is allocated 5×5 grid cells (thin lines). The boundaries for the VPRs are the same as for the underlying cell grid. The maximum number of VPRs is equal to the number of grid cells.[3]

The VPRs represent separate regions within the image plane that can be allocated to a processing element (PE). In the example of Fig. 6 there are 64 VPRs which can be distributed among up to 64 processing elements in a task/data parallel fashion. Each PE processes the ATUs (textured squares) and performs feature detection in untracked grid cells (white squares) which are contained within its assigned VPR. Since the VPRs are spatially allocated their image data requirements are fixed. Therefore, as an ATU tracks a feature across a VPR boundary, the VPR passes the ATU to the appropriate neighboring VPR before the next image set is acquired. Currently each VPR is given enough image pixels[4] surrounding the

[3]In the case of 512×512 pixel images with 11×11 pixel cells there can be as few as one VPR or as many as 2116 VPRs.

[4]Each VPR is currently given a ten pixel wide image strip from each of its neighbors. The size of this

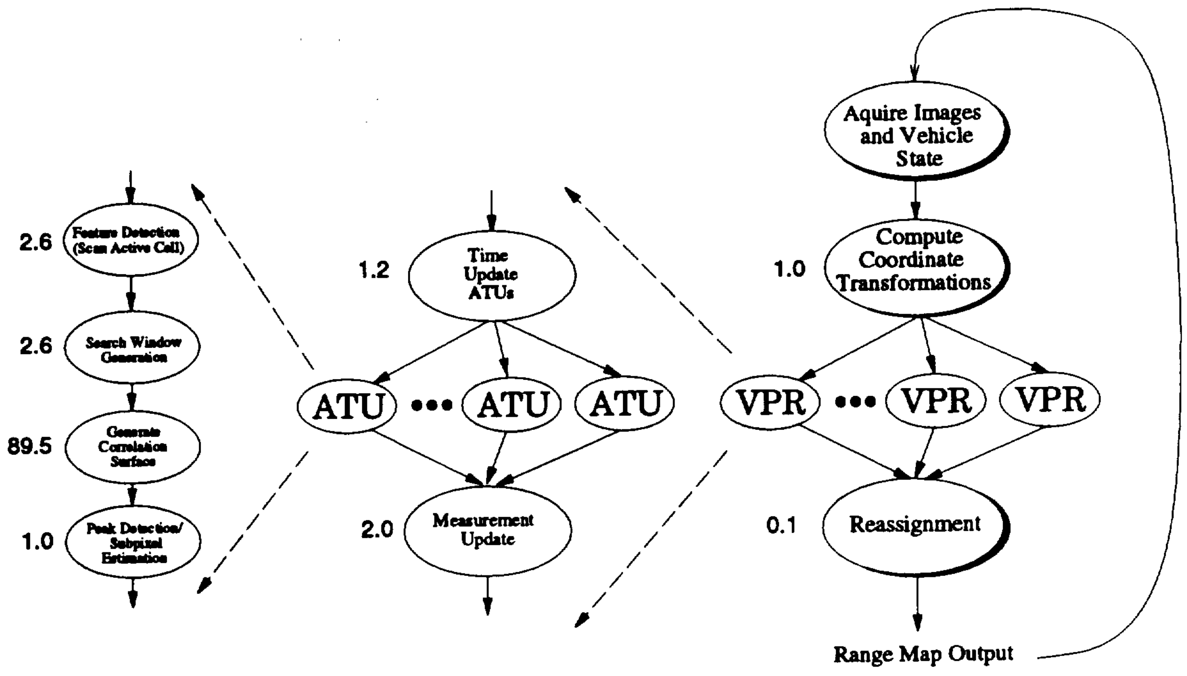

Aquire Images and Vehicle State
1.0
Compute Coordinate Transformations
VPR
•••
VPR
VPR
0.1
Reassignment
Range Map Output
1.2
Time Update ATUs
ATU
•••
ATU
ATU
2.0
Measurement Update
2.6
Feature Detection (Scan Active Cell)
2.6
Search Window Generation
89.5
Generate Correlation Surface
1.0
Peak Detection/ Subpixel Estimation

Figure 7: Feature tracking algorithm flowchart.

VPR proper such that ATUs can be tracked into a neighboring VPR's image space without the need for interprocessor communication. This is an interim solution allowing for more flexibility in the algorithm such that a wider range of architectures may be considered. If an architecture has very cheap nearest-neighbor communication then ATUs would be designed to be transferred between VPRs during the tracking phase of the algorithm. If the tracked features have inter-image shifts greater than can be handled by the extra pixels sent along with a VPR, then interprocessor communication during the tracking phase will be necessary.

Fig. 7 depicts the feature tracking method as a flow chart using the definitions of autonomous tracking units and virtual processing regions. The feature tracking algorithm based on virtual processing regions exhibits Single Program Multiple Data (SPMD) parallelism. Each VPR can be processed in parallel as soon as its input data have been supplied. The VPR however can exploit further parallelism at the ATU level if it contains more than one grid cell. Each ATU/grid cell can be processed in parallel. The data requirements for each ATU are implicitly supplied by the parent VPR. The ATU in turn is composed of a series of serial matrix-like operations which exhibit vector-like parallelism. The number next to each ellipsoid indicates the aggregate percentage of total computation needed by each function.

The motivation behind the ATU/VPR construct is flexibility. The feature tracker can be configured to use as few as one or as many as a couple thousand VPRs. Changing the number of VPRs obviously affects the ATU per VPR ratio. On a highly parallel machine (with several thousand processors) each processor would be assigned either an empty grid

strip is based on the highest inter-image shift expected during tracking.

cell or an active ATU; there would be no need for VPRs. Since the algorithm has been designed to be ported to different architectures, the VPR construct is necessary to reduce the number of task/data parallel units to the optimal number for a given architecture and load balancing scheme.

The ellipsoid labeled *Reassignment* in Fig. 7 indicates the section of the code which detects when an ATU crosses a VPR boundary. As stated previously, since this section could use interprocessor communication it is left as a serial section until an architecture port is made. Since reassignment is relatively cheap on a uniprocessor computer, its parallelization has not been considered in this paper.

7 Load Balancing

Each virtual processing region is task and data independent. The computational load represented by each VPR is proportional to the number of ATUs being managed by that VPR. If the feature distribution in a scene is nonuniform then the number of ATUs per VPR may vary greatly over the set of VPRs. If this occurs then a load balancing technique would be needed to most effectively utilize every PE in a parallel system. Three load balancing techniques have been explored: uniform partitioning and static and dynamic scheduling [10]:

7.1 Uniform partitioning

If scene content is such that features are uniformly distributed over the image plane, then an allocation scheme which creates equal sized partitions would be optimal. Given an N processor machine the image plane would be divided into N equal sized VPRs; one VPR per PE. No explicit load balancing would be necessary to equally utilize each PE due to the uniform distribution of features in the image plane.

7.2 Static scheduling

If scene content is such that features are not distributed uniformly (which is most often the case), then a load balancing technique will be needed to make efficient use of every PE in a system. The major computational load of each VPR is performing the correlations necessary for feature tracking. If the time to scan a cell for a new feature is t_d, the time to generate a correlation surface is t_c and the time to perform measurement and time update is t_f; then if the ith VPR has A_i untracked cells and B_i ATUs (tracked features), the computation time for the ith VPR, τ_i, can be approximated by

$$\tau_i \approx A_i t_d + B_i(t_c + t_f + t_d) \tag{28}$$

If the number of features per VPR does not change too rapidly during steady-state feature tracking, then it would be possible to perform static scheduling for the current frame time based on each VPR's estimated computation time τ_i from the previous frame.

Given that the master image is divided into M VPRs, static scheduling attempts to distribute all M VPRs from the current frame onto a set of N processors so as to minimize completion time. It is required that $M > N$ so that the scheduler has the resolution to properly distribute the load. More precisely: Given N processors, a deadline D and an M element set, $\mathcal{X}$, of VPRs each with estimated computation time τ_i, select a disjoint partition of $\mathcal{X} = \mathcal{X}_1 \cup \mathcal{X}_2 \cup \cdots \cup \mathcal{X}_N$ such that

$$\max\left\{\sum_{i \epsilon \mathcal{X}_j} \tau_i : 1 \leq j \leq N\right\} \leq D \tag{29}$$

The estimated time necessary to process all VPRs with a uniprocessor is

$$T = \sum_{i=1}^{M} \tau_i \tag{30}$$

Thus the best case deadline D possible, given N processors, is T/N. This is known as the Multiprocessor Scheduling Problem and has been shown to be NP-complete [21]. The challenge of static scheduling is to choose the partitions $\mathcal{X}_j$ in a computationally efficient manner such that the maximum PE computation time approaches T/N.

7.3 Dynamic scheduling

If scene content changes rapidly such that the number of tracked features per VPR fluctuates from one frame to the next or the number of correlations necessary to generate a correlation surface fluctuates from one feature to the next, then equation (28) will not be an accurate representation of the computation time required by a VPR. To correct for such occurrences a higher order model of the computation time may be formulated by taking the influencing dynamic factors into account. This approach, though, may lead to an overly complicated computational model or a model which has dominant factors which cannot be predicted efficiently. In such cases a dynamic approach may be used [10]. A simple method is to have a controlling processor distribute VPRs to slave processors from a task queue of VPRs. Processing elements are assigned new VPRs as they finish processing their current VPR. This method of dynamic VPR allocation is practical only if the communication network between the controlling task scheduler and the PEs does not saturate with the necessary communication overhead.

None of the scheduling schemes presented above explicitly takes into account the communication time necessary to download data to each PE. Uniform partitioning will be adversely affected if communication time becomes comparable to computation time. For static scheduling the communication time can be factored into the scheduler, if the delays are deterministic, by introducing it as another element in the cost function. Dynamic scheduling has the benefit of inherently adapting to nondeterministic communication delays and time varying unbalanced architectures [10].

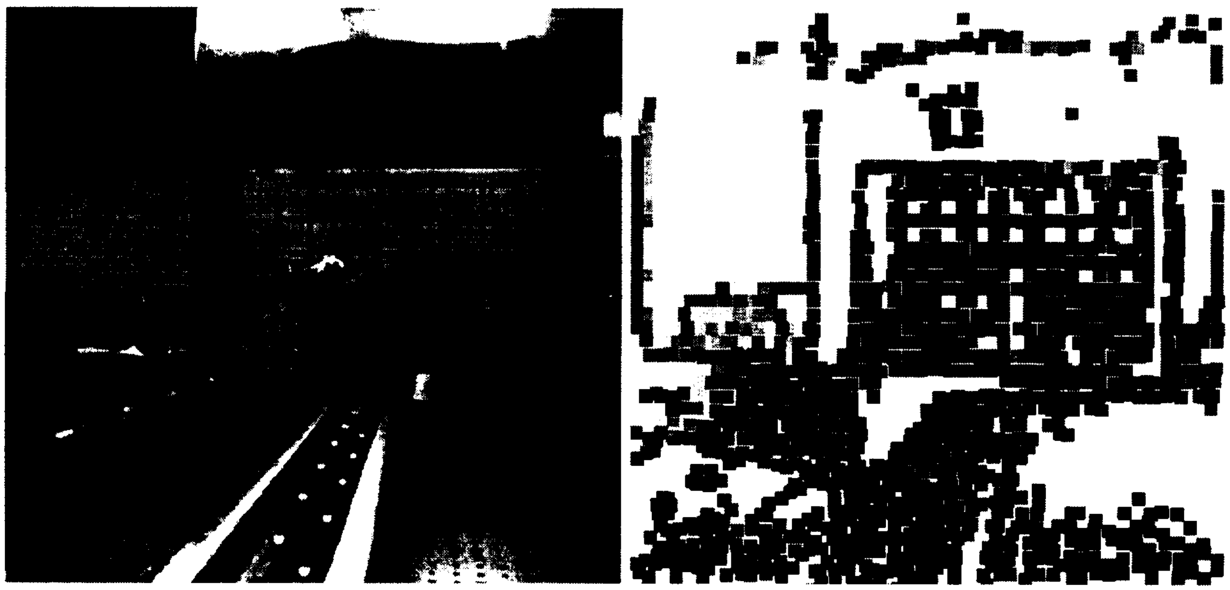


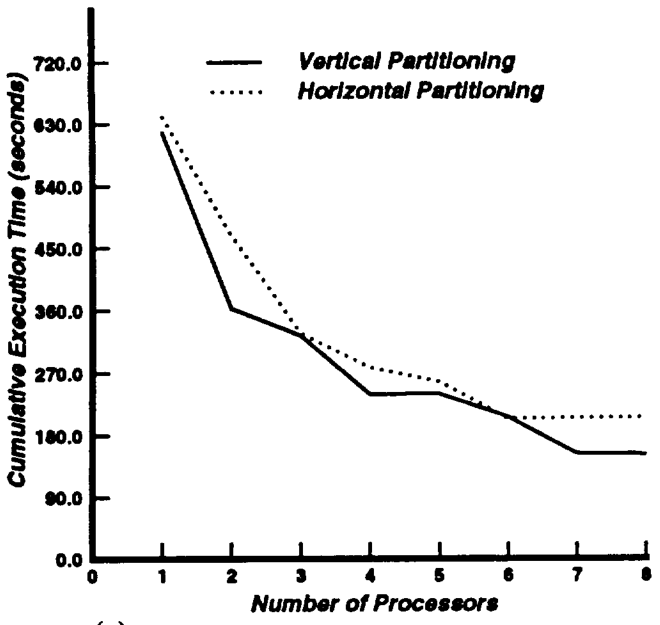
Figure 8: Image with corresponding intensity coded range map.

8 Implementation Results

The parallel constructs and load balancing methods described earlier allow implementation of the feature tracking algorithm on distributed-memory computers or multithreaded shared-memory computers or a combination of both. An ideal evaluation consists of comparing each architecture/load balancing combination and choosing the scheme which performs best. Due to limitations in available hardware and software, only a subset of the schemes has currently been implemented and compared. We present several schemes which represent trends in current parallel computer systems design which have a strong effect on the performance of the algorithm. The execution time and speedup results of the various implementations will suggest further architectures and software design issues for investigation.

The following subsections describe a distributed-memory machine based on a network of workstation-class computers and a modern shared-memory multiprocessor. The feature tracking algorithm was ported to each architecture and, if the operating system software allowed, each load balancing scheme was examined.

Fig. 8 shows the 20th master sensor image along with its corresponding range map. The image is from a sequence of 240 stereo image pairs generated using a computer controlled motion table and scaled helicopter dynamics recorded from flight [22, 23]. The range map, composed of 1450 tracked features, has been projected onto the master image plane with the range coded by intensity. The cumulative execution time to process the first 20 image pairs of the sequence was used as the basis of comparison for each of the computer/load balancing schemes tested.



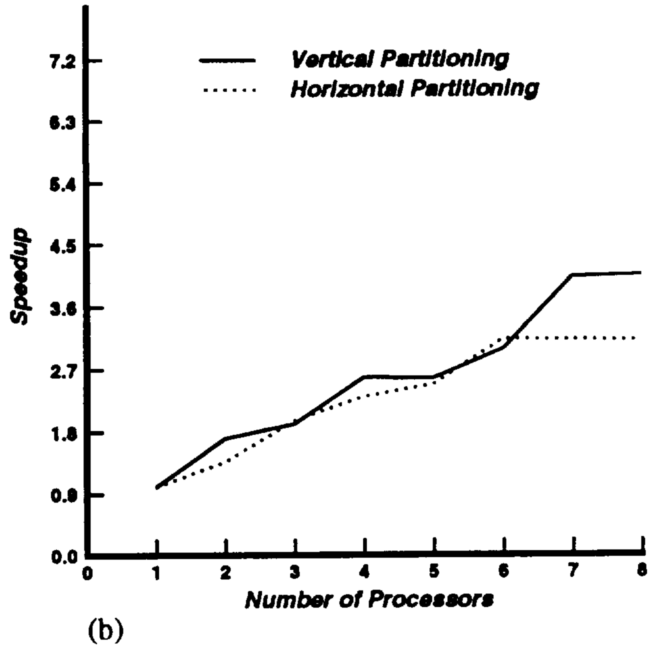


Figure 9: Effect of VPR partitioning scheme and number of PEs on cumulative execution time and speedup for distributed-memory.

8.1 Loosely coupled distributed-memory machine

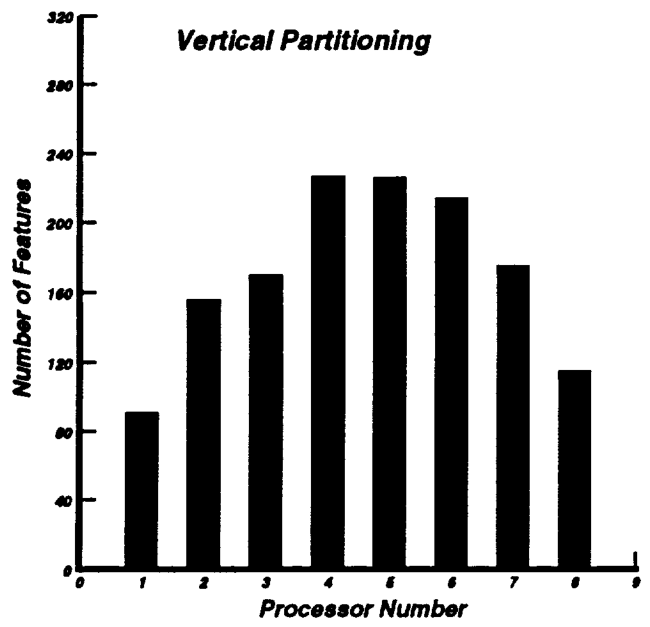
A loosely coupled distributed-memory machine may easily be constructed with an Ethernet network of UNIX[5] workstations using the Berkeley sockets library. The benefits of such an implementation are low cost and ease of configurability. The disadvantage is that the resulting multicomputer is very limited by the bandwidth and latency of the local area network.

The feature tracking algorithm was structured so that a single PE (control node) would run code that would schedule and distribute VPRs to slave PEs (compute nodes). The slave PEs would contain the bulk of the feature tracking code and could each process a set of VPRs. Load balancing is directly controlled by the control node; therefore, each load balancing scheme discussed may be tested.

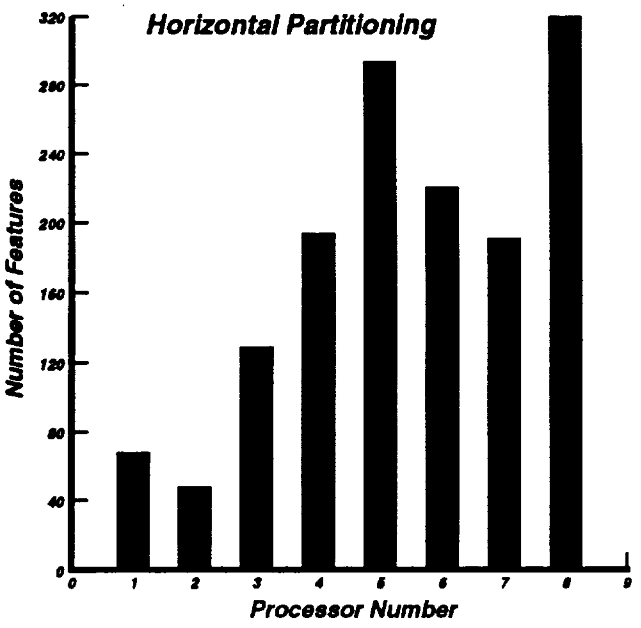
The distributed machine uses a network of nine Sun workstations. Each compute node is a SparcStation2 and the control node is a SparcServer 630MP. During the timing test, only the feature tracking software and routine low-overhead operating system support software were executing on each node.

Fig. 9(a) and (b) and 11(a) and (b) show graphs of the cumulative execution time and the speedup for the distributed-memory machine as the number compute nodes is increased. Fig. 9(b) shows the effect that feature distribution has on speedup using a simple uniform scheduler. For this figure the master image was divided into eight equal-area vertical VPRs and compared with eight equal-area horizontal VPRs. By comparing the speedup graph in

[5]UNIX is a registered trademark of AT&T.

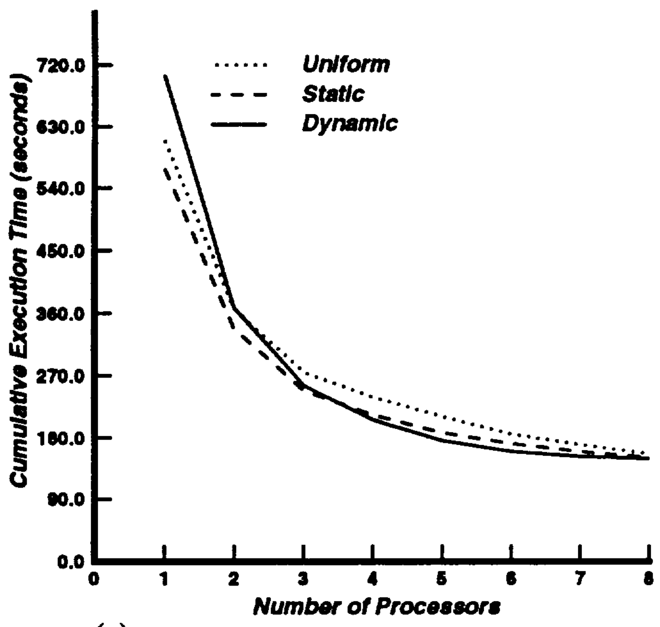


(a)

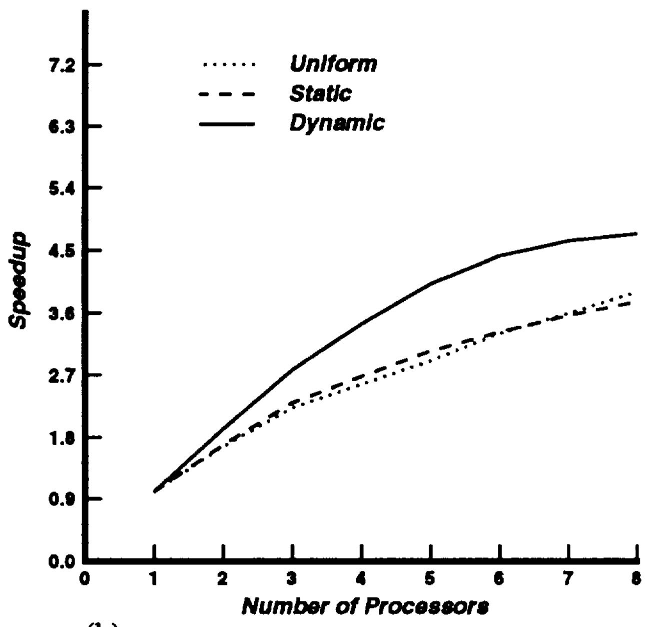


(b)

Figure 10: Feature distribution for eight vertical VPRs and eight horizontal VPRs.

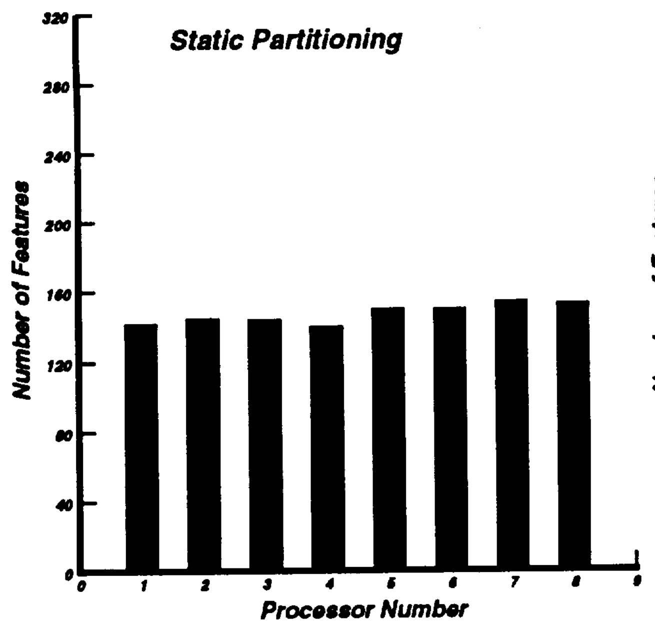


(a)

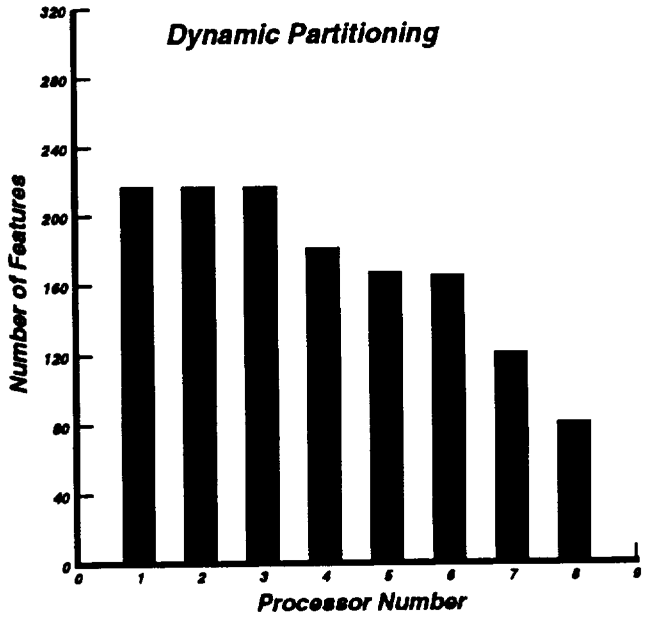


(b)

Figure 11: Effect of load balancing schemes and number of PEs on cumulative execution time and speedup for distributed-memory.



(a)



(b)

Figure 12: Feature distribution for static allocation compared with dynamic allocation using eight PEs.

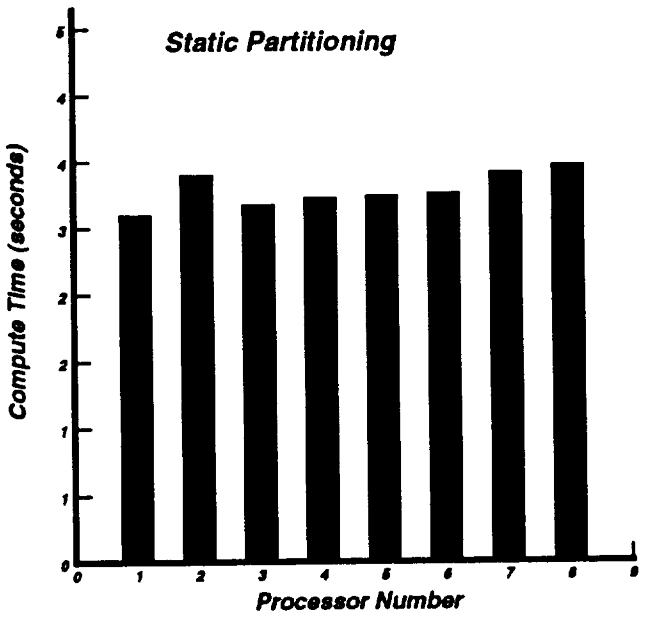


Figure 13: Computational distribution for static allocation.

Fig. 9 with the feature distribution in Fig. 8, one can see that a vertical partitioning will more evenly allocate the ATUs among the processors than a horizontal partitioning for this particular image sequence. Fig. 10(a) and (b) shows these feature distributions graphically for eight compute nodes during the last frame.

Fig. 11(a) and (b) compare the uniform, static, and dynamic load balancing schemes using $M = 64$ VPRs (much like Fig. 6). We found that choosing $M = N^2$ when N is small (i.e., $N \leq 8$), will give fine enough resolution for the static or dynamic scheduler to perform an efficient schedule for the tested feature distributions.

The VPRs of the uniform method are distributed by vertical columns plus fractions of a column when necessary. This method schedules $64/N$ VPRs to each compute node, where N is the number of nodes. When $N = 8$ the feature distribution for 64 VPRs is identical to the vertical VPRs of Fig. 10(a).

We solved the static scheduling problem stated earlier (i.e., computation of the partitions $\mathcal{X}_j$) with a computationally efficient *bin packing*[6] heuristic, "*First Fit Decreasing*" (FFD). The FFD algorithm operates as follows: VPRs are taken in order of nonincreasing τ_i and assigned to the first PE which has enough computational capacity to accommodate it. The major computational burden of this method is the initial $O(M \log_2 M)$ sort of estimated compute times. The benefits of this algorithm are twofold: first, it is easy to implement and modify and, second, it has been shown to have several strong properties of asymptotic optimality [24].

One slight modification to the theoretical static scheduler was made prior to implementation. The estimated compute time τ_i in equation (28) is heavily dependent on the correlation surface time t_c. A computer trace (SparcStation2) of the feature tracker gives the following results: $t_c = 20.25$ ms, $t_d = 1.08$ ms, and $t_f = 0.68$ ms. In light of these numbers the static scheduler was implemented such that B_i from equation (28) was the estimate of the computational load of each VPR (i.e., letting $t_d = 0$ and $t_f = 0$, ignoring any reference to true time). VPRs with no actively tracked features are evenly divided among all the processors. Fig. 12(a) shows the feature distribution of the static scheduler during the 20th frame, while Fig. 13 shows the compute time distribution during the same frame. Comparisons of these graphs lead us to believe that load balancing using only the number of active features per VPR may be accurate enough to perform the processor allocation. The benefit of this modification is also twofold: first, accurate estimates of t_c, t_d and t_f are not necessary as long as $t_c \gg (t_d + t_f)$ and, second, the static scheduler uses only integer mathematics which speeds computation.

The feature distribution for the dynamic scheduler is shown in Fig. 12(b). We can see that only the first six nodes are highly utilized. This is because of the low speed of the node interconnect (Ethernet). From the cumulative time graphs it is clear that the dynamic scheduler outperforms uniform partitioning for more than two nodes. It also outperforms the static scheduler for more than three nodes. Fig. 12(a) would seem to indicate that the static scheduler should outperform the dynamic scheduler because it does a better job of distributing the load. This is not the case, because, as the number of nodes increases, the communication time begins to dominate the total processing time. With the nondynamic

[6]Bin Packing is closely related to the Multiprocessor Scheduling Problem [21].

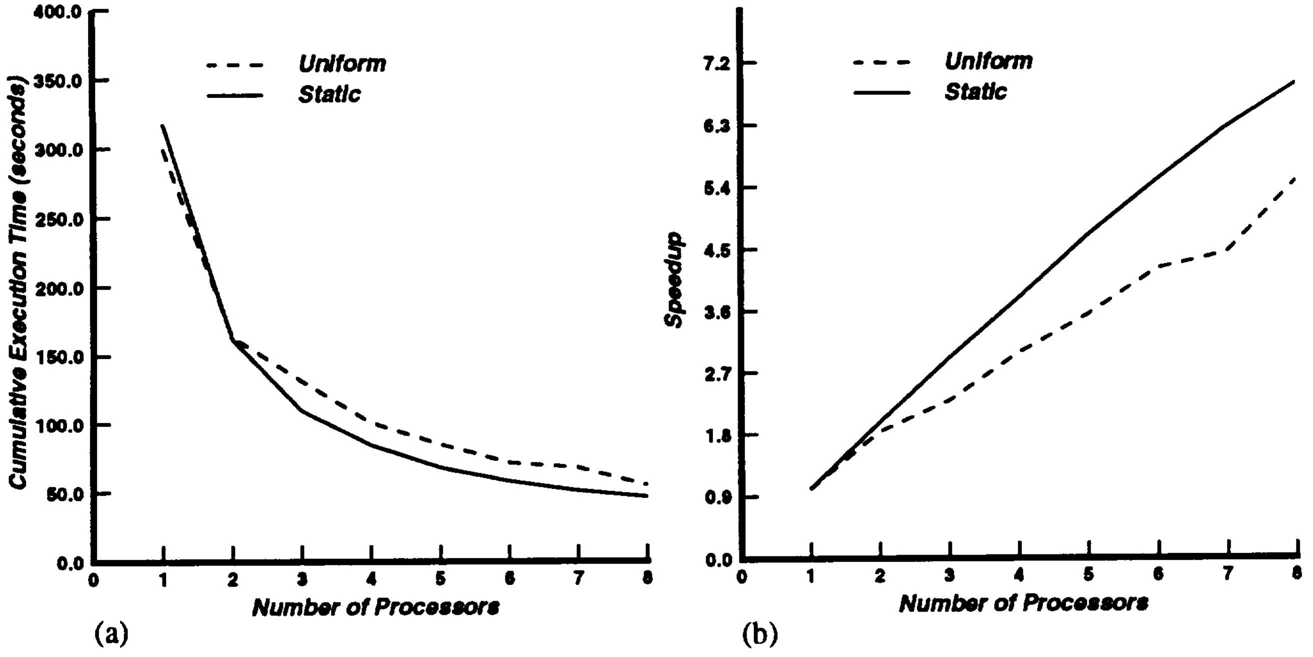


Figure 14: Cumulative execution time and speedup for shared-memory.

schedulers there will be wasted time after a node finishes processing its VPRs until the distribution node retrieves the results. From these graphs it is clear that the Ethernet saturates at six nodes for this application regardless of the scheduler.

The caveat for the dynamic scheduling is the poor scalability of the communication overhead needed to support nonblocking I/O between the distribution node and the compute nodes. Since the static scheduler performs nearly as well as the dynamic scheduler, we predict that the static method will win out as the number of nodes increases.

8.2 Shared-memory machine

A Silicon Graphics IRIS 4D/480 was used to implement a multithreaded shared-memory version of the feature tracker. The 4D/480 has eight RISC processors in a shared-memory configuration. It is typical of modern shared-memory multicomputers in that the processing elements are unbalanced with respect to the shared-memory interconnect [25]. This results from the fact that memory interconnect speeds have not kept pace with the increased speed of modern RISC CPUs. The 4D/480 also comes with a limited multithread support library based on the Sequent Computer Systems parallel programming primitives.

Ideally for dynamic scheduling, each VPR would be assigned a thread, where the number of threads is greater than the number of processors. The operating system would then perform dynamic thread management. Such a method would be comparable to the dynamic scheduler for the distributed-memory machine. The Sequent/SGI primitives allow for thread process management of a limited number of threads (default maximum of eight). With so few threads available, any load balancing scheme based on a large number of threads is unworkable. Therefore the only scheduling schemes which could easily be implemented were those based upon a limited number of threads: uniform and static scheduling.

The uniform partitioning method was the easiest to implement. The master image was partitioned into vertical VPRs, starting with a single VPR and working up to eight VPRs. A thread was assigned to each VPR. Since the number of threads was always less than or equal to the number of processors, the thread count was equal to the utilized processor count. Fig. 14 shows graphs for the cumulative execution time and the speedup for uniform partitioning.

Also in Fig. 14 are results from an implementation of static scheduling on the 4D/480. Instead of assigning one VPR to each thread the load balancer generates an index map whereby each thread can address the appropriate set of VPRs from shared-memory. The overhead of the static scheduler is outweighed by the increased efficiency as can be seen in the graphs.

9 Conclusions

The parallelization of the multisensor feature-based range-estimation software has proven quite successful. The method has shown good speedup with up to eight processors. We have shown that the algorithm, even though it is complex and data-driven, can be efficiently parallelized into many independent task/data units which may be processed by a distributed-memory or a shared-memory parallel computer. The Silicon Graphics 4D/480, using all eight processors and the static scheduler, was able to process the 1450 features of the 20th frame in 2.59 seconds. Thus to reach ten frames a second with a maximum of 1500 features we will need to speed the processing up 26 times. This is a realistic goal which can be achieved in the near future by increasing the number of processing elements, their performance, and interconnect speeds.

The most detrimental aspect of our distributed-memory computer was the low bandwidth of the node interconnect. To combat this problem we will consider new systems with much faster node interconnect speeds. We are planning to port the algorithm to an Intel iWarp[7] systolic mesh computer [26] and possibly the next generation of Silicon Graphics Multiprocessors.

In this paper we have not focused on parallel feature reassignment or image data acquisition and distribution. These are important issues in a real-time system and will be topics in the next phase of our research.

References

[1] V.H.L. Cheng and B. Sridhar. Considerations for automated nap-of-the-earth flight. In *Proceedings of American Control Conference*, Atlanta, GA, June 1988.

[2] V.H.L. Cheng and B. Sridhar. Integration of active and passive sensors for obstacle avoidance. *IEEE Control Systems Magazine*, 10(4):43–50, June 1990.

[7]iWarp is a trademark of Intel Corp.

[3] B. Sridhar, V.H.L. Cheng, and H. Swenson. Automatic guidance of rotorcraft in low altitude flight. In *AGARD 53rd Symposium of the Guidance and Control Panel on Air Vehicle Mission Control and Management*, Amsterdam, Netherlands, October 1991.

[4] B. Sridhar and A.V. Phatak. Simulation and analysis of image-based navigation system for rotorcraft low-altitude flight. In *Proceedings of the AHS Meeting on Automation Application for Rotorcraft*, Atlanta, GA, April 1988.

[5] P. K. Menon, G. B. Chatterji, and B. Sridhar. Vision-based optimal obstacle avoidance guidance for rotorcraft. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, New Orleans, LA, August 1991.

[6] Y. Barniv. Velocity filtering applied to optical flow calculations. *IEEE Trans. on Aerospace and Electronic Systems*, to be published, October 1992.

[7] B. Sridhar, R.E. Suorsa, and B. Hussien. Passive range estimation for rotocraft low altitude flight. *Journal of Machine Vision and Applications*, 1991.

[8] A. Choudhary and S. Ranka. Parallel processing for computer vision and image understanding. *Computer*, 25(2):7–10, February 1992.

[9] L.S. Haynes, R.L. Lau, D.P. Siewiorek, and D.W. Mizell. A survey of highly parallel computing. *Computer*, 15(1):9–27, January 1982.

[10] A.N. Choudhary and J.H. Patel. *Parallel Architectures and Parallel Algorithms for Integrated Vision Systems*, chapter 6, pages 123–128. Kluwer Academic Publishers, Norwell, MA, 1990.

[11] B. Sridhar, P.N. Smith, R.E. Suorsa, and B. Hussien. Multirate and event driven kalman filters for helicopter passive ranging. In *Proceedings of the 1st IEEE Conference on Control Applications*, Dayton, Ohio, September 1992.

[12] T.S. Chang, K. Qiu, and J.J. Nitao. An obstacle avoidance algorithm for an automatic land vehicle. *International of Journal of Robotics and Automation*, 2(1):21–25, 1987.

[13] K. Skifstadt and R. Jain. Range estimation from intensity gradient analysis. *Machine Vision and Applications*, 2(1):81–102, 1989.

[14] P.K. Menon and B. Sridhar. Passive navigation using image irradiance tracking. In *AIAA Guidance, Navigation and Control Conference*, Boston, MA, August 1989.

[15] J. Ma and S.I. Olsen. Depth from zooming. *Journal of Optical Society of America*, 7(10):1883–1890, 1990.

[16] G. B. Chatterji P. K. Menon and B. Sridhar. Passive obstacle location for rotorcraft guidance. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, New Orleans, LA, August 1991.

[17] P.N. Smith, B. Sridhar, and B. Hussien. Vision-based range estimation using helicopter flight data. In *Proceedings of the 1992 IEEE Computer Society*, pages 202–208, Champaign, Illinois, June 1992.

[18] B. Sridhar and R.E. Suorsa. Integration of motion and stereo sensors in passive ranging systems. *IEEE Trans. on Aerospace and Electronic Systems*, 27(4):741–746, 1991.

[19] S.T. Barnard and W.B. Thompson. Disparity analysis of images. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 2(4):333–340, 1980.

[20] J.L. Hennessy and Patterson D.A. *Computer Architecture A Quantitative Approach.* Morgan Kaufmann Publishers, Inc, 1990.

[21] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completness*, chapter 3, pages 63–65. W.H. Freeman and Company, San Francisco, CA, 1979.

[22] R.E. Suorsa and B. Sridhar. Validation of vision based obstacle detection algorithms for low altitude flight. In *Proceedings of the SPIE International Symposium on Advances in Intelligent Systems*, Boston, MA, November 1990.

[23] P.N. Smith. Flight data acquisition for validation of passive ranging algorithms for obstacle avoidance. In *Proceedings of the American Helicopter Society Forum*, Washington, D.C., May 1990.

[24] B.J. Lageweg, J.K. Lenstra, A.H.G. Rinnooy Kan, and L. Stougie. Stochastic integer programming by dynamic programming. Technical memorandum, Centre for Mathematics and Computer Science, Amsterdam, P.O. Box 4079, 1009 AB Amsterdam, April 1985.

[25] T.J. LeBlanc and Markatos E.P. Shared memory vs. message passing in shared-memory multiprocessors. Technical memorandum, Computer Science Department, University of Rochester, Rochester, NY 14627, May 1992.

[26] S. Borkar, R. Cohn, G. Cox, T. Gross, H.T. Kung, M. Lam, M. Levine, B. Moore, W. Moore, C. Peterson, J. Susman, J. Sutton, J. Urbanski, and J. Webb. iWarp: An integrated solution to high-speed parallel computing. In *Proceedings of Supercomputing 88*, pages 330–339, November 1988.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE January 1993	3. REPORT TYPE AND DATES COVERED Technical Memorandum

4. TITLE AND SUBTITLE A Parallel Implementation of a Multisensor Feature-Based Range-Estimation Method	5. FUNDING NUMBERS 505-64-13
6. AUTHOR(S) Raymond E. Suorsa and Banavar Sridhar	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Ames Research Center Moffett Field, CA 94035-1000	8. PERFORMING ORGANIZATION REPORT NUMBER A-93036
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Washington, DC 20546-0001	10. SPONSORING/MONITORING AGENCY REPORT NUMBER NASA TM-103999

11. SUPPLEMENTARY NOTES
Point of Contact: Raymond E. Suorsa, Ames Research Center, MS 210-9, Moffett Field, CA 94035-1000; (415) 604-6334

12a. DISTRIBUTION/AVAILABILITY STATEMENT Unclassified — Unlimited Subject Category 04	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

There are many proposed vision based methods to perform obstacle detection and avoidance for autonomous or semi-autonomous vehicles. All methods, however, will require very high processing rates to achieve real time performance. A system capable of supporting autonomous helicopter navigation will need to extract obstacle information from imagery at rates varying from ten frames per second to thirty or more frames per second depending on the vehicle speed. Such a system will need to sustain billions of operations per second. To reach such high processing rates using current technology, a parallel implementation of the obstacle detection/ranging method is required. This paper describes an efficient and flexible parallel implementation of a multisensor feature-based range-estimation algorithm, targeted for helicopter flight, realized on both a distributed-memory and shared-memory parallel computer.

14. SUBJECT TERMS Computer vision, Load balancing, Range estimation	15. NUMBER OF PAGES 26
	16. PRICE CODE A03

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102