Real-time Motion Vision for Robot Control in Unstructured Environments

John Woodfill* Ramin Zabih† Oussama Khatib‡

Abstract

The control of robots in unstructured environments demands real-time, general purpose vision. We have designed a visual tracking system which makes use of motion to pick out and track moving objects without requiring information about their shape. The vision system has been implemented, and runs in real-time on a graphics accelerator in a Sun workstation. It has been used to provide visual feedback for a Puma manipulator controlled in operational space. The manipulator keeps a camera trained on a moving object. This robotic camera robustly follows people walking around in an unmodified room.

1 Introduction

Real-time sensor-based control of robots has been a major research theme in robotics. Visual sensing has enormous potential for providing information about the environment. However, computational vision has proven to be very difficult, especially under the constraint of real-time operation. As a result, most robots using real-time visual feedback are capable of operation only in highly structured environments. For unstructured environments, a more general purpose real-time vision system is required.

In this paper, we describe a general purpose vision system which performs real-time tracking of moving objects. By relying purely on motion information, the system is capable of finding and tracking objects whose shape is not known in advance, even when the camera itself is moving.

The visual capabilities of this system can be divided into two parts: motion primitives that take images as input; and the higher level operations, tracking and segmentation, that use the

*Interval Research, 1801 Page Mill Road, Building C, Palo Alto, CA 94304

†Robotics Laboratory, Computer Science Department, Stanford University, Stanford California

‡Robotics Laboratory, Computer Science Department, Stanford University, Stanford California

results of these primitive computations. The input to the vision system is a consecutive set of images, and the resulting output is a stream of binary images. For each input image, the output, called an *object bitmap*, indicates the image extent of the tracked object.

The basis for the motion measurement algorithm is a dense optical flow computation which is performed at every point in the image. The resulting flow field describes the motion of scene elements between consecutive images. This method can also be used to supply stereo depth information.

Once the optical flow computation is completed, a tracking algorithm uses the result to update the position of a moving object, while a segmentation routine uses it to identify the camera-relative extent of the moving object. To find and track an object, the robot control module merely requests that segmentation be used to find a moving object and passes along the result to be tracked. Since there is no requirement that the object's appearance be specified in advance, the system is well-suited for unstructured environments. In particular, it can handle objects of unknown or changing shape (i.e., unmodeled or non-rigid objects), as long as they remain in motion.

This vision algorithm has been implemented, and runs in real-time using a graphics accelerator on a Sun workstation. It processes 8-bit gray level images whose resolution is 128 by 128 at 10–15 frames per second, a rate that is suitable for tracking control. The system has been used to provide visual feedback for a Puma 560 manipulator controlled in operational space to keep a camera trained on a moving object. This robotic camera is capable of robustly following people walking around in an unmodified room.

We begin the paper with a survey of some related work, and then turn to the vision system. In section 3.1 we present the overall system architecture. There are three major components: real-time motion and stereo algorithms, described in section 3.2; an algorithm for tracking a moving object, described in section 3.3; and segmentation techniques for finding a moving object, discussed in section 3.4. Finally, we discuss the robotic camera and show some experimental results.

2 Related work

Prior knowledge of the objects to be seen or of their visual properties can be used to simplify vision tasks. Making use of such prior knowledge has allowed several researchers to build real-time vision capabilities. Lowe [5] has built a system that finds and keeps track of the three-dimensional (3-D) position and orientation of an articulated but otherwise rigid object. The pre-specification of the dimensions and articulations of the object allow the vision system to focus on salient details that are expected to be in the scene. Yamauchi [10] has constructed a system that allows a robot arm to juggle a balloon. The balloon is known to be of a dark color, and the environment is known to be light colored. The vision system can rapidly pick out the balloon in an image by looking for a dark region. The position of the balloon in two binocular images can be used to determine the 3-D position of the balloon with ease.

These vision systems work exceptionally well in dynamic environments for objects that have been previously specified. However, in a natural, unstructured environment, there can be no exhaustive list of specifications of all the objects that can be seen. Nor can it be the case in an unstructured image sequences that all objects of interest can be distinguished by some previously specified set of visual properties. A vision system for unstructured image sequences must have additional capabilities that do not depend on prior knowledge of a very restricted subject matter.

Although much work has been done on computer vision for natural, unstructured scenes, little has addressed issues of real-time performance. We return to the topic of general real-time vision capabilities at the end of section 3.3.

3 Real-time motion tracking

In this section we describe a vision system that picks out a moving object and keeps track of its camera-relative extent in the scene. The system is general enough to be suited in unstructured environments, as there is no requirement that the object's appearance be specified in advance. In particular, the system can handle objects of unknown or changing shape (i.e., unmodeled or non-rigid objects), as long as they are moving.

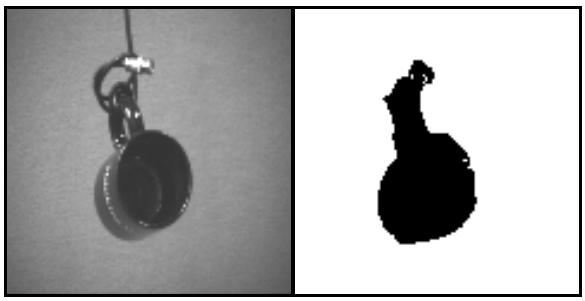

Figure 1: An example of the input and output of the tracking system: a swinging mug

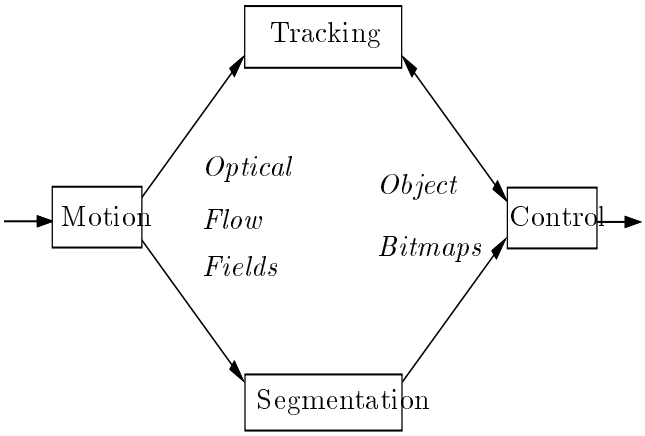

Figure 2: Vision system architecture

The input to the vision system is a stream of images. The output is a stream of binary images. For each input image, the output, called an *object bitmap*, indicates the image extent of the tracked object. A typical image depicting a swinging mug is shown in figure 1, along with the corresponding output.[1]

3.1 System architecture

The visual capabilities of our system can be divided into motion primitives that take images as input, and the higher level operations tracking and segmentation that use the results of the primitive computations. The overall architecture of the vision system can be seen in figure 2. Arrows signifying data paths show that information flows primarily in a single direction, from images (which enter at left) to object centroids (which exit at right).

The motion measurement algorithm computes a dense optical flow field, which describes the motion of scene elements between consecutive images. Tracking uses the optical flow fields to update the position of an object as it moves, while segmentation uses them to identify a moving object. The control module tells segmentation when to look for moving objects, and passes the results along to be tracked.

First we describe the motion measurement algorithm, and then turn to tracking and segmentation. A more complete discussion of the algorithms appears in [9].

[1] The vision system is capable of tracking several objects at the same time, but for simplicity of explanation we will focus on a single tracked object.

3.2 Motion measurement

Given a temporally sequential pair of images, the motion computation must produce a dense optical flow field that determines for each pixel in the first image a corresponding pixel in the second image (i.e., where it has moved). The stereo depth task is similar, but starts with a stereo pair of images and produces a depth map that determines for each pixel on one image a corresponding, laterally displaced pixel on the other image.

Our motion and stereo computations use an area based approach 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 objects in the world tend to cohere) how well do neighboring points agree on their displacement. These two criteria need to be combined to 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 [8] suggests, for example,

$$\iint \left[\nabla^2 G * (I_L(x,y) - I_R(x + D(x,y), y))^2 + \lambda(\nabla D)^2 \right] dx\, dy$$

as a regularization functional for stereo depth, where I_L and I_R are the left and right images, D is the disparity map, λ is a constant, G is a Gaussian and $*$ denotes convolution. The first term measures the goodness of match, while the second 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 involves no iteration. Our algorithm has two phases: first, we find a good initial motion estimate using Sum of Squared Differences (SSD) correlation; then we smooth these motion estimates using mode filtering.

Initially, we find for each pixel on the image I_k the best corresponding pixel on the next image I_{k+1}. This correspondence is determined using SSD correlation on intensity values for each point in a small local radius. For two consecutive images I_k and I_{k+1}, given a correlation window Δ, we compute an SSD measure of the dissimilarity of the pixels p and p'

$$E(p,p') \equiv \sum_{\delta \in \Delta} (I_k(p+\delta) - I_{k+1}(p'+\delta))^2.$$

For each pixel p in image I_k, the result of initial motion estimation is a pixel p' in image I_{k+1} lying within a small local radius of p, that minimizes $E(p,p')$. In the current implementation the local radius is an ellipse containing 37 pixels.

Our motion computation uses a correlation window Δ that consists of only 5 pixels (the center pixel and its 4-connected neighbors). This is considerably smaller than typical correlation windows. The small window size allows us to handle rotations and non-uniform motions.

The disadvantage of a small window size is that it can give noisy results. We handle this by a second stage in our algorithm, where the initial motion estimate is smoothed to enforce neighborhood agreement. The final optical flow field is generated by determining for each pixel the most popular initial disparity estimate surrounding the pixel. This step, called mode filtering, preserves discontinuities at edges while smoothing the optical flow fields to reduce noise.

The output of the motion measurement algorithm is an optical flow field, which can also be viewed as a map from I_k to I_{k+1}. Both phases of the motion computation can be computed efficiently without iteration.

The stereo algorithm is analogous, except that the search window is linear rather than elliptical. Its output is a dense disparity field in which nearby scene elements are represented by large disparities and distant elements are represented by small disparities.

Other researchers such as Coombs [1] and Matthies [6] have obtained real-time stereo using image-processing boards from Datacube. Nishihara [7] and Inoue et al. [3] use custom hardware to implement real-time correlation for motion (as well as stereo in Nishihara's case). Inoue's work is particularly close to ours in its reliance on gray-level correlation (although the correlation measure he uses is $|x-y|$ rather than $(x-y)^2$). Nishihara relies on binary correlation of the

sign bit of the result of convolving with a Laplacian. Our approach differs from theirs in our use of small correlation windows and mode filtering. In addition, we produce a motion estimate at every point in the image, while Inoue and Nishihara produce a sparser, but still dense, output.

Computing optical flow is necessary for picking out moving objects and for determining the correspondence between image components. However, tracking of a moving *object* requires more than individual motion estimates; it requires some amount of cross-temporal aggregation and integration.

3.3 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 an object bitmap, or tracked object — the representation of the object being tracked — is merely an arbitrary set of pixels.

The tracking algorithm is an iterative one. On each iteration, it takes as input an optical flow field showing the motion between images I_k and I_{k+1}, and an object bitmap representing the location of an object in image I_k. It produces as output a new tracked object representing the location of the object in image I_{k+1}.

The tracking algorithm has two steps: projecting the tracked object through the optical flow field, and improving or adjusting the object bitmap by using regions of similar motion. We will describe each in turn.

Projecting the tracked object through the optical flow field is a simple notion. The optical flow 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. The output tracked object, the result of projection, is the set of successor pixels of pixels in the input tracked object.

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 optical flow field tends to be slightly inaccurate, and projecting through the optical flow field tends to distort the tracked object. Improving the tracked object is possible since an object moving in a scene will tend to produce discontinuities in the optical flow field at its perimeter. The adjustment step attempts to align the edges of the tracked object with these motion discontinuities. It makes use of local consensus among like-moving pixels to determine membership in the tracked object.

In order for tracking to begin, moving objects must be picked out. Furthermore, the tracking algorithm occasionally loses a 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 resume tracking it.

3.4 Finding moving objects

Segmentation, that is, picking out a moving object in the scene is a key element of the vision system. Static segmentation cues such as intensity or texture cannot 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 [2], 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 motion vectors. 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 optical flow field generated from a single pair of images may not distinguish the motions of the object from those of the rest of the scene clearly. However, composing the motions 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. Objects must move over the course of several frames in order to be found. In addition, the

segmentation scheme is theoretically restricted to certain camera motions, namely rotations about the horizontal and vertical axes passing through the camera's center of projection.

We have explored a more general algorithm for picking out moving objects, however it has not yet been implemented to run in real-time. By utilizing the results of the stereo depth computation, moving objects of unknown shape can be picked out under fairly arbitrary camera motion.

The combination of a motion flow field and a stereo depth map produces a 3-D motion vector at every point in an image. Each 3-D motion vector is consistent with a number of rigid motions of an object in 3-space. The individuation algorithm partitions a set of 3-D motion vectors into subsets such that each subset is consistent with a single common rigid motion of an object. This can be done efficiently without ever reconstructing any of the constituent rigid motions.

By making use of some distance geometry, a set of points that are undergoing a single motion can be picked out from a set of n points in $4n$ simple operations. The underlying idea of the approach is that the inter-point distances between any set of points on a rigid object will stay constant in successive frames regardless of the motion the object is undergoing. Distances between points on the object and points in the background will in general not remain constant from frame to frame (since the object is moving with respect to the background). This algorithm for picking out moving objects is described in [11] in detail.

3.5 Implementation notes

The algorithms we have developed are well-suited for efficient implementation because of their uniform and local nature. We make extensive use of a form of dynamic programming (sometimes referred to as *sliding sums*) to share intermediate results between neighboring pixels.

The tracking algorithm runs on a VX/MVX graphics accelerator in a Sun workstation. The VX/MVX contains 5 Intel i860 processors, each of which is capable of 40 MIPS. The motion computation is done on 4 processors, which divide the image into (overlapped) slices. The remainder of the computation is done on a single processor.

The tracking system has been shown to work on a large variety of natural image sequences. It works both on indoor and outdoor scenes, although it is important that the scene have texture. The tracking system has been validated by mounting a camera on a PUMA robot arm and controlling camera rotations using tracking. The combined system, described in more detail in section 4, picks out an object and tracks it, rotating the camera in order to keep the object in the field of view.

Although the stereo algorithm has not yet been incorporated into the real-time system, it runs at approximately 8 Hz on 128 by 128 images on a uni-processor 40Mhz Sun Sparc-10.

4 The robotic camera

We have used the visual tracking system to provide real-time feedback for a robotic camera. We have configured a Puma 560 to rotate a camera to follow a moving object tracked by the vision system.

The architecture of the robotic camera is shown in figure 3. The camera's video signal is sent to the vision system, which runs on a VX/MVX graphics accelerator. The vision system computes the trajectory of the object's centroid, and send it via shared memory to a Motorola 88k which is used to control the robot. The robot motion controller has been implemented using COSMOS, an object-level control system based on the operational space approach [4]. The operational point is the camera center of projection.

Figure 4 shows a sequence of images taken from a camera mounted on the moving robot arm. The white contour shows the outline of the tracked object. In this sequence the camera is panning to the right to follow the person.

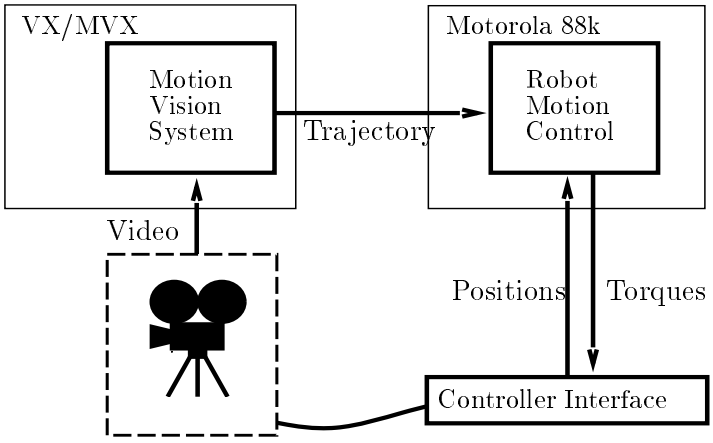

Figure 3: The robotic camera system

4.1 Handling latency

One complication arises in the control of the robot, due to the latency involved in the vision system. By the time an object centroid has been computed, as much as a quarter second may have passed. So the object centroid that is emitted by the tracking service specifies where the object was relative to where the camera was pointed when the image was captured a quarter second ago. There is no way to get around the fact that the information is old. However, given that the camera may be panning, information about object positions relative to the camera heading of some time ago is quite useless, unless the camera heading at the time of image capture is known.

Thus, the robot systems are set up so that whenever an image is captured, the current heading of the camera is also recorded. When a new object centroid is produced, it is combined with the recorded camera-heading information to compute a heading that would have been correct when the centroid was computed. This heading is out of date since it says how the camera should have pointed when data that produced the current centroid was captured. However, it is the best information available, and hence is specified as the current desired heading for the camera.

An alternative scheme would involve predicting the object's position, based on its velocity or perhaps its history. We have not yet investigated prediction-based approaches.

5 Conclusions

We have described a real-time vision system which can track objects based on their motion. This provides a basic capability for visual feedback, which can be used for controlling robots. In addition, we believe that our tracking system can supply a bottom-level visual service, and that higher level visual capabilities can be constructed that make use of the information our system supplies. We hope that the system's ability to handle unmodeled non-rigid objects, combined with its real-time performance, will make it suitable for robot control in challenging unstructured environments.

Acknowledgments

John Woodfill has been supported by an NSF postdoctoral grant and a fellowship from the Shell Corporation. Ramin Zabih has been supported by a fellowship from the Fannie and John Hertz Foundation. We wish to acknowledge additional financial support from Xerox PARC, CSLI, SRI and CIFE. We are grateful to Harlyn Baker, David Heeger, Dan Huttenlocher, and Jim Mahoney for useful comments in the course of this work.

References

[1] David Coombs and Christopher Brown. Real-time smooth pursuit tracking for a moving binocular head. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 1992.

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

[3] Hirochika Inoue, Tetsuya Tachikawa, and Masayaki Inaba. Robot vision system with a correlation chip for real-time tracking, optical flow and depth map generation. *ICRA*, pages 1621–1626, 1992. Nice, France.

[4] Oussama Khatib. A unified approach for motion and force control of robot maniuplators: The operational space formulation. *IEEE Journal of Robotics and Automation*, RA-3(1):43–53, February 1987.

[5] David Lowe. Integrated treatment of matching and measurement errors for robust model-based motion tracking. In *Proceedings of International Conference on Computer Vision*, pages 436–440, 1990.

[6] Larry Matthies. Stereo vision for planetary rovers: Stochastic modeling to deformation of image curves. *International Journal of Computer Vision*, 8(1), 1992.

[7] H. Keith Nishihara. Practical real-time imaging stereo matcher. *Optical Engineering*, 23(5):536–545, Sept–Oct 1984. Also in *Readings in Computer Vision: Issues, Problems, Principles, and Paradigms*, edited by M.A. Fischler and O.Firschein, Morgan Kaufmann, Los Altos, 1987.

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

[9] John Woodfill. *Motion Vision and Tracking for Robots in Dynamic, Unstructured Environments*. PhD thesis, Stanford University, August 1992.

[10] Brian Yamauchi and Randal Nelson. A behavior-based architecture for robots using real-time vision. In *IEEE International Conference on Robotics and Automation*, pages 1822–1827, 1991.

[11] Ramin Zabih. *Individuating Unknown Objects by Combining Motion and Stereo*. PhD thesis, Stanford University, 1993 – Forthcoming.

Frame 1 Frame 2 Frame 3 Frame 4

Frame 5 Frame 6 Frame 7 Frame 8

Frame 9 Frame 10 Frame 11 Frame 12

Frame 13 Frame 14 Frame 15 Frame 16

Frame 17 Frame 18 Frame 19 Frame 20

Figure 4: Tracking results