A Real-time Computer Vision Platform for Mobile Robot Applications

Sandor Szabo, David Coombs, Martin Herman, Ted Camus, Hongche Liu
{sszabo,dcoombs,mherman,tcamus,hongche.liu}@nist.gov
http://isd.cme.nist.gov/
National Institute of Standards and Technology
Intelligent Systems Division
Building 220, Room B-124
Gaithersburg MD 20899

ABSTRACT

A portable platform is described that supports real-time computer vision applications for mobile robots. This platform includes conventional processors, an image processing front-end system, and a controller for a pan/tilt/vergence head. The platform is ruggedized to withstand vibration during off-road driving. The platform has successfully supported experiments in video stabilization and detection of moving objects for outdoor surveillance, gradient-based and correlation-based image flow estimators, and indoor mobility using divergence of flow. These applications have been able to run at rates ranging from 3 to 15 Hz for image sizes from 64x64 to 256x256.

1 Introduction

There are many applications for autonomous mobile robots, including military scout missions, urban patrol missions, surveillance and security, highway driving, lunar and planetary exploration, hazardous waste handling, delivery of material to machines for manufacture, delivery within hospitals, household cleaning, and many more. All of these applications have in common the fact that their behaviors can be composed from a common set of generic activities. Many of these generic activities may use vision as a primary form of perception. Examples of such generic activities include detecting and avoiding obstacles, detecting stationary and moving targets, tracking moving targets, camera fixation, image stabilization, landmark recognition and localization, landmark-based pose determination, and egomotion determination.

This paper describes a real-time computer vision platform designed for mobile robot applications, particularly outdoors. The platform consists of a stand-alone enclosure containing electronic and computing equipment. The enclosure is shock-mounted to withstand vigorous vibration encountered during off-road driving. The enclosure contains Sparc computers, an image frame grabber, a real-time image-processing board, and a motion-control board for operating a pan/tilt/vergence head.

Although the platform was designed to handle a target acquisition and tracking task, it can also be used for many of the vision-based activities enumerated above. The task for

which it was designed involves reconnaissance, surveillance and target acquisition (RSTA) from a moving vehicle (termed "RSTA on the Move" [10][11]). The scenario is the following: While the vehicle is moving, a camera mounted on a computer-controlled pan/tilt platform scans the terrain searching for moving targets (such as tanks, trucks, and other vehicles). Once a target is detected, it is tracked by the camera through control of the pan/tilt platform. This task requires several of the generic activities above, including digital image stabilization, detection of independently moving targets, tracking these targets, and computer control of the pan/tilt head. This paper provides examples of the computer vision platform performing real-time image stabilization and real-time moving target detection.

The platform has also been used to experiment with real-time optical flow extraction. Two different algorithms have been implemented: one uses image derivatives to find flow and another uses a correlation matching method. Finally, the platform has been used to perform experiments in real-time obstacle avoidance by an indoor mobile robot using only optical flow for navigation.

This paper is organized as follows. The next section describes the NIST hierarchical control architecture for which the computer vision platform was designed. Then a description of the design and system components of the platform is presented. This is followed by an overview of several applications which have been implemented and tested on the platform. These include RSTA on the Move, real-time optical flow extraction, and real-time obstacle avoidance.

2 Architecture

The computer vision platform described in this paper is designed to be readily used in systems designed according to the Real-Time Control System (RCS) hierarchical architecture [1]. RCS decomposes goals both spatially and temporally to meet system objectives. It monitors its environment with sensors and updates models of the states of the system itself and the world. Figure 1 maps the functionality of the RSTA application into the first four levels of the RCS hierarchy. Figure 12 maps the functionality of the indoor obstacle avoidance system into the first three levels of the RCS hierarchy.

RCS is composed of three parallel legs, *sensory processing (SP)*, *world modeling (WM)*, and *behavior generation (BG)* that interact to control complex systems. The hierarchical levels run in parallel and are labelled, from highest to lowest, *tribe, group, task, e-move (elemental-move), prim (primitive)* and *servo*. The BG modules control physical devices. The WM modules supply information to both the BG hierarchy and the SP hierarchy. It maintains a database of system variables and filters and analyzes data using support modules. The SP modules monitor and analyze 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 and current states of the world and to predict future states of the world.

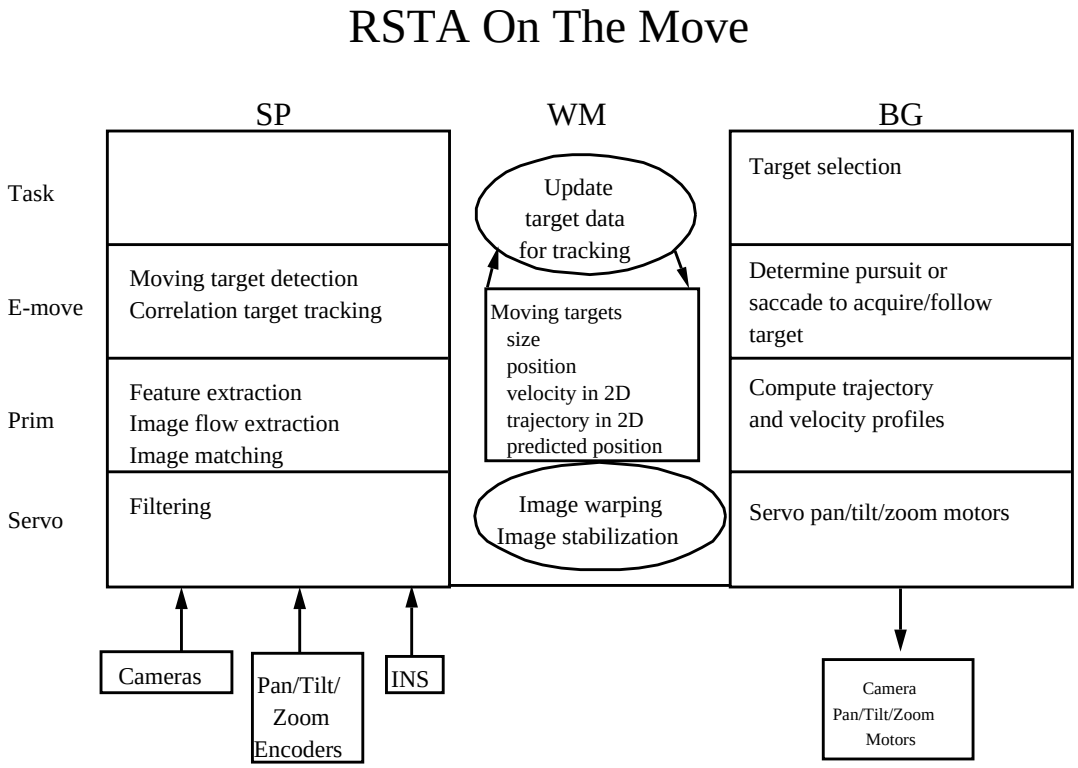

Figure 1: Design of the RSTA on the Move system described using the RCS Architecture.

3 System Description

3.1 Design requirements

The architectures described above are useful in determining what requirements must be met by the computer vision platform. Still, because of the nature of our research and evaluation goals, the requirements are not entirely fixed. A considerable amount of flexibility is necessary in order to experiment with many types of algorithms. A variety of researchers typically work on implementing, evaluating and improving algorithms over a long period of time, so an easily understood environment is essential. Implemented algorithms or components may be obtained from a multitude of sources, so a common platform is desirable. The complexity of vision processing for both indoor and outdoor moving robots demands real-time performance which is often not achievable using single processor systems. A clear path must exist to add additional general purpose processors and specialized processors. Thus, besides the functional requirements outlined by the architectures, the platform must be flexible, easy to use, support a common open architecture and achieve real-time performance.

Based on the above design criteria, a Sparc[1] VME-based multiprocessor system, running the Solaris operating environment, was selected as the core of the computer system. Solaris provides real-time performance and multiprocessor resource management. The

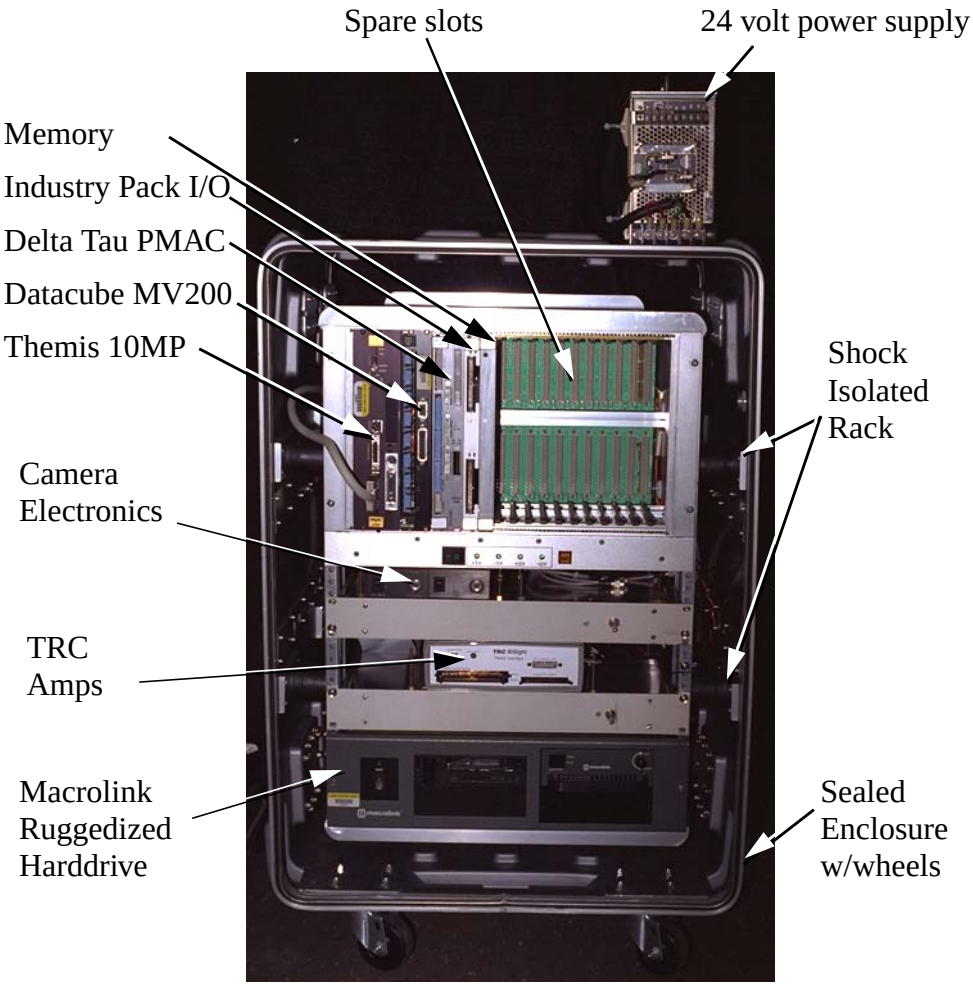

Figure 2: Electronics enclosure.

VME backplane provides the ability to integrate a wide range of specialized board components in a compact card cage. The entire system, with ruggedized hard drive, electronics and power conditioning is housed in a shock isolated 19-inch rack mount enclosure (Figure 2). The enclosure allows the system to be easily mounted on any vehicle that provides 24 volts dc.

In order to provide additional computational bandwidth, several low level functions are implemented on dedicated boards. Low level image processing is performed on a Datacube MV-200. Low level servo control of the TRC head is performed on a Delta Tau

1. Certain commercial equipment, instruments, or 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 best for the purpose.

PMAC motion controller. More details of each system component is presented in the following section.

3.2 System components

3.2.1 Sparc and Bus architecture

The Sparc multiprocessor computer is a Themis 10MP VME board. The board has slots for two MBus cards, each capable of holding one or two HyperSparcs. The current system is configured with one 80 MHz and two 90 MHz HyperSparc processors. Each processor contains a floating-point unit, a memory management/cache controller and a local cache.

The MBus cards provide communications between the processors and memory and to the VME bus interface. The Mbus is a 64-bit multiplexed address (36-bit) and data (64-bit) bus which operates at 40 MHz[9]. During burst transfers it can achieve a peak bandwidth of 320 Mbytes/sec. The VME is a 32-bit address, 32-bit data bus which operates at 40 MHz and can achieve a peak bandwidth of 160 Mbytes/sec[26].

3.2.2 Operating environment

The operating environment is Solaris 2.4 [22] which is based on the SunOS 5/SVR4 (Unix System V Release 4) operating system. Solaris 2.4 is compliant with existing POSIX standards (NIST POSIX Conformance Test Suite Certificate of Validation 151-2SUN003). See Appendix A for a more detailed description of this environment.

3.2.3 Datacube MV-200

The MaxVideo 200 is a VME-based image processing board capable of performing several types of image operations in real-time [19]. Imageflow is an extensive library of software that is used to program and interact with the MV-200. The MV-200 is based on the concept of image pipelines. A pipeline consists of a 1-dimensional stream of pixels which may be operated at a maximum clock rate of 20 MHz. (RS170 formatted video requires a bandwidth of 7.4 MHz.) At the heart of the MV-200 is a 32x32 crosspoint switch which routes pipelines among image processing modules. The MV-200 performs arithmetic operations (multiply, add, normalize), logic operations, table look-ups, 8x8 convolutions, statistics (sums, min/max), feature listings (maximum of 512 per frame), histograms and morphological operations. The MV-200 is also configured with a mini-warper module that performs first and second order polynomial image warps on images as large as 1024 x 1024.

3.2.4 TRC Head

The Transitions Research Corporation (TRC) head consists of a UniSight pan/tilt base that supports a BiSight Vergence head (Figure 3). Each axis is independently driven by encoded DC brush motors. The absolute position is accurate to 1 arc minute. The maximum angular velocities are 650 deg/sec for the pan, 500 deg/sec for the tilt, and 1000 deg/sec for the vergence axes. Maximum angular accelerations are 1350 deg/sec^2 for pan/tilt axes and 12000 deg/sec^2 for the vergence axes [24].

3.2.5 Delta Tau PMAC

The PMAC [12] is a general-purpose VME-based motion control board that performs low level control of the TRC head. The board contains a digital signal processor that is

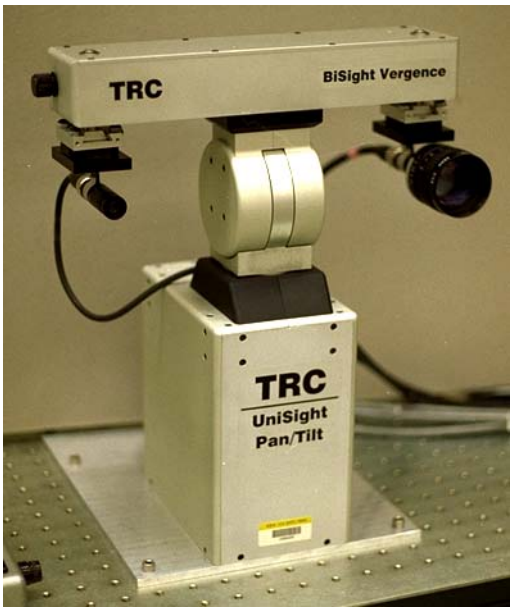

Figure 3: TRC Bisight head with two types of lens.

capable of performing servo updates at a frequency of 2 kHz. The PMAC provides a high level programming interface which contains a suite of trajectory generation algorithms. In the linear blended move, a motor is commanded to move toward a goal at a constant velocity. The acceleration can be constant, resulting in a trapezoidal velocity profile, or can be ramped, yielding an S-curve velocity profile. The PMAC also provides a more direct interface in which the programmer can specify the end position or distance, the end velocity, and the time period for the move. The PMAC computes a third order position trajectory to the goal (ramped acceleration). The programmer must provide new goals prior to the end of the specified time period. The time period can be very small enabling the programmer to specify precise trajectories for visual saccade or pursuit behaviors.

4 Experiments

Several applications have been tested on the vision platform. In this section we examine the implementation of several algorithms and show results of several experiments with them.

4.1 RSTA on the Move

Two algorithms for a RSTA on the Move system were tested on the vision platform. The first algorithm performs image stabilization based purely on image registration. The second algorithm identifies pixels in the image which represent motion in the scene independent of camera induced motion.

4.1.1 Image Stabilization

This algorithm is based on registering features between images. The registration assumes that image motion is attributable to four motion parameters: two translations in the

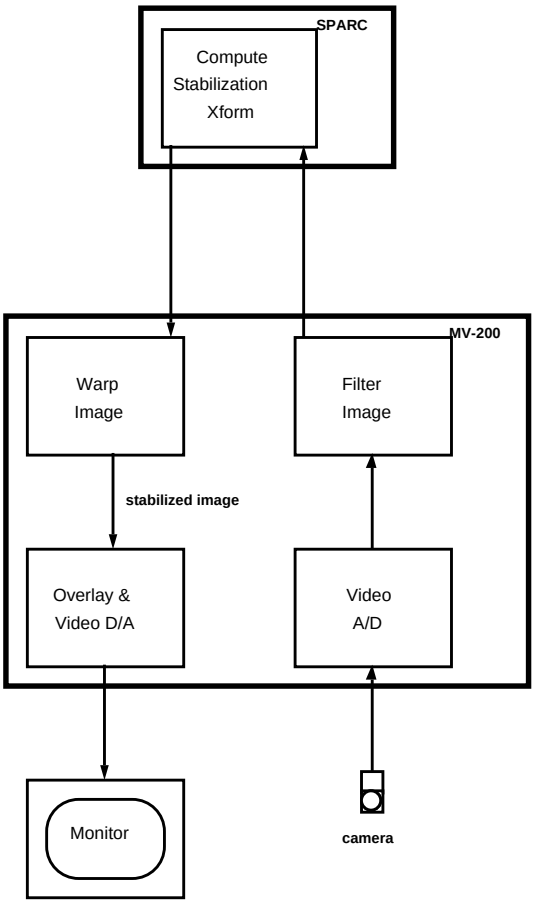

Figure 4: Allocation of stabilization functions to hardware resources.

image plane, rotation about the image axis and scale. Since perspective is not taken into account by the model, the algorithm works best when stabilizing distant portions of the scene. This is appropriate for RSTA applications in which targets are usually at a significant distance.

A detailed description of the algorithm can be found in [20]. Figure 4 shows where components of the algorithm run on the vision platform. Basically, the algorithm consists of extracting and matching features between consecutive image frames. An affine image motion model is then fit from frame to frame. Finally, all images are aligned to a reference image. The following are more implementation details. First, a new image is filtered to remove noise and to accentuate features. Then the image is convolved with a 5 x 5 feature detector kernel. The image is divided into vertical rectangular strips, and convolution maxima are located in these strips, with an attempt to find maxima located along the horizon in the scene. An 11 x 11 patch around each convolution maximum is then correlated with each patch in the previous frame. The best three correlations are chosen to fit rotation, translation, and scale parameters. To obtain subpixel displacement, a second order polynomial surface is fit around the correlation peak instead of using the original grey lev-

els. Scale is estimated based on the Euclidean distance between feature points. The rotation and translation parameters are obtained by solving the overdetermined linear system derived from the set of matched features. The transformations between consecutive frames are composed to compute the transformation between the reference frame and the current frame. Finally, the current frame is warped to the reference frame using bilinear interpolation.

Results of the algorithm are shown in Figure 5. The images were obtained from a U.S. Army HMMWV while traveling cross country at approximately 8 kmph (5 mph). The unstabilized images are shown on the right. A cross hair has been graphically inserted to show how the image moves due to camera motion. On the left are stabilized frames in the same sequence. Again, the cross hairs in this sequence help illustrate that the central portion of the image remains fixed. The algorithm operates on real-time video imagery at a rate of seven frames per second on imagery digitized at 256x256 pixels.

4.1.2 Independent motion detection

This algorithm, developed by Nelson [21], takes advantage of the fact that images from a translating camera appear to expand or flow outwards (i.e., radially) from the focus of expansion (FOE). The FOE lies at the point in the image toward which the camera is moving. In a world full of stationary objects, one can predict the flow at each point in the image. Objects that exhibit flow inconsistent with the predicted flow are flagged as independently moving objects (IMOs).

The algorithm first computes the normal (gradient parallel) flow field of the image using spatial and temporal derivatives. Some estimate of the camera motion is required in order to create the expected motion flow field. In the algorithm, a Hough transform is used to develop a coarse representation of the motion field (4x4 motion field quantized to 8 directions). The transform accumulates the flows of the background as well as the independently moving objects - thus it assumes the background motion is the dominant contributing factor. In the general case of the algorithm, the elements of this transform matrix form a feature vector which is used to index into a library of canonical motion fields. The canonical motion fields correspond to expected flows from various translations and rotations of the camera. For optimization purposes, the algorithm as implemented on the vision platform does not search through the motion field library but assumes the flow is due to a forward moving camera. The expected forward motion field is used to construct a filter which is then compared to the original flow field. Flow vectors inconsistent with the predicted vectors are assumed to be caused by motion of independent objects. Figure 6 shows how this algorithm is mapped onto the vision platform. Figure 7 shows a sequence of imagery where a moving van is highlighted as the HMMWV travels toward the van. The highlighted pixels have been determined by the algorithm to lie on independently moving objects. The algorithm operates on real-time video imagery at a rate of 15 frames per second on imagery digitized at 256x256 pixels.

4.2 Gradient-based optical flow

A gradient algorithm for computing optical flow has been implemented and tested on the vision platform. The algorithm, developed by Liu, *et al.* [15][16], uses up to third-order spatio-temporal derivatives and a generalized motion model that accommodates ex-

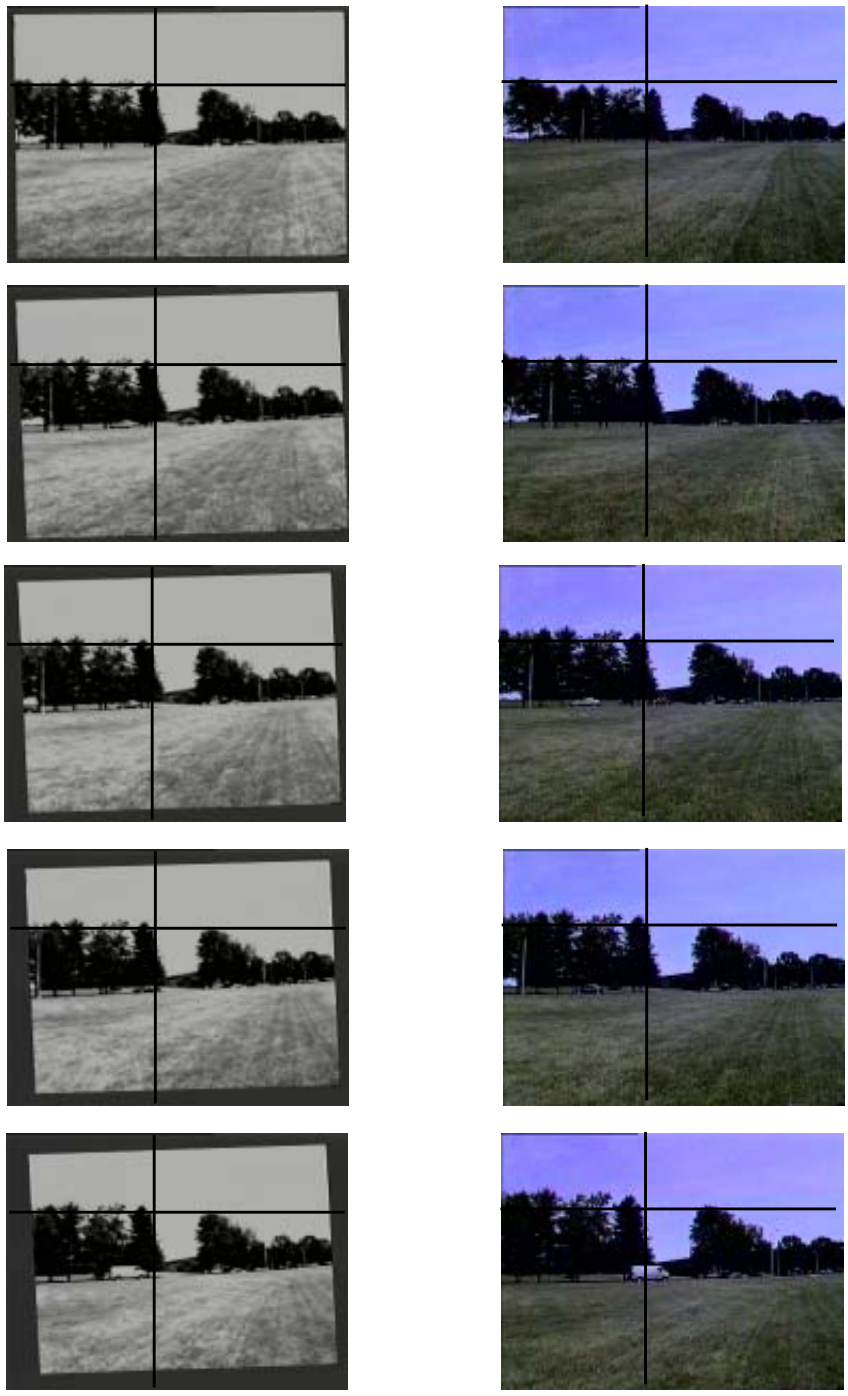

Figure 5: Stabilized frames (in left column) vs. unstabilized frames in the same sequence. Cross-hairs highlight the stability or instability in the image.

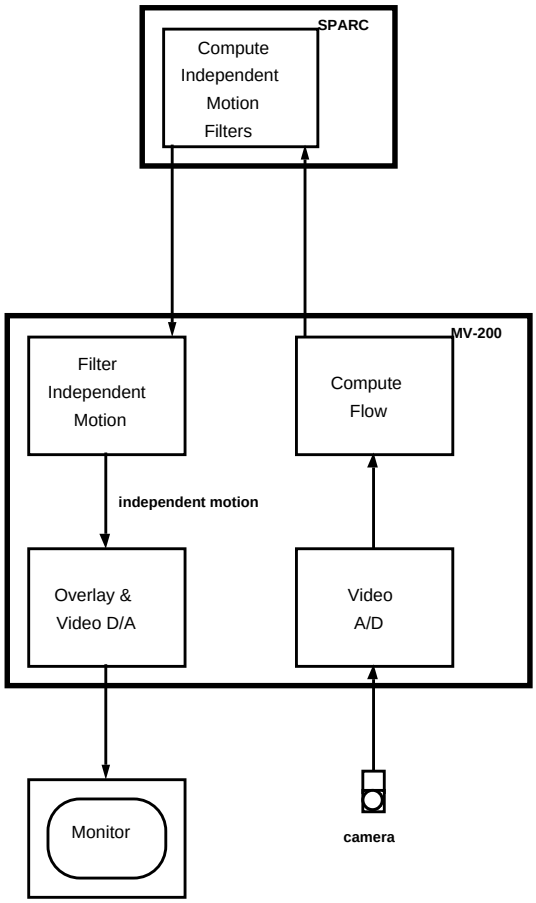

Figure 6: Allocation of independently moving object functions to hardware resources.

pansion and translation. In this approach, the spatial and temporal derivatives are fit to a single coherent motion model (i.e., corresponding to a single object), which leads to a linear system of multiple motion constraint equations. The goodness with which image derivatives fit the model determines the confidence assigned to the motion estimated using the linear system.

The speed is roughly 3-10 frames per second depending on the window size, median filtering, order of derivatives used, density of output, etc. The real-time implementation is depicted in Figure 8. Five successive image frames are smoothed and subsampled to 64x64 pixels. Image derivatives up to third-order are then obtained by applying separable 3-D Hermite polynomial differentiation filters to the neighborhood of each pixel [15][16]. This produces an overdetermined linear system which is then solved using a least squared error (LSE) method. An estimate of optical flow at each pixel is thus obtained.

The accuracy and efficiency of this algorithm, as compared to other state-of-the-art optical flow algorithms (including Horn and Schunck [14], Lucas and Kanade [18], Uras, *et al*. [25], Anandan [2], Fleet and Jepson [13], Bober and Kittler [4], and Camus [6]), is

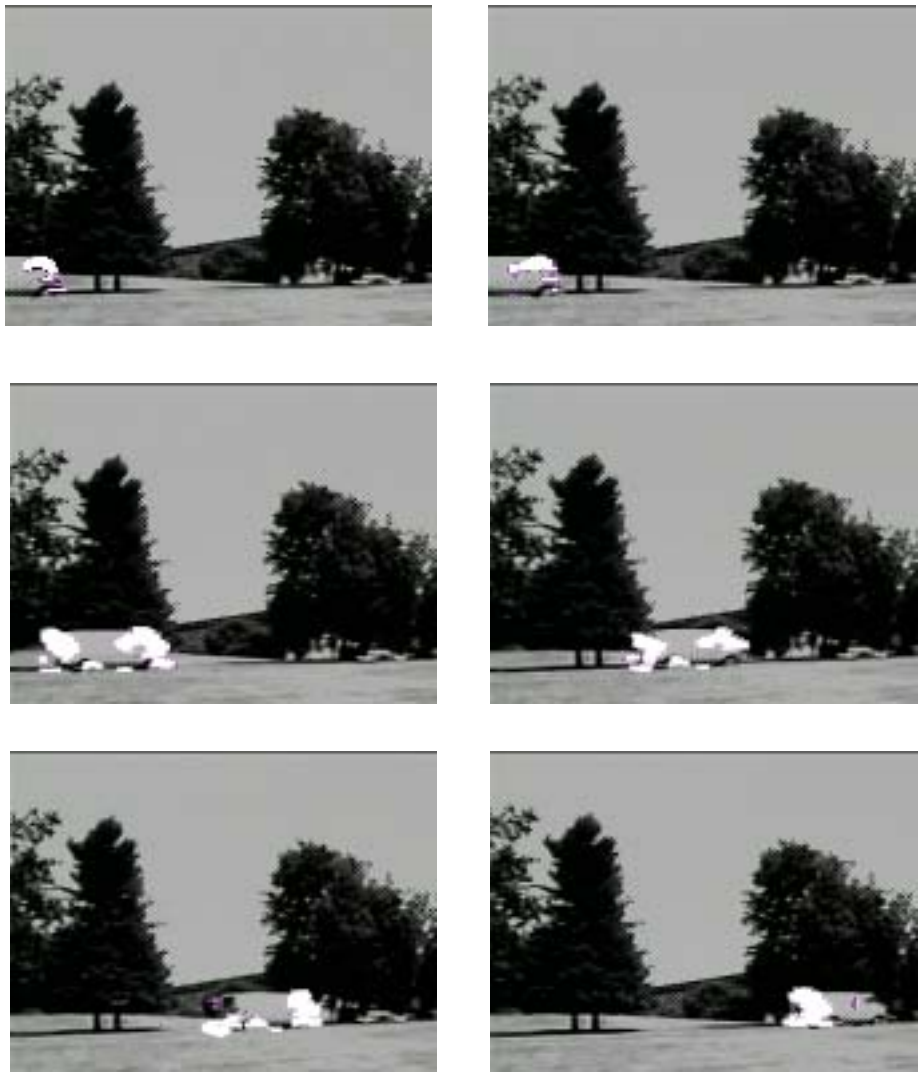

Figure 7: Detecting independently moving objects from a moving vehicle. The images were obtained from a U.S. Army HMMWV while traveling cross country at approximately 8 kmph (5 mph).

depicted in Figure 10 (see [17]). This figure shows accuracy (or error) as one coordinate and efficiency (or execution time) as the other. Two-dimensional accuracy-efficiency (AE) curves in this figure characterize an algorithm's performance. A curve is generated by setting parameters in the algorithm to different values. For optical flow using correlation methods, the template window size and the search window size are common parameters. For gradient methods, the (smoothing or differentiation) filter size is a common parameter. More complex algorithms may have other parameters to consider.

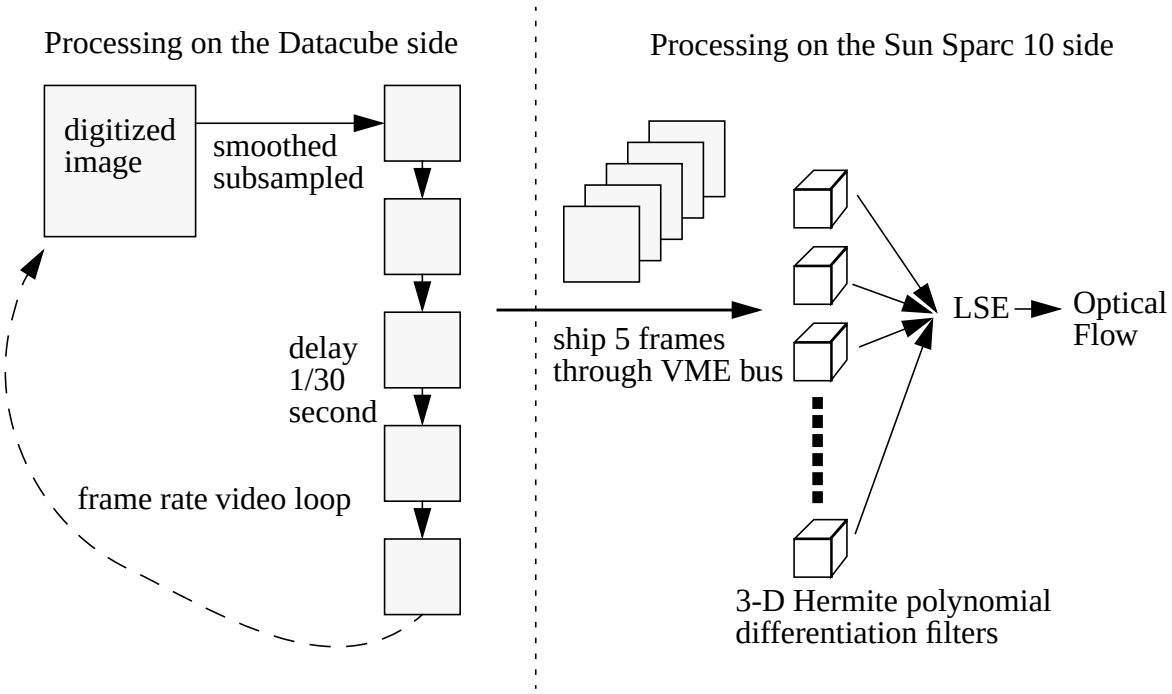

Figure 8: Real-time implementation of the optical flow algorithm of Liu, *et al.* [15]

For optical flow, accuracy has been extensively researched in Barron, *et al.*[3]. Figure 10 uses the error measure in [3], that is, the angle error between computed flow $(u_c, v_c, 1)$ and the ground truth flow $(u, v, 1)$, as one quantitative criterion. For efficiency, we use throughput (number of output frames per unit time) or its reciprocal (execution time per output frame) as the other quantitative criterion. Therefore, in Figure 10, the x axis represents the angle error and the y axis the execution time. The results in this figure are derived from running the algorithm on the diverging tree sequence (Figure 9).

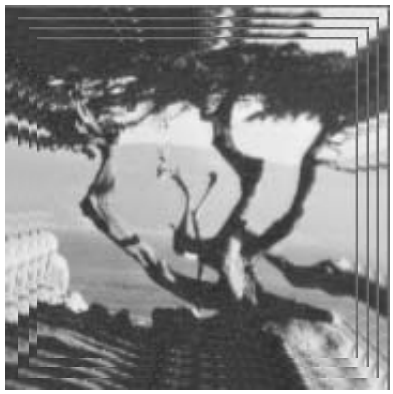

Figure 9: Diverging tree sequence

A point in the performance diagram corresponds to a certain parameter setting. The closer the performance point is to the origin (small error and low execution time), the bet-

ter the algorithm. An algorithm with different parameter settings spans a curve, usually of negative slope. The distance from the origin to the AE curve represents the algorithm's AE performance. In Figure 10, there are two AE curves and several points[1]. It can be seen

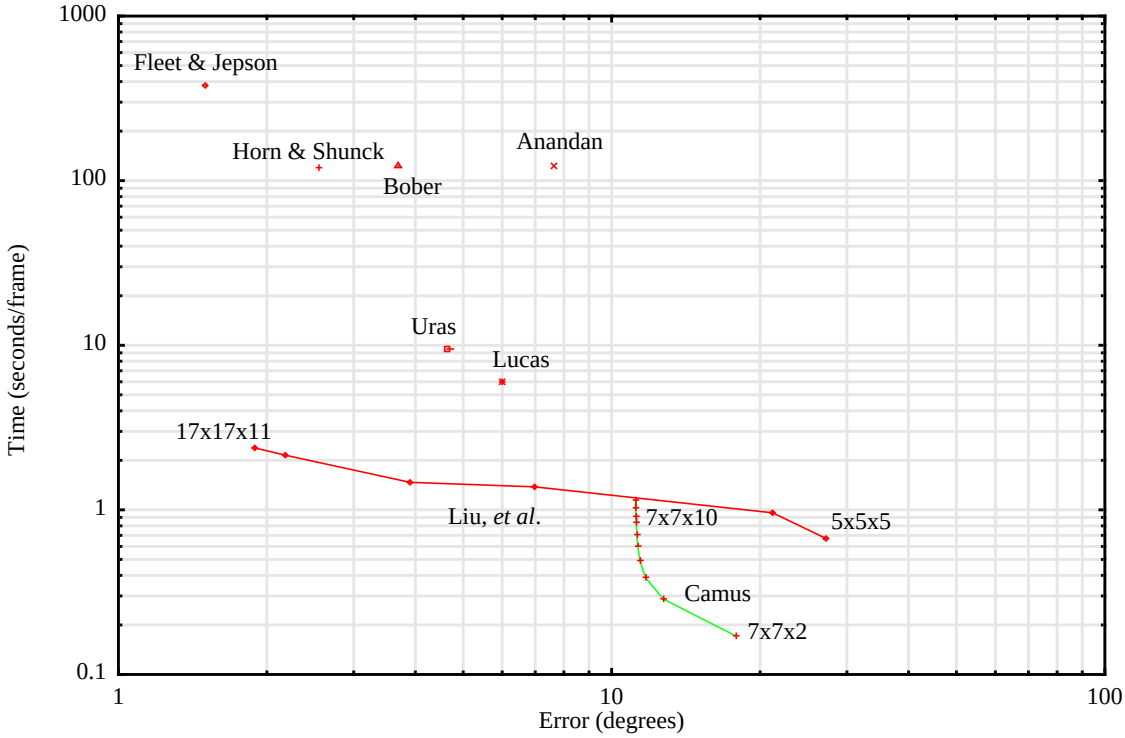

Figure 10: 2-D performance diagram

that some algorithms (e.g., Fleet & Jepson[13]) may be very accurate but very slow while other algorithms (e.g., Camus[6]) may be very fast but not very accurate. The algorithm of Liu *et al*. [15] [16], on the other hand, is very flexible since it can be very accurate for some parameter settings while very fast for other settings.

The algorithm has achieved high efficiency and accuracy by clever design as well as exploiting the vision platform capabilities. It uses the real-time digitizing, convolution and image scaling capabilities of the Datacube MV-200 to smooth and subsample the input images. This way, it avoids an overwhelmingly huge volume of input data to the Sparc and at the same time reduces aliasing (by pre-smoothing). The fast floating-point operations provided by the Sparc make the general motion modeling and differentiation filtering feasible. The separable filter design also contributes to the efficiency of the algorithm.

---

1. The implementations of all algorithms except Liu, *et al*. and Camus were provided by Barron [3]. Some of the algorithms produce different output density; we simply project the error by extrapolation. In Liu's curve, the filter size used ranges from 5x5x5 to 17x17x11. In Camus's curve the template size ranges from 7x7x2 to 7x7x10. The execution time for all algorithms is the approximate elapsed time running on our vision platform (with a single 80MHz HyperSparc 10 board).

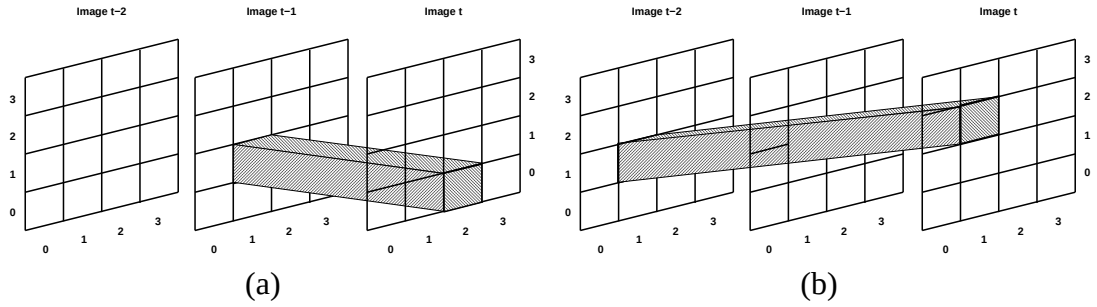

Figure 11(a): Visualization of pixel (1,1) in image T-1 moving to pixel (2,0) in image T, an optical flow of (1,-1) pixels per frame. (b): Visualization of pixel (1,1) in image T-2 moving to pixel (1,2) in image T, an optical flow of (0,1/2) pixels per frame.

The algorithm is capable of handling motion velocities up to about a quarter of the filter size used. It is also very good at extracting very small motion. Generally, it can handle 0.05 to 4 pixels per frame of motion.

In the implementation depicted in Figure 8, five successive image frames are used to compute flow. The flow is computed for the point in time when the middle frame is acquired. Therefore, two past frames as well as two future frames are used. For a throughput of 5 frames per second, the latency of the algorithm is $2 \times 33ms + 200ms = 266ms$, that is, two times the frame latency (33 ms per frame at normal frame rate) plus the algorithm computation time (200 ms).

The fact that we can transport 5 frames of images through the VME bus limits undesirable latency incurred by symmetric filtering of the algorithm. Without this bandwidth, we would have to slowly ship the images one by one to the host (while the images are captured live with the Datacube MV-200). This would increase the latency and also allow only larger image motions to be detected.

4.3 Correlation-based optical flow

A real-time correlation algorithm for computing optical flow, developed by Camus [6],[7] has been implemented and tested on the vision platform. The speed is about 9 frames per second running on the platform (when configured with a single 80 MHz HyperSparc 10) on 64x64 images. It is currently one of the most efficient general purpose optical flow algorithms. The basic idea of this algorithm is to subsample the image and thus constrain the motion velocity so that a quadratic search in space can be reduced to a linear search in time. This temporal matching concept is depicted in Figure 11. This algorithm produces quantized flow estimates. The number of quantization levels (for magnitude and orientation) is related to the range of the temporal search. For the performance mentioned above, the temporal search range is 10 frames. Since the algorithm's computational complexity is linear in the temporal search range as well as the image size, optical flow may be computed at video rates (30 frames per second) on 32x64 images using a temporal search range of 6 frames. The template window size, which does not affect the algorithm's computational complexity [6], is 7x7. Despite the algorithm's simplicity and quantization, it is sufficiently accurate for computing time-to-contact robustly [8].

The correlation measure used is the sum-of-absolute-differences of the intensity values of corresponding pixels within the template window. Due to the two-dimensional scope of the matching window, this algorithm generally does not suffer from the aperture problem except in extreme cases, and tends to be very resistant to random noise. Since the patch of a given pixel largely overlaps with that of an adjacent pixel, match strengths for all displacements for adjacent pixels tend to be similar (except at motion boundaries), and so the resultant optical flow field tends to be relatively smooth, without requiring any additional smoothing steps. Conversely, any noise in a gradient-based flow method can result in errors in the basic optical flow measurements due to the sensitivity of numerical differentiation. In fact the correlation-based algorithm's "winner-take-all" nature does not even require that the calculated match strengths have any relation whatsoever to what their values should theoretically be; it is only necessary that the best match value correspond to the correct motion. For example, a change in illumination between frames would adversely affect the individual match strengths, but need not change the best matching pixel shift. Conversely, a gradient-based algorithm's image intensity constraint equation would not apply since the total image intensity does not remain constant. The robustness of the sum-of-absolute-differences correlation match has been demonstrated on images with extremely low texture [6]. As a result, optical flow measurement density for this algorithm is 100 percent.

4.4 Real-time obstacle avoidance using flow divergence

A real-time obstacle avoidance system has been implemented on the vision platform [5]. The system uses only the divergence of the optical flow field for both steering control and collision detection. The robot has wandered about the lab at 20 cm/s for as long as 26 minutes without collision.The entire system is implemented on the vision platform configured with a single 80 MHz HyperSparc 10 without taking advantage of real-time operating system support (which should be able to further improve performance). Dense optical flow estimates are calculated in real-time (as described in section 4.3) across the entire wide-angle image. The divergence of this optical flow field is calculated everywhere and used to control steering and collision behavior. Divergence alone has proven sufficient for steering past objects and detecting imminent collision. The major contribution is the demonstration of a simple, robust, minimal system that uses flow-derived measures to control steering and speed to avoid collision in real time for extended periods.

Figure 12 sketches the obstacle avoidance system. Video images are obtained from an on-board uncalibrated camera with a 115° field of view. The robot's view from this camera is shown in Figure 13(a). The camera is mounted on a pan motor. The images are subsampled and full flow is computed in this half-height image. Flow divergence is estimated and spatio-temporal median filters are applied to reduce momentary fluctuations in the divergence field. Hazard maps are derived from the divergence field's middle row, the previous steering decision, and the goal direction. A composite hazard map (in this case simply a row vector) is used to steer the robot around objects as it drives in the goal direction. (The temporal concatenation of the one-dimensional hazard vectors is shown in Figure 13(b).) Using active gaze control, the camera is rotationally stabilized to reduce the magnitude of the flows in the image stream. When the camera points too far away from the

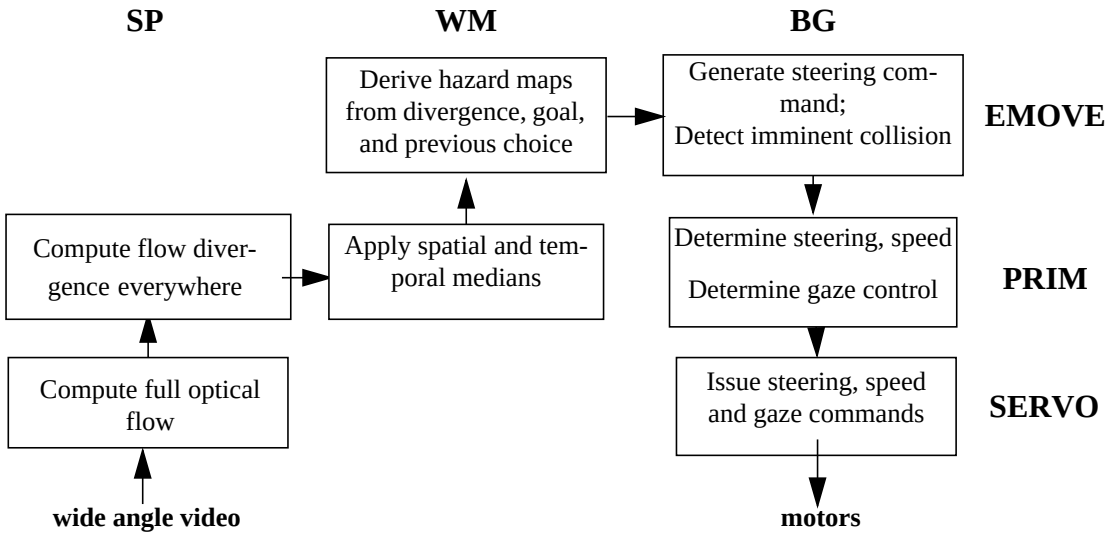

Figure 12. Obstacle Avoidance Architecture

heading, a saccade is made to realign gaze with the anticipated heading. These saccades introduce momentary disturbances of the flow data, but the temporal median filter effectively eliminates disruptive effects. When divergence data indicate imminent collision ahead, the robot stops, turns away, and resumes wandering. The inputs to the body and gaze controllers consist of driving, steering, and gaze velocities. The path of the robot appears in Figure 13(c) for the gauntlet of office chairs seen in (a) from the robot's viewpoint before the trial began. The accumulated composite hazard map (b) shows the succession of one-dimensional maps used by the robot to control driving throughout the trial.

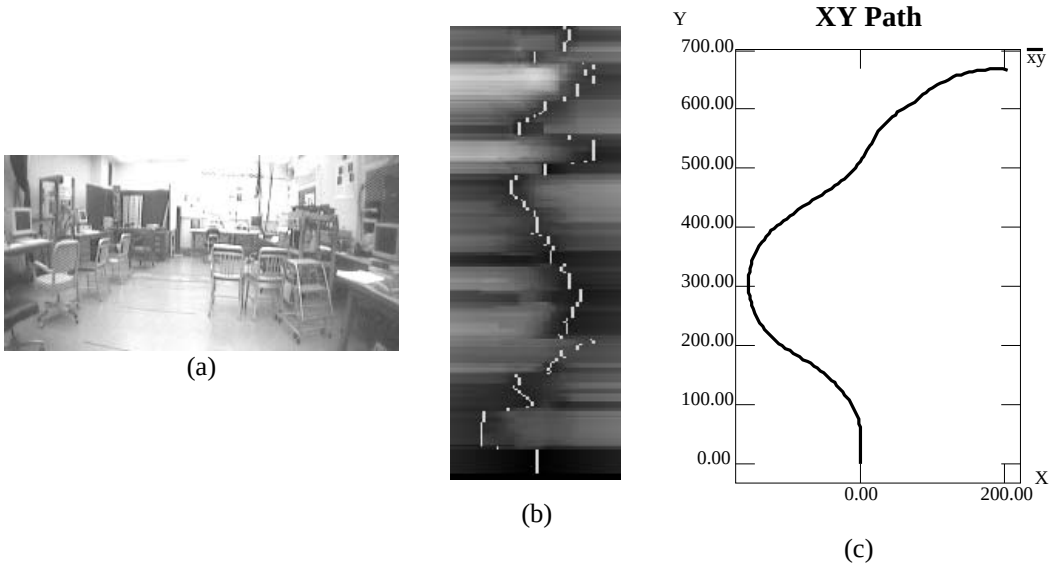

Figure 13: (a) Robot's view of the gauntlet of office chairs before the trial: (b) hazard map (time increasing upward): (c) XY path trace begins at (0,0).

The selected heading is highlighted, and the path can be seen to veer around the stool on the right and then back from the chairs on the left.

The software modules produce and consume data at various rates, and the interactions of the unequal cycle times have considerable consequences. Flow and divergence estimates are produced approximately every 260 ms (3.85 Hz). The robot accepts speed and steering commands at about 3 Hz. Hence, at a robot velocity of 20 cm/s, visual data become available about every 5 cm of robot travel and steering is adjusted about every 7 cm. To avoid losing valuable data, especially time-critical impending collision indications, the behavior controller runs at 20 Hz, evaluating any fresh data and writing appropriate steering and speed commands. These commands are only single buffered, so only the most recent command is read by the robot controller when it is ready for a new one. The systems are designed to require only approximate knowledge of the robot's current motion state if they use any at all. As well, robust data filters are employed to ignore momentary noise and artifacts that result from system module interactions. This approach enables modules to cooperate without delicate synchronization.

Although image motion has long been considered a fundamental element in the perception of space, attempts to use it in real-world mobility tasks have been hampered by noise, brittleness, and computational complexity. These results demonstrate that real-time robot vision and control can be achieved with careful implementations on ordinary computing platforms and environments. Similarly, an extensible framework can combine simple robust components in a manner than minimizes requirements for tight synchronization.

5 Conclusion

A portable computer vision platform has been described. The platform has been used to implement and test not only fundamental computer vision functionality such as image flow estimation but also applications supported by such capabilities, including outdoor surveillance and indoor mobility. The platform has served well as a focal point for bringing together components developed at separate sites by the partners of the RSTA project. Systems were developed using Datacube and ordinary workstations, and the code was readily ported. The platform provides the convenience of a workstation development and testing environment with the rugged portability of a field unit. Future work will focus on integrating the platform onto a HMMWV and performing real-time field tests.

6 References

[1] Albus, J., "Outline for a Theory of Intelligence," *IEEE Transactions on Systems, Man and Cybernetics*, 21(3):473-509, 1991.

[2] Anandan, P., "Measuring Visual Motion from Image Sequences", Ph.D. Thesis, COINS TR 87-21, University of Massachusetts, Amherst MA, 1987.

[3] Barron, J. L., Fleet, D. J. and Beauchemin, S. S., "Performance of Optical Flow Techniques", *International Journal of Computer Vision*, vol. 12, no. 1, pp. 43-77, 1994.

[4] Bober, M. and Kittler, J., "Robust Motion Analysis", *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, Seattle, WA, pp. 947-952, 1994.

[5] Camus, T., Coombs, D., Herman, M. and Hong, T., "Real-time Single-workstation Obstacle Avoidance Using Only Wide-field Flow Divergence", NISTIR, Gaithersburg, MD, in press 1996.

[6] Camus, T., "Real-Time Quantized Optical Flow", *Proceedings of IEEE Conference on Computer Architectures for*

Machine Perception, Como, Italy, 1995.

[7] T. Camus, "Real-Time Quantized Optical Flow", to appear in *The Journal of Real-Time Imaging* (special issue on Real-Time Motion Analysis), Academic Press, 1996.

[8] Camus, T., "Calculating Time-to-Contact Using Real-Time Quantized Optical Flow", NISTIR-5609, Gaithersburg, MD, 1995.

[9] Catanzaro, B., Multiprocessor System Architectures, SUN Microsystems, Mountain View, CA, 1994.

[10] Davis, L.S., Bajcsy, R., Herman, M. and Nelson, R. "RSTA on the Move", *Proceedings of the ARPA Image Understanding Workshop*, Monterey, CA, November 1994.

[11] Davis, L.S., Bajcsy, R., Herman, M. and Nelson, R. "RSTA on the Move: Detection and Tracking of Moving Objects from an Autonomous Mobile Platform", *Proceedings of the ARPA Image Understanding Workshop*, Palm Springs, CA, February 1996.

[12] Delta Tau PMAC Users Manual, Version 1.13, December 1992.

[13] Fleet, D.J. and Jepson, A.L., "Computation of Component Image Velocity from Local Phase Information", *International Journal of Computer Vision*, vol. 5, no.1, pp. 77-104, 1990.

[14] Horn, B. K. P. and Schunck, B. G., "Determining Optical Flow", *Artificial Intelligence*, vol. 17, pp. 185-204, 1981.

[15] Liu, H., "A General Motion Model and Spatio-Temporal Filters for Motion Implementation", Ph. D. Dissertation, University of Maryland, September 1995; NIST-IR 5763, Gaithersburg, MD, March 1996.

[16] Liu, H., Hong, T., Herman, M. and Chellappa, R., "A General Motion Model and Spatio-temporal Filters for Computing Optical Flow", University of Maryland TR -3365, November 1994; NIST-IR 5539, Gaithersburg MD, November 1994, to appear in *International Journal of Computer Vision*.

[17] Liu, H., Hong, T., Herman, M. and Chellappa, R., "Accuracy vs. Efficiency Trade-offs in Optical Flow Algorithms", *Proceedings of the Fourth European Conference on Computer Vision*, Cambridge, England, 1996.

[18] Lucas, B. D. and Kanade, T., "An Iterative Image Registration Technique with an Application to Stereo Vision", *Proceedings of the DARPA Image Understanding Workshop*, pp.121-130, 1981.

[19] MaxVideo 200 Hardware Reference Manual, Datacube Inc., Danvers, MA, October, 1993.

[20] Morimoto, C.H., DeMenthon, D., Davies, L.S., Chellappa, R. and Nelson, R., "Detection of Independently Moving Objects in Passive Video", *Proc. of IEEE Intelligent Vehicles Symposium*, I. Masaki (ed.), Detroit, MI, Sept. 1995.

[21] Nelson, R., "Qualitative Detection of Motion by a Moving Observer", *International Journal of Computer Vision* 7, 33-46, November 1991.

[22] Reference Manual for Solaris 2.4, Product # SOL-24-RD, SUN Microsystems, Mountain View, CA, 1994.

[23] Solaris Multithreaded Programming Guide, SUN Microsystems, Mountain View, CA, 1995.

[24] TRC Pan/Tilt/Vergence Platform Specifications, Transitions Research Corporation, Danbury, CT, 1995.

[25] Uras, S., Girosi, F., Verri, A., Torre, V. "A Computational Approach to Motion Perception", *Biological Cybernetics*, vol. 60, pp. 79-97, 1988.

[26] The VMEbus Specification, VME International Trade Association, Scottsdale, AZ, October 19, 1987.

A. Solaris 2.4 Description

The Solaris environment is based on the SunOS 5/SVR4 (Unix System V Release 4) operating system. Several features of the environment make it a desirable operating system for a general-purpose, real-time, image-processing platform.

The environment defines an Application Binary Interface (ABI) Standard which enables developers to run their applications on a wide variety of hosts. Different ABI's exist for each processor architecture, including the Sparc, 680X0, 88000, 80x86, and MIPS RISC. The ABI lets a developer write an application and have the executable code run on any processor within an architecture. The source code can be compiled to produce executable code for each processor architecture. Thus, applications developed on Sparc architectures can be ported to run on lower cost personal computers.

The environment provides an extensive set of utilities for multiprocessor applications. The core of the utilities is a multithreaded kernel that supports symmetric multiprocessing (SMP). A multithreaded process can execute several instruction streams concurrently on individual processors. In an SMP system each processor shares the kernel image and each

processor can execute kernel instructions simultaneously. Alternatively, in an asymmetric MP system only one processor can run kernel operations simultaneously. Finally, in other MP systems, each processor has its own copy of the operating system with the major disadvantage of increased complexity in communications between instructions streams.

In the environment, parallelism and efficient use of processors is achieved using threads. SunOS uses a two-level thread model [23]. At the user level are those threads identified by the programmer. These threads contain the sequences of instructions that the programmer determines may be executed in parallel. The second level involves the interface to the operating system. At this level, threads are mapped to light weight processes (LWPs). The LWPs, also termed virtual processors, are scheduled by the kernel. The programmer can rely on the thread scheduler (part of the thread library) to handle mapping of threads to LWPs (called unbound threads), or can explicitly map threads to LWPs (called bound threads). The threads scheduler maps threads on available LWPs based on the relative priority of threads within a process.

One advantage of bounded threads is that the programmer can explicitly assign the thread to a physical processor. This might be done in order to create an optimal processor architecture that reflects the programmer's insight into how best to parallelize the algorithm. A second advantage is that the programmer can explicitly specify the priority of the bound thread, i.e., the LWP which may be critical in a real-time application.

The scheduling of LWPs is handled by the kernel. The kernel scheduler is fully preemptive and supports three classes for scheduling LWPs. In the lowest level class, called Time Share (TS), each LWP is scheduled a fixed time quantum (a few hundred milliseconds) and switching takes place in a round-robin fashion. At the next higher level of priority is the System class which consists mostly of kernel processes. The programmer can not schedule LWPs to run within the system class.

The highest priority level of scheduling is the real-time class (RT). Real-time LWPs are scheduled on the basis of their priority and time quantum. An LWP with infinite time quantum runs until it terminates, blocks or is preempted by a higher priority real-time process (typically as a result of an interrupt). An LWP with finite time quantum has the same stop conditions but also ceases to run when the time quantum expires.