Real-time Obstacle Avoidance Using Central Flow Divergence and Peripheral Flow

David Coombs, Martin Herman, Tsai Hong, Marilyn Nashman
National Institute of Standards and Technology
Intelligent Systems Division
Building 220 Room B-124
Gaithersburg MD 20899

ABSTRACT

The lure of using motion vision as a fundamental element in the perception of space drives this effort to use flow features as the sole cues for robot mobility. Real-time estimates of image flow and flow divergence provide the robot's sense of space. The robot steers down a conceptual corridor, comparing left and right peripheral flows. Large central flow divergence warns the robot of impending collisions at "dead ends." When this occurs, the robot turns around and resumes wandering. Behavior is generated by directly using flow-based information in the 2-D image sequence; no 3-D reconstruction is attempted. Active mechanical gaze stabilization simplifies the visual interpretation problems by reducing camera rotation. By combining corridor following and dead-end deflection, the robot has wandered around the lab at 30 cm/s for as long as 20 minutes without collision. The ability to support this behavior in real-time with current equipment promises expanded capabilities as computational power increases in the future.

1. Introduction

Mobile robots that drive at reasonable speeds (*e.g.*, 30 cm/s indoors) must robustly sense and avoid obstacles in real-time. Image motion provides powerful cues for understanding scene structure. Divergence of image flow (the sum of image flow derivatives in two perpendicular directions) is theoretically not affected by camera rotation, so it gives a robust measure of scene structure for a moving observer. In the direction of the camera's heading, divergence is inversely related to time-to-contact (T_c). The system described in this paper avoids obstacles while it wanders. It uses time-to-contact estimates derived from flow divergence to detect imminent head-on collision. This is combined with a centering behavior that compares left and right peripheral flows to steer the robot down a conceptual corridor [6]. That is, the robot attempts to move straight ahead, but will adjust its heading to maintain clearance on both sides. When the corridor-following behavior drives the robot into a "dead end" in the "corridor," the central time-to-contact predictions warn the robot of the impending collision. The robot stops, turns, and resumes wandering, following a new "corridor." These integrated behaviors have driven the robot around the lab at 30 cm/s for as long as 20 minutes without collision. Because this wandering behavior is already a real-time capability, there is promise that future increases in computational power will fuel development of both increasingly robust basic skills and additional behaviors for robots.

Our approach achieves real-time intelligent behavior by using minimalist visually-derived representations. In such representations, a minimal amount of information required to achieve the given task is extracted from the imagery [2][4]. The representations contain only task-relevant information (*i.e.*, relevant to obstacle avoidance) and the information is represented in 2-D image coordinates only. The control algorithms directly use observable image information represented in the 2-D image sequence; a 3-D reconstruction is not required [16]. This approach requires fewer calibrations, fewer scene hypotheses, and less computation. It is therefore simpler and faster.

Figure 1 sketches the demonstration obstacle avoidance system. The system consists of five processing modules. Video images are obtained from two on-board cameras. One of the cameras is above the other, and both are mounted on a single pan motor. Neither camera is calibrated. Central/peripheral visual sensing is achieved with narrow and wide lenses. The narrow central camera has a $40°$ field of view. It is tilted downward so that its visual axis intersects the floor 2 or 3 meters in front of the robot. The wide camera, mounted slightly below the narrow one, has a $115°$ field of view. The robot's view from these cameras is shown in Figure 2. The images from the two cam-

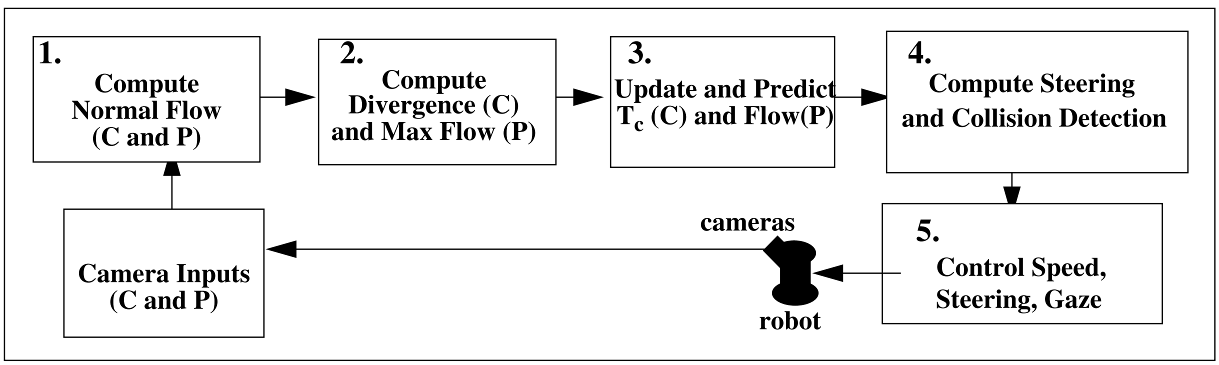

Figure 1. Obstacle Avoidance System

Central Lens: $40°$ Peripheral Lens: $115°$

Figure 2. Robot's View of the Scene

eras are processed independently, and they are independently used to control behavior. The first process in Figure 1 computes normal flow, the component of motion perpendicular to edges in the images of the two cameras. In Figure 1, “C” represents video input from the narrow-angle central camera and “P” represents video input from the wide-angle peripheral camera. The magnitude and quantized direction of the normal flow are presented to the second processing module, which computes divergence and time-to-contact in three overlapping windows of the central image. In the wide image, the process estimates maximum flow in two peripheral visual fields (left and right). Accuracy is sacrificed to achieve real-time performance. The resulting errors in normal flow estimates produce errors in divergence and time-to-contact measurements as well. The third process temporally filters maximum flow and T_c to reduce errors. Recursive estimation is used to update current flow and T_c estimates and to predict flow and T_c at the next sample time.

The fourth process uses flow to steer the robot down the conceptual corridor estimated by comparing peripheral flows in the wide camera. Using active gaze control, the cameras are rotationally stabilized so that their motion is approximately a translation. If the flow is larger on one peripheral side than the other, then objects in the scene are closer on that side, and the robot steers away. When the camera points too far away from the heading, a saccade is made toward the heading. When T_c predictions in the central camera indicate imminent collision in a “dead end” in the conceptual corridor, the robot stops, turns away, and resumes wandering. The inputs to the body and gaze controllers consist of driving, steering, and gaze velocities.

2. Real-Time Control System (RCS)

The obstacle avoidance system we describe in this paper is designed in accordance with the Real-Time Control System (RCS) hierarchical architecture described in [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 3 maps the functionality of the obstacle avoidance system into the first three levels of the RCS hierarchy.

RCS is composed of three parallel legs, sensory processing (SP), world modelling (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.

3. Computation of Normal Flow

Process 1 (Figure 1) computes the magnitude and quantized direction of normal flow in the image sequences from each of the two cameras mounted on a mobile robot, as sketched in Figure 4. These flow values are computed in real-time on PIPE[1] (Pipelined Image Processing Engine) [11].

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.

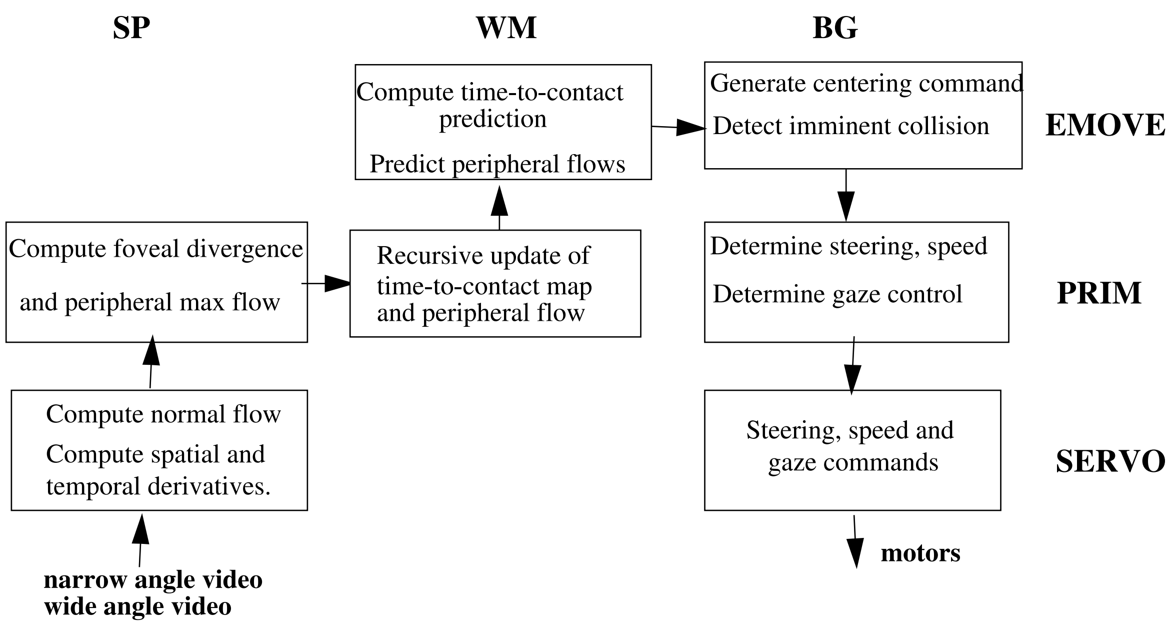

Figure 3. Obstacle Avoidance Architecture

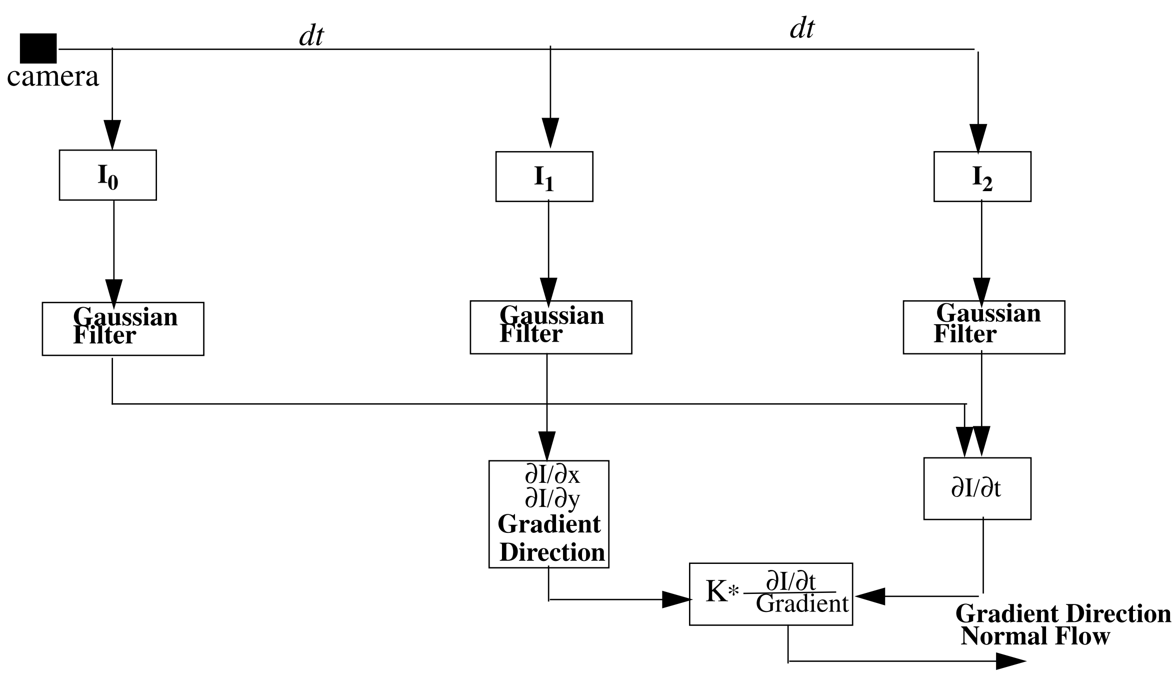

Figure 4. Sketch of Normal Flow Algorithm on PIPE

Normal flow is the component of image motion in the direction of the intensity gradient (*i.e.,* perpendicular to intensity edges). Motion information is extracted from the sequences using temporal and spatial derivatives [10]. For each camera, sequential images (video *fields* not frames) I_0, I_1, and I_2, are digitized and stored. To satisfy the continuity constraint for the gradient based flow computation, each image is filtered with a Gaussian function ($\sigma = 1$ pixel) using a 5×5 convolution kernel. The temporal derivative is computed as:

$$\frac{\partial I}{\partial t} = I_2 - I_0 \tag{1}$$

It is important that *alternate video fields* are used to estimate the temporal derivative because the interlacing of consecutive fields would distort the result. The spatial derivatives, $\frac{\partial I}{\partial x}$ and $\frac{\partial I}{\partial y}$, are computed on image I_1. For every point in I_1, intensity gradient magnitude and orientation are computed using 3×3 spatial gradient operators [17]. The gradient direction, θ, of each point in the image is

$$\theta = \operatorname{atan}\left(\frac{\partial I}{\partial y} \Big/ \frac{\partial I}{\partial x}\right), \text{ for } 0 \le \theta < 2\pi. \tag{2}$$

θ is quantized into four directions: north-south, northeast-southwest, east-west, and southeast-northwest.

The gradient magnitude ∇I is:

$$\nabla I = \sqrt{\frac{\partial I}{\partial x}^2 + \frac{\partial I}{\partial y}^2} \quad . \tag{3}$$

Best flow estimates are obtained at local extrema of ∇I, so the ∇I image is thinned by non-extrema suppression [22]. The normal flow $O_{\nabla I}$ is:

$$O_{\nabla I} = K \frac{\partial \mathrm{I} / \partial t}{\nabla I} \tag{4}$$

where K is a scaling factor that permits the result to be expressed in 8-bit integer format to meet hardware constraints. In addition, because of noise, the temporal derivative, $\frac{\partial I}{\partial t}$, is used only if above a threshold. The gradient, ∇I, is also used only if above a threshold to reduce the difference between the motion field and the corresponding optical flow [10]. K is chosen empirically to allow representation of flows arising from obstacles between 1.5 and 2.5 meters from the camera at robot speeds up to 30 cm/s. Closer objects give rise to image flows too large to represent. Extremely small flows can be noisy.

The result of this process is a histogram of the quantized gradient directions, a list of image coordinates of each quantized direction, and the corresponding scaled flow values in each direction.

4. Maximum Flow in the Visual Periphery

One of the tasks of process 2 (Figure 1) is to compute maximum flows in the left and right peripheral visual fields of the wide-angle camera. As described in Section 8, the cameras are rotationally stabilized using active gaze control so that their motion is approximately a translation. For comparing clearance to obstacles on each side of the path, the maximum flow value in each peripheral receptive field indicates the distance to the nearest object in that field [6]. The optical flow processing in each field is implemented in two parts: first, dense normal flow is estimated as described above, then the maximum magnitude flow in each receptive field is identified by examining the histogram of flows in each field. The histogram is transferred from PIPE to a host processor, where the largest magnitude flow is determined. Spurious outlying high magnitude data are avoided by ignoring a small fraction of the highest magnitude flows (*e.g.*, 4%). The maximum flows are also recursively estimated and predicted (Section 6).

It should be noted that this implementation does not attempt to compute range for objects in the image. To do so would require calibrating the focal length, the wide-angle lens distortion, and the vehicle velocity in order to compensate for the eccentricity from the heading at each flow estimate. An object near the focus of expansion (FOE) or contraction (FOC) will generate less optical flow than an object at the same range to the side of the robot's path. For these reasons, the control system is designed to keep the gaze nearly aligned with the heading (Section 8).

5. Divergence and Time-to-contact for Obstacle Detection

A second task of process 2 (Figure 1) is to compute flow divergence in the central camera. Divergence of flow can be used to estimate time-to-contact (T_c). Both theory and implementation are discussed here as well as considerations for employing T_c estimated from divergence for obstacle detection by a moving robot.

The equations for the x and y components of optical flow (O_x, O_y) due to general camera motion (arbitrary translation and rotation) in a stationary environment are

$$O_x = (1/Z)\,(-T_x + xT_z) + \left(xy\omega_x - \left(1 + x^2\right)\omega_y + y\omega_z\right) \tag{5}$$

$$O_y = (1/Z)\,(-T_y + yT_z) + \left(\left(1 + y^2\right)\omega_x - xy\omega_y - x\omega_z\right) \tag{6}$$

where Z is the depth of the object in the environment relative to the camera, (T_x, T_y, T_z) and $(\omega_x, \omega_y, \omega_z)$ are the translational and rotational motion of the environment relative to the camera[23]. The divergence of an optical flow field is defined as:

$$\nabla^\circ (O_x, O_y) = \frac{\partial O_x}{\partial x} + \frac{\partial O_y}{\partial y}. \tag{7}$$

Note that

$$\frac{\partial O_x}{\partial x} = \frac{\partial \rho}{\partial x}(-T_x + xT_z) + \rho T_z + y\omega_x - 2x\omega_y \tag{8}$$

$$\frac{\partial O_y}{\partial y} = \frac{\partial \rho}{\partial y}(-T_y + yT_z) + \rho T_z + 2y\omega_x - x\omega_y \tag{9}$$

where $\rho = 1/Z$. From equations (7) through (9), at (x, y) = (0, 0):

$$\nabla^\circ (O_x, O_y) = \frac{\partial \rho}{\partial x}(-T_x) + 2\rho T_z + -\frac{\partial \rho}{\partial y}(-T_y) \tag{10}$$

From equation (10),

$$\nabla^\circ (O_x, O_y) = 2\rho T_z \tag{11}$$

whenever the imaged surface is a mostly perpendicular surface *or* the gradient of the imaged surface is perpendicular to the transverse velocity (T_x, T_y).

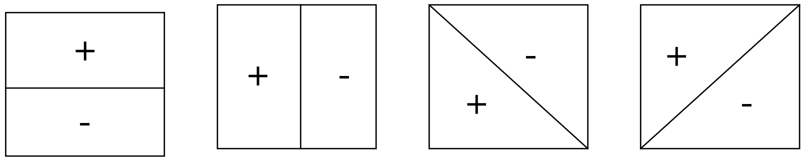

Figure 5. Flow Divergence Templates

Time-to-contact (T_c) or nearness in time, can be estimated directly from divergence (Equation (11))

$$T_c = Z/T_z = 2/\nabla^\circ(O_x, O_y) \ . \tag{12}$$

This measurement is particularly useful for obstacle avoidance during visual navigation because divergence is invariant under the rotational motion of the sensor that is inevitable due to imperfect stabilization.

Equation (12) suggests that T_c has only time as its dimension. The values of T_c over any significant area represent the time needed to reach an object at distance Z with velocity T_z in the z direction. T_c is twice the inverse of divergence. Therefore, a family of simple fixed flow divergence templates can be applied to any image sequence to estimate divergence [15]. Each template is symmetrically divided into positive and negative halves (Figure 5). Flow divergence is calculated by convolving the template with a window in the flow image and computing the sum of the image flow derivatives in perpendicular directions. Since there are few accurate flow values in the implementation, a large window is needed to accumulate multiple samples. (The estimates are inaccurate particularly because they are only normal flow and their directions are quantized.) Each time-to-contact value has an associated confidence, which is the number of points in the window that have measurable flow values. In order to improve the consistency of the estimation of T_c, we apply a recursive least squares update procedure.

6. Recursive Least Squares Update of Time-to-Contact and Maximum Flows

Process 3 (Figure 1) computes updates and predictions of time-to-contact and flow values. One of the significant contributions of our approach is its use of a temporal model to estimate and predict T_c values. A linear model is maintained for the time-to-contact in each of the three windows:

$$T_c = a_1 + a_2 t \tag{13}$$

where t = 0, -1, -2,... For each measurement of time-to-contact, model parameters a_1 and a_2 are updated by a weighted recursive least squares computation with exponential decay [5][9][13][19]. This involves determining a_1 and a_2 such that the residual J is minimized:

$$J = \sum_{t=0}^{-n} \lambda^t w_t [T_c - (a_1 + a_2 t)]^2 \tag{14}$$

where $t = 0, -1, -2, \ldots, -n$; $t = 0$ is the present; λ is the forgetting factor; and w_t is the confidence of the t^{th} T_c measurement (the number of flow data points in the window).

In order to solve for a_1, a_2, the square root information filter (SRIF) algorithm [5] is used. The SRIF provides an efficient, numerically stable, closed form solution to the least squares problem.

It is also a recursive algorithm, i.e. the model is updated as new data becomes available without having to explicitly store old data. The SRIF algorithm is described in [5][13].

Since the camera is approximately aligned with the robot's heading, the objects viewed in the estimation windows change as the robot steers away from obstacles. Recursive estimates derived over time are sensitive to the underlying data, which should ideally consist of the same sample points for the estimation period. However, the windows' sample data change constantly when the vehicle is turning, *e.g.,* old obstacles disappear from view and new obstacles become visible. The forgetting factor, λ, adjusts for this behavior by controlling the relative weighting of new and old data. λ is changed dynamically as a function of the control system's commanded angular velocity. In this way, the weight of past events is reduced when the robot turns faster and a new scene becomes visible. Similarly, past observations weigh more heavily when the robot moves straight ahead.

Statistical analysis of the recursive least squares predictions has produced a robust indicator for imminent collision. Figure 9 plots a history of time-to-contact predictions for straight line motion of the robot. The last entries in the plot are decreasing and approaching zero. However, although the general trend is downward, the progression is non-monotonic. No single time-to-contact datum provides sufficient confidence to conclude that collision is likely. There appears to be a correlation between collision, the time-to-contact value (a_1), the slope of the predicted line (a_2), and the confidence (w_0) which represents the number of normal flow pixels contributing to the current divergence estimate. A combination of short predicted time-to-contact ($a_1 < 0.07$), negative slope of T_c predictions ($a_2 < 0$), and at least a moderate number of current data ($w_0 > 20$) seem to reliably indicate imminent collision. a_1 represents a measure of time-to-contact, but because of repeated scaling operations, it is expressed only in terms of t units which are not calibrated to standard time units. We have empirically found that at least 10 T_c samples are necessary for reasonable accuracy of recursive estimates.

In our approach to filtering maximum flows, we use a temporal linear model similar to equation (14) in each receptor field. The forgetting factor, λ, is a constant which controls the degree of smoothing and the length of past history.

7. Driving Control

Processes 4 and 5 (Figure 1) are used to control driving and gaze. The robot's task is to avoid obstacles while achieving mobility goals. In general, such goals might be specified by coordinates in a map, features that uniquely identify a location, or simply features that satisfy a precondition required for the next subtask (*i.e.* the mobility goal might be positioning the robot to pick up an object.) Ideally, the robot would survey the visual data to identify the direction nearest its desired path that is also a safe direction in which to travel. In these experiments, the goal is to develop the ability to maneuver without collision using only visual image flow to sense the environment. The robot's behavioral goal is simply to drive forward, steering away from obstacles, and to stop and turn when it senses that collision is imminent.

The robot begins driving at 30 cm/s. Driving at a fixed speed allows the use of a lookup table to estimate time-to-contact. Hardware constraints on the real-time processing machine prevent the use of multiple tables which could be used for different velocities. The centering behavior uses the peripheral flows to steer the robot down its conceptual corridor. Indication of imminent collision

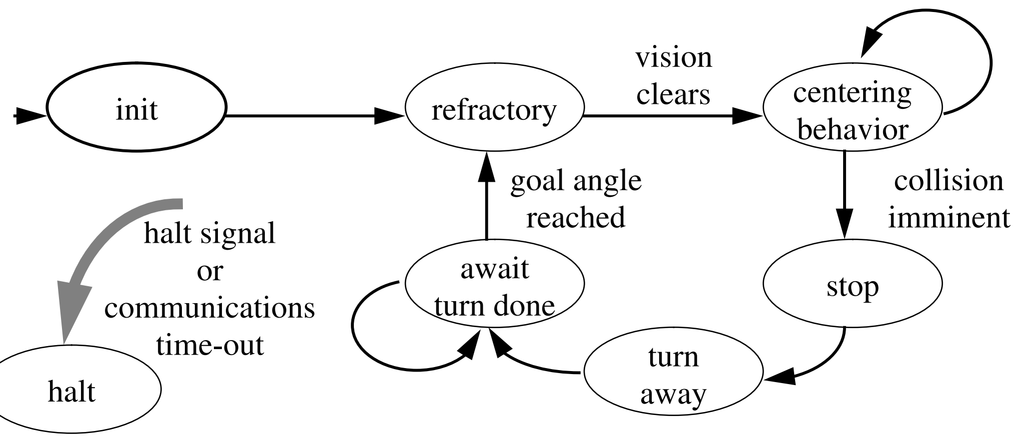

Figure 6. Body Control Automaton

(based on time-to-contact) in any of the central receptive fields causes the robot to stop, turn away and resume wandering. This is implemented with a finite state automaton, with a command associated with each state (Figure 6). Some state transitions are triggered by sensed events, and others merely provide command sequencing.

The robot steers smoothly to the new desired heading with saturated negative visual feedback controls [7]. The desired heading attempts to balance the peripheral flows by choosing a desired heading, θ_v, in retinotopic (visual) coordinates.

$$\theta_v = k_h \cdot \left(O^L_{\nabla I} - O^R_{\nabla I} \right) \tag{15}$$

$O^L_{\nabla I}$ and $O^R_{\nabla I}$ are the left and right maximal normal flows in the peripheral visual fields. The gain k_h is chosen empirically to produce a sensible desired heading in the visual space. Desired change in heading, $\Delta\theta$, is then calculated, accounting for the current gaze angle, θ_g, with respect to heading.

$$\Delta\theta = \theta_v + \theta_g \tag{16}$$

The steering control policy is simply a saturated steering velocity proportional to the desired heading.

$$\dot{\theta} = Saturate\left(k_s \cdot \Delta\theta \cdot \frac{1}{T_b}, s \right) \tag{17}$$

The gain k_s (usually <1) determines how quickly the steering is servoed to the desired heading. Time is normalized to seconds by dividing by the body control cycle time, T_b. Thus angular velocity is expressed in degrees/s rather than degrees/cycle. For instance, setting $k_s = 0.3$ will command a velocity that would reduce the error by 30% in the next control cycle (assuming nearly instantaneous acceleration). The angular velocity is saturated at $\pm s$ deg/s to limit the peak rotation rate to reasonable levels. There are three reasons for this limit. First, the latency of robot command execution is quite large, and the command cycle is not entirely uniform. Therefore it is possible to overshoot the desired heading if the rotational velocity is too high, since the controller might not be able to stop steering at the right time. Second, motion estimation suffers if the cam-

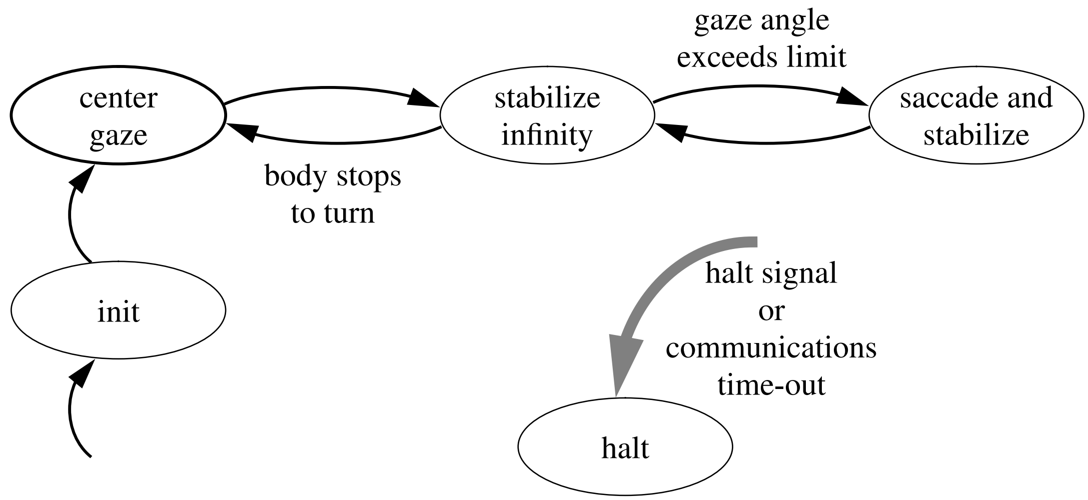

Figure 7. Gaze Control Automaton

era rotates too fast, since our computation of flow is based on the gradient method which limits motion to less than two pixels per frame. Poor flow measurements degrade T_c estimates. This impacts the quality of steering and stopping. Third, slower steering improves the temporal consistency of flow data for recursive estimation. The second and third issues would not be of concern if gaze were perfectly stabilized, but stabilization is not perfect and the residual camera rotation is correlated with the steering rate.

The robot steering and collision detection improve when the robot turns relatively slowly for two reasons: (1) temporal consistency of spatial samples, and (2) accuracy of motion estimates. Also, in order for the centering behavior to work properly, the cameras should be translating approximately along the heading. In this way, the relationship between flow and range in the two peripheral fields will be the same. Therefore, the behavior and motor control systems must minimize rotation of the cameras. This is accomplished by stabilizing the cameras with active motor commands and by limiting rotation of the body so the gaze stabilization is not overstressed. Despite these precautions, gaze stabilization is imperfect and some data are contaminated. The sensory signals must be examined to detect corrupted estimates that should not be used to control driving. For instance, poor gaze stabilization injects a smooth flow (arising from rotation) across the visual field. These flow patterns are not consistent with pure forward translation of the camera. This condition can be detected, and such data can safely be ignored for short periods.

8. Gaze Control

The nonlinear gaze control is a *nystagmus*, a repetitive eye motion of slow phase rotations punctuated by quick phase rapid returns. It is also implemented as a finite state automaton (Figure 7). The camera is rotated at velocity $\dot{\phi} = -\dot{\theta}$ to counter the body rotation and stabilize the camera images. The gaze control also checks the deviation of the gaze angle, θ_g, from the robot's heading and snaps the camera back to the heading if the limit

(a) (b) (c)

Figure 8. Sequence of Images As Robot Approaches Obstacles

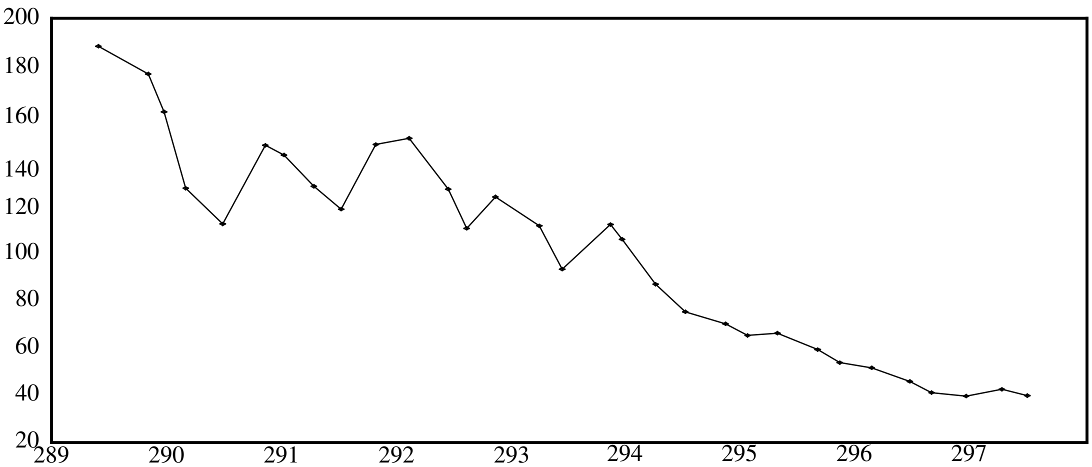

Figure 9. Time-to-contact versus Time

is exceeded. With the cameras rotationally stabilized near the heading, the peripheral flow estimates can be interpreted as relative distances to obstacles.

9. Experiments and Results

Experiments with the obstacle avoidance system were performed in a laboratory containing office furniture and robot and computing equipment. Furniture and equipment lined the walls and there was free space roughly 5 m by 3 m in the center of the lab. Office chairs provided obstacles.

In typical experiments, the robot began the trial at one end of the lab. Obstacles were placed in the camera's view (Figure 8a). The camera has a $40°$ field of view. The robot drove forward at 30 cm/s. As images were processed, time-to-contact was estimated. Figure 8 shows a sequence of images from the robot's viewpoint as it approached a stack of chairs. Figure 8b was captured midway through the experiment, and Figure 8c shows the view when the robot detected imminent collision. Figure 9 plots the T_c recorded during

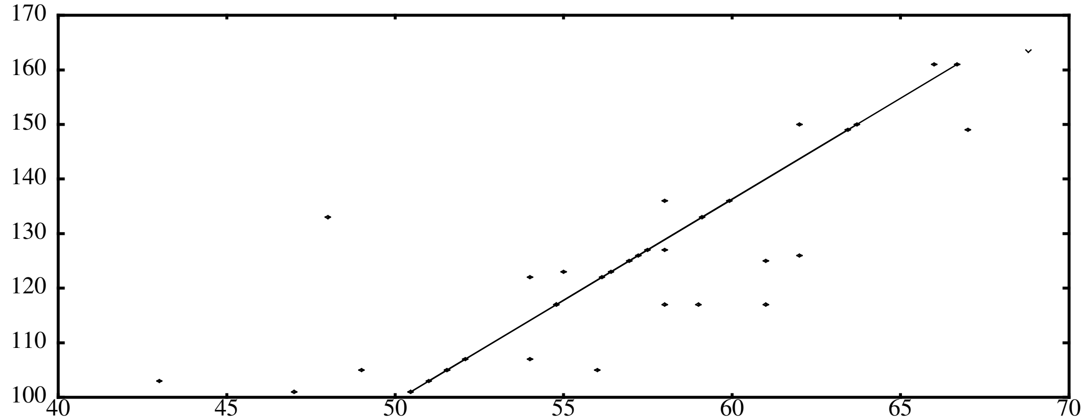

Figure 10. Range (cm) versus Time-to-Contact

such a trial against time, represented by sampling numbers. The time-to-contact dimension is not calibrated.

The collision trend seems apparent to our eyes by $t = 292$ since the general slope of the line seems to be approaching zero. Our real-time image processing system can accurately compute and represent a narrow range of normal flow values. Driving at 30 cm/s, normal flow can be estimated arising from objects located between 1.5 m and 2.5 m ahead of the camera. This range corresponds to time values from 292 to 298 in the plot starting when the robot is approximately 2.5 m from the obstacle.

Figure 10 is a graph of range (in cm on the y axis) versus time-to-contact (in arbitrary time units on the x axis) estimated at the time the robot stopped to avoid collision in each of 20 trials. In this experiment, centering behavior was deactivated and the robot was programmed to stop (rather than turn) when it detected an obstacle. The distance between the robot and the obstacle was physically measured.The corresponding time-to-contact value represents the last value predicted by the system before the robot was commanded to halt. The graphed points plot raw data; the line represents a least square linear fit of measured range versus time-to-contact value. The stopping distance varied between 101 cm. and 150 cm. and the time-to-contact values varied between 43 and 66. Ideally, we would expect the computed time-to-contact values and the measured values to be linear. The standard deviation of this fit is 4.75 cm.

A typical trace of the robot's path using obstacle avoidance and wandering behaviors is plotted in Figure 11. The lab was lined with chairs forming a free space roughly 5 m by 3 m. As the dead-reckoned trace shows, the robot traversed most of the space in these five minutes of recorded data. Variability in steering and stopping depends in part on the textures visible from a particular approach. Stopping distances may vary just because the angle of approach varies. In an average run, robot motion starts at any open area of the lab. As time-to-contact and peripheral flows are computed, the robot moves forward toward open areas while centering itself in the open space. Upon detection of an imminent collision, it turns to avoid the obstacle and continues its wandering behavior. The system has run successfully for up to 20 minutes.

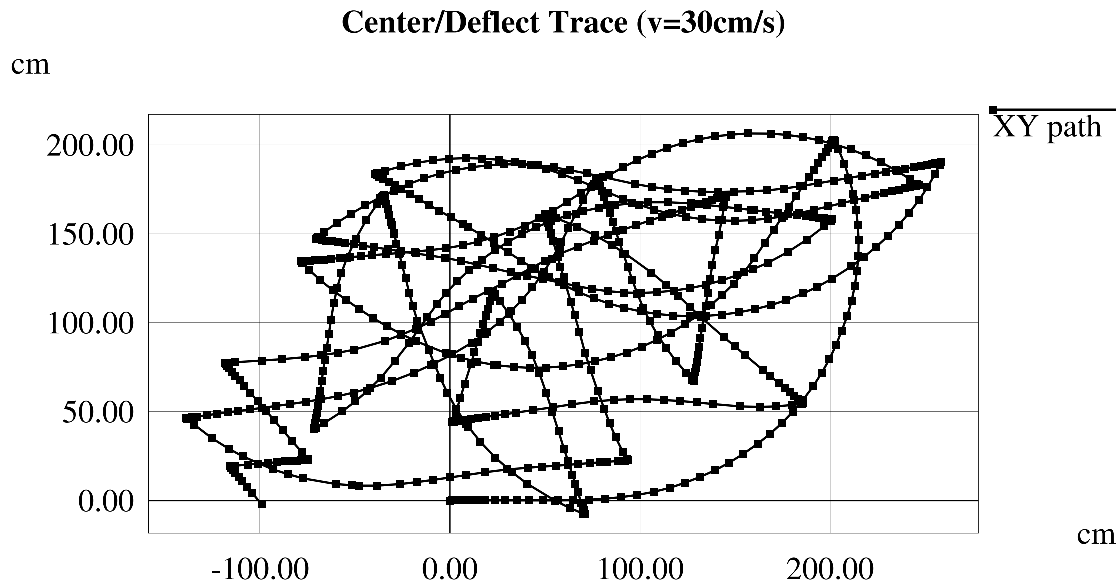

Figure 11. Robot's path

There are three primary factors that lead to system failure. First, there is a variable time delay between detection of an event and the behavioral response to this event. The variability is a function of inter-communication delays caused by processes (eye and robot controls) running in a Unix environment. The effect of this variability can delay the robot from reacting quickly enough to avoid an obstacle. Second, the flow computation assumes that there will be sufficient texture in the field of view from which to compute flow. If this is not the case, *e.g.*, the robot is steering toward a blank wall, no flow is detected and the robot collides with the wall. The third cause of failure is the rectangular geometry of the robot base and the presence of bumpers which trigger stopping upon contact. There are instances where the robot is turning to avoid an obstacle, and in so doing, brushes its bumper against a close-by object. This is not a short-coming of the system per say, but nevertheless, results in a system shut-down.

10. Discussion

Some researchers [12][20][21] have proposed using divergence or flow derivatives for visual cues, but they do not provide real-time implementations of these ideas. Nelson and Aloimonos [14][15] used directional flow divergence for stop-and-look obstacle avoidance (not real-time smooth driving). Also, their system did not simultaneously handle forward and lateral obstacle avoidance, under both translational and rotational robot motion. Finally, their environments were much simpler than ours and they did not demonstrate extensive robust behavior over extended periods of time.

Corridor following behaviors using wide field peripheral flow have been demonstrated with [6] and without [18] gaze stabilization. The choice depends on several factors. As noted, a rotationally stabilized visual frame may improve temporal integration of data. In addition, image flows may be interpreted directly as proximity estimates because the camera is approximately translating. In principle, steering based on differences of peripheral flows should be independent of rotational flow components due to egorotation. Assuming egomotion in a static environment, the rotational flow components would simply cancel. In practice, however, most flow estimators have limited spatiotemporal range, and rapid camera rotation could easily dominate the image flows observed by a freely rotating observer. This would significantly degrade the resulting steering of a system using flow estimators with limited dynamic range, as it does in this demonstration system.

All these systems suffered from a gaping central blind spot. Our work combines central and peripheral vision to avoid collision over the entire field for vision-based mobility.

System performance depends on many factors. Underlying the instantaneous T_c estimates are image flow measurements. Although divergence is theoretically unaffected by camera rotation, rotation contributes directly to image flow. The system calculates image flow using the gradient method, which, like all techniques, has limited spatiotemporal sensitivity. In particular, large flows are not estimated accurately, so fast camera rotation can corrupt the image flow estimates on which the divergence and T_c estimates rely.

In addition, image flow depends on the field of view and scene texture.The obstacle must appear in the field of view in order to be detected. A logical extension would be construction of a local map of space, though this introduces new problems. Also, since the normal flow computation performs best on textured scenes, the system fails in the absence of texture. Further, the real-time estimation of image flow on PIPE suffers from limited resolution in flow estimates. The translational components of the flow which indicate the scene structure are very small near the heading direction, where T_c is directly related to divergence. Therefore T_c estimates are noisy, and it becomes necessary to detect trends in the data since we cannot base behavioral decisions on any single datum.

Sensorimotor interactions affect overall performance. There are significant latencies in the sensing, estimation, and control modules. These modules are distributed and to some extent asynchronous. Although in general the latency equals the cycle time, in some cases the latency is a bit greater. The modules produce and consume data at various rates, and the interactions of the unequal cycle times have considerable consequences. Normal flow estimates are produced at 7.5 Hz. In parallel, the divergence estimation task runs at a mean rate of 7 Hz, though the rate varies with the density of the normal flow data. On the same processor, the recursive estimator executes in 12 ms. The robot accepts speed and steering commands at about 3 Hz. Hence, at a robot velocity of 30 cm/s, visual data become available about every 5 cm of robot travel and steering is adjusted about every 10cm. 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. This minimizes the overall latency in the system. However, it also means the behavior controller does not know exactly which command is being executed unless the robot controller sends an acknowledgment that identifies the accepted command. This condition, coupled with considerable latencies in actually effecting a robot command, means it is difficult for the control (and sensing) systems to know precisely what the robot is doing, short of installing high-speed low-latency feedback sensors. Consequently, the systems are designed to require only approximate knowledge of the robot's current motion state if they use any at all.

11. Conclusions and Future Plans

A demonstrated system is presented that uses only real-time motion cues to wander while avoiding obstacles in a lab containing office furniture and robot and computing equipment. The robot has wandered around the lab at 30 cm/s for as long as 20 minutes without collision. To our knowledge, this is the first such demonstration of real-time wandering using only image motion cues.

The paper describes how flow, divergence of flow, and maximal flows are computed in real-time to provide the robot's sense of space, and how steering, collision detection, and camera gaze control together accomplish safe wandering. The major contribution of this work is in developing and demonstrating this integrated system that robustly coordinates real-time behaviors in a complex environment.

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 always been hampered by problems such as noise, brittleness, and computational complexity. We demonstrate for the first time that robust image motion cues can be extracted and used as the sole perceptual cues to safely move about a complex environment in real-time for extended periods.

This work includes the integration of several visual mobility behaviors, including forward obstacle avoidance, lateral obstacle avoidance, and camera gaze control. It also demonstrates the integration of foveal/peripheral visual processing, image motion processing, retinal 2-D representations and behavior control. The manner in which this integration is achieved is unique and represents an important contribution in integration methods. The approach requires fewer calibrations, fewer scene hypotheses, and less computation than standard techniques. It is therefore simpler and faster.

Future work will benefit from a number of additions or modifications to our system. Among these factors is the need for a wider visual field of view. Motion planning for smooth trajectories will benefit from having as much visual information from the scene as possible. Faster update rates and reduced lag time are also necessary for more robust performance.

We plan to install all computing capabilities on-board the robot. This will allow experimentation in larger areas. We also plan to explore more robust real-time flow algorithms in order to overcome the problems caused by limited numerical precision in our present system. The current robot behavior will be extended to include the ability to go to a goal position while avoiding obstacles.

12. References

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

[2] J. Aloimonos, I. Weiss, and A. Bandyopadhyay, "Active Vision," *International Journal of Computer Vision* 1: 333-356, 1988.

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

[4] D. Ballard and C. Brown, Principles of Animate Vision, *CVGIP: Image Understanding*, 56(1):3-21, 1992.

[5] G. Bierman. *Factorization Methods for Discrete Sequential Estimation*. Academic Press. New York, 1977.

[6] D. Coombs and K. Roberts. "Centering behavior using peripheral vision", In *Proc. of CVPR 1993, the IEEE Conference on Computer Vision and Pattern Recognition*, New York, June 15-17, 1993.

[7] R. Dorf. *Modern Control Systems,* Addison-Wesley, 1980.

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

[9] G. Goodwin and K. Sin. *Adaptive Filtering, Prediction and Control.* Prentice-Hall. Englewood Cliffs, New Jersey, 1984.

[10] B.K. Horn and B. Schunck. "Determining Optical Flow", *Artificial Intelligence Journal*, 17, 1981.

[11] E. Kent and M. Shneier and R. Lumia. "PIPE (Pipelined Image-Processing Engine)", *Journal of Parallel and Distributed Computing*, 2:50-78, 1985

[12] J. Koenderink and A. van Doorn., "Optic Flow", *Vision Research*, 26(1):161-180, 1986.

[13] C. Lawson and R. Hanson. *Solving Least Squares Problems*. Prentice-Hall. Englewood Cliffs, New Jersey, 1974.

[14] R. Nelson and Y. Aloimonos. "Using Flow Field Divergence for Obstacle Avoidance in Visual Navigation", In *Proc. DARPA Image Understanding Workshop*, pp. 548-567, April 1988.

[15] R. Nelson and Y. Aloimonos. "Obstacle Avoidance Using Flow Field Divergence", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(10):1102-1106, October 1989.

[16] D. Raviv and M. Herman, "Visual Servoing from 2-D Image Cues," In *Active Perception*, Y. Aloimonos, ed., Lawrence Erlbaum Associates, Hillsdale, NJ, pp. 191-226, 1993.

[17] A. Rosenfeld and A. Kak. *Digital Picture Processing*, Vol. 1, Second Edition, Academic Press, 1982.

[18] J. Santos-Victor, G. Sandini, F. Curotto, and S. Garibaldi. "Divergent stereo for robot navigation: Learning from bees", *Proceedings, CVPR 1993, IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (New York, NY, June 15-17) pp. 434-439, 1993.

[19] H. Schneiderman and M. Nashman. "Visual Tracking for Autonomous Driving", *IEEE Transactions on Robotics and Automation, special issue on Perception Based Real-World Navigation*, in press.

[20] W.B. Thompson and J.K. Kearney, "Inexact vision," *Proceedings of Workshop on Motion: Representation and Analysis* (Charleston, SC, May 7-9, 1986), pp. 15-21.

[21] M. Tistarelli and G. Sandini, "On the advantages of Polar and Log-polar Mapping for Direct Estimation of Time-to-Impact from Optical Flow," *IEEE-PAMI*, April, 1993.

[22] A. Verri and T. Poggio. "Against quantitative optical flow", In *Proc. First International Conference on Computer Vision*, London, England, pp. 171-180, June 8-11, 1987.

[23] G.S. Young, T.H. Hong, M. Herman, and J.C.S. Yang, "New visual invariants for obstacle detection using optical flow induced from general motion," *Proc. IEEE Workshop on Applications of Computer Vision*, Palm Springs, CA, November 30-December 2, 1992, pp. 100-109.

Acknowledgments

The authors thank Henry Schneiderman for his help with the recursive estimation.

A1. Implementation Details

Our development environment includes a Labmate mobile robot, a Sun SPARC2 workstation, a Pipelined Image Processing Engine (PIPE), and a VME-based multiprocessor system. Figure 11 shows the breakdown of processing across hardware. In this figure, the large gray rectangles represent distinct software modules. Each of these modules is labelled by its functionality (SP = sensory processing, WM = world modeling, BG = behavior generation) and level within RCS. A complete control system architecture proposed for intelligent machines is found in [1].

Video input is read into PIPE from CCD cameras mounted on the mobile robot body. The incoming images are digitized to provide 8-bit gray-scale images that are 242x256 pixels in size. Normal flow is extracted from the image sequences. In addition, the quantized flow orientation values are computed and passed to the Iconic-to-Symbolic Mapper

(ISMAP) stage of PIPE. ISMAP converts this information from an image format to a symbolic list and stores the quantized orientation data as a list of pixel positions. The corresponding flow values are stored in the ISMAP iconic buffer where they are mapped onto the memory of one of the microprocessors via a specialized PIPE-VME interface board. The normal flow extraction and symbolic mapping operations are indicated by black parallelograms in Figure 12. For additional information on PIPE, see [3].

The remaining processing is divided among microprocessors in the VME backplane. Most computations are pipelined and are implemented on designated processors. Interprocessor communication is done through semaphored global memory. For a detailed description of our software engineering practices refer to [8]. Program development for the VME-based multiprocessor system is done on a Sun SPARC2 workstation. Code for the PIPE communication process and the least squares recursive filtering process is written in the Ada programming language. Program development for PIPE is done on a personal computer using the PIPE graphical programming language, ASPIPE [3]. The remaining processes are written in the C programming language and, except for the motion controller and eye controller processes which run on the SPARCstation 2, operate in a VxWorks operating system environment.

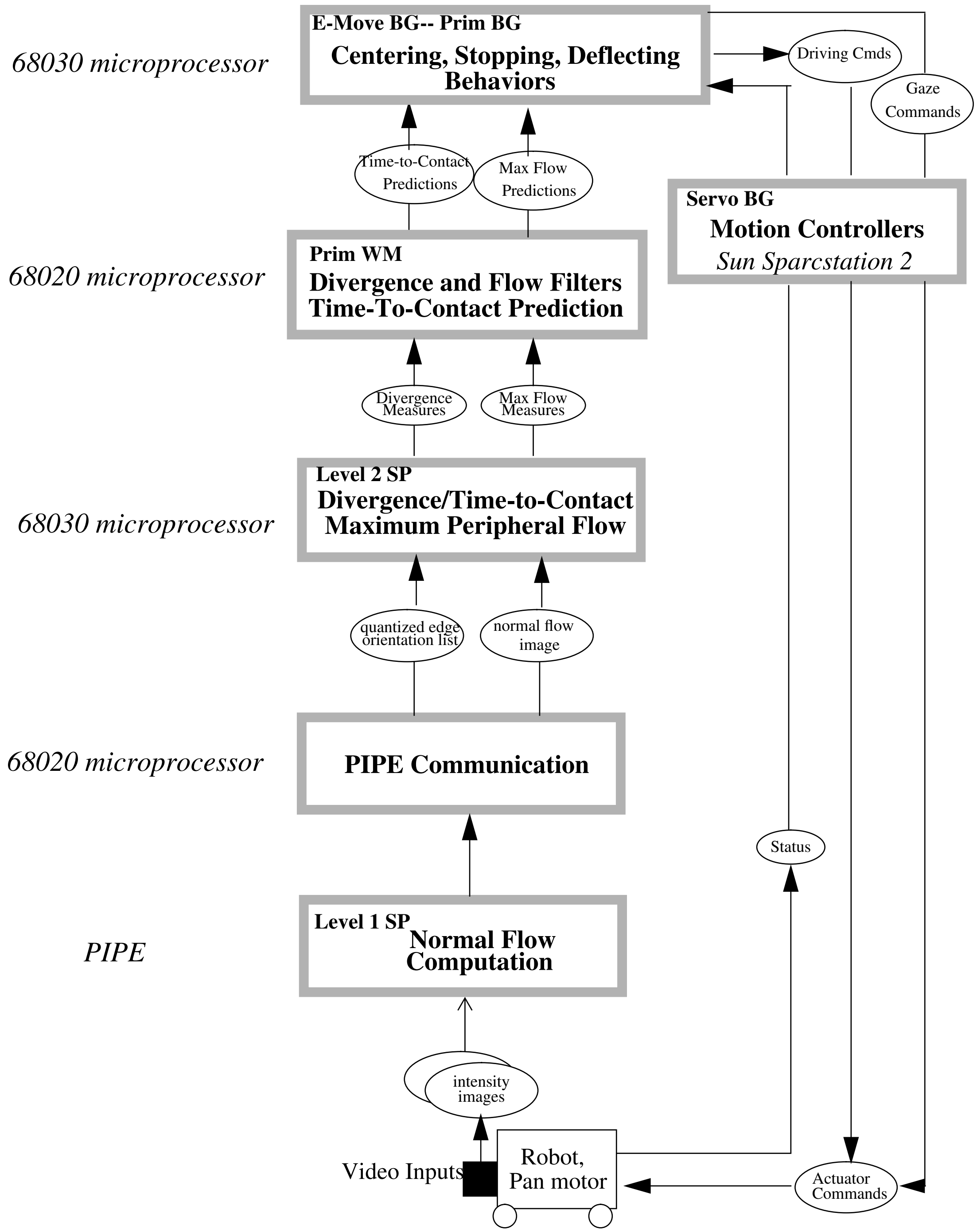

Figure 12. Distribution of Hardware in Processing