Real-Time Imaging **3**, 213–228 (1997)

Optical Normal Flow Estimation on Log-polar Images. A Solution for Real-Time Binocular Vision

This paper is concerned with a computational solution for normal optical flow estimation using space-variant image sampling. The article describes one solution for the problem based on log-polar images, including the algorithm implementation on digital signal processing hardware. The log-polar structure achieves high data compressing rates by maintaining a high resolution area—the fovea—and a space-variant resolution—the retinal periphery. This sampling scheme enables the implementation of selective processing, characteristic of attentive vision. The solution was developed to be used with an active vision system to pursue moving objects, assuming that the cameras could also move. Some properties of the log-polar structure are also described and the main results of the implementation are presented.

Jorge Dias, Helder Araújo, Carlos Paredes and Jorge Batista

Instituto de Sistemas e Robótica-Coimbra Pole,
Departamento de Engenharia Electrotécnica,
Universidade de Coimbra, Largo Marquês de Pombal, 3000 Coimbra, Portugal

Introduction

The development of computational solutions for simulation of visual behavior deals with the problem of integration and co-operation between different computational processes for control of active vision systems. In general these computational processes could emulate different and specific artificial visual behavior and their integration presents distinct facets and problems. The first is the interaction and co-operation between different control systems, and the second is the use of common feedback information provided by images captured.

One example is when a human observer has to accomplish a navigational task such as driving a car. In that case the entire field of view is used to estimate the 3D motion and to control the vehicle. However, it is known that the resolution of the human eye decreases towards the periphery of the retina. This fact does not prevent the observer from detecting alarming events and from exploiting the image periphery to sense and estimate the ego-motion. If, during the observer's navigational task, another vehicle (or object) is approaching, an estimate of the relative motion is calculated, including the direction of a possible impact, in order to enable an appropriate reaction. During that phase the observer directs his gaze towards the moving vehicle (or object) and pursues it for a short time interval. From this scenario it is evident that both motion detection and estimation are important for artificial simulation of a navigational task. This simulation is more complete if it is accomplished with space-variant resolution.

1077-2014/97/030213 + 16 $25.00/ri960053

In this article we address this problem by proposing an algorithm for the estimation of optical normal flow on log-polar images. This article describes the advantages of this sampling scheme, its properties and a fast solution for normal flow estimation using this type of space-variant data structure.

Fast algorithms for optical flow estimation are crucial to implement solutions for visual tracking or problems related with it. These include surveillance, automated guidance systems and robot manipulation [1–5].

Papanikolopoulos proposed a tracking process by using a camera mounted on a manipulator for tracking objects with a trajectory parallel to the image plane [1]. The information supplied by the vision system is processed by an optical flow based algorithm that is accurate enough to track the objects in real-time. The efficient use of windows in the image improves the performance of the method. Allen also proposed a similar control process for tracking moving objects, but extended to 3D [2].

The concept of visual behavior and active/animate vision and the work around this subject originated from several publications that relate concepts from robotics and also from computer vision. Coombs studied the real-time implementation of a *gaze holding* process using binocular vision [3]. The system used a binocular system mounted on a manipulator and Coombs studied different control schemes for visual *gaze holding* using the information from two images. These studies included predictive control schemes to avoid delays in the control system [3,6]. The work of Coombs [7] was one of the first to show real-time performance with an active vision system. However, this solution has some constraints regarding the type of targets that can be pursued. These constraints are related with the zero-disparity filtering used to extract the target.

The problem of integration of different visual processes has also been studied by Ballard, and in Grosso *et al*. [4] the authors proposed an approach based on multiple independent processes and on different vision strategies, to control different degrees of freedom of an eye-head system supported by a manipulator. In that system, visual fixation was an essential feature to control the system, but the performance of the system in achieving that state is not reported.

The co-operation between visual processes is also important and Pahlavan [8,9] considered the co-operation between focusing and vergence for dynamic fixation. More recently Uhlin [10] proposed a system based on the concept of an open architecture, where the integration of different modules is used to fixate and pursue a moving target. The work describes the architecture of the system and, as an open architecture, integrates different techniques proposed by other authors. In that work special and expensive hardware is used to perform the task.

There are also other results reporting the use of vision information in *real-time* processes For example, Burt reports a real-time feature detection algorithm based on hierarchical pyramid data structures for surveillance and robotics [11]. These hierarchical data structures have been used by Pahlavan [8] and Uhlin [10] in experiments for real-time tracking with an artificial vision system. Also, Murray developed a monocular active system for real-time pursuit [12]. This implementation relies on the extraction and tracking of corners, but its performance on very patterned or textured environments is not reported.

More recently other image data structures, based on the log-polar mapping, have been proposed for active vision vergence control [13,14]. Our work follows the same line, and we believe that log-polar sampling, by its properties, is a good alternative for the image data structure providing good mechanisms for *real-time* processing when compared with pyramidal structures applied to regular sampled images.

Searching for a Computational Solution for Active Vision

The visual *fixation* is a state of our visual system where the object of our attention is projected on to the central part of the human eye (fovea). Fixation is a consequence of the control of our visual system to achieve that state. The fixation state is part of more elaborate visual behaviors such as gaze-changing behaviors and gaze-holding behaviors [15,16].

The *gaze-changing* behavior includes behavior for *pursuit* in which the eye is moved smoothly to track moving targets, and *saccades*, which abruptly shifts the gaze from one target to another. The two types of behavior are distinct. The pursuit movements, in general, attempt to match the target velocity, independently of the target's position. The saccadic movements ignore target velocity and attempt to match target position without processing the images during the trajectory of the eyes. Vergence movement is a third type of gaze-changing behavior. If the objects are to be viewed with both eyes simultaneously, their related angles must be changed to achieve the fixation state. These angles

depend on the distance and orientation of the object relative to our visual system. This is accomplished by a distinct slow eye movement behavior named vergence. The movement is driven by the retinal disparity vergence or by changes in the distance where the eyes are focused (accommodative vergence) [15].

Gaze-holding behavior uses movements to maintain the projection of the environment in the retina stationary, despite our locomotion or head's movements. Two types of behavior, named reflexes, are used to stabilize the images [15]. The optokinetic and vestibulo-ocular reflexes stabilize the eyes and the vestibulocollic reflex stabilizes the head.

In the gaze-changing behavior the fixation state is an integrated part of the process. That feature is essential for visual sensing and, from the artificial systems point of view, it is important to define the computational approach to achieve that state. That could be part of the solution for more elaborate processes such as body orientation [4], vergence control [8], pursuit simulation with mobile robots [17] or gaze-holding [3]. Computationally, these types of behavior can be defined as processes and these processes can be described graphically with states and transitions between states. One example of such a description is given in Figure 1. The figure represents the process diagram of a system that simulates pursuit using an artificial system. The *target* of interest is moving relative to the world in front of the cameras and the vision system must perform movements to hold the *target* in the image center. Two types of movements, known as *pursuit* and *vergence*, are used. These movements are the consequence of the control performed by the process that includes the *fixation* of the visual system on the object. Previously, the system executes a saccadic movement and stabilizes the target in the image center (fixation stabilization). After achieving fixation the system changes to the state named pursuit. Figure 2 represents the geometrical model of the active vision system. We assume that the pan angle for each camera can be controlled separately through the angles θ_l and θ_r and their tilt by using the ψ angle. The angles θ_l, θ_r and ψ can be used as variables to control the active vision system during a process of visual fixation. These are the basic variables necessary to control in order to perform the simulation of this visual task. If the active vision system exhibits redundancy on some degrees of freedom, the correspondent variables can be related in order to achieve the goal (see [17] for an example).

Many advantages can be achieved from this *fixation* process, where the selected *target* is always in the image center (the foveal region in mammals). This avoids the segmentation of all the image to select the *target* and allows the use of relative coordinate systems, which simplifies the spatial description of the *target* itself (relationship between the observer reference system and the object reference system). Another advantage is that the computational effort is concentrated in the image center to perform the correct execution of the process. This advantage suggested a different type of structure for the sensor used in image sampling. Examples of sensors based on these *retina like* structures are described by Manzotti *et al.* [13], Sandini *et al.* [18] and Schwartz [19]. Fortunately, this structure can be simulated by re-mapping the image array using a log-polar transform and look-up tables simulating this sampling.

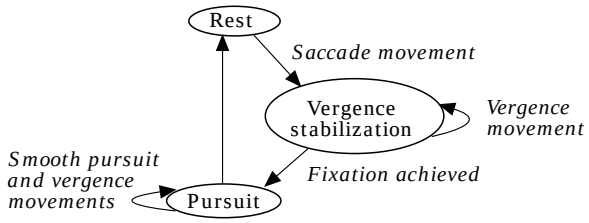

Figure 1. State diagram of the *pursuit* process.

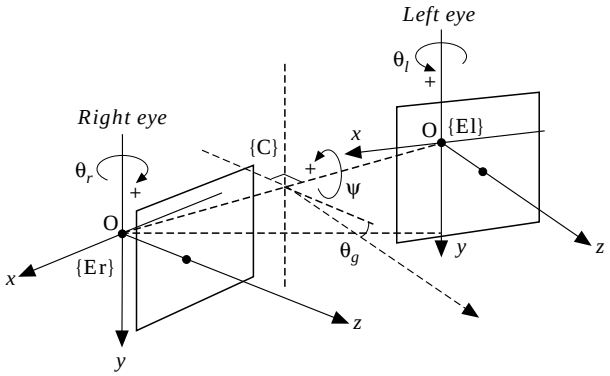

Figure 2. The geometric model used to represent the active vision system. The angles θ_l and θ_r control the vergence of each camera and can be independently controlled. Angle θ_g is responsible for the *gaze* of the system and can also be controlled independently. The angle ψ is responsible for the cameras' tilt. We assume that the system can, by calibration, be adjusted to align the different degrees of freedom.

Space-variant Image Sampling for Active Vision

The implementation of visual behaviors in artificial systems is strongly related to the performance of the image analysis algorithms used. To achieve a good performance it is necessary to develop fast algorithms for image processing and control, but exhibiting stability and robustness, to fit

the goals of the vision system activity. This could include the integration of the set of different algorithms running in the computational structure used in the active vision system and their adjustment to the geometry of the mechanical system.

The performance of these algorithms has a significant impact on the system performance and sampling period. The sampling period can affect the stability of the system and how rapidly disturbances are responded to. So it is always desirable to develop fast algorithms and efficient data structures to achieve the goal of a short sampling period. This section describes algorithms for real-time image processing operating on image data structures based on the principles of log-polar transformation. The properties and potentialities of these data structures for real-time image processing are described in this section.

One interesting feature of the human visual system is the topological transformation [18,19] of the retinal image into its cortical projection. In our own human vision system, as well as in those of other animals, it has been found that the excitation of the cortex can be approximated by a log-polar mapping of the eye's retinal image. In other words, the real world projected in the retinas of our eyes is reconfigured onto the cortex by a process similar to log-polar mapping before it is examined by our brain [19].

In the human visual system the cortical mapping is performed through a space-variant sampling strategy, with the sampling period increasing almost linearly with the distance from the fovea. Within the fovea the sampling period becomes almost constant. This retino-cortical mapping can be described through a transformation from the *retinal plane* (ρ, θ) onto the *cortical plane* ($\log(\rho)$, θ) as shown in Figure 3. This transformation presents some interesting properties as scale and rotation invariance about the origin in the Cartesian plane which are represented by shifts parallel to real and imaginary axes, respectively. This transformation is applied just on the *non-foveal* part of a retinal image. If (x, y) are Cartesian coordinates and (ρ, θ) are the polar coordinates, by denoting as $z = x + jy = \rho e^{j\theta}$ a point in the complex plane, the complex logarithmic (or log-polar) mapping is

$$w = ln(z), \tag{1}$$

for every $z \neq 0$ where $Real(w) = ln(\rho)$ and $Im(w) = \theta + 2k\pi$. However, we constrain angle θ to the range of $(0, 2\pi)$. This logarithmic mapping is a known conformal mapping preserving the angle of intersection of two curves.

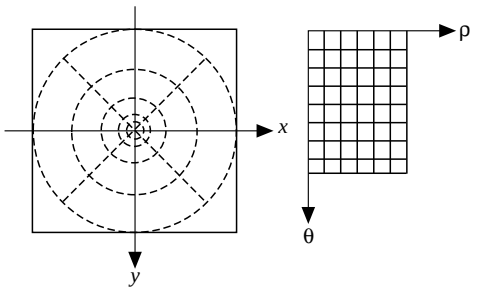

Figure 3. Log-polar transformation. Any point (x_i, y_i) in the image plane (left) can be expressed in terms of (ρ, θ) in the cortical plane (right), by ($\ln_b(\rho)$, θ).

Log-polar Mapping and its Properties

Log-polar mapping can be performed from regular image sensors by using different types of space-variant sampling structures [20–24]. For example, the mapping described in Massone *et al*. [20] is characterized by a linear relationship between the sampling period and the eccentricity, defined as the distance from the image center.

The space-variant sampling structure described by Massone *et al*. [20] can be modified into a more simplified sampling structure, as is illustrated in Figure 4.

This spatial variant geometry of the sampling points is also obtained through a regular tessellation and a sampling grid formed by concentric circles with N_{ang} samples over each circle. The number of samples for each circle is always constant and they differ by the arc

$$\frac{2\pi}{N_{ang}} \tag{2}$$

between samples.

For a given N_{ang}, the radius ρ_r of the circle i is expressed by an equation of the type

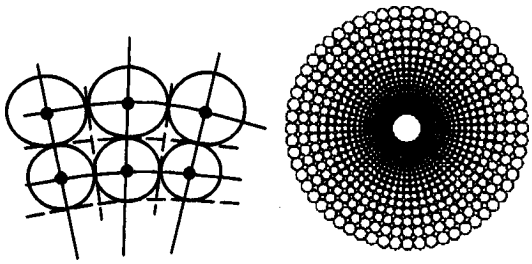

Figure 4. Graphical representation of a simpler sampling structure. The figure represents a structure with N_{ang} = 60 angular samples.

$$\rho_{r_i} = \rho_{fovea} b^i \tag{3}$$

with $i = 0 \ldots N_{circ}$ and $\rho_{fovea} > 0$ representing the minimum radius of the sampling circles.

In this case the radius basis b is expressed by

$$b = \frac{N_{ang} + \pi}{N_{ang} - \pi} \tag{4}$$

with $i = 0 \ldots N_{circ}$ and ρ_{fovea} the minimum radius of the sampling circles.

The concentric circles are sampled with the period described by expression (2) and each sample covers a patch of the image corresponding to a circle with a radius given by

$$T_{radius} = \frac{\pi \rho_{ri}}{N_{ang}}. \tag{5}$$

The value ρ_{fovea} could be chosen equal to the minimum sampling period to cover all the image center without generating oversampling in the retinal plane. This constraint is expressed by

$$\rho_{fovea} \geq \frac{N_{ang}}{\pi}. \tag{6}$$

The intensity value at each point of the *cortical plane* is obtained by the mean of the intensity values inside the circle centered at the sampling point (ρ_{ri}, θ_{ri}). Therefore this model is similar to the models described by Spiegel *et al*. [22], Weiman [21,24] and Wallace *et al*. [23].

This simplified version of space-variant structure needs less storage than the one described by Massone *et al*. [20], as we can verify in Table 1.

Table 1. Different sampling schemes also require different storage in memory.

	Reg. Sampling (512×512)	Log-polar ($N_{ang} = 60$)
Massone *et al*. [20]	262144 samples	6540 samples
$b = \frac{N_{ang} + \pi}{N_{ang} - \pi}$	262144 samples	1860 samples

Log-polar Properties

The log-polar mapping has a number of important properties that make it useful as a sampling structure. The mapping of two regular patterns as shown in Figure 5 results in similarly regular patterns in the other domain. From Figure 5(a) the concentric circles in the image plane become vertical lines in the *cortical* plane. A single circle maps to a single vertical line, since the constant radius r at all angles θ of the circle gives a constant ρ_c coordinate for all θ_c coordinates. Similarly an image of radial lines which have constant angle but variable radius results in a map of horizontal lines.

These mapping characteristics are fundamental for some properties such as rotation and scaling invariance. Rotation and scaling result in shifts along the θ_c and ρ_c axes, respectively. For rotation invariance it should be noted that all possible angular orientations of a point at a given radius will map to the same vertical line. Thus, if an object is rotated around the origin between successive images, this will result only in a vertical displacement of the mapped image. This same result is valid for radial lines. As a radial line rotates about the origin, its entire horizontal line mapping moves only vertically. Also, from Figure 5 we can conclude that as a point moves out from the origin along a radial line, its mapping stays on the same horizontal line moving from the left to the right. The mappings of the concentric circles remain vertical lines and only move horizontally as the circles change in size. These properties were fundamental for the development of algorithms for pattern recognition [20,25]. If we model image formation as perspective projection, scaling invariance implies that changes in focal length (or in the distance along the optical axis) will result in shifts along the ρ_c axis.

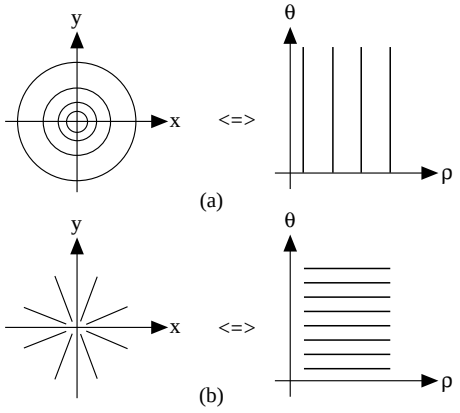

Figure 5. The log-polar mapping applied to regular patterns. (a) Concentric circles in the *image* plane are mapped into vertical lines in the *cortical* plane. (b) Radial lines in the *image* plane are mapped into horizontal lines in the *cortical* plane.

Normal Optical Flow on Log-polar

The space-variant resolution and sampling exhibits interesting properties for the optical flow. At this point we study some of these properties of the optical flow and an algorithm for normal flow detection is described.

Consider the monocular imaging situation where the observer and the scene are in motion relative to each other. In order to obtain the equations relating the scene to the image measurements in a general form we define two coordinate systems. The coordinate system {**CAM**} is fixed to the observer and used as reference and another co-ordinate system {**W**} is fixed at a point in the scene.

If this motion is rigid then it can be described by a translational velocity $\mathbf{v}_t = (v_x, v_y, v_z)^T$ and angular velocity $\omega = (\omega_x, \omega_y, \omega_z)^T$ relative to a stationary environment. Each scene point described by $\mathbf{P} = (X, Y, Z)$ measured with respect to a coordinate system *OXYZ* fixed to the camera, moves relative to the sensor's frame with velocity **V**, where

$$\mathbf{V} = -\mathbf{v}_t - \omega \times \mathbf{P}. \tag{7}$$

The 2D motion on an imaging surface is the projection of this 3D motion field of the scene points moving relative to that surface. This is equivalent to projecting the 3D motion vectors on an image surface of a given shape, generating an image of the motion field. A planar or a spherical imaging surface can be considered, but in this work we will use a planar one.

Consider that the image is formed in a plane **I** orthogonal to the *optical axis* (equivalent to the $Z-axis$ of the camera referential—see Figure 6).

If the center of projection is at the origin **O** and the image is formed in a plane orthogonal to the $Z-axis$ at distance *f* (*focal length*) from **O**, the relationship between the image point $\mathbf{p} = (x, y, f)$ described and the scene point **P** under perspective projection is

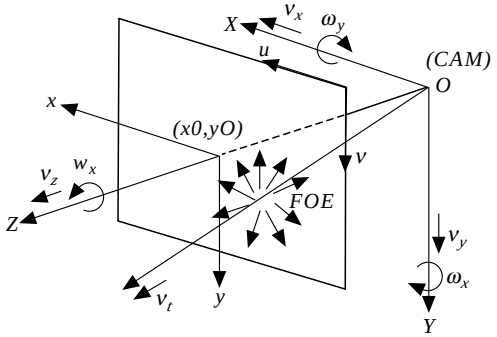

Figure 6. Image formation and motion representation.

$$\boldsymbol{p} = \frac{f}{\boldsymbol{P}^T . \hat{\boldsymbol{z}}} \boldsymbol{P}, \tag{8}$$

where $\hat{\mathbf{z}}$ is an unit vector and '$\cdot$' denotes the inner product of vectors. Note that the image for processing is actually a sampled version of the continuous image. Therefore it is necessary to convert the coordinates obtained from Eqn (8) to the image coordinates in pixels. For that we consider the relationship between the units used in the {**CAM**} referential and the pixels of the image. That relationship is obtained by a calibration process described by Batista *et al*. [26]. That process enables the estimation of the scale factors for both the *x* and *y* axes, the focal length *f* and the coordinates of the image center.

If we now differentiate **p** with respect to time, and substitute for **V**, we obtain the following equation for **v**:

$$\boldsymbol{v} = \frac{f}{\boldsymbol{P}^T \cdot \hat{\boldsymbol{z}}} ((\boldsymbol{v}_t \cdot \hat{\boldsymbol{z}}) \cdot \boldsymbol{p} - \boldsymbol{v}_t) + f(\boldsymbol{p} \cdot (\boldsymbol{\omega} \times \hat{\boldsymbol{z}} - \boldsymbol{p} \times \boldsymbol{\omega})). \tag{9}$$

This equation represents the analytical equation of the optical motion field projected in the image plane. The first term of **v** in Eqn (9) denotes the translational motion component which depends on the depth $Z = \mathbf{P}^{\mathbf{T}} \cdot \hat{\mathbf{z}}$, while the second term denotes the rotational component which does not depend on depth, but only on the rotational velocity.

To relate the optical flow field in log-polar coordinates with Eqn (9), let us write

$$\begin{bmatrix} \dot{\rho} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{\partial \rho}{\partial x} & \frac{\partial \rho}{\partial y} \\ \frac{\partial \theta}{\partial x} & \frac{\partial \theta}{\partial y} \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} \tag{10}$$

where $\dot{\rho}$, $\dot{\theta}$, $\dot{x}$, $\dot{y}$ stand for the derivatives with respect to time. Substituting the partial derivatives by their expressions we obtain

$$\begin{bmatrix} \dot{\rho} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\frac{\sin\theta}{\rho} & \frac{\cos\theta}{\rho} \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix}. \tag{11}$$

Defining

$$\xi = \ln_b \frac{\rho}{\rho_{fovea}} \tag{12}$$

and $\gamma = \theta$, the relationship between the time derivatives of ξ and ρ is given by

$$\dot{\xi} = \frac{1}{\ln b}\frac{\dot{\rho}}{\rho} = \frac{1}{\ln b}\frac{\dot{\rho}}{\sqrt{x^2 + y^2}}. \tag{13}$$

From (11) and using (13) we obtain the relationship between the motion field in Cartesian coordinates ($\dot{x}$, $\dot{y}$) and log-polar coordinates ($\dot{\xi}$, $\dot{\gamma}$) as

$$\begin{bmatrix} \dot{\xi} \\ \dot{\gamma} \end{bmatrix} = \frac{1}{\sqrt{x^2 + y^2}} \begin{bmatrix} \cos\gamma & \sin\gamma \\ \frac{\ln b}{-\sin\gamma} & \frac{\ln b}{\cos\gamma} \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix}. \tag{14}$$

Assuming that the sensor is approaching the scene with translational motion, the point where the axis of translation intersects the retina is called the *Focus of Expansion* (FOE), since at this point the projection of the translational motion field is zero and all projections of the translational motion vectors emanate from this point.

For an observer with translational motion with an opposite direction, this point has the projection of the motion field vectors pointing towards it, in which case we speak of the *Focus of Contraction* (FOC). The direction of these projected vectors is determined by the location of this vanishing point and their lengths depend on the 3D positions of the points in the scene.

If the observer wants to translate in the direction toward which he or she looks, this means that translation direction must intersect the retina plane. If the observer is approaching the scene, that intersection will be a point with characteristics of a FOE. To construct an autonomous artificial system that can be guided, it is necessary to acquire information about its own motion. That is a prerequisite to perform navigational tasks. Since we are using an active vision system, we need to develop algorithms to determine the projected motion field (*optical flow field*) in the images. The determination of this motion can be useful, in a more advanced phase, to control the system and keep the FOE point around the image center or, at least, stable in the image.

The images of Figure 7 illustrate the mapping of the optical flow vectors for different types of translational motion of the sensor. Notice that when the sensor translates in the same direction as the optical axis, the optical flow generated appears as vectors diverging from the image center. The effect in the cortical plane is a set of vectors with the same orientation.

The relative motion of the observer with respect to the scene gives rise to motion of the brightness patterns in the image plane. The instantaneous changes of the brightness

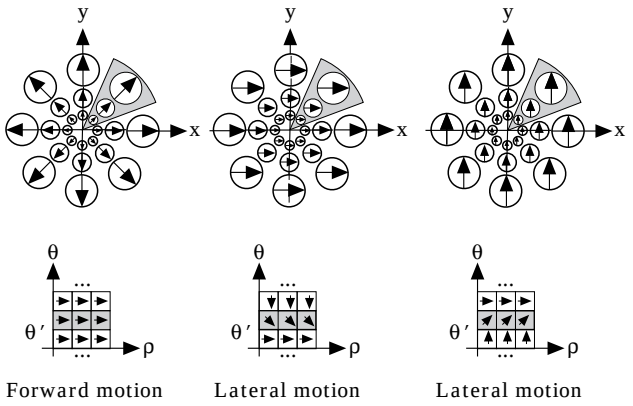

Figure 7. The optical flow vectors for different types of translational motion. For lateral motion the optical flow vectors generate in the cortical plane streamlines of vectors with the same orientation. For forward motion these lines are equal in all the plane.

pattern in the image plane are analysed to derive the optical flow field, a two-dimensional vector field (u, v) reflecting the image displacements.

The optical flow value of each pixel is computed locally—that is, only information from a small spatio-temporal neighborhood is used to estimate it. In general, it is not possible to compute the true velocity of an image point just by observing a small neighborhood. Suppose that we are watching a feature (a piece of contour or a line) at two instants of time and through a small aperture smaller than the feature (see Figure 8).

Watching through this small aperture, it is impossible to determine where each point of the feature has moved to. The only information directly available from local measurements is the component of the velocity which is perpendicular to the feature, the *normal flow*. The component of the optical flow parallel to the feature cannot be determined. This ambiguity is known as the *aperture problem* and exists independently of the technique employed for local estimation of flow. However, in cases where the aperture is located around an endpoint of a feature, the true velocity can be computed, because the exact location of the endpoint at two instants of time can be computed. Thus, the aperture problem exists in regions that have strongly oriented intensity gradients, and may not exist at locations of higher-order intensity variations, such as corners.

Any optical flow procedure involves two computational steps. In the first, assuming the local conservation of some form of information, only local velocity is computed. In a second step, in order to compute the other component of the optical flow vectors, additional assumptions have to be

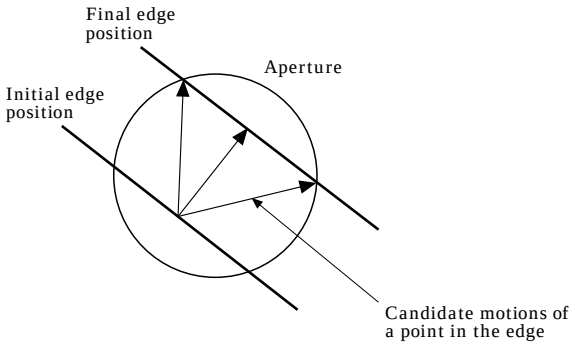

Figure 8. A line feature or contour observed through a small aperture at time t moves to a new position at time $t + \delta t$. In the absence of knowledge of camera motion, when we are looking at a viewpoint-independent edge in an image through an aperture, all we can say about the evolving image position of an indistinguishable point along the edges is that this position continues to lie somewhere along the evolving image of the edge. In the limit, we can locally determine only that component of the image motion of an image-intensity edge that is orthogonal to the edge—the *normal flow*.

made. Generally they are related with some kind of smoothness on the optical flow values.

The gradient-based approach introduced by Horn *et al*. [27] is based on the assumption that for a given scene point the intensity I at the corresponding image point remains constant over a short time instant. This corresponds to a brightness constancy assumption, $\frac{dI}{dt} = 0$, that gives a relationship that can be used to estimate the flow parameters directly from the spatial and temporal grey-level gradients. If a scene point projects onto image point (ξ, γ) at time t and onto the image point $(\xi + \delta\xi, \gamma + \delta\gamma)$ at time $(t + \delta t)$, we obtain the optical flow constraint equation [27],

$$\frac{\partial I}{\partial \xi} u + \frac{\partial I}{\partial \gamma} v + I_t = I_\xi u + I_\gamma v + I_t = 0 \qquad (15)$$

which relates the flow (u, v) to the partial derivatives (I_ξ, I_γ, I_t) of the image I. From this constraint alone, without making any additional assumptions we can only compute the normal flow u_n, equivalent to the projection of optical flow on the gradient direction:

$$u_n = -I_t \frac{1}{\|\Delta I\|}. \qquad (16)$$

Since our goal is to develop robust algorithms that can perform successfully in general environments, we abandoned all computational processes that are based on unrealistic assumptions or assumptions that can fail to verify, and the development of the algorithms was based on normal optical flow.

Let $I(\xi, \gamma, t)$ denote the image intensity, and consider the optical flow field $\mathbf{v} = (u, v)$ and the motion field $\mathbf{v}_m = (u_m, v_m)$ at the point (ξ, γ), where the normalized local intensity gradient is $\mathbf{n} = (I_\xi, I_\gamma)/\sqrt{I_\xi^2 + I_\gamma^2}$. The normal motion field at point (ξ, γ) is by definition

$$u_{m_n} = \mathbf{v}^T \mathbf{n} = \left(\frac{d\xi}{dt}, \frac{d\gamma}{dt}\right) \cdot \frac{\Delta I}{\|\Delta I\|} = \frac{1}{\|\Delta I\|}\left(I_\xi \frac{d\xi}{dt}, I_\gamma \frac{d\gamma}{dt}\right). \qquad (17)$$

If we approximate the differential $\frac{dI}{dt}$ by its total derivative we get a relationship between the equations (16) and (17)

$$u_{m_n} - u_n = \left(\frac{1}{\|\Delta I\|} \frac{dI}{dt}\right), \qquad (18)$$

which shows that the two fields are close to equal when the local image intensity gradient ΔI is high. This confirms that the normal flow is a good measurement of the normal motion field in locations where the intensity gradient exhibits high magnitude [28].

Figure 7 illustrates that a uniform motion in the log-polar plane is not uniform in the retina plane. This fact can be explored to generate an algorithm that, using this data structure, detects and determines large motions in the periphery of the image and smaller motions around the retina center. In the next section an algorithm for finding normal flow using this data structure is described.

Practical Implementation and Results

The main hardware components of the system are the mobile robot and the active vision system. These two basic units are interconnected by a computer designated *Master Processing Unit*. This unit controls the movements of the active vision system, communicates with the robot's on-board computer and is the host computer of the DSP (Digital Signal Processors) image processing hardware. The connections between the different processing units are represented in the diagram shown in Figure 9, and a photograph of the system is presented in Figure 10.

The hardware for image processing is based on TMS320C40* DSP CPUs, including one frame grabber for acquisition of the images from the two cameras. The active

* Trademark of Texas Instruments Inc..

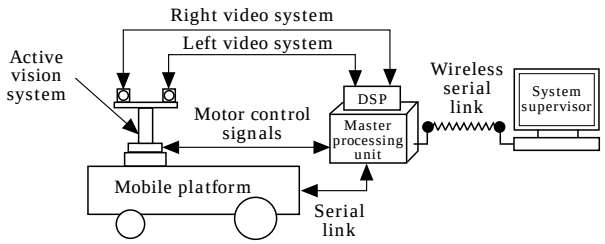

Figure 9. System architecture.

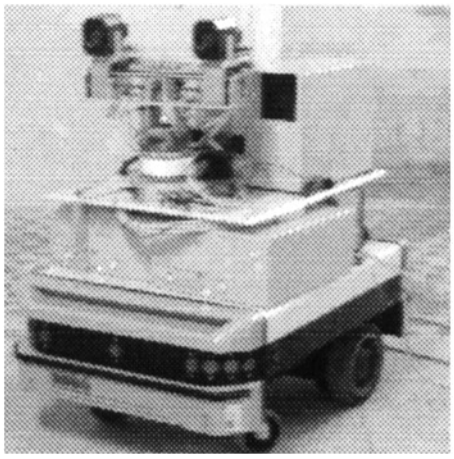

Figure 10. The active vision system and the mobile robot.

vision system has two CCD monochromatic video cameras with motorized lenses (allowing for the control of the iris, focus and zoom position) and five step motors that confer an equal number of degrees of freedom for the vergence of each camera, baseline shifting, *eyes* tilt and *neck* pan. The *Master Processing Unit* is responsible for the control of these degrees of freedom, using step motor controllers.

The *Master Unit* is also responsible for the platform's trajectory. This unit sends commands to the platform by using a RS-232 serial port that is the physical link of the communication system. The mobile platform has a multitasking operating system running on a 68020 CPU that is responsible for generating and controlling the commands received from the *Master Unit*. The supervision and management of the whole system is done by an external computer, connected to the *Master Processing Unit* and using a serial link with a wireless modem.

This system is used to perform the pursuit of moving targets so that the distance from the mobile robot to the target is kept approximately constant. Normal optical flow is used to enable visual fixation. After fixation is achieved the distance and heading of the target can be estimated and pursuit performed.

Computing normal flow on binocular log-polar images—the C40 solution

The log-polar sampling and the normal flow algorithm are implemented in software for DSP TMS320C40 from Texas Instruments. This software runs on a special board based on C40s (DSP TMS320C40). The board corresponds to one of the possible configurations of a modular hardware system based on these processors that explores the technical and communications capabilities of these devices. The modular nature of the system enables a scalable system if more processing power is required. With this architecture it is also possible to explore parallel processing implementations. The carrier board provides the modules with power, an interface for the ISA bus of the *Master Unit* and the infrastructure that allows data communication between modules. The processor TMS320C40 provides six communications ports that are bidirectional and can carry data at up to 20 MB/s. A six-channel DMA co-processor allows for concurrent I/O and CPU operation, maximizing sustained CPU performance. The CPU, running at 50 MHz, is capable of 275 million operations/s and 50 M FLOPS. It utilizes a 40/32 bit floating/fixed point multiplier/ALU and has hardware support for divide and inverse square root operations. The addressing modes include special addressing as linear, circular and bit-reversed addressing. A single-cycle barrel shifter for 0–31 bits right/left ensures fast bit manipulation. The TMS320C40 has two identical l00 MB/s buses for data and addresses (local and global).

The board used on our implementation comprises two modules, with their own processor C40 at 40 Hz, local and global memory and fast communications ports (see Figure 11). One of the modules is a frame grabber for image acquisition and processing. The board has 4 MB of local memory and $1024 \times 1024 \times 24$ bits of video memory. The

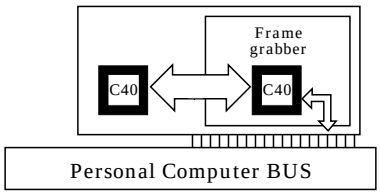

Figure 11. Block diagram of the image processing architecture based on DSPs (Digital Signal Processors).

board is able to acquire color images with 1024 × 1024 pixels by sampling the R, G and B signals, codifying each sample using 8 bits. The frame grabber is also able to capture images from different cameras and into a private video array buffer, without the intervention of the local C40. The other module has 1 MB of local fast RAM and 4 MB of global RAM. The processors are programmed to co-operate and to achieve the minimum execution time during the processing.

For real-time operation, we developed routines that are able to compute the several operations in the least amount of time possible. In order to apply a 'divide and conquer' technique, each time an image is captured the C40 is used to convert it into its log-polar equivalent, thus freeing the private memory of the grabber for the next image, and leaving the resulting image in its local memory for further processing. The additional processing is, in this case, the computation of the normal flow.

The algorithm to convert an image to the log-polar plane, and the method used to compute the normal flow, were implemented taking advantage of the set of special instructions provided by the C40 (parallel, block repeat, etc.). Care should be taken when using those instructions, particularly the parallel instructions (refer to [29] for detailed descriptions of the instructions).

Implementation of Log-polar Mapping

To implement the conversion of a common video array image into a log-polar image, the cortical pixel values should be computed as averages of the receptive fields, as described previously. Several implementations described in the literature use this principle [22, 30–32]. However, in our case we decided to use just sampling (without averaging) to speed up the whole computational process. For that purpose we need to sample the video image at the correct coordinates. One way of doing it is to program the mathematical equations needed to convert (ξ, γ) into (x, y), and then use them for every needed value of (ξ, γ). Although simple, that method would represent a big load to the processor, since it needs to perform multiplications and compute logarithms to do it. Also, after the (x, y) values were obtained from the desired (ξ, γ) coordinates, the processor would need to convert them into the correct displacement on the video array image, adding one more multiplication to the method for each coordinate.

The procedure to convert a single (ξ, γ) pixel into its corresponding displacement in the video array image can be speeded up if a conversion table is built. That table gives the displacement in the video array image for all the possible values of (ξ, γ) and needs only to be initiated at start-up. However, like all the other methods involving tables it is quite fast, but needs large memory resources. The process of table access must be also reversible, and once the video image has been captured, we want to convert it completely into its log-polar equivalent. So we can take advantage of the C40 repeat instructions that allow us to repeat one or more instructions using no cycles to branch back to the start of the loop (they are thus called zero-overhead looping instructions). Also, we are using a processor that is able to perform three bus accesses per cycle: one instruction read and two operand loads (reads) or stores (writes). In fact, the C40 takes advantage of this facility through parallel instructions that allow two loads, two stores, or one load and one store to be performed in one instruction.

Unfortunately, each conversion using the table involves one load for getting a displacement from the table, then another load when using the displacement in the video array to get a pixel, and one store for saving on a memory buffer the pixel taken from the video array. So we cannot perform the conversion with a single parallel instruction, but we can still take advantage of them. Using *C language-style* expressions we can write two cycles of the conversion process like:

```
video_array_address_plus_displacemen_1 = *table++;
pixel_1 = *video_array_address_plus_displacement_1;
*buffer++ = pixel_1;
video_array_address_plus_displacement _2 = *table++;
pixel_2 = *video_array_address_plus_displacement_2;
*buffer++ = pixel_2;
```

Although there are several ways to implement two cycles of the conversion process, the suggested expressions were written based on several aspects that should be noted:

(i) The conversion table contains the displacements added to the base address of the video array, instead of just the displacements.

(ii) Although the parallel instructions can perform one operand load and one operand store with the same instruction, they cannot load a value and store the same value in the same instruction. So an expression like $*a = *b$ could not be converted directly into a parallel instruction. The loaded value can only be used by the next instruction, and the stored value must have been loaded with the previous instructions.

(iii) The C40 assembly code, and the parallel instructions in particular, must follow certain rules, like the need to use auxiliary registers when pointers are needed, and the impossibility to load those registers with parallel instructions. So the expressions that load the *video_array_address_plus_displacement_N* pointers cannot be performed by parallel instructions.

Based on this, we can re-arrange the conversion process as:

```
video_array_address_plus_displacement_1 = *table++;
video_array_address_plus_displacement_2 = *table++;
pixel_1 = *video_array_address_plus_displacement_1;
pixel_2 = *video_array_address_plus_displacement_2;
*buffer++ = pixel_1;
*buffer++ = pixel_2;
```

The first two expressions must be implemented by two separate load instructions (loading of auxiliary registers), but the last four expressions may be performed by two parallel instructions. The first parallel instruction will execute a double load, and the second a double store. Defined in that block, we need only to repeat it as many times as half the number of entries in the table (since the block will read two of them). In the end, our buffer will contain the log-polar equivalent of the video array image.

Notice that the C40 uses words with 32 bits. So each entry in the table will be a 32 bit pointer, and the table elements must be consecutive in memory. The above routine is executed each time an image is captured. The destination buffer is located in the C40 local memory, allowing further processing without interfering with the acquisition process. Each entry in the destination buffer will also be 32 bits, although only the least significant 8 bits are important.

Computing the Normal Flow Using DSPs

Assuming that we have captured two images in two consecutive instants of time, and converted them into their log-polar equivalent, this section will show a fast way to compute the normal flow between them. We start by smoothing the images, but we will not describe it, since it is a simple operation. Extracting the normal flow is, basically, a problem of computing the partial derivatives of the image intensity on a certain pixel, with respect to ξ, γ and t (time). To do that, we use the first differences of the pixel values on a cube (see Figure 12). Let $I_{\xi,\gamma,t}$ be the intensity of the pixel located at (ξ, γ) in the image taken at time t, then the partial derivatives D_ξ, D_γ and D_t can be obtained as follows:

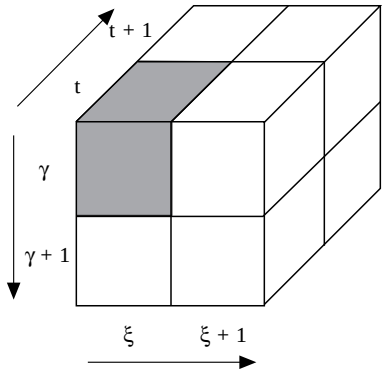

Figure 12. The 8 pixels involved in the computation of the partial derivatives on the shaded pixel.

$$
\begin{aligned}
D_\xi &\approx \frac{1}{4}(I_{\xi+1,\gamma,t} + I_{\xi+1,\gamma,t+1} + I_{\xi+1,\gamma+1,t} + I_{\xi+1,\gamma+1,t+1}) - \\
&\quad - \frac{1}{4}(I_{\xi,\gamma,t} + I_{\xi,\gamma,t+1} + I_{\xi,\gamma+1,t} + I_{\xi,\gamma+1,t+1}) \\
D_\gamma &\approx \frac{1}{4}(I_{\xi,\gamma+1,t} + I_{\xi,\gamma+1,t+1} + I_{\xi+1,\gamma+1,t} + I_{\xi+1,\gamma+1,t+1}) - \\
&\quad - \frac{1}{4}(I_{\xi,\gamma,t} + I_{\xi,\gamma,t+1} + I_{\xi+1,\gamma,t} + I_{\xi+1,\gamma,t+1}) \\
D_t &\approx \frac{1}{4}(I_{\xi,\gamma,t+1} + I_{\xi,\gamma+1,t+1} + I_{\xi+1,\gamma,t+1} + I_{\xi+1,\gamma+1,t+1}) - \\
&\quad - \frac{1}{4}(I_{\xi,\gamma,t} + I_{\xi,\gamma+1,t} + I_{\xi+1,\gamma,t} + I_{\xi+1,\gamma+1,t}).
\end{aligned} \tag{19}
$$

To apply Eqn (19) directly, we need to perform 21 additions/subtractions to reach the expected results. In an attempt to reduce that number, let us start by defining the following relations:

$$
\begin{aligned}
a &= I_{\xi,\gamma,t+1} - I_{\xi+1,\gamma+1,t} \\
b &= I_{\xi,\gamma+1,t+1} - I_{\xi+1,\gamma,t} \\
c &= I_{\xi,\gamma+1,t} - I_{\xi+1,\gamma,t+1} \\
d &= I_{\xi,\gamma,t} - I_{\xi+1,\gamma+1,t+1} \\
e &= b - d \\
f &= a - c.
\end{aligned} \tag{20}
$$

We can now simplify the partial derivatives using the above relations, resulting in only 11 additions/subtractions to compute:

$$
\begin{aligned}
D_\xi &\approx \frac{1}{4}(-a - b - c - d) \\
D_\gamma &\approx \frac{1}{4}(-a + b + c - d) = \frac{1}{4}(e - f) \\
D_t &\approx \frac{1}{4}(a + b - c - d) = \frac{1}{4}(e + f).
\end{aligned} \tag{21}
$$

Another problem we must face is reading the pixels in the two images. Each time the partial derivatives are computed, we need to read 4 pixels from each image. Since accessing memory is always a slow task, we should try to reduce the number of accesses needed. That is not a hard goal to accomplish. In fact, if we compute the derivatives sequentially, we only need to access the memory four times for each new set of partial derivatives, since 4 of the pixels used in the last computation will be used again. Hence, we can store those pixels in the machine registers, and just read the next 4 pixels.

After computing the partial derivatives, how will we use them? The answer depends on the problem at hand, and on what we intend to do with the values. As an example, let us suppose we want to compute the angle of the vector with coordinates (D_ξ, D_γ). Notice that this angle would be computed in the $\xi - \gamma$ space, and therefore one should be careful in relating this value with the orientation in image space. As was the case with the sampling of the log-polar image, the inverse tangent is not going to be computed each time the angle is needed. The fastest way is to build a table with the inverse tangent of several pairs of values of D_ξ and D_γ. How big will the table be? Since the pixel intensity is a value between 0 and 255, the values of D_ξ and D_γ are always between -255 and 255. Hence, a table with all the possible combinations of values of D_ξ and D_γ would have 511×511 entries. This is obviously an enormous table. In general, we do not need a table with that many entries. What we should do is just ignore one or more of the least significant bits of the values, and use them as indexes to the table.

Timing Considerations for Binocular Real-time Operation

The system throughput depends very much on the algorithms implemented, on the size of the images we are working on, and on our needs and hardware solutions. Figure 13 shows a very fast processing scheme, taking full advantage of the hardware available. Looking back at Figure 11, we see that only the C40 supporting the frame grabber can communicate with the PC host. For that reason, that C40 is called the master, and all the others (just one in the case described) are called the slaves. The frame grabber has multiple video inputs, but can only capture from one of them at a time. Since the images captured are interlaced PAL video signals, and since we decided not to use the full resolution of the images, we only capture one field from each camera. Figure 13 shows that both video signals are synchronized. For that to happen, the cameras must have external synchronization signals. If they are not synchronized, there will be a delay between the ending of a capture and the beginning of the next, which will reduce the global processing speed.

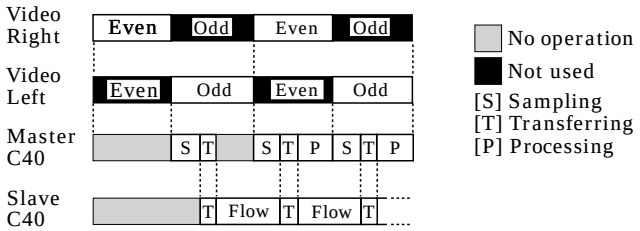

Figure 13. The timing diagram. The video signal from both cameras is shown, and also the processing stages of both C40s.

The timing diagram in Figure 13 describes the starting stages of the processing scheme. The rest of the processing will be a simple repetition of the final steps shown. Below we enumerate all the steps:

(i) While the even field of the right camera is being acquired, both C40s are in busy wait.
(ii) When the field ends, the master C40 switches the frame grabber to acquire the odd field of the left camera, and starts the log-polar sampling of the right image, using the method described above. After the log-polar image is complete, it will be downloaded to the slave C40 for processing.
(iii) The slave C40 receives the log-polar image, and applies the normal-flow algorithm to it. The slave will store that image until another image from the right camera is received, since the algorithm must be used with two images.
(iv) When the odd field from the left camera ends, the frame grabber is switched back to the right camera, and the left image will be converted and downloaded. When the time comes to download the left log-polar image, the results from the slave C40 are available, and the master C40 will upload them.
(v) The master C40 will then process the results, and inform the PC host of the actions to take (in our case, the control of an active vision system).

With this scheme we are able to reach a processing cycle of 50 Hz (the frequency of one field of the video signal), with a maximum processing delay of 40 ms. Since we are interested on the results from both images, the accumulated processing reduces the frequency to 25 Hz. What if the algorithms are too slow to fit in that diagram? Then additional C40s should be used, and instead of the results from one slave being uploaded by the master, they would be sent to another slave, and so on. The master would receive the final results from the last slave in the chain,

adding a processing delay of 20 ms per slave, but keeping the same processing frequency. The amount of time spent in transferring between the C40s is negligible. Indeed, DMA is used, allowing the C40s to process while receiving and sending data.

Results obtained by the application of this normal flow algorithm are illustrated in Figure 14.

After the computation of the log-polar normal optical flow, additional processing has to be done in order to enable the active vision system to perform fixation. The basic idea is to use the "amount" of motion taking place in the image as an indication of the region that should be *fixated*, i.e., that should be in the foveal area of the image. As a measure of the amount of motion, the length of the flow vectors is used. Let $\mathbf{v}_n$ represent a normal flow vector and $||\mathbf{v}_n||$ its magnitude. Two functions of the magnitude of the flow vectors are computed: the integral along the γ-axis for each one of the ξ values and the integral along the ξ-axis for each one of the γ values. Because we are dealing with discrete values, these functions are

$$G(\xi_j) = \sum_{\gamma_i} \|v\|_{\gamma_i} \tag{22}$$

and

$$H(\gamma_j) = \sum_{\xi_i} \|v\|_{\xi_i} , \tag{23}$$

where ξ_i and γ_i represent the discrete values of the variables. Figure 15 displays the values of these two functions computed for the sequence of images of Figure 14. After these two functions are computed, the centers of mass of both distributions are used as the location of the moving target. During this process several thresholds are applied, namely to the magnitude of the vectors and to the values and extent of the functions. These thresholds prevent the system from reacting to noisy and small motions. The control system (running in the *Master Unit*) takes these values as estimates of the locations and directs both cameras so that *fixation* is performed. The control structure includes predictive filters. Further processing may be required to confirm that both cameras are fixating the same object. Fixation of the object enables the robot to track it.

Discussion and Conclusions

This paper describes a computational solution for normal optical flow estimation using space-variant image sampling. The article describes some of the most important properties of the log-polar structure for fast and real-time processing. These include the high data compressing rates by maintaining a high resolution area—the fovea—and a space-variant resolution—the retinal periphery. This sampling scheme is useful for the implementation of selective processing, characteristic of attentive vision and active vision svstems.

The implementation of normal optical flow detection was developed around digital signal processors (DSP) TMS C40. The normal optical flow computation is performed in real time (video frame rates) for both the left and right images. The use of the log-polar sampling scheme was crucial for this type of performance.

The computation of the normal optical flow is part of a robotic system that uses binocular visual fixation in order to track moving objects so that the distance between the target and the mobile robot is kept approximately constant. Binocular visual fixation enables easy computation of the target distance and heading. Despite the fact that globally the image is much smaller, the higher resolution in the central region of the image enables a more robust operation of the control structure of the active vision system while performing visual fixation.

To fixate the target, its location in both images is computed from the distribution of the normal flow. Further prediction filtering is required to enable smooth fixation control. The system described demonstrates the use of normal optical flow computed in log-polar sampled images to perform binocular visual fixation in real time with an active vision system integrated in a mobile robot.

In the future we plan to implement more complex visual behaviors that will enable the robot to perform obstacle avoidance as well as other reactive behaviors. The goal is to equip the mobile system with behaviors that enable the development of co-operation with other robots.

References

1. Papanikopoulos, N., Khosla, P. & Kanade, T. (1993) Visual tracking of a moving target by a camera mounted on a robot: a combination of control and vision. *IEEE Trans. on Robotics and Automation*, **9**(1): 14–34
2. Allen, P., Timcenko, A., Yoshimi, B. & Michelman, P. (1993) Automated tracking and grasping of a moving object with a robotic hand-eye system. *IEEE Trans. Robotics and Automation*, **9**(2): 152–165.
3. Coombs, D. (1992) *Real-Time Gaze Holding in Binocular Robot Vision*, PhD Dissertation, University of Rochester, USA, June.

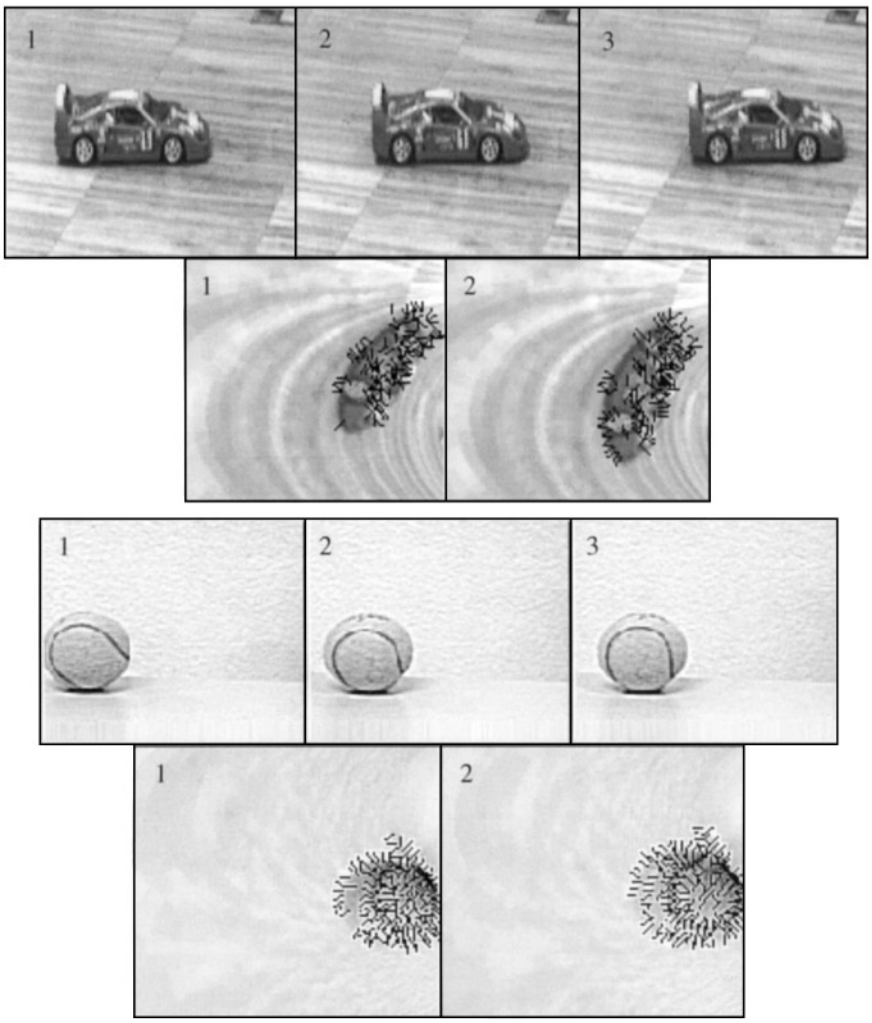

Figure 14. Two different image sequences showing the computation of the normal optical flow. For each sequence the top row displays the original images and the second row displays the computed normal flow vectors on log-polar. In the log-polar images the horizontal axis corresponds to the coordinate ξ and the vertical axis corresponds to the coordinate γ. The origin of each log-polar image is located at the bottom left corner.

4. Grosso, E. & Ballard, D. (1992) Head-Centered Orientation Strategies in Animate Vision, *Technical Report 442*, Comp. Science Dept., University of Rochester, USA, October.
5. Cipolla, R. & Hollinghurst, N. (1994) Visual robot guidance from uncalibrated stereo. In: Brown, C. & Terzopoulos, D., *Real-time Computer Vision*, University Press, pp. 169–187.
6. Brown, C. (1990) Gaze controls with interaction and delays. *IEEE Transactions on Systems, Man and Cybernetics*, IEEE TSMC 20, May, pp. 518–527.
7. Coombs, D. & Brown, C. (1992) Real-time smooth pursuit tracking for a moving binocular robot. In: *Proc. 1992 IEEE Comp. Soc. Conf. on Computer Vision and Pattern Recognition*, pp. 23–28, Champaign, USA.
8. Pahlavan, K., Uhlin, T. & Eklund, J.O. (1992) Integrating primary ocular processes. In: *Proc. Second European Conference on Computer Vision*, Santa Margherita, Italy: Springer-Verlag, pp. 526–541.
9. Pahlavan, K. (1993) *Active Robot Vision and Primary Ocular Processes*, PhD thesis, Royal Institute of Technology, Sweden.
10. Uhlin, T., Pahlavan, K., Maki, A. & Eklund, J.O. (1995) Towards an active visual observer. In: *Prof. Fifth International Conference on Computer Vision*, pp. 679–686, Cambridge, MA, June 20–23.
11. Burt, P., Bergen, J., Hingorani, R., Kolcynski, R., Lee, W., Leung, A., Lubin, J. & Shvaytser, H. (1989) Object tracking

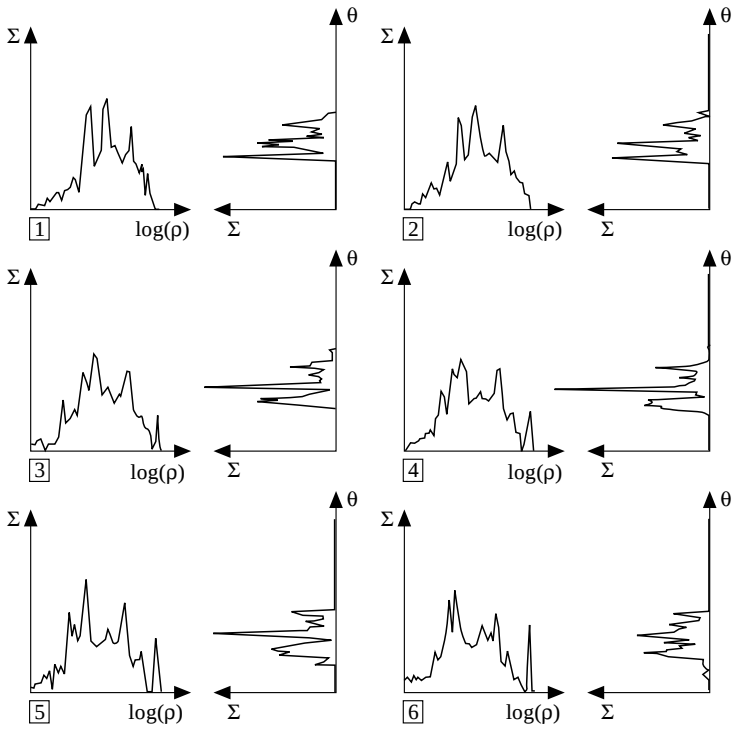

Figure 15. Computation of the integrals of the normal flow vector magnitudes.

with a moving camera. In: *Proc. IEEE Workshop Visual Motion*, Irvine.
12. Murray, D., McLaughlin, P., Read, I. & Sharkey, P. (1993) Reactions to peripheral image motion using a head/eye platform. In: *Proc. 4th International Conference on Computer Vision*, pp. 403–411.
13. Manzotti, R., Tiso, R., Grosso, E. & Sandini, G. (1994) Primary ocular movements revisited. *Technical Report 7/94*, LIRA Lab, University of Genoa, Italy, November.
14. Panerai, F., Capurro, C. & Sandini, G. (1995) Space variant vision for an active camera mount. *Technical Report TR 1/95*, LIRA Lab, University of Genova, Italy.
15. Carpenter, H. (1988) *Movements of the eyes*, 2nd edn, London: Pion Limited.
16. Hacisalihzade, S., Stark, L. & Allen, J. (1992) Visual perception and sequences of eye movement fixations: a stochastic modeling approach. *IEEE Trans. on Systems, Man and Cybernetics*, **22**(3): 474–481.
17. Dias, J., Paredes, C., Fonseca, I., Batista, J., Araújo, H. & de Almeida, A. (1995) Simulating Pursuit with Machines – Experiments with Robots and Artificial Vision. In: *1995 IEEE Int. Conf. on Robotics and Automation*, Nagoya, Japan, May.
18. Sandini, G. & Tagliasco, V. (1980) An anthropomorphic retina-like structure for scene analysis. *Computer, Graphics and Image Process.*, **14**: 365–372.
19. Schwartz, E. (1984) Anatomical and physiological correlates of visual computation from striate to infero-temporal cortex. *IEEE Trans. on Systems, Man, and Cybernetics*, **SMC-14**(2): 257–271.
20. Massone, L., Sandini, G. & Tagliasco, V. (1985) Form-invariant topological mapping strategy for 2D shape recognition. *Computer Vision, Graphics and Image Processing*, **30**: 169–189.
21. Weiman, C.F. (1988) Exponential sensor array geometry and simulation. In: *Proc. SPIE Conf. on Pattern Recognition and Signal Processing*, **938**, pp. 129–137, Orlando, FL.
22. der Spiegel, J.V., Kreider, G., Claeys, C., Debusschere, I., Sandini, G., Belutti, P. & Soconi, G. (1989) A foveated retina-like sensor using CCD technology. In: Mead, C. (ed.) *Analog VLSI implementations of neural systems*, Boston, MA: Kluwer.
23. Wallace, R.S., Ong, P.-W., Bederson, B.B. & Schwartz, E.L. (1994) Space variant image processing. *Int. J. Computer Vision*, **13**(1): 71–90.
24. Weiman, C.F. (1994) *Log-polar binocular vision system – final report*, SBIR Phase II Contract #NAS 9–18637, T.R.C., Danbury, December.
25. Bederson, B.B., Wallace, R.S. & Schwartz, E.L. (1995) A miniaturized space-variant active vision system: cortex-I. *Machine Vision and Applications*, **8**: 101–109.
25. Reitboeck, H. & Altmann, J. (1984) A model for size- and rotation-invariant pattern processing in the visual system. *Biol. Cybern.*, **51**: 113–121.
26. Batista, J., Dias, J., Araújo, H. & de Almeida, A. (1993) Monoplanar camera calibration, *British Machine Vision Conference*, 21–23 September, Surrey, UK, **2**: 476–488.
27. Horn, B. & Schunk, B. (1981) Determining optical flow. *Artificial Intelligence*, **17**: 185–204.
28. Fermuller, C. & Aloimonos, Y. (1995) Qualitative egomotion. *Int. J. Computer Vision*, **15**: 7–29.
29. TMS320C4x User's Guide, Texas Instruments (editor), 1992.
30. Bailey, J.G. & Messner, R.A. (1990) Docking target design

and spacecraft tracking system stability. *SPIE – Intelligent Robots Computer Vision VIII: Algorithms and Techniques*, **1192**: 820–831.
31. Weiman, C.F. (1990) Video compression via log-polar mapping. In: *Proc. SPIE Conf. on Real Time Image Processing II*, **1295**: 266–277, Orlando, FL.
32. Bederson, B.B., Wallace, R.S. & Schwartz, E.L. (1993) A miniaturized space-variant active vision system: cortex-I. In: Mammone, R.J. (ed.), *Artificial Neural Networks for Speech and Vision*, Chapman and Hall, pp. 429–456.
33. Aloimonos, Y., Weiss, Y. & Bandopadhay, A. (1988) Active vision. *Int. J. Computer Vision*, **7**: 333–356.
34. Aloimonos, Y. (1990) Purposive and qualitative active vision. *Proc. Image Understanding Workshop*, 816–828.
35. Bajcsy, R. (1988) Active perception. *Proceedings of the IEEE*, **76**: 996–1005.
36. Ballard, D. (1991) Animate vision. *Artificial Intelligence*, **48**: 57–86.
37. Bederson, B.B., Wallace, R.S. & Schwartz, E.L. (1992) A miniaturized active vision system. In: *Proc. 11th IAPR International Conference on Pattern Recognition Vol. IV, Conference D: Architectures for Vision and Pattern Recognition*, pp. 58–61.
38. Bergen, J., Burt, P., Hingorani, R. & Peleg, S. (1990) Computing two motions from three frames. *Technical Report from David Sarnoff Research Center*, subsidiary of SRI International, Princeton, NJ, 08543–5300.
39. Blackman, S. (1986) *Multiple-target tracking with radar applications*. Artech House, Inc., pp. 19–25.
40. Burt, P. (1981) Fast filter transforms for image processing. *Computer Vision, Graphics, and Image Processing*, **16**: 16–20.
41. Burt, P. & Adelson, E. (1984) The pyramid as a structure for efficient computation. In: Rosenfeld, A. (ed.), *Multiresolution image processing and analysis*, Spring Series in Information Sciences, Vol. 12, New York: Springer.
42. Burt, P., Anderson, C., Sinniger, J. & van der Wal, G. (1986) *A pipelined pyramid machine*, NATO ASI Series, Vol. F 25.
43. Crowley, J. (1981) *A Representation for Visual Information*, PhD Thesis, Carnegie-Mellon University, Robotics Institute, Pittsburgh, PA.
44. Dias, J., Batista, J., Simplício, C., Araújo, H. & de Almeida, A. (1993) Implementation of an Active Vision System. In: *Proceedings of International Workshop on Mechatronical Computer Systems for Perception and Action*, Halmstad University, Sweden, June 1–3.
45. Meer, P., Baugher, E. & Rosenfeld, A. (1987) Frequency domain analysis and synthesis of image pyramid generating kernels. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **PAMI-9**(4): 512–522.
46. Paul, R. (1984) *Robot manipulators: mathematics, programming, and control.* Cambridge, MA: The MIT Press, pp. 9–11.