PARALLEL PROCESSING SYSTEMS FOR PASSIVE RANGING DURING HELICOPTER FLIGHT

Banavar Sridhar and Raymond E. Suorsa
Mail Stop 210-1 NASA Ames Research Center
Moffett Field, CA 94035-1000, U.S.A.

Abstract

The complexity of rotorcraft missions involving operations close to the ground result in high pilot workload. In order to allow a pilot time to perform mission-oriented tasks, sensor-aiding and automation of some of the guidance and control functions are highly desirable. Images from an electro-optical sensor provide a covert way of detecting objects in the flight path of a low-flying helicopter. Passive ranging consists of processing a sequence of images using techniques based on optical flow computation and recursive estimation. The passive ranging algorithm has to extract obstacle information from imagery at rates varying from five to thirty or more frames per second depending on the helicopter speed. We have implemented and tested the passive ranging algorithm off-line using helicopter-collected images. However, the real-time data and computational requirements of the algorithm are beyond the capability of any off-the-shelf microprocessor or digital signal processor. This paper describes the computational requirements of the algorithm and uses parallel processing technology to meet these requirements. Various issues in the selection of a parallel processing architecture are discussed and four different computer architectures are evaluated regarding their suitability to process the algorithm in real-time. Based on this evaluation, we conclude that real-time passive ranging is a realistic goal and can be achieved within a short time.

1 Keywords

Passive Ranging, Helicopter Control, Parallel Processing, Computer Vision

2 Introduction

The design of intelligent guidance systems for helicopter will require information about objects in the vicinity of the flight path of the vehicle. The sensor system on the helicopter should 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. Object location information over the field of view (FOV) is referred to as the range map. Several techniques have been proposed for determining range using electro-optical cameras [1]-[2]. These techniques use optical flow resulting from the relative motion between the camera and objects on the ground together with the helicopter states from an inertial navigation system to compute range to various objects in the scene. In this paper, we refer to the algorithm used to compute the range map starting from a sequence of images as the passive ranging algorithm (PRA).

We have implemented and tested the passive ranging algorithm off-line using both laboratory and helicopter flight images. The algorithm performs satisfactorily and provides range estimates with 5 to 10% accuracy. However, the real-time data and computational requirements of the algorithm are beyond the capability of any off-the-shelf microprocessor or digital signal processor. Parallel processing technology offers a solution to the computational needs of the PRA [3]. The challenge lies in the selection of the appropriate multiprocessor technology and the design of an architecture which takes into account the degree of parallelism present in the algorithm and additional considerations such as weight, volume and ruggedness due to its installation on the helicopter.

This paper is organized as follows: Section 2 provides a brief introduction to the PRA. The computational requirements of the algorithm is described in section 3. Section 4 describes the nature of parallelism in the PRA and some of the factors to be considered in the selection of a parallel computing architecture. Section 5 describes the implementation of the algorithm on different computer architectures. Ideas on future research and some concluding remarks are presented in section 6.

3 Passive Ranging Algorithm

Consider a helicopter-mounted camera that observes a point \( P \) on a stationary object in the environment as shown in fig. 1. The image point \((u, v)\) corresponding to the point \( P \) is given by the perspective projection equations as follows

\[

t = f x / z, \quad v = f y / z
\]

where \( x, y, z \) are components of \( \rho \), the object's position relative to the camera, and \( f \) is the focal length of the camera lens. As the camera moves, the image of \( P \) will also move. If \( P \) is assumed fixed in the Earth frame, the rate of change of \( \rho \) in the camera's axes system can be determined using the Coriolis equation as follows

\[
\dot{\rho} = -V \times \omega \times \rho
\]

where \( V \) and \( \omega \) are the camera's translational and rotational velocities with respect to the Earth frame. The
velocity \((u, v)\) associated with the image of point \(P\) is referred to as optical flow. A feature is defined as a small region of interest within the image.

The passive ranging algorithm consists of estimating range to objects in the field-of-view of the camera given a sequence of images \(k = 1, 2, \ldots, N\). The algorithm consists of two major parts: (a) a feature tracking part which extracts the measurement \((u(k), v(k))\) from the image and (b) estimation of range given the measurements. The feature tracking mechanism divides the initial image into a number of square pixel regions or cells. In this implementation features are squares of \(11 \times 11\) pixels which exhibit intensity variation greater than some user-specified minimum threshold value. Fig. 2 shows the selection of features and the propagation of features between successive images. The optic flow measurements are obtained from the difference between an object’s location in successive images. The optic flow measurements are obtained from the difference between an object’s location in successive images. The number of features for which range estimates can be obtained depends directly on the ability to select robust features that can be unambiguously tracked between images. The quality of range estimate for each feature depends on the ability to accurately track the feature between images. A normalized correlation method is used to determine the feature’s location in each new image.

The computation of range is formulated as a state estimation problem using a Kalman filter. The Kalman filter is well-suited to this application because it combines redundant measurements to recursively improve its estimate over time. In addition, the state covariance matrix provided by the Kalman filter gives an indication of the estimate accuracy. The Kalman filter also yields a prediction of the state vector, the state covariance matrix, and an expected location of the feature for the next sample time. This latter information is used to constrain the search area for locating the feature in the next image. As the range information improves, the search window becomes smaller and less computation is required to locate the feature.

Several Kalman filter implementations were studied in [1], where the best results were obtained by selecting the state vector \(X = \left[ p_s \right]^T\) and the measurement vector \(Z = \left[ u, v \right]^T\). With these definitions, the Coriolis equation (2) becomes the state equation and the perspective projection equations (1) become the measurement equations. The state and measurement equations can be written as follows

\[
\dot{X} = -[\omega_s]X - V_s,
\]

\[
Z = h(X) = [f x_s/z_s, f y_s/z_s]^T
\]

where

\[
[\omega_s] = \begin{bmatrix}
0 & -\omega_s & \omega_y \\
\omega_s & 0 & -\omega_x \\
-\omega_y & \omega_x & 0
\end{bmatrix}
\]

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 \(\omega_s\) are constant during the sampling interval \(\Delta T\). The discrete time system equations are

\[
X(k+1) = \Phi(k)X(k) + \Gamma(k)U(k) + \Gamma_d(k)G_d(k)
\]

\[
Z(k) = h(X(k)) + \zeta_d(k)
\]

where \(\Phi(k)\) is the state transition matrix, \(\Gamma(k)\) is the input distribution matrix, \(U(k) = -V_s(k)\) is the control matrix, \(\Gamma_d(k)\) is the disturbance distribution matrix, and \(\zeta_d(k)\) and \(\zeta_e(k)\) model the process noise and measurement noise, respectively. Zero-mean Gaussian white noise is assumed such that \(R(k) = \text{cov}(\zeta_d)\) and \(Q(k) = \text{cov}(\zeta_e)\). The state transition matrix and the control distribution matrices have been derived in [1]. The measurement equation is linearized about the current estimate of \(X\) giving

\[
Z(k) = H(k)X(k) + \zeta_e(k)
\]

\[
H(k) = \frac{\partial h(X_k)}{\partial X}
\]

\[
= f \begin{bmatrix}
1/z_s & 0 & -z_s/x_s^2 \\
0 & 1/z_z & -y_s/z_s^2
\end{bmatrix}
\]

where \(H(k)\) is computed based upon the best state estimate available just before the measurement update. The
discrete-time state equation (5) and the linearized measurement equation (7) are used 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 to the system dynamics. Before each iteration of the Kalman filter, we know \(Q(k)\) and \(R(k)\) and we have estimates of \(X(k)\) and \(P(k)\). The measurement update is then performed according to the following equations:

\[
\bar{X}(k) = X(k) + K(k)[Z(k) - H(k)X(k)]
\]

\[
\bar{P}(k) = [I - K(k)H(k)]P(k)
\]

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

\[
K(k) = P(k)H^T(k)[H(k)P(k)H^T(k) + R(k)]^{-1}. \quad (9)
\]

The time update equations are:

\[
\tilde{X}(k+1) = \Phi(k)\tilde{X}(k) + \Gamma(k)U(k)
\]

\[
\tilde{P}(k+1) = \Phi(k)\tilde{P}(k)\Phi^T(k)
+ \Gamma(k)Q(k)\Gamma^T(k)
. \quad (11)
\]

As noted above, the Kalman filter requires initial estimates for \(X\) and \(P\). The initial estimate for \(X\) can be derived from the optic flow equations and the perspective projection equations given a feature’s location in two images and the camera’s translational and rotational velocities which are assumed constant during the interval between images. The initial estimate of the state covariance matrix is chosen \textit{a priori}.

The passive ranging algorithm has been tested extensively using laboratory images and helicopter flight image sequences. The algorithm performs satisfactorily with range errors under 10% for objects in the range of 100 to 1000 feet. The next section examines the computational requirements of the algorithm.

4 Computational Requirements

Typically, the passive ranging algorithm will receive 512 \times 512 8-bit images at frame rates varying from 5 to 30 frames per second. The amount of data and the processing required may double for stereo (two camera) algorithms. The amount of processing time varies from 200 ms to 33 ms depending on the frame rate. Another major variation in the amount of computation is caused by the number of features in an image. The number of features can vary from a few hundred to a few thousand depending on the scenery. However, an upper limit can be placed on the number of features processed by the passive ranging algorithm. The major part of the computation involves the creation and optimization of the correlation function to determine the position of a feature in the next image. It is estimated that more than 90% of the time is spent in correlation. The number of floating point operations (FLOPs) in each correlation coefficient is approximately 1200. The total number of FLOPs in correlating a feature with an elliptic search area with a major-axis of 11 pixels and a minor axis of 3 pixels is approximately 36000. The maximum number of features in a 512 \times 512 image is close to 2000. At frame rates of 30Hz, this results in a grand computational requirement (60000 features/second) in excess of 2 giga FLOPs/sec (FLOPS). A more modest computational requirement of 90 mega FLOPS is sufficient to track 250 features at a frame rate of 10Hz (2500 features/second). The actual computational requirement will vary substantially depending on how the passive ranging algorithm is used. It is clear from the above discussion that there is no single off-the-shelf microprocessor which can adequately meet the requirements of an unconstrained flexible passive ranging algorithm operating at or near 30 frames per second.

5 Parallel Processing

Parallel processing technology offers a solution to the computational needs of the passive ranging problem. The challenge lies in the selection of the appropriate multiprocessor technology and the design of an architecture which takes into account the characteristics of the algorithm. This section will examine issues in parallel processing, the degree of parallelism present in the algorithm and describe additional considerations in the selection of a parallel computer architecture.

5.1 Issues

Passive ranging involves algorithms from diverse areas such as image processing, numerical analysis, Kalman filtering and visualization. The levels of processing required by different computations in a vision algorithm can be classified into low, intermediate and high. Each type of processing can be done efficiently in a particular kind of parallel processing architecture. In the PRA the various levels of processing need to occur concurrently and, in addition, communicate with each other. Thus, a hybrid multiprocessor architecture is most appropriate for the algorithm. A multiprocessor computer requires (a) the algorithm to be partitioned into sub-tasks, (b) subtasks and data to be distributed among the various processors and (c) communication and synchronization between the processors. There are two different ways for data sharing among processing elements in multiprocessors: shared-memory in tightly coupled processors and message passing in loosely coupled processors. In shared-memory processors a single memory can be accessed uniformly by all processors. The processes communicate through shared variables in memory and synchronization must be available to co-ordinate processes. An alternative model to sharing memory is distributed memory where processes communicate by sharing messages. It is hard, in general, to write computer programs for multiprocessors that achieve close to linear speedup as the number of processors assigned to the task increases. Unlike the uniprocessor case, efficient programming on a multiprocessor requires the knowledge of the memory architecture to achieve a fast and
scalable program. This tight coupling makes parallel programs difficult to port and is one of the biggest challenges facing computer science.

5.2 Algorithm Parallelism

The algorithm has parallelism at two levels: region level and feature level. An image is divided into a number of processing regions. Each region is searched for features. Several regions can be handled by a single processor. Load balancing is achieved by allocating a different number of regions to each processor such that all the processors have the same estimated computation time. Each feature is tracked independently and there is no communication between features. As a feature is tracked, its location may change from one region to another. Thus, there is intercommunication at the region level. The parallel architecture has to acquire the image, distribute the regions to the processors and ensure all processors have finished their tasks before attempting the next cycle.

5.3 Other Factors

Clearly, computing power is the dominant criterion in the selection of a processor. Other factors in the choice of the computing architecture are:

1. The computer has to be installed on a helicopter. Thus, special attention should be paid to the weight, volume and ruggedness of the hardware.

2. The computer should be able to interface with imaging sensors and accept helicopter state information from inertial navigation system and other sensors. Further, it should be able to output the range information either to a guidance algorithm or to a display system.

3. Due to the rapid changes in computer technology, for research purposes, the computer should be general purpose as far as possible and should be capable of executing other vision algorithms. The architecture should be programmable using high level programming languages. Use of assembly languages, special languages and proprietary software should be avoided as far as possible.

4. The computer should be assembled using off-the-shelf microprocessors and RISC modules. Special purpose architectures and design of special purpose VLSI chips are to be avoided if possible.

6 Computer Architecture

Several different architectures were considered as a cost effective real-time candidate for implementing the PRA. These were evaluated using the criteria described in the previous section. The parallelization of the PRA software has proven quite successful. The method has shown good speedup with up to 20 processors. The algorithm, even though complex and data-driven, can be efficiently parallalized into many independent task/data units which may be processed by a distributed-memory or a shared-memory parallel computer. Here, we describe our experience with four different architectures: a shared-memory machine, two distributed-memory machines and a custom hardware.

6.1 Shared-memory machine

The shared-memory machine tested was a 20 processor Silicon Graphics Onyx. We chose the Onyx due to the promising speedup numbers (6.88 for eight processors) that was achieved by an SGI IRIS 4D/480 in previous research [3]. The Onyx system tested had four 150 MHZ and sixteen 100 MHZ MIPS R4400 processors and an interconnection backplane bandwidth of 1.2 Gbytes/second [4]. The Onyx predecessor, an IRIS 4D/480, has eight 40 MHZ MIPS R3000 Processors and a interconnection backplane bandwidth of 64 Mbytes/second.

The implementation of the PRA on the Onyx was of minimal effort compared to the that needed for the other architectures. The Onyx has a multithreaded operating system (OS) which lends itself to easy coding of data-parallel tasks.

The Onyx with sixteen 100 MHZ processors is able to process 200 to 250 features at 6 frames/second. The speedup in the performance going from 100 to 150 MHZ processors, based on four processor configuration, is approximately 1.47. We estimate a twenty 150 MHZ Onyx system should be able to process 200 to 250 features at 10 frames/second. Thus the Onyx is able to meet the modest requirement assuming only 75% parallel efficiency. The Onyx has real-time image I/O capability.

6.2 Distributed-memory machine

The iWarp is a distributed-memory multicomputer and is the product of a joint development effort between the Intel Corporation and Carnegie Mellon University. The system supports both tightly and loosely coupled parallel processing and was designed to support high-performance signal processing via balanced communication and computation [5]. The iWarp is distinguished from other distributed-memory multicomputers by three significant features: low-latency high-bandwidth inter-node communications, systolic processing, and multiplexed physical connections [6].

Under the iWarp architecture, processing elements, or "nodes", are connected in a 2-D toroidal mesh "array". Each node is a RISC like processor theoretically capable of 20 MIPS and 2 MFLOPS. Each node also has a communication agent able to support four duplex 40 Mbytes/second channels, 320 Mbytes/second aggregate, to other nodes in the array. Input/output for the array is handled by special interface nodes. A schematic diagram of the iWarp architecture is shown in fig. 3.

The PRA was implemented on the Oak Ridge National Laboratory (ORNL) iWarp. The system consists of 16 nodes with 2 Mbytes of static RAM per node. The ORNL iWarp is able to process 524 features/second.

The performance of the PRA on the iWarp suffered
in two different areas. First, current iWarp machines do not have enough memory per CPU to run the PRA without modifications. Second, the compiler and debugging environment are rudimentary. Hours were spent unrolling loops by hand and changing the correlation algorithm's instruction mix to help the compiler keep the integer and floating point units busy.

6.3 Honeywell Datacube

The Honeywell architecture is designed to achieve real-time performance in a single chassis of commercial off the shelf (COTS) hardware [7]. The PRA maps into one VME chassis of hardware consisting of 10 VME cards occupying a total of 12 VME slots. The hardware consists of three types:

- 7 Datacube image processing boards
- 1 Motorola MVME 147 general purpose processor
- 3 Supercard i860 CPUs

Fig. 4 shows the connections between the various processors. The Datacube image processing hardware consists of 3 Max20 boards and 4 VFIR MKIII boards. A Max20 board digitizes the video signal and uses its image memory to store three consecutive frames. The rotorcraft and sensor data from the inertial navigation unit and other sources are fed to the i860B board which performs the necessary co-ordinate transformations and the computation of state equations. The Datacube performs the mean, variance and correlation coefficient computations on square pixel regions. The variance image is available to the MVME 147 over the VME bus from the dual-ported image memory onboard a Max20. MVME 147 performs the simple task of selecting a feature given the variance and directs the Datacube hardware. Datacube generates the correlation function over a rectangular window. The i860C has the task of searching for the peak given the correlation function in the rectangular window. The i860C board is unique due to the presence of a MaxBus interface that allows it to have direct access to image memory that exists within the Datacube hardware. The Kalman filter updates are performed by i860A. At present, the range map is archived during the processing of the images and will be transmitted over Ethernet to a harddrive for storage. The raw range can also be displayed on a terminal connected to the real-time testbed. Based on timelines, it is estimated that the system can process 100-200 features at a frame rate of 30 Hz with a latency of 3 frames.

The Honeywell architecture consists of both special purpose hardware and custom software. It cannot be easily adapted to other vision algorithms and does not scale very well. It meets the modest requirement and, since it is based on hardware which has been flight-tested in automatic-target-recognition, provides an early flight test capability.

6.4 ICI/SKY architecture

The ICI/SKY architecture was designed by Innovative Configurations, Inc. (ICI) based on extensive analysis of the PRA software, evaluation of various COTS hardware and the performance of the algorithm both on a single and four processor SKY i860 based computer [8]. The computation time to process 911 features was reduced from 9.76 sec to 2.56 sec in the 4 processor configuration resulting in a speedup of 3.8. If we assume a degradation to 80% utilization for a 36 processor configuration then the system could process 512 features in 0.19 sec or at the rate of 512 (approximately 2700 features/second).

Fig. 5 shows an overview of the ICI/SKY architecture. It consists of two or more SKY-mp multiprocessors. Each multiprocessor has a 9U VME form factor SKYbolt-mp motherboard with a single INTEL i860 system processor and 3 or 4 arithmetic processor daughter cards. The i860 is a 32-bit general purpose integer processor which runs at 20 MHz. It handles most I/O transfers, data management and general program man-
agement. Each daughter card has 4 i860 arithmetic processors and 16 MB of memory. The i860 runs at 40 MHz and has a dual-ported local memory system with a bandwidth of 160 MB/sec. Each i860 can be used to process a separate sub-task, or a task can be distributed to 4 processors. A crossbar provides four independent data paths for communication between the arithmetic processors and the I/O interfaces. The VME interface connects the processors to the system bus and host processor. The i860 uses this interface to communicate with other processors in the system, memory devices, and remote data acquisition devices. The VSB (VME Systems Bus) provides a dedicated link between the processors and other devices. The IBUS is a 32-bit high-speed interface card. It enables data transfer of 40 MB/sec between the processors and the RS 170 input containing image sequences and frees the host VME bus from heavy communication loading. Range information associated with a feature is sent over the VSB bus and superimposed on the original image after a time delay. The entire information about the feature is sent over Ethernet to a hard drive for further analysis.

The SKY/ICI architecture is based on general purpose computing engines and meets the modest requirement. It has high level language support with custom computer library and custom image distribution software. It scales well and can be programmed to run other computer vision algorithms.

7 Concluding Remarks

We have considered four different computer architectures based on parallel processing technology to achieve real-time passive ranging. Various issues in the selection of a parallel processing architecture were discussed and the architectures were evaluated regarding their suitability to process the algorithm in real-time. All the architectures, except the iWarp, should meet or exceed the modest requirement of 2500 features/second. The iWarp currently lacks real-time image distribution capability. The rapid advancement in processor technology and the large installed base of SGI and other similar general purpose parallel machines will provide the computing power to meet the grand requirement of 60000 features/sec. Today, the Honeywell and ICI/SKY architectures provide a means for achieving real-time passive ranging on a modest scale. This would enable us to evaluate passive ranging for pilot-aiding in low-altitude helicopter flight.

References


