A Smart Buffer for Tracking Using Motion Data

James J. Little and Johnny Kam*
Department of Computer Science
University of British Columbia
Vancouver, BC, Canada V6T 1Z2

Abstract

Responsive vision is vision responding to the environment. The characteristics of a responsive system are: active response in dynamic environment, real time computation, and using multiple modalities in a multi-purpose system. The Vision Engine[?] is a general purpose for general vision tasks. Early vision processing, e.g., optical flow and stereo is implemented in near real-time using the Datacube, producing dense displacement fields at near video rates, which are then transferred to a Transputer subsystem, where data dependent processing occurs in parallel on subimages.

We use the Vision Engine for complex processing under real-time constraint, the differences between the processing rates in a robotic system require smart buffers, objects that can buffer data between perception, reasoning and action processes. Smart buffers offer a simple interface between asynchronous processing tasks and simplify the structure of multiprocessor vision systems. We describe a simple motion tracker that uses a smart buffer to mediate between early and middle vision processing. The smart buffer permits the system to sense during action by letting the sensing component accumulate visual data in the course of action.

*The authors may be contacted at little@cs.ubc.ca. This research was supported by a grant from the Natural Sciences and Engineering Research Council of Canada and the Networks of Centres of Excellence Institute for Robotics and Intelligent Systems, Project A-1.

1 Introduction

Responsive vision is vision responding to the environment: a response is "an answer, an action, stimulus, movement, or change elicited by a stimulus or influence", responsive is "reacting readily or favorably". The characteristics of our responsive system are: active response in dynamic environment, real time computation, and using multiple modalities in a multi-purpose system. A vision system is termed "active" when the system uses changes in its viewing parameters to improve its results. How the parameters, such as viewpoint, change can be directly controlled by the nature of the task, in which case the critical issue is how to change the parameters, e.g., [?, ?], or the parameters may be changing independently, as in sensing from a moving vehicle, in which case the question is how to use additional information to add robustness or reduce ambiguity, e.g., [?]. Responsive adds to the goals of active vision the goal of real-time action.

The goal of real-time performance is mandatory for any responsive vision system, but the goal of performance conflicts with the computational intensity of most vision tasks. The necessity of quick response is driven by interaction with a dynamic world. The luxury of long computation times vanishes when confronted with the urgency of the physical world. Vision systems need tasks to keep them honest and nothing is more honest than real-time deadlines, aptly named.

Usually vision systems become trivial to respond to real time constraints—if there is little time to do something, do very little. Many robotic systems employ specialized sensors de-

signed for particular tasks so that the connection between sensor and action can be direct and fast. [?]. But of course this is not vision.

Parallel processing is one way out of the dilemma, and much recent work has been dedicated to discovering how to utilize parallel processors for general vision algorithms [?, ?]. Likewise many specialized processors have been designed, from analog chips [?] to special-purpose digital designs[?], such as reconfigurable meshes[?]. These machines are as yet unavailable for common comsumption or, if available, are quite specialized and solve only the very early stages of vision processing. But the machines on which complex vision algorithms have been implemented are typically large and expensive. Robotic tasks demand small and inexpensive machines. So a vision module must operate in a physical context where there are not only physical constraints, but also economic constraints. A vision module must be part of a system that is configured in a realistic fashion.

To solve the problems of implementing non-trivial vision processing on realistic machines, a designer must confront the realities of multirate processing: due to technology mismatch between early processing engines and more limited intermediate and high level processing capability, as well as limited communication bandwidth, the rate at which components in the system produce and consume data vary enormously.

1.1 The Vision Engine

A short discussion of the structure of the UBC Vision Engine [?] will make the problem concrete. The system is designed to be general purpose and support computation for all levels of vision. The levels of vision processing include early vision, which is spatially homogeneous, involving dense processing, such as filtering; middle vision, which is spatially distributed, regular but sparse, such as line following, aggregation; and late vision, which is symbolic, such as matching[?].

Our system, the Vision Engine, consists of multiple architectures, each commonly available. The first stage is pipelined, a Datacube MaxVideo200

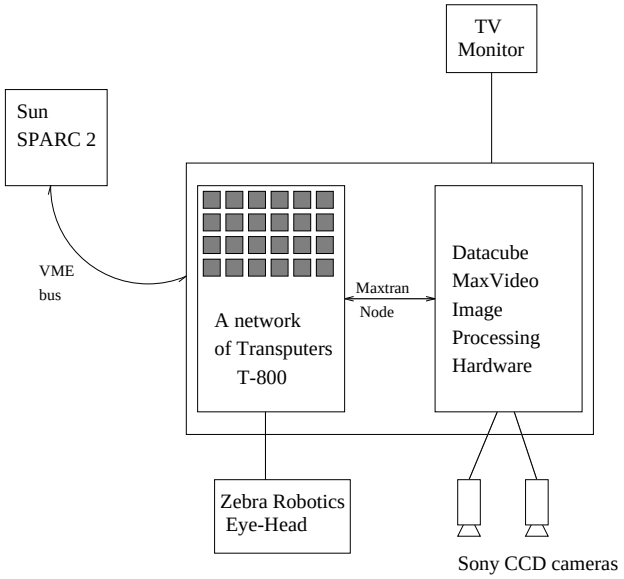

Figure 1: Configuration of the LCI Robot Head and Vision Engine

image processor with a Digicolor color image digitizer. The second stage is a MIMD multicomputer: 20 T800 Transputers, each with 2MB memory, with programmable interconnections through a crossbar. These subsystems are connected by a Maxtran board, a bidirectional interface operating at video rates. The Maxtran board maps data from the video bus of the Datacube system into video RAM attached to a Transputer. Figure ?? depicts the organization of our system. The entire system resides on the VME bus connected to a Sun SparcStation 2 host. The Transputer system communicates directly with a specialized eye/head platform with vergence, pan and tilt control, built by Zebra Robotics. The Transputer system can also control a CRS A-460 six degree-of-freedom robot arm and a collection of independently controlled mobile robot vehicles that play soccer [?].

1.2 Vision Operating Systems

The right model for the design of a vision system can be taken from the design of operating systems. A central problem in machine architec-

ture is mating the central processor, the fastest processing element in the system, with a large and slow memory hierarchy. To simplify programming of the CPU, the machine presents the abstraction that the memory is homogeneous and comparable in speed to the CPU. Likewise, a vision system may be composed of many different speeds of processors, but the critical central processing should be programmed at an abstraction level where the details of processing speeds in the more peripheral elements can be ignored.

What must a vision operating system provide to allow this abstraction? Temporal considerations motivate this feature so that processing sequences of images or more precisely vision processing in time. Unlike the memory hierarchy, early vision processing can be quite rapid because its homogeneous nature lends itself to implementation in simple processors such as mesh and pipeline processors, and in specialized hardware. Later processing stages are data dependent and complex and must be implemented on more general processors. It is necessary to create a buffer between the high-speed process and the slower stages to follow.

In our architecture, a smart buffer operates between the high-speed processing in early vision and the more complex data-dependent processing in the middle and late vision processes.

The smart buffer creates the illusion is that the vision system continuously views the world and that the robot can sample the visual world at any time. In vision applications, data input is not continuous, rather, it is quantized in time by video frames. The delay between data input and response is mediated by the intervening layers. However, unlike a continuous system, it makes sense to speak of the "last" result. When the inter-frame interval is sufficiently small, the last result may still be usable data.

We structure real-time vision systems as processes that operate on data streams. Each process typically operates on an image, then optionally subdivide an image into smaller subimages and sends the image(s) on to later processing stages. The later stages can collect several image streams, representing subimages of an original image, to form aggregates. More simply, a stage can just process the subimage and pass it on. Consider processed images arriving in a vision system at a later interpretation process. If the later process cannot process arriving image in the interframe period, data will be lost. A simple ring of buffers would suffice if the processing time is sufficient to process the data eventually; if not, the queue of data stored in the buffers would grow indefinitely. A smart buffer performs intermediate processing on arriving data to connect two processes at different speeds, providing current data without burdening the receptor process with details of how it is buffered.

There is some resemblance between the smart buffer and some aspects of the "whiteboard" concept supported by CODGER (COmmunications Database with GEometric Reasoning) in CMU NAVLAB [?, ?]. CODGER supports data flow between parallel modules using a central blackboard database. Unlike CODGER, our smart buffer essentially operates just to collect data. Like CODGER, synchronization between the producer and consumer is handled.

In order to be responsive, each component of our vision system needs to be able to provide results to successor components, at any time. The interface between successive layers must provide for this ability, either by tailoring the algorithm to this need, or by explicit storage of the results of the previous step. Computations that can be interrupted any time and provide approximate results have been termed "anytime". The smart buffer connects with the anytime concept by allowing the later stages to get the data "anytime".

2 An Example

The section describes an example that shows how we can use these ideas to build a tracker that has several processes occurring at different rates. We have implemented a complex, integrated tracking system to demonstrate our Vision Engine in action, and to develop the core of a responsive vision system, which needs dense results, but perhaps

not detailed results, to provide quick response to changing situations. Coverage by the vision system should be broad—one function of a vision system is to monitor the environment for threats, albeit as simple as an obstacle during locomotion. Moreover, it should be able to support higher functions like recognition and so should provide general, not task-based results.

The tracker demonstrates how complex vision processing can be applied in a vision task at near real-time rates[?]. Therefore it uses motion processing to select a target and tracks a moving object, without any knowledge of the particular object that it tracks. Moreover, it uses dense optical flow data as its input and does not reduce tracking, essentially, to following a white spot on a black background, as do many systems.

2.1 Tracking Systems

Following the largest moving object is a popular choice in motion tracking. A simple one camera motion tracking system has been implemented to pick the second largest moving area as target, assuming that the dominant motion to always be the motion of the background [?]. Such system is running on a 16K processor Connection Machine; the tracking algorithm is sequential, and it is currently dealing with only one object. There is no guarantee the background motion is always the largest in magnitude or in size. Our system takes the approach of attempting to cancel out the background motion, or at least minimizing its effect causing it to be the least significant motion, i.e., all moving objects should have flow values larger than background motion after cancellation. Eventually, we will pick the target based primarily on the magnitude of motion, instead of the size of the region.

The approach taken in [?] is first to verge the cameras and then to use a Zero-Disparity Filter to pick out the target. Their system is quite robust and has good performance with a prediction module. It is not clear, however, how such system can deal with multiple objects, or to shift attention. The approach taken at the University of Rochester is to throw away, or filter out, irrelevant data or useless information as quickly as possible. Our datacube program produces stereo data, so it would be simple to implement ZDF in our system; we have decided to attack a more difficult problem, using motion data. Our current system uses more output data, and therefore, has a much better record of the motion paths of *all* moving objects in view for further analyses.

An attentive control system [?] has been implemented to track features. Shifts in focus of attention are accomplished by using a saliency map and by altering feedback gains applied to the visual feedback paths in the position and velocity control loops of the binocular camera system. Since we do not have any knowledge about any object we are dealing with, our system simply uses motion field to drive attentional processing.

The KTH Head has 13 degrees of freedom and 15 different motors, simulating the essential degrees of freedom in mammalians [?]. Their current work suggests the integration of low-level ocular processes for fixation, and the use of cooperative vergence-focussing to assist the matching process. Their current work realizes dynamic fixation, i.e., changes in focus and vergence in real time [?].

The Oxford group[?, ?] tracks foveated corner clusters; the affine structure of a moving cluster of corners (each tracked using an image-velocity Kalman filter) determines the cluster point.

The absolute simplest tracking system is a "bright-spot" tracker: one that chooses some feature, e.g., a white dot on a black background, on a single image to follow, without even computing for motion. Even simple differencing of two images grabbed at different time, creating a binary image by thresholding the differences, and then finding the centroid for tracking has been often used.

The design of a tracker can also be simplified by observing the world only when the robot head is not moving, finding out what is to be tracked, and then moving the robot head without paying any attention to the changes in the world. The "stop-and-look" mechanism excludes the idea of imaging while moving, and the use of sequential

operations is extremely easy to implement and manage. This system cannot continue to detect changes to any object while data is being analysed and the head is being moved to a new location. The fact that fast saccades in the human visual system do in fact suppress visual processing, up to 200ms, shows that this mode of operation is feasible. Given a sufficiently high-speed eye-head platform, such as Yorick [**?**], one can operate on pairs of frames, in a "stop-and-look" fashion.

2.2 Our Tracker

We have chosen to explore how to use dense optical flow and stereo data for tracking. The processing requires that pairs of frames have limited image displacements. Our simple eye-head system is slow compared to the speed that would be required to move stop to stop in time to acquire new information. In addition, we wish to be able to track continuously during slow motion of an object, observing all intermediate frames. The deficencies of not being attentive to the world at all time motivate our use of parallel processes so that we can eventually handle multiple moving objects. We continuously monitor the changes in the world so that we know exactly what had happened while our reasoning processes were busy working on something else. The effect of our decisions, first, to gather visual data during head movements, and, second, to identify the target solely on the basis of the current largest moving object, we must compensate for the apparent motion caused by the panning and tilting of the eye-head itself. We do this by estimating the image motion caused by camera motion and *cancelling* the apparent motion in the accumulated optical flow during a camera movement.

The tracker is composed of multiple stages:

- correlation motion and stereo on the Datacube
- accumulation of optical flow over multiple frames
- connected component labeling
- target selection
- control of the eye-head

The first two stages comprise the Perception component of the tracking system; note that it is distributed over two different machines. The next two stages form the Reasoning component, so termed since the input of the Perception component is analysed to determine the target to be tracked. The final computational stage translates the target location in image coordinates into controls for the eye-head. Figure **??** depicts the data flow in the system.

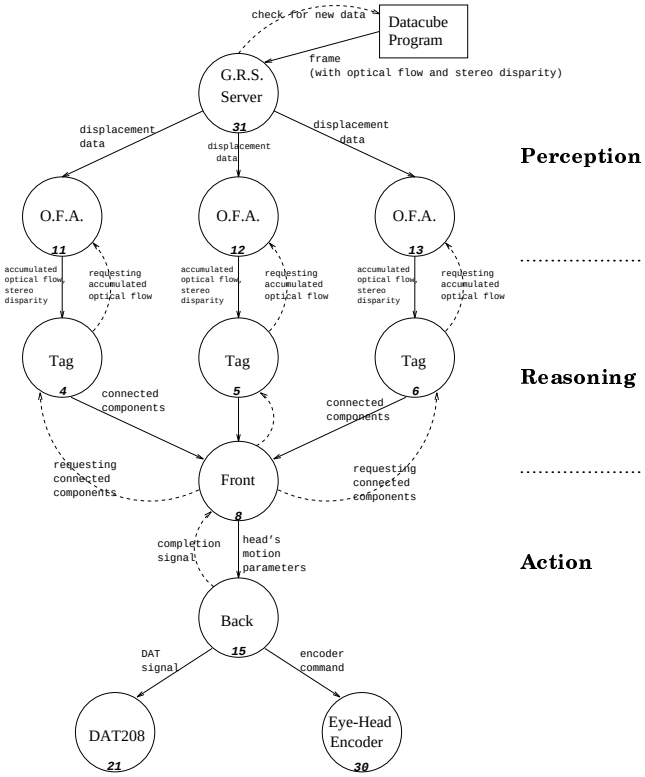

Figure 2: Software Components and Data Flow

2.2.1 Perception

The input images are grabbed from the pair of stereo cameras. The images are taken at 30 frames per second; we only use one field of the frame. Each field is 512x240; our Sony cameras average successive odd and even pairs so that we do not have missing data in the vertical direction. We smooth with a Gaussian before subsam-

pling to 128x120 [?]. An output image of size 128x512 is returned with the following four subframes, which are in order: optical flow, stereo disparity, edges (zero-crossings of the Laplacian of Gaussian), and Laplacian of Gaussian images (see figure ??). Each of these subframes has a default size of 128x120, which can be scaled to a lower resolution of 64x60 if needed.

The common range of motion used is [-2, +2] for vertical flow and [-3, +3] for horizontal flow, and the range for stereo disparity to be considered is usually [-13, +13]. Optical flow (a 2D vector) is encoded as a single scalar value at each pixel. The simple sum of absolute values of differences (SAD) correlation technique is used in measuring optical flow and stereo disparity. A 7x7 correlation window is used in the computation. Only the correlation range specified in the input data file will be considered. Optical flow [?] takes $(2d+1)^2$ passes through the images to compute a maximum displacement of magnitude d. Subsampling down from 480x512 has several benefits: it reduces the number of pixels, making the operations of the Datacube faster; it also reduces the maximum displacement so that there are fewer iterations. The matching program determines where the Laplacian of Gaussian image contains insufficient information, and marks the correlation result at that particular point. Optical flow and stereo are computed at 10Hz on the Datacube Maxvideo200 board.

Optical Flow	Stereo Disparity	Edges	$\nabla^2 G$

Figure 3: The Four Subframes in the Output Image of the Datacube Program

An essential component of our motion smart buffer is a a process responsible solely for receiving the motion data stream. This frame grabbing process, which we denote as the *active monitor*, waits for data to arrive, and keeps track of the data until it has been properly stored for later retrieval by the reasoning system. This observing or monitoring process is termed active since it is an independent process which has the initiative to grab a frame whenever it is available without having to wait for instructions to do so.

This particular observing process is simple but plays an extremely important role in synchronizing the communication and uniting the different modules. Optical flow and stereo disparity images will be pumped out continuously regardless of whether or not the reasoning system is prepared to process the data. It is extremely important that this observing process be executed at a rate *no slower* than the rate the displacement measures are being pumped out, or otherwise, the loss of any data frame might greatly contribute to the inaccuracies of the overall system. This is especially critical since we accumulate displacements over several frames. Figure ?? shows the structural relation between the active monitor and other vision processes.

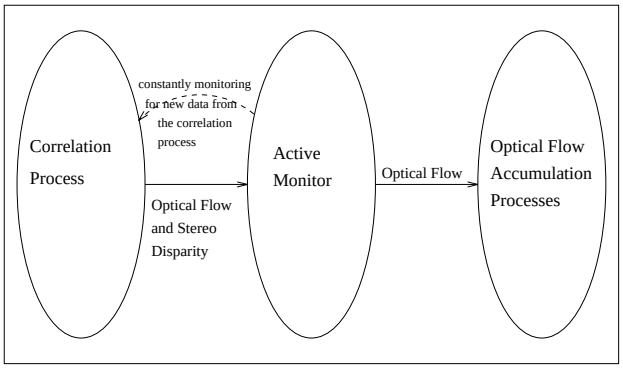

Figure 4: Perception System

The active monitor observes the data arriving from the Datacube at the Maxtran. The 128x512 data is transferred into the Maxtran memory by the monitor. A slight delay is incurred to synchronize image capture in the Maxtran to the video stream from the Datacube. The flow data is then downsampled into 64x60 to accomodate the speed of the downstream processing. The speed of the Transputer is such that it can just han-

dle downsampling the 128x120 data in 0.1s. We could downsample in the Datacube to 64x60 and operate at 15 Hz, but the motion data is more reliable at 128x120. At that resolution, smaller motions are not cut off by downsampling.

2.2.2 Reasoning: Determining the Target

The fundamental processing in the tracker is locating the target, the largest connected region moving with uniform velocity. For a static sensor, processing is confined to labeling a region of uniform motion, finding its centroid, and driving the eye-head to center that point in the visual field. Then processing can continue. Unfortunately, sensor motion induces optical flow. There are two solutions: first, stop and shoot, where the vision system becomes blind during motion, not unlike our own human visual system during rapid saccades; and, second, compensation for the known flow. We have adopted the second since it leads to a faster, more effective tracker.

Based on our control theories, a *cancellation* process is used to reduce the unstable effect of the background optical flow caused by ego-motion, before the segmentation process can work on partitioning the flow field. Such cancellation process is integrated into the segmentation process in our implementation, in the way that whenever the flow values of a pixel are retreived from the flow field, the expected background flow computed for such pixel will be subtracted from the flow values, resulting in a revised pair of horizontal and vertical accumulated optical flow.

The expected background flow values for each pixel is, in theory, calculated by multiplying the ego-motion parameters with the mapping values corresponding to the depth of such pixel. A special mapping table, indexed by stereo disparity, is recommended for use with the cancellation procedure. However, in our current implementation of the motion tracking system, *average mapping* values, one for vertical motion and another for horizontal, are used instead of a special table. This approach is adapted because of the difficulty of establishing a reference point for each depth, without knowing for sure that such a point will exist when the cameras are pointing at a random direction during initialization. It is also our belief that since cancellation will not completely zero out the background optical flow due to round off errors and correlation errors, using average numbers are as good in approximation and creation of "pop-out" effects as using a complete table, whose entries are also subject to contain errors due to the use of correlation matching. As a result, the initialization process for our tracker moves the head in a certain degree on each axis, and compute how much optical flow has been incurred, assuming that the whole scene is stationary. Such procedure will be repeated a number of times, using different degrees of motion, before the average mapping values are derived.

We have implemented a simple distributed memory connected component labeling program that processes the 64x64 in three horizontal strips of 22 rows each. Unfortunately processing occupies three to five frame times. This is the crucial multirate interface in the system. We insert a smart buffer here to connect the two processes. The buffer is embodied in an accumulation process that receives flow fields from the Maxtran and adds successive flows to the initial flow. This is relatively straightforward except for the fact that the data must be moved from the initial coordinate system (frame0) to the current frame and then added to the current displacement, at each step. To the connected component labeler (CCL), the smart buffer simply looks like a data object that contains the optical flow. To the image acquisition process on the Maxtran, it also appears as a repository for the stream of optical flow frames coming from the Datacube.

The interpretation cycle of CCL and the target selector can operate now during egomotion—when the process that moves the head is finished, it then requests the largest region from the CCL process, which then returns the accumulated optical flow. Before computing components, CCL subtracts the estimated flow due to egomotion from the accumulated flow. The estimated flow has been calibrated empirically. Then a correct flow relative to the new position of the camera

system can be computed. Particularly simple processing may then be applied, since the moving object should then "pop out" relative to a static background. We have not yet implemented the simplified processing but are using a full connected component analysis.

The full cycle, optical flow computation and accumulation, flow cancellation and target identification requires 800ms, mostly because of the time required by component labelling. An important constraint required to cancel apparent motion is that the eye-head complete its commanded movement before the reasoning system requests optical flow data from the smart buffer. If the motion is not complete, incorrect cancellation occurs. We are also limited in eye-head velocity by the velocity range of the correlation system so that the apparent motion of the background does not exceed the flow velocity limits. Currently the system is able to track a person moving at a normal walking pace 2 meters from the cameras.

The system can be improved in several ways. First, the optical flow computation on the Datacube can be speeded up, increasing the maximum velocity it can sense, allowing the head to move more rapidly. Also we are replacing the eye-head apparatus with a newer model, with significantly less backlash, that moves faster. Finally, by simplifying the analysis to determine a target we hope to reduce the total response time below 500ms.

3 Discussion

A smart buffer is a necessary component in a vision system where substantial early vision computation occurs at high rates and later stages operate at slower cycles. The buffer serves to synthesize a virtual data source that can be interrogated at any time by middle and high-level vision processes, independent of the processing cycle of the interpretation process or the data production cycle of the early vision process. It is implemented as an object that presents particularly simple interfaces both to the early modules and later modules. Like an object, it can hide significant internal processing that may depend on the model of the image acquisition process, such as noise elimination or stabilization. Unlike an interface built on a world model, the buffer keeps the data close to the source sensor model. Moreover, the buffer can be implemented simply so that it is fast.