In: Proceedings of the AAAI Fall Symposium on Sensory Aspects of Robotic Intelligence. Asilomar, California, November 1991.

Using Motion Vision for a Simple Robotic Task

John Woodfill and **Ramin Zabih**
Computer Science Department
Stanford University
Stanford, California 94305

Abstract

We are interested in real-time visual capabilities for robots in unstructured domains. We have developed motion, stereo, and motion-based tracking algorithms that run in real-time on a Connection Machine. Using a camera placed on a HERO-2000 robot, we have constructed a system that pans to follow a large moving object in the field of view. This system is capable of reliably tracking a moving person for several minutes. In this paper, we describe the visual and robotic components of our system, and review some of the challenges that face researchers attempting to construct robots with real-time visual sensing.

1 Introduction

We are interested in building autonomous robotic systems that operate in unstructured, dynamic environments. Sensing capabilities for such systems are a major research challenge, as it will be necessary to deal with a wide variety of possible events as they arise.

We have implemented a vision system suitable for robots in such domains. That the environments are unstructured, argues that neither model based approaches with limited model libraries, nor techniques that rely on special properties of the scene or of the object to be tracked (e.g., light objects against dark backgrounds) will suffice. That the environments are dynamic, suggests that real-time response will be critical. We have developed motion, stereo, and tracking algorithms with an emphasis on breadth of coverage and real-time performance.

Our implementations of these algorithms runs in real-time on a Connection Machine at Xerox PARC. We have mounted a camera on a HERO-2000 robot, that can pan under the control of the tracking algorithm. The robot has been operated several times in the PARC auditorium, and has reliably tracked a person walking around for several minutes.

We begin with a brief summary of our research goals, that is intended to justify the importance of unstructured dynamic environments. We next describe the vision algorithms and their integration with the panning robot. Finally, we survey some of the challenges of real-time vision for robotics in unstructured domains.

2 Research Goals

We believe that the central task of Artificial Intelligence is to produce autonomous artifacts. We are particularly interested in robots that interact with environments that are dynamic and unstructured. Many robots have been constructed that deal with some carefully structured domain, such as a conveyer belt or a simple, static environment. Unstructured domains, such as outdoor parks or offices shared with humans, are unpredictable; one cannot, for instance, assume that only a small known set of objects can be encountered.

It appears that robots without visual sensing will be capable only of very restricted interactions with their environments.[1] Visual sensors produce a two-dimensional projection of the robot's environment, which is far richer than that available from sonar or touch.

We are therefore interested in visual sensing for robots in dynamic, unstructured domains. We have chosen to concentrate on motion vision, and on tracking in particular. Motion is extremely important in a dynamic, unstructured environment; it is critical for detecting and avoiding moving obstacles. In addition motion can be a crucial cue for segmentation and manipulation of objects.

Tracking is a visual capability which is likely to be useful for a large number of purposes. Given the difficulty of real-time visual sensing, it seems implausible to construct an independent visual system for every robotic behaviour (as has been argued for under the rubric of "sensor fission" [2]). Instead, one will need some general purpose modules that can be used for several different behaviours. Tracking seems like a particularly useful such module.

[1]While it is true that bats, for instance, perform many interesting tasks, their sonar is significantly better than that possessed by commercially available robots.

3 Vision for a Robot Task

We have developed real-time motion, stereo, and motion based tracking algorithms for use in unstructured environments. We have used the motion based tracking algorithm as the vision component of a feed-back controlled panning camera system. We will describe the various vision algorithms, and then discuss the integrated system.

3.1 Visual Capabilities

The visual capabilities of our system can be divided into motion and stereo primitives that take images as input, and the higher level operations tracking and segmentation that use the results of the primitive computations. In this section we describe the primitive operations, followed by the tracking algorithm, segmentation, and finally implementation details. The discussion of motion, stereo and tracking is terse. A more complete discussion appears in [7].

3.1.1 Motion and Stereo

Given a temporally sequential pair of images, the motion computation must produce a dense motion field that determines for each point on the first image, a corresponding point on the second image (i.e., where it has gone to). The stereo depth task is similar, but starts with a stereo pair and produces a horizontal displacement map in which smaller displacements correspond to greater depth.

It seems unlikely that one can rely on the existence of stable distinctive features in natural scenes, which feature based approaches require. Instead, we use an area based computation that relies on correlating intensity values. Area based motion and stereo algorithms usually rely on two criteria to determine the most likely displacement for each pixel: how similar are the local area on the first image and its corresponding local area on the second image, and (because things in the world tend to cohere) how well do neighboring points agree on their displacement. These two criteria need to be combined in some way to uniquely determine motion at each pixel.

One approach to computing motion or depth fields is to define a global optimality criterion and then to optimize. Poggio [4] suggests, for example,

$$\iint (\nabla^2 G \circ (I_L(x,y) - I_R(x+D(x,y),y))^2) + \lambda(\nabla D)^2 \, dx\,dy$$

as a regularization functional for stereo depth. The left term measures the goodness of match, while the right term measures smoothness of the resulting disparities. A depth map can be generated from a stereo pair by numerical minimization.

Given our concern with real-time performance, our approach to combining the two criteria is direct. Initially, we find for each pixel on the first image the best corresponding pixel on the second image. This correspondence is determined using sum of squared differences correlation on intensity values for each point in a small local radius. The initial disparity estimate must be smoothed to enforce neighborhood agreement. The final disparity map is generated by determining for each pixel, the most popular initial disparity estimate surrounding the pixel. This smoothing step is similar to that used by Spoerri and Ullman for detecting motion edges [5]. Both the initial estimate and the smoothing step can be computed efficiently without iteration.

3.1.2 Tracking

The goal of tracking is to maintain an object's location across multiple images. Due to the need for real-time performance, the representations used in the vision system are all retinotopic maps at the same scale as the image. In particular a tracked object — the representation of the object being tracked — is merely an arbitrary set of pixels (sometimes called a boolean image).

The tracking algorithm is an iterative one. On each iteration, it takes as input a sequential pair of images and a tracked object, or boolean image, representing the location of an object in the first image. It produces as output a new tracked object representing the location of the object in the second image.

The tracking algorithm has three steps: computing a motion field, improving the estimate of the object's old location, and projecting the tracked object through the motion field. Computing a motion field is done by the motion computation described above.

Projecting the tracked object through the motion field is also a simple notion. The motion field is a map from pixels to pixels, that determines for each pixel on the first image, a corresponding successor pixel on the second image. The input tracked object is a set of pixels on the first image. Intuitively, the output tracked object, the result of projection, is the set of successor pixels of pixels in the input tracked object. However, complications arise because the motion field is not an automorphism; a pixel may be the successor of no pixels, or of two or more pixels, some tracked, and some untracked.

The need to improve the estimate of the object's location arises because the tracking algorithm is iterative. The input tracked object on one iteration results from previous motion computations and projections. The motion field tends to be slightly inaccurate, and projecting through the motion field tends to distort the tracked object. Improving the input tracked object is possible since an object moving in a scene will tend to produce discontinuities in the motion field at its perimeter. The adjustment step attempts to align the edges of the tracked object with these motion discontinuities.

This tracking algorithm has been shown to work on a large variety of real image sequences. It works both on indoor and outdoor scenes, although it is important that the scene have texture.

Because the tracking algorithm is applied iteratively, it needs to be initialized: the tracked object in the initial image must be somehow selected. Furthermore, the tracking algorithm occasionally loses the moving object, either because the object stops for too long, or due to camera noise, or for some other reason. Here too it is necessary to find the approximate location of a moving object so that the tracking algorithm can track it. We refer to this process as segmentation.

3.1.3 Segmentation

Segmentation, that is, picking out a moving object in the scene, is of critical importance to our system. The task cannot be performed by hand — there is no time. Nor can static segmentation cues such as intensity or texture be relied on, as we intend to deal with arbitrary moving objects. Our approach is similar to the use of gray-level histograms for segmentation [3], but we use motion rather than gray-levels as the segmentation modality. The pixels in a scene containing a moving object will tend to fall into two classes when grouped by their trajectories. The pixels corresponding to the moving object will tend to have moved along with the object, while the pixels corresponding to the rest of the scene will tend to have moved in opposition to camera motion.

The motion field generated from a single pair of images does not tend to distinguish the motions of the object from the rest of the scene clearly. However, adding up the motion vectors from several sequential pairs of frames to form cross temporal trajectories generates a reasonably clusterable histogram, provided that the object has been moving. Once the trajectories have been histogrammed, the motions under the largest peak are considered to be the motion of the background, and the motions under the second largest peak to be those of the object. Pixels that have exhibited the motions determined to correspond to the motion of the object are labeled as part of the tracked object.

This histogramming approach works quite well in practice, although it has some limitations. If the moving object is deforming, the motions corresponding to the moving object may fall under two or more peaks of the histogram. If the camera is dirty, an additional peak may appear due to perceived stationarity in low texture regions. We are exploring ways of integrating spatial coherence to mitigate these problems. Although we currently deal with only one object, there is no intrinsic reason for not segmenting and tracking multiple objects.

3.1.4 Implementation Details

We have implemented the above described algorithms on a 16 kilo-processor Connection Machine at Xerox PARC. The tracking algorithm working on 128 by 128 images processes 15 pairs of images per second including time for digitization and shipping images into the Connection Machine. The motion computation can detect motions of up to four pixels per image pair. This limit combined with the 15 frames per second processing speed, results in a maximum camera relative motion of 60 pixels per second.

There are two noteworthy details related to the speed of the algorithm and the operation of the Connection Machine. First, shipping images from digitizer memory into the Connection Machine is fairly slow; it accounts for approximately one third of our processing time.

Second, the more images that are processed at a time on the Connection Machine, the more efficient processing becomes. This economy of scale results from the fact that a serial front-end processor (in our case a Sun-4) sends instructions to the Connection Machine. The resulting per-instruction overhead can be amortized by processing multiple images simultaneously. Our algorithm runs most efficiently processing 8 or 16 pairs of images at a time. However, at 15 frames per second, this entails latency of more than one half or a whole second. Currently we process 4 pairs of frames at a time. This causes latency of about a quarter of a second between the time something happens in the world, and the time the tracking algorithm has finished processing the image.

Since stereo processing would entail shipping twice as much data into the Connection Machine as motion, and since there are technical problems with digitizing simultaneous stereo pairs, we currently do not use our stereo algorithm. Performance on stored data is around 50 stereo pairs per second.

3.2 Robotic system

We have incorporated the above motion-based tracking algorithm into a robot system that pans to keep a moving person centered in its field of view. Just as the task has a simple description, the basic robot control strategy is also simple. Whenever the tracking system has determined the centroid of the object it is tracking, it requests the robot to center the camera on this object centroid. In this section we describe the configuration of the whole system along with the Hero's primitive operations. We touch on the positioning of the camera and on relating camera rotations to pixel shifts in images. Finally we discuss difficulties arising from processing latency and time relative information, along with a solution to these difficulties.

3.2.1 System configuration

The robotic system is comprised of an RS-170 8-bit gray-level camera mounted on a Heathkit Hero-2000 robot with masking tape. The robot and camera are tethered to a Sun-4 that contains a Datacube digitizer, and a Connection Machine interface. Images pass from the camera to the digitizer. Digitized images are shipped into the Connection Machine. The results of the tracking computation are passed back to the Sun as object centroids. In turn, the Sun sends rotation commands to the Hero robot via a serial line.

3.2.2 Robot primitives

The Hero robot uses angles relative to the front of the robot base, termed ***home***, for specifying torso orientation. Since our current system only pivots about its center torso joint, and does not rotate its base, these angles can be considered absolute. Hero provides two basic orientation operations, one for setting the desired torso angle, and one for testing the current torso angle. Rotating the torso is slow, however, both setting and testing may be done asynchronously, before a previous torso rotation request has been fulfilled. A new rotation request supersedes any previous requests. Suppose, for instance, a request is made to turn 5 degrees left of home followed by a request for a rotation of 7 right of home. If the robot is homed to start with, this might result in an actual left rotation of 3 degrees followed by a right rotation of 10 degrees.

3.2.3 Camera mounting

If images are being captured while the camera is panned about an axis other than its center of projection, motion fields computed from these images will have multiple discontinuities arising from depth disparities and parallax. If the camera is panned about the center of projection, and if the camera parameters are sufficiently close to a pin hole camera model the resulting motion field is much simpler. A camera rotation of α degrees results in a uniform image shift of k pixels, for stationary parts of the scene.[2]

3.2.4 Rotation and image shift

Knowing the camera's field of view, and which subset of the pixels are being processed one can compute a Pixels/Degree coefficient p, that given a camera rotation α can predict an image shift of $k = p\alpha$ pixels. However, it may be difficult to determine the camera and digitization parameters, and the robot's rotation control may not be properly scaled. Instead, we rely on self-calibration. Our robot pans various fixed angles α and measures the image shift k produced by our motion computation. Each such pan results in an estimate $\frac{k}{\alpha}$ for the Pixels/Degree coefficient p. This is done several times to produce a good estimate for p.

[2]As an aside, this point argues for building stereo robot camera heads with parallel independently panning cameras.

3.2.5 Control

The task of the control system is to map the output of the tracking computation (object centroids) to robot rotation requests. Three factors in the tracking algorithm and robot combine to complicate this control task. First, the result of the tracking algorithm is the angle between the center of the tracked object and the principle axis of the camera when the image was captured. Second, the tracking algorithm, processing four pairs of images at a time, engenders a latency of about a quarter of a second. Finally, the Hero requires rotation requests in terms relative to home (the center of the robot base).

The result of the tracking computation is an object centroid in camera relative coordinates. The horizontal coordinate of the centroid can be straightforwardly mapped to a camera relative angle using the above Pixels/Degree coefficient. However, since the camera is moving, camera relative coordinates and angles are in turn relative to the orientation of the camera at the time when the image was captured.

By the time images have been processed, however, the camera may have rotated several degrees, and there is no way to map the result of tracking to a home relative rotation request which is what Hero expects. The information we have at the end of a quarter second tracking cycle (which is how long it takes for our algorithm to process four image pairs) is about how the world was a quarter second ago, and in terms of coordinates relative to where the camera was pointing a quarter second ago. The uncertainty about how the world may have changed since an image was taken is unavoidable. The problem arising from the mismatch between the dated visual information and robot's need for rotation requests in terms of the center of its base can be finessed.

In order to get around the temporal mismatch we have implemented an additional pair of rotation primitives. These are, first, a request to record the current heading (i.e., the current angle away from home); and second, an indexical rotation request that specifies an angle relative to the last recorded heading. If one records the current heading at the same time that one captures an image, then the system can request a rotation in terms of the position the camera had when the image was taken.

Thus, every four frames, the system records the current robot heading, and begins to process the four frames. When vision processing is done, and an object centroid is produced, the centroid is converted to the angle α between the camera's principle axis and the centroid of the object. Finally, a request is made to rotate the camera α degrees relative to the recorded robot heading. If this rotation request were to complete, the camera would end up pointing to where the tracked object was a quarter of a second ago. In the absence of any more recent information, and without extrapolating the object's motion, this would be the optimal place for the camera to point. However, a quarter of a second later, before most rotation requests could finish, a new object centroid is available, and a new request is issued.

4 Research Challenges

In the process of getting our robot to perform this relatively simple task, we have come upon a number of difficulties that are likely to confront other researchers attempting to use real-time vision in unstructured domains. In this section we will review some of the challenges that we have faced, and also point out some issues that we would like to see addressed in future research.

The challenges can be grouped into three classes. First, there are many practical difficulties involved in obtaining real-time vision; most of these are algorithm independent, and pertain to the conceptually simple task of getting image data into the system. Second, there are algorithmic challenges that must be addressed. Finally, there are a number of issues that involve robotics.

4.1 Practical challenges

Data acquisition has been one of the principle obstacles to building our system. Conceptually, the task is quite simple, but in practice it has proved to be difficult and time-intensive to solve. Present day cameras, digitizers and computer I/O bandwidth are quite unsatisfactory for real-time vision.

Current camera technology is problematic in a number of ways, largely as a consequence of the 1950's era RS-170 (NTSC) specification, which almost all cameras support. The RS-170 standard supplies a frame of data every 1/30 second. When digitized, such a frame looks like a two-dimensional array of 8-bit intensity values (the array is slightly larger than 512 by 512). However, not all the data in the array comes from the same time instant. This presents no problem for static scenes, but in dynamic environments it causes tremendous trouble.

An intensity value in an array of digitized video data represents the light level that some imaging element in the camera recorded over a specific time interval. Ideally, these time intervals would be the same for every element in a frame, but the RS-170 specification (as usually implemented) stands in the way. In a typical video camera, the time interval for the array elements in a single row are simultaneous (or very close). However, the even numbered rows of data come significantly earlier than the odd rows — usually by about 1/60 of a second. The RS-170 specification requires that the even numbered camera scan lines be read out before the odd ones. The way most cameras implement this specification results in the even lines coming from a different time instant than the odd ones.

In a dynamic environment, real changes occur in 1/60 of a second. Suppose that there is a hanging pendulum moving to the left, and consider what the digitized data will look like. In the even numbered rows of data, the pendulum will not have moved as far as in the odd ones. So the pendulum's appearance will be jagged; it will have moved farther to the left in the odd rows than in the even ones. This situation is a disaster for motion vision.

The best solution to this problem is to use either the even rows of data or the odd ones.[3] It is not unreasonable to use only half the available data, as cameras generate far more data than can be processed in real time. But a single field corresponds to a strange subset of the scene before the camera: if a point in the scene is present in that field, so is the point to its left or right, but not the point directly above or below! Correlation-based schemes in particular can have trouble with this, because an image feature found in one image may not even appear in the next image; if it is sufficiently small, it may reside on the other field, and thus disappear from view completely.

It is worth noting that cameras exist that do not interlace. However, these cameras are expensive, and the different pixels still don't all correspond to the same time instant.

Obtaining reasonable lighting conditions has also proven to be difficult. CCD cameras tend to be sensitive to infrared radiation, so outdoor scenes are often saturated unless an IR filter is used. Automatic gain control is provided by many cameras, but a large automatic change in gain from frame to frame will cause most motion computations to fail. The alternative is to set the gain statically, resulting in the obvious dependence on uniform light levels. We have also observed some peculiar effects that may be due to fluorescent lighting.

Camera lenses, and the imaging surfaces themselves, are inevitably dirty. This presents a problem in low-texture scenes, as real motion may be masked by dirt that appears as stationary texture. Cleaning imaging surfaces and lenses is the simplest solution, but it is hard to keep everything perfectly clean, and it's unsatisfactory to have a robot's vision system that fails unless the robot is kept in a Clean Room. Smoothing images can remove the effects of dirt, but may also have the effect of throwing real texture out as well. A third (perhaps preferable) alternative would be to dy-

[3]In RS-170 terminology, the entire array of data available in 1/30 of a second is called a *frame*, the even and odd halves of a frame are called *fields*, and the practice of reading out first the even and then the odd field is called *interlacing*.

namically compute a compensation function for each cell on the imaging surface.

Finally, our system has detected spurious motions that result from aliasing effects in scenes with sharp edges. An example of such an effect would occur with a black diagonal line on a white background. Consider two images taken as the camera pans across the scene. The first image might have a repeating pattern of pixels in varying shades of gray resulting from discretizing patches in the world that lie on the edges of the black line. On the second image, these patterns may again appear, but they will have shifted phase. The patterns that best match the first image may be above or below the original pattern, inducing a spurious motion estimate. This problem results from the discretization required to produce television images. Although we have not found any solutions to this problem, it is encouraging that people looking at such image sequences can also perceive this illusion.

Digitization is likely to be a major obstacle to real-time vision. Very few commercially available digitizers are capable of digitizing consecutive frames (in other words, running at 30 hertz). The ones that are tend to be expensive, and difficult to program.

Another problem with real-time vision is the enormous rate at which data is generated. A single 8-bit gray-level camera will generate 10 megabytes of data per second. It would not be unreasonable to want a stereo pair of color cameras, that would produce 60 megabytes per second. This is far too much data to process in real-time, even with a supercomputer. Instead, one must use subwindowing or subsampling, or a combination of the two.

It seems that images that are smaller than 128 by 128 simply do not contain enough detail for tasks such as tracking. Furthermore, most motion computations require that the time interval between images be relatively small. Another consequence of the RS-170 standard is that to obtain input data from a single frame evenly spaced it is necessary to run at a frequency that divides 30 hertz.

This entire discussion argues for a model of visual processing much more like Meade's proposed analog retina. Ideally, a network of small processors should be mounted as close to the imaging surface as possible. This would make it possible to perform a number of computations which cannot be done using current technology.

4.2 Challenges for algorithm design

The major difficulty from the algorithmic point of view is to design robust real-time algorithms. The necessity for real-time performance is obvious, but in an unstructured environment, robust performance is critical.

4.2.1 Robustness

We believe that robustness, in fact, is the key challenge for robotic vision. A robust vision algorithm is one that works in a wide variety of environments. This requirement is especially important if one is interested in autonomous robots that act in unstructured domains. A vision algorithm that is intended to find particular objects in highly constrained scenes can legitimately use highly specific properties of the objects and scenes. For instance, a vision system to detect broken computer chips can use the fact that the chips are rectangular, have a known size, or are imaged against a blue background. In an unstructured domain such assumptions are not justified.

Numerical parameters (such as thresholds) are a major source of non-robustness. Their use is inevitable, but their exact values make a great deal of difference to the performance of most algorithms. Unfortunately, the parameter values that cause an algorithm to perform well in one environment can cause failure in a slightly different environment. In unstructured domains, very little can be assumed about the inputs, so the parameter values need to produce good performance on a wide range of images.

We know of no theoretical tools that can measure vision algorithm robustness (i.e., the ability of an algorithm to perform in a variety of environments). Nor do we know of any theoretical tools that can aid in the design of such algorithms.

Synthetic data in our experience tends to be misleading; we have seen several cases in which algorithms that perform well on synthetic images yet fail miserably on real data. (Indeed, the description in section 4.1 suggests that it is hard to synthesize data that is as bad as what one actually gets from a camera.) Models of object shape or motion, surface reflectances and lighting conditions appear unlikely to obtain the broad coverage essential for unstructured domains.

In the absence of any theoretical tools, the only option left is empirical verification. If one is interested in vision in unstructured domains, this requires examining an algorithm's performance in many different environments. We have collected over 600 megabytes of data, and examined the performance of our tracking algorithm on tens of thousands of images.

Relying solely on empirical verification is quite unsatisfactory, but it is not clear how the situation can be improved. Robust algorithms are hard to design. It is difficult to verify that an algorithm has any robustness. Relying on purely empirical evaluation is always problematic, but especially so when the task at hand is ill-defined. For instance, one can expect to be able to judge if a stereo or motion algorithm is giving good results, as one has some idea of the physical properties of the scene. But for any higher-level task, such as tracking, performance is difficult to evaluate. For instance, by what firm standard can one determine that tracking is "failing", or worse yet "tracking the wrong object"?

Theoretical tools could conceivably alleviate some of these problems. However, given the kinds of tools that are available, we are not optimistic. Empirical verification of algorithms, while performing robotic tasks in as wide a variety of environments as can be found, seems to be the only alternative for the near future.

4.2.2 Real-time performance

Any vision system for a robot in a dynamic domain needs to run in real time. This presents two related difficulties. First, it is difficult to design an algorithm that runs that fast. Even when such an algorithm has been designed, the implementation has a tendency to be fragile: minor changes can easily destroy the real-time performance. Secondly, even when an such an algorithm is available, a great deal of unpleasant systems programming is required to produce a system with real-time performance.

There is another interesting property of the real-time constraint. For most motion algorithms, it furnishes a limit on how fast an object can move. Any scheme based on matching, such as gray-level intensity correlation or edge matching, will look for a match within a fixed radius r. The cost of the matching step is at least $O(r^2)$, so r must be limited to achieve real-time performance.

4.2.3 Task-directed visual algorithms

Another challenge for algorithm design is to somehow make use of the notion of task without entirely losing robustness. Most research in computational vision assumes that the goal of vision is to produce as complete and accurate a description of the environment as possible. But if vision is not seen as an end unto itself, but as a part of an autonomous system performing some task, certain problems may become easier. Taken to an extreme, this view results in entirely special purpose vision algorithms that (for instance) detect corridors in a particular building, or track green objects against non-green backgrounds.

However, it does seem that there is an intermediate point between taking vision as "the problem of deriving a symbolic description of a scene" [3], on the one hand, and writing entirely non-robust visual algorithms, on the other. [6] and [1] are two examples of worthwhile attempts to find a middle ground, but it remains to be seen if this will result in valuable algorithms.

4.3 Robotic challenges

Because the robotic component of our system is so minimal, we have not dealt with any difficulties that would bear discussion. However, in thinking about how to couple our vision system with more complex robots, we have noted some shortcomings in the way robot arms are currently built.

In general, robot arms seem to be designed to optimize precision and speed of motion. Given the state of the art in perception, this seems to be a poor design choice, especially if one is interested in unstructured domains. (To give the arm designers their due, they have mostly been interested is automatic assembly or similar tasks in highly structured domains, where their design decisions make a lot of sense.)

If real-time visual feedback is available, it seems that there is little point to precision, or at least to precision in terms of positioning. The data available from such visual sensing is likely to be quite noisy, at least in the near future, and it makes little sense to have an arm that can move precisely and quickly to the wrong place! Furthermore, it is desirable that the arm move slowly; as mentioned in section 4.2.2, the faster something moves the more computationally intensive it is to track. It seems that a slow-moving imprecisely controlled arm, which is what we want, would be far easier and cheaper to build than the currently available robot arms.

5 Conclusions

We have attempted to present a detailed example of what is currently possible in terms of robots with visual sensing for unstructured domains. Hopefully our experiences will provide other researchers with an idea of the problems that need to be overcome to advance to the next level of sensory competence for such robots.

5.1 Future work

In the near future, we intend to improve our panning camera's performance. In the slightly longer term, we plan to begin experimentation with an arm in order to perform more challenging robotic tasks. We are also exploring small scale parallel implementation of the vision algorithms to increase the practicality of the results.

Acknowledgements

We wish to thank John Goldman of Thinking Machines for technical support, and also Hal Moroff of Datacube. We have also received valuable advice from Harlyn Baker, Dan Huttenlocher, Jim Mahoney, and many members of Xerox PARC.

John Woodfill is supported by a Shell Doctoral fellowship. Ramin Zabih is supported by a fellowship from the Fannie and John Hertz Foundation. Additional financial support was provided by SRI through Harlyn Baker and by CSLI through Meg Withgott.

References

[1] J. Aloimonos, I. Weiss, and A. Bandopadhay. Active vision. *International Journal of Computer Vision*, 1:333–356, 1988.

[2] Anita Flynn and Rod Brooks. Building robots: Expectations and experiences. In *Proceedings of the IEEE International Robotics and Systems Conference, Tsukuba, Japan*, September 1989.

[3] Berthold Horn. *Robot Vision.* The MIT Press, 1986.

[4] Tomaso Poggio, Vincent Torre, and Christof Koch. Computational vision and regularization theory. *Nature*, 317:314–319, 1985.

[5] Anselm Spoerri and Shimon Ullman. The early detection of motion boundaries. In *International Conference on Computer Vision*, pages 209–218, 1987.

[6] Shimon Ullman. Visual routines. *Cognition*, 18:97–159, 1984.

[7] John Woodfill and Ramin Zabih. An algorithm for real-time tracking of non-rigid objects. In *Proceedings of AAAI-91, Anaheim, CA.*, pages 718–723. The MIT Press, 1991.