Robotics for Challenging
Environments, Proc. ASCE '94
New York, pp. 10-18

Real-time Motion Vision for Robot Control

John Woodfill, Ramin Zabih and Oussama Khatib

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

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 real-time constraint. As a result, most robots that have used real-time visual feedback have depended upon highly structured environments. For an unstructured environment, real-time general purpose vision is required.

In this paper we describe a general purpose vision system which performs real-time tracking of moving objects. The vision system is capable of finding and tracking objects whose shape is not specified in advance, by relying purely on their motion. This can be done even from a moving camera.[1]

The vision system runs in real time on a graphics accelerator in a Sun workstation. It processes 8-bit gray level images whose resolution is 128 by 128 at 10–15 frames per second. The basis for the motion tracking is a dense optical flow computation which is performed at every point in the image; this can also be used to supply stereo depth information.

[1]It is, of course, easy to detect a moving object from a stationary camera by looking for intensity changes.

In this paper we begin 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; a technique for finding a moving object, discussed in section 3.4; and an algorithm for tracking a moving object, described in section 3.3. Finally, in section 4 we describe the use of the vision system to control a robot camera.

2 Related work

Because real-time computer vision is so difficult, most visually controlled robots have relied on highly structured environments. These robots have often performed impressive tasks, such as playing ping-pong [1] or juggling a balloon [10]. However, the visual task that these robots handle is extremely simple binary vision; they find a white object against a black background. These robots are thus limited to carefully constructed workspaces.

However, many potentially useful robotic tasks require real-time visual feedback in unstructured environments. Examples of such tasks include construction, autonomous outdoor navigation and office cleaning. These tasks require real-time general purpose vision.

Some recent attempts have been made to provide real-time visual capabilities by using special-purpose hardware. Inoue *et. al.* [4] and Nishihara [7], for example, have used special-purpose processing chips for real-time stereo and motion. A related approach has been taken by researchers such as Coombs [2] and Matthies [6], who have obtained real-time stereo using image-processing boards from Datacube.

3 Real-time motion tracking

In this paper we describe a vision system which picks out a moving object and keeps track of its camera-

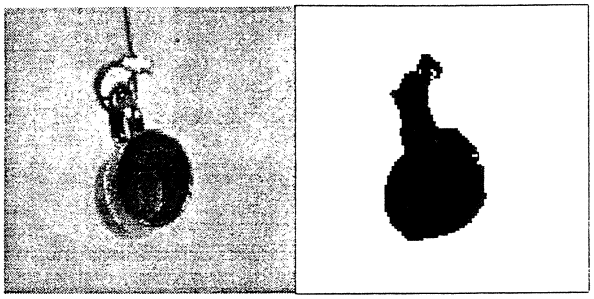


Figure 1: An example of the input and output of the tracking system

relative extent in the scene. The system is general enough to be suited for unstructured environments, as there is no requirement that the object's appearance be specified in advance. In particular, it can handle objects of unknown or changing shape (i.e., unmodeled or non-rigid objects), as long as they are moving.

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, with the corresponding output, is shown in figure 1, which shows a swinging mug.[2]

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 requests that segmentation find a moving object, and passes the result along to be tracked.

We will describe the motion measurement algorithm (which also computes real-time stereo depth fields) first, followed by tracking and then segmentation.

[2]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.

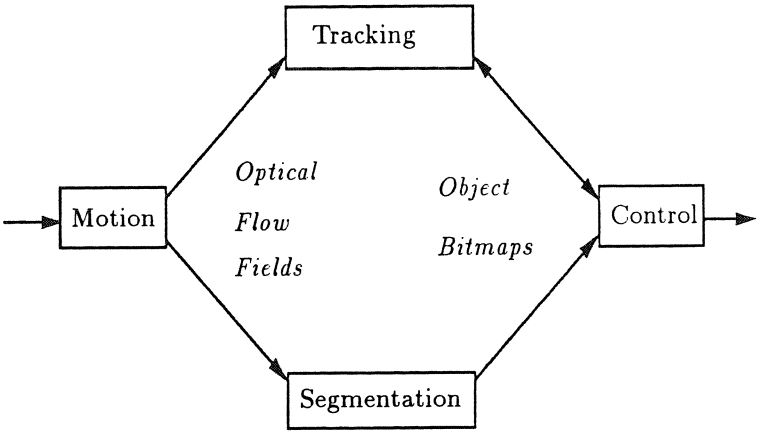


Figure 2: Vision system architecture

The discussion of motion, tracking and segmentation is terse. A more complete discussion appears in [9].

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 scene element in the first image, a corresponding scene element in the second image (i.e., where it has moved). 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.

Our motion and stereo computation 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 things 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,

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

as a regularization functional for stereo depth, 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 is direct. 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'

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

The SSD motion estimate will be for the pixel p to move to the pixel p' within a local radius of p minimizing $E(p,p')$. (In our 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 results that are not smooth. 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.

It is worth noting that Inoue [4] and Nishihara [7] also rely on correlation to produce real-time motion. Inoue's work is particularly close to ours in its reliance on gray-level correlation (although his correlation measure is $|x-y|$ rather than $(x-y)^2$, which we use). Nishihara relies on binary correlation of the sign bit after convolving with a Laplacian. Both approaches differ from ours in their use of large correlation windows. In addition, we produce a motion estimate at every point in the image, while Inoue and Nishihara produce a sparser (but still dense) output.

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 a tracked object — the representation of the object being tracked — is merely an arbitrary set of pixels (sometimes called an object bitmap).

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 the object bitmap by using motion boundaries. 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.

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 approxi-

mate location of a moving object so that the tracking algorithm can track it. We refer to this process as segmentation.

3.4 Finding moving objects

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 optical flow 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, 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 a couple of seconds in order to be found. In addition, the segmentation scheme is restricted to certain camera motions, namely rotations about the camera center of projection. We believe that with real-time stereo it will be possible to handle arbitrary camera motions in the near future.

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 dynamic programming to share intermediate results between neighboring pixels.

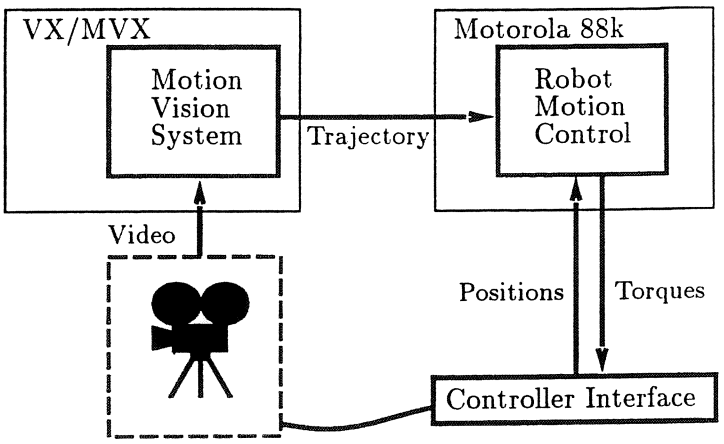


Figure 3: The robotic camera system

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.

Figure 4 shows a sequence of images captured while the robotic camera was operating. The white line shows the outline of the tracked object. In this sequence the camera is panning to the right to follow the person.

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 [5]. The operational point is the camera center of projection.

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.

Acknowledgements

John Woodfill has been supported by 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. Alain Fidani and Jon Goldman provided help with the robots.

References

[1] Russell Andersson. *A Robot Ping-Pong Player: An Experiment In Real-Time Intelligent Control.* MIT press, 1988.

[2] 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.

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

[4] 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.

[5] 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.

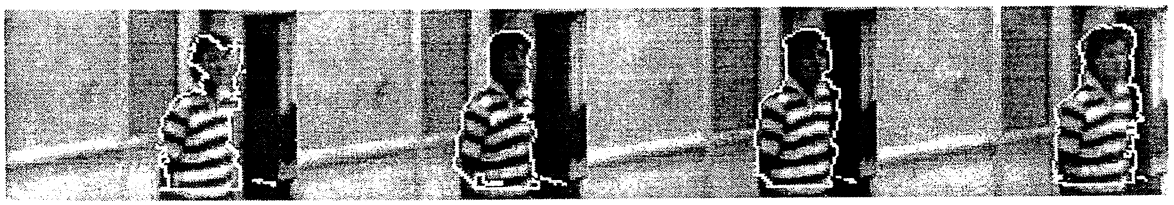
[6] Larry Matthies. Stereo vision for planetary rovers: Stochastic modeling to deformation of image curves. *International Journal of Computer Vision*, 8(1), 1992.

[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.



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: Robotic camera data