MOTION VISION AND TRACKING FOR ROBOTS IN DYNAMIC, UNSTRUCTURED ENVIRONMENTS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
John Iselin Woodfill
October 1992

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Michael Genesereth
(Principal Adviser)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Oussama Khatib

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Harlyn Baker
(SRI International)

Approved for the University Committee on Graduate Studies:

Abstract

The use of robots in dynamic and unstructured environments poses difficult challenges for machine vision. The vision systems for such robots must report changes in the world quickly and must deal with objects that have not been previously seen or specified.

A promising approach to building such vision systems is to construct generally applicable, modular vision services that compute summarizations of camera data that are of direct utility for action. In this thesis it is argued that such modular vision services can serve to connect machine vision to robots in dynamic, unstructured environments.

The thesis describes a real-time vision service that picks out and monitors non-rigid moving objects in natural scenes. The tracking service uses correlation-based optical flow for both picking out, and monitoring the camera-relative extent of moving objects. The algorithms that comprise this service have been implemented on both massively parallel SIMD, and small-scale parallel MIMD hardware. As these algorithms are intended to be used for real-time robot systems, the algorithms and their complexity are described in detail.

The tracking service has been used to build two camera-panning robot systems. Whenever a large object moves into view, the object is picked out and tracked, while the robot pans the camera to keep the object centered in the field of view. The robots have successfully tracked people walking around for long periods. These functioning robots demonstrate both the utility of the tracking vision service itself, and the manner in which such vision services can be integrated into complete systems.

Acknowledgements

I would like to thank my thesis adviser, Michael Genesereth, and the members of my reading committee, Harlyn Baker and Oussama Khatib.

I am deeply indebted to Ramin Zabih with whom I have been collaborating for more than three years. Much of the work described in this thesis is the result of our joint efforts.

Devika Subramanian rekindled my enthusiasm for A.I. several times. Terry Winograd, Jim Mahoney, and Brian Smith made it difficult to embrace the more paradoxical elements of A.I. and machine vision.

I am grateful for the three-year doctoral fellowships provided by the NSF and the Shell Corporation, and tow years of summer support from SRI International. Xerox PARC provided extensive access to the Connection Machine and other facilities.

Ramin Zabih, Karin Meyer and Harlyn Baker read drafts of the thesis beyond the call of duty.

Ana Haunga was extremely supportive during a period of grief. Officemates Yungjen Hsu and Eric Berglund provided an atmosphere of solidarity.

My family, Jacqueline Woodfill, Celia Woodfill, Thomas Woodfill and the late Walter Woodfill deserve thanks.

My friends and housemates kept me sane: Ned Black, Jobst Brandt, Karen and Lonn Johnston, Elsbeth Meyer, Marc Meyer, Kate Morris, Jack Newlin, Giovanna Petrone, Lara and Des Garner, John and Alice Kenney, Elgar and Pegaret Schuerger-Pichler, Marion Sturtevant, and Bob Walmsley.

Finally, I wish to embrace Karin Meyer who has given love and warmth for more than seven years.

— De quoi t'occupes tu exactement?

— De la réîfication.

— Je vois, c'est un travail trés sérieux,
avec de gros livres et beaucoup de papiers
sur une grande table.

— Non, je me promène.
Principalement je me promène.

Quoted in Greil Marcus' *LIPSTICK TRACES* [63]
from André Betrand's "Le Retour de la colonne Durutti"
as published in "Ten Days That Shook the University"

Contents

List of Figures

Chapter 1

Introduction

One goal of Artificial Intelligence is the construction of autonomous robots that can move about performing useful tasks in our everyday world. A robot that is to fetch a toy from the city park must be able to avoid a fast-moving bicycle, just as it must not be confounded by a troop of halloween-costumed youths.

If vision is to be a significant component in such robots, it must produce relevant output quickly, and its performance must not degrade when normal but heretofore-unseen objects appear. The need for fast response from vision arises from the *dynamic* nature of environments such as the city park; a fast-moving bicycle may burst on the scene. The need to deal with unrecognized or unmodeled objects arises from the *unstructuredness* of these environments; halloween costumes span an arbitrary range of shapes, sizes and appearances.

A promising approach to building vision systems for such robots is to construct generally applicable, modular *vision services* that compute summarizations of camera data that are of direct utility for action. This thesis introduces the idea of vision services as an approach and describes an implemented, real-time tracking service in detail. The described vision service uses optical flow for picking out and monitoring the camera-relative extent of moving objects in natural scenes. It has been used to implement two systems that pan a camera to follow a moving object.

This introductory chapter discusses two paradigms of machine vision that individually appear to be insufficient for robots in dynamic, unstructured environments,

model-based vision and surface reconstruction. Next, the approach of vision services is presented. As an instance of the vision services approach, and as the subject of this thesis, the general capabilities of the tracking service are presented. Validation is discussed since vision services present both novel issues for validation, as well as partial solutions to these issues. Having introduced both the idea of vision services and a particular vision service, the contemporary situation of this work is laid out. The chapter concludes with a summary of the contributions of the thesis and a reader's guide.

1.1 Approaches to vision

Model-based vision is one paradigm of machine vision that has allowed robots to act using visual input in dynamic environments. In model-based vision, prior knowledge of the objects to be seen or prior knowledge of distinctive visual properties of the objects to be seen is used to simplify vision tasks. A model-based vision system that knows what object is to appear in the scene can focus its resources on trying to find that object and reporting on its whereabouts. Such prior knowledge allows the vision system to access relevant visual information rapidly.

Lowe [60] 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 prespecification 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 [103] 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 an 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 environment, that all objects of interest can be

distinguished by some previously specified set of visual properties. A vision system for dynamic, unstructured environments must have additional capabilities that do not depend on prior knowledge of a restricted environment.

Surface reconstruction is another paradigm of machine vision that differs from model-based vision in that it makes minimal assumptions about the environment. Instead, the goal of surface reconstruction is to extract as much information as possible about the 3-D structure of the world in front of the camera or cameras, without recourse to knowledge about the particular scene.

David Marr [64] was an early advocate of surface reconstruction as a step towards vision systems with human-like capability. He proposed building a "$2\frac{1}{2}$-D sketch" that encodes the orientation and depth of visible surfaces. This surface model could then be used as a kind of map for determining the shapes and positions of things in the scene.

In an unstructured, static world, building up a surface model of the environment has great utility. For example, a mobile robot could form a surface model of the environment in front of it, and plan a route through the part of the scene that is in view. The system would work regardless of what kinds of objects were in the scene.

However, surface reconstruction can only be of limited help in dynamic environments. Building and maintaining an up-to-date surface model may be computationally prohibitive in a changing environment. As environments can be complex, so too can surface models be complex data structures. Extracting relevant information from a rapidly changing surface model may be comparable in difficulty to extracting relevant information from an intensity image. A vision system for dynamic, unstructured environments must have additional capabilities beyond the ability to construct surface models at widely spaced intervals.

To summarize, model-based vision can be sufficient for limited dynamic environments in that it can directly produce relevant information that can be used for action. It is insufficient for unstructured environments since it presupposes knowledge of the kinds of things in the scene. Surface reconstruction is sufficient for unstructured, static environments since it assumes nothing about the scene. Yet, it is insufficient for dynamic environments as it is unlikely that model construction and access can

keep pace with the world.

1.2 Vision services

I propose that the vision needs of robots in unstructured domains can be met by constructing general, modular *vision services*. Three principal properties characterize a vision service: a vision service *summarizes* camera data in a form that can be directly used for action; the summarization is *general* in that it does not depend on prior knowledge of objects or of the environment; a vision service is *taskable* in the sense that once started, it can continue to perform a service for long periods with minimal input other than images. The idea of vision services is similar to Ullman's notion of visual routines [96], in that it emphasizes computing useful properties directly from camera data. It is dissimilar in that a vision service is an ongoing process, working on a stream of images, and that the results of a vision service are intended to be used for acting rather than for further image interpretation.

To clarify the concept and motivation behind vision services, imagine the control component of a general purpose robot as a stupid, slow, perceptually under-equipped homunculus. The homunculus has a set of single-purpose knobs and buttons that control its effectors, a panel of multi-digit numeric displays that tell it what is being perceived, and even a few buttons for controlling perceptual processes. It has difficulty making decisions about what to do. The more often it must make a decision, the worse its decisions become. Such a control component is of interest, since current computer technology can probably be the basis for such homunculi.

The question is, what kind of inputs should the homunculus receive from its perceptual system? Digital images would be fairly useless to the homunculus since they would be presented to it as long sequences of numbers on the display panel. Similarly, a surface model by itself would be of little help, since it would have to be encoded as a sequence of numbers to be pored over.

A model-based object tracking system would be useful when the right kind of object was in view. Perhaps, three numeric displays would encode the coordinates of such an object when they were in view. The homunculus could then grasp the object

by pushing appropriate buttons corresponding to the provided object coordinates.

Vision services are intended to be of use in a robot that is controlled by such a homunculus in a dynamic, unstructured environment. Because vision services reduce camera data to a small number of bits that encode relevant aspects of the situation, effector actions can be chosen simply. Because vision services do not depend on *a priori* knowledge of objects or the environment, they are suitable for robots in unstructured environments. Because vision services are taskable, the robot's control system need not devote much of its decision resources to directing the vision service itself.

Charters of some potential vision services include:

- Report when some large, independently moving stuff appears.
- Report where the nearest stuff is.
- Report fast moving stuff.
- Report on looming stuff that is coming near.
- Keep track of the largest, nearby patch of moving stuff.

1.3 An optical flow-based tracking service

This thesis presents a vision service that picks out and keeps track of moving objects. Throughout the thesis, "tracking" involves keeping track of the image-relative extent of an object across time. Figure 1.1 shows an image on which a white line marks the approximate perimeter of the image-relative extent of a moving cat. The area inside the white line could be the output of the tracking service for one image. For each subsequent image, tracking would be expected to provide updated information on the cat's new position.

The tracking service makes use of a visual property that all visible moving objects have, namely, that they move. The idea of an optical flow field is useful when talking about perceived motion. Consider the two images in Figure 1.2. The lower image was captured $\frac{1}{30}$th of a second after the upper one. The cat is moving to the right.

Figure 1.1: The image-relative extent of a cat

Figure 1.2: Two images of a cat taken $\frac{1}{30}$th of a second apart

The camera is panning right, and is moving slightly faster than the cat. Thus the cat appears to have shifted a little to the left, while the background has shifted a good deal more to the left. Each image is composed of a grid of cells or pixels. An optical flow field is an approximate answer to the question of where each pixel on the first image went to in the second image. Figure 1.3 contains a needle diagram showing a subportion of an optical flow field computed from the two cat images. Each cell in the needle diagram contains a displacement vector (the thick end of the needle is the point of the arrow) indicating where each cell is estimated to have gone.

The tracking service makes use of computed optical flow fields in two ways. Initially, flow fields are used to pick out, or *segment* new moving objects. Subsequently, flow fields are used in keeping track of where the objects have gone.

Looking again at the optical flow field in Figure 1.3, it appears plausible that the pixels corresponding to the cat could be picked out (segmented) by selecting those pixels that underwent a distinguishing motion. Selecting pixels that move together is the basis for the segmentation technique used by the tracking service for picking out moving objects.

Recall that the optical flow field maps pixels on the first image to the second image. Thus if the image-relative extent of the cat is known on the first image, in principle, it ought to be possible to derive the image-relative extent of the cat in the second image by following the vectors of the optical flow field. The idea of using the optical flow field to map the image-relative extent of an object on one image to that on the next image is the basis for the part of the tracking service that keeps track of where objects have gone.

Figures 1.4–1.6 show an extended tracking sequence that demonstrates the output of the tracking service over 27 frames. The motions from frames 0–3 are used to pick out a moving cat. The white lines on subsequent images indicate the outline of the tracked blob.

1.3.1 Is this a vision service?

Vision services were defined on the basis of three characteristics: summarization, generality, and taskability. The tracking service summarizes each image as a set of

Figure 1.3: An optical flow field of cat motion

Frame 0 Frame 1 Frame 2

Frame 3 Frame 4 Frame 5

Frame 6 Frame 7 Frame 8

Figure 1.4: Cat tracking sequence part 1

Frame 9 Frame 10 Frame 11

Frame 12 Frame 13 Frame 14

Frame 15 Frame 16 Frame 17

Figure 1.5: Cat tracking sequence part 2

Frame 18 Frame 19 Frame 20

Frame 21 Frame 22 Frame 23

Frame 24 Frame 25 Frame 26

Figure 1.6: Cat tracking sequence part 3

pixels that mark the image-relative extent of an object or, even more simply, the image-relative coordinates of the center (centroid) of the object. The summarization can be used for visual servo-control to keep a camera aimed at an object. If the centroid lies away from the center of the image, the control system turns the camera in the appropriate direction to center the object in the image.

The tracking service is general, in that it depends neither on knowing what objects will look like, nor on knowing about specific visual properties that will distinguish these objects. If something moves in such a fashion that enough of its motion is apparent to the system, the object can be picked out and tracked.

Given no instruction, the tracking service will attempt to track the moving object that takes up the largest region of its field of view. If some higher level system were to give it the image-relative extent of an object to track, it would try to track the object as long as possible. The tracking service is taskable, in that it can go on producing useful information without external direction; on the other hand, when direction is given, it can change its behavior.

1.3.2 Is it useful?

General arguments for the usefulness of tracking follow two lines: people and animals appear able to do it, and it has been useful for building systems in Artificial Intelligence. Pylyshyn [78] shows evidence suggesting that humans can keep track of a small number of independently moving icons. Agre and Chapman [3, 25] have used simulated tracking to build systems for playing video games.

A more practical demonstration of the usefulness of the optical flow-based tracking service is its use in implementing two systems that perform *camera pursuit*. The goal of camera pursuit is to keep a camera aimed at an object of interest.

In these pursuit systems, a camera is mounted on a robot, and when a person or other large moving object appears in the camera's field of view, the system attempts to center the camera on the moving object. The systems perform visual-servoing to control the camera using the centroids of moving object that are the output of the tracking service.

These camera pursuit systems function in dynamic, unstructured environments.

The algorithms are suitable for massively parallel implementation, and currently run at between 10 and 15 frames per second. The algorithms do not depend on properties of the objects other than that they move and that they be sufficiently visible on the imaging surface.

The tracking service for the first camera pursuit system runs on a Connection Machine at Xerox PARC. The tracking service for the second camera pursuit system runs on five Intel i860 processors in the robotics laboratory at Stanford.

1.4 Validation

Very early machine vision work was characterized by *ad hoc* techniques that only functioned on very limited classes of scenes. Guzman [40], for example developed a system that inferred the 3-D structure of blocks world scenes by relying on very good edge detection and a large set of heuristics. If a spurious edge was detected, some edges were overlooked, or some case not covered by heuristics arose, the system would fail.

In reaction to the early *ad hoc* period, researchers tended to follow the role model supplied by the folklore of physics, in the hopes of producing work that could be validated. By assuming knowledge of light sources, object surfaces, and cameras, by understanding image formation, and by devising general vision problems, the plan was to solve vision analytically. The solutions to vision problems in this paradigm tend to be equations. The proposed advantage to the approach is that one can prove things about the resultant equations and their relation to the strongly constrained, assumed world.

This analytic approach to machine vision suffers from several practical difficulties. Computationally, it is often the case that the derived equations are expensive to solve, and in some cases numerically unstable. Empirical problems with the approach arise from the fact that enough may not be known about light sources and object surfaces, cameras may not agree with camera models, and in general the world may not satisfy the strong assumptions that were made to derive the equations. The consequences of these difficulties are that although one can prove how the equations will operate in

idealized scenes, one has no idea how they will operate in practice.

An alternate approach is to define modular vision services that can actually be used for some robot task, and to attempt to design algorithms that support the services. Two key properties are required of algorithms that are to support vision services: they must actually work on real images, and they must be suitable for actual implementation.

One issue in designing vision algorithms for particular tasks is how to validate them. Given the output of a vision algorithm on one input, how can the output be evaluated? Supposing that the algorithm works on one input, how can it be shown to work on many other different kinds of images from different scenes?

Take for example, an optical flow algorithm. Given a sequence of input images, how can the resulting flow field be evaluated? One approach is to generate synthetic data based on a model of image formation. With synthetic data, the resulting optical flow field can be mechanically compared with the intended answer. However, a successful comparison only shows that the algorithm is consistent with the model of image formation.

A better approach to validating an optical flow algorithm is to produce an actual environment that is completely calibrated. Then an optical flow field can be mechanically compared with the predicted scene motion of the known calibrated scene. Although an algorithm can be shown to work in one actual environment using this technique, no information is gained about how the algorithm will work in other environments.

A third approach is to run the optical flow algorithm on many different kinds of images, and examine the resultant needle diagrams by hand. This subjective approach is tedious and produces scant information about the validity of the algorithm. It is extremely difficult to judge the accuracy of a single needle diagram after much examination. Attempting to evaluate thousands of needle diagrams by eye would be without point.

Vision services also provide a partial answer to the question of validation. Image services are intended to compute summarizations of camera data that are useful for action. If one actually implements a vision service that can run at suitable speeds

and connects it to the appropriate robotic device, then the whole system, assuming that it works, provides ongoing empirical validation. The performance of all of the components of the system, the optical flow algorithm for example, can be evaluated based on the performance of the whole.

The advantage of this validation procedure is that it results in real-time vision systems that actually work. The disadvantage of the validation procedure is, of course, that one must build real-time vision systems that actually work.

1.5 Contemporary situation

The content of this thesis is most closely tied to two threads of contemporary research in A.I. and machine vision, active vision, and bottom-up A.I. My own motivation for pursuing the work is related to a third thread in A.I. and philosophy, namely, practical reification: the question of how perception can carve up the world into relevant pieces in a sufficiently plastic way.

1.5.1 Active vision

Active vision has become a significant subfield in the A.I. and vision community. Judging from a report written by attendees of the Active Vision Workshop [88] that took place in August of 1991, active vision has three essential elements: active control of camera parameters, selective sensing, and tight coupling of perception and action. Vision services and the motion-based tracking service are consistent with the idea of a tight coupling between vision and action. However, the emphasis in the proposal for vision services is distinct from those espoused by authors of the seminal papers in the field.

Aloimonos' view of active vision emphasizes active control. In his paper *Active Vision*, he defines "an observer [to be] active when engaged in some kind of activity whose purpose is to control the geometric parameters of the sensory apparatus" [5, P. 35]. The advantage of controlling camera parameters is that "problems that are ill-posed, non-linear or unstable for a passive observer become well-posed, linear or

stable for an active observer."

Bajcsy in her paper *Active Perception*, defines "Active Perception (Active Vision specifically) ... as a study of Modeling and Control strategies for perception" [9, P. 996]. Her emphasis is on "modeling of the sensors, the objects, the environment, and the interaction between them for a given purpose." The notion of paying attention to the interactions is consistent with that of vision services, however the idea of modeling objects and the environment goes against the idea of an unstructured environment.

Ballard, in his paper *Animate Vision* emphasizes that the "central asset of animate vision is ... the collection of different mechanisms for keeping the fovea over a given spatial target" [11, P. 61].

The idea of vision services emphasizes the tight coupling of vision and action and de-emphasizes other aspects such as selective sensing, and scene and sensor modeling. The tracking service itself provides support for selective sensing, in that it chooses an area of interest — a tracked region. Similarly, the systems pan a camera, but do not use the ability to achieve better measurement.

1.5.2 Bottom-up AI

Recently, Brooks *et al.* [17, 18, 19, 35] have argued that Artificial Intelligence ought to be incrementally working up by building very simple robot "insects" that actually work in the world, rather than starting with very high-level ideas about intelligence and attempting to work down. The ideas of vision services and of the tracking service itself are consonant with the bottom-up approach. Vision services would be ideal for use in such robots. The simple control mechanisms of Brooks' insects could be straightforwardly coupled with the tracking service.

1.5.3 Practical reification

Underlying this thesis there is a perplexing issue. How is it that a vision system can, almost without fail, map visual input to some bits / representations / stimuli that provide a control system with sufficient relevant information for survival in a dynamic and unstructured world?

Literally, *reification* means the act or result of viewing something as a material thing. Practical reification is the process of continually splitting the world up into units that can be of use in responding to that world. The idea of vision services and the tracking service perhaps hint at a way of approaching practical reification. A few quotations may serve to make the issue of practical reification clearer.

Winograd [99], in his discussion relating Heidegger's concept of *blindness* to computer systems, points out that "[i]n writing a computer program, the programmer is responsible for characterizing the task domain as a collection of objects, [and] properties," and continues by pointing out that "[t]he program is forever limited to working within the world determined by the programmer's explicit articulation of possible objects, properties and relations among them"(P. 97). This observation can be directly mapped to problems of perception. If the vision system is programmed to characterize the world in terms of a previously specified ontology, then the vision system may be incapable of characterizing things and events that fall outside of this ontology.

René Descartes, in his Second Meditation of 1642, makes clear that objects are not exclusively the simple sorts of things that can be easily entered into a vision system's model library as wire frame models and characterized with simple predicate symbols.

> Let us now consider the commonest things, which are commonly believed to be the most distinctly known and the easiest of all to know, namely, the bodies which we touch and see. I do not intend to speak of bodies in general, for general notions are usually somewhat more confused; let us rather consider one body in particular. Let us take, for example, this bit of wax which has just been taken from the hive. It has not yet completely lost the sweetness of the honey it contained; it still retains something of the odor of the flowers from which it was collected; its color, shape, and size are apparent; it is hard and cold; it can easily be touched; and, if you knock on it, it will give out some sound. Thus everything which can make a body distinctly known is found in this example.
>
> But now while I am talking I bring it close to the fire. What remains of the taste evaporates; the odor vanishes; its color changes; its shape is lost; its size increases; it becomes liquid; it grows hot; one can hardly

> touch it; and although it is knocked upon, it will give out no sound. Does the same wax remain after the change? We must admit that it does; no one denies it, no one judges otherwise. What is it then in this bit of wax that we recognize with so much distinctness? [1] [32, P. 29]

Somehow, a vision system must report on the continued presence of a ball of wax. Something other than prior knowledge of its outward appearance, something about the coherence and stability of the ball of wax makes it perceivable.

Opening his essay, *Things and Their Place in Theories*, Quine puts things on level ground.

> Our talk of external things, our very notion of things, is just a conceptual apparatus that helps us to forsee and control the triggering of our sensory receptors in the light of previous triggering of our sensory receptors [79, P. 1].

The conceptual apparatus that is our notion of things is not a fixed structure; what is to be taken as a thing is open. "There is room for choice, and one chooses with a view to simplicity in one's overall system of the world." [79, P. 10] Quine's promiscuity towards objects is far-reaching.

> We need all sorts of parts or portions of substances. For lack of a definable stopping place, the natural course at this point is to admit as an object the material content of any portion of space-time, however irregular and discontinuous and heterogeneous. This is the generalization of the primitive and ill-defined category of bodies to what I call physical objects [79, P. 10].

Quine's primary concern is with language and theories. However, the observation that the carving up of the world into objects is a pragmatic issue applies as well to perception.

[1] The rest of the paragraph is part of an argument for Idealism that diverges from the present discussion: "Certainly it cannot be anything that I observe by means of the senses, since everything in the field of taste, smell, sight, touch, and hearing are changed, and since the same wax nevertheless remains."

Thus, the attempt to pick out and maintain an individual object solely on the basis of motion is an attempt to get at a perceptually grounded ontology. More generally, properties such as motion and depth can be used to carve up simple worlds into usable entities independent of any interpretation.

The thesis contributes to both bottom-up A.I. and active vision. It is also a demonstration of a vision system that distinguishes some objects in a dynamic, unstructured environment.

1.6 Joint work and prior publication

This work was done at Stanford, SRI, and Xerox PARC. It was carried out in the context of a joint research programme with fellow Ph.D. student Ramin Zabih. Many of the ideas for algorithms in the tracking service arose during joint experimentation and discussion over many months. In an ideal world no more would be said. However, academe demands additional detail. The ideas of the optical flow and tracking algorithms are joint. The algorithms for efficiently computing the desired results are primarily mine. As I was particularly interested in actually connecting the tracking algorithms to the world, I am mostly responsible for the motion segmentation technique and the control regimen that allows the system to run on incoming video data.

Again, the robot systems were jointly constructed by Ramin and me. Jon Goldman of Thinking Machines produced special purpose microcode to speed certain computations on the Connection Machine at Xerox PARC. Alain Fidani provided an interface to Khatib's COSMOS system for controlling a PUMA robot arm at Stanford. Penni Sibun untiringly performed expert modeling for several hours. David Sobeck and the Symbolics Corporation lent me a Lisp machine for several years.

Some of this work has appeared previously in [100, 101, 102].

1.7 Contributions

This thesis presents an optical flow-based tracking and segmentation system. Novel algorithms that perform efficient optical flow computation, flow-based tracking, and cross-temporal, trajectory-based object segmentation are described and analyzed. The idea of *vision services* as a model for vision systems that are to be used in dynamic, unstructured domains is introduced, and shown to subsume the tracking system. The tracking service and its component vision algorithms are validated with two real-time robotic camera pursuit systems.

1.8 Reader's guide

The second chapter of the thesis contains a discussion of basic notions including a model of cameras, camera and object motions, and various notation. The chapter also introduces dynamic programming techniques that make real-time computation feasible. Chapter 3, *Architecture*, introduces the architecture of the tracking service, and describes the embedding of this architecture in two implemented camera pursuit systems. The next three chapters cover the main components of the tracking service. Chapter 4, *Motion measurement*, describes the algorithm used to compute optical flow fields. Chapter 5, *Motion segmentation*, introduces the motion trajectory-based segmentation scheme used to pick out independently moving objects. Chapter 6, *Motion tracking*, presents the algorithm that tracks the position of an object from frame to frame. Finally, Chapter 7, concludes the thesis. Appendix A contains a table of special symbols along with the values of various important constants. Appendix B gives a more complete explanation of the dynamic programming techniques introduced in Chapter 2. In order to better communicate the described algorithms, they are also presented in Appendix C as LISP code.

Each chapter presenting descriptions of developed algorithms has a section discussing related algorithms and a section containing an analysis of the computational complexity of the algorithms. Section 2.6 touches on the general model of complexity used. The complexity of the presented algorithms is summarized in Section 7.2.

Chapter 2

Preliminaries

This chapter presents assumptions, notation and ideas that recur throughout the thesis. First, a model of cameras is introduced, and the effects of camera motion are examined. Next, the ideas of motion fields and optical flow fields are presented. Analogous notions for object representation are discussed in Section 2.4. A small amount of mathematical shorthand is introduced in Section 2.5. The chapter concludes with a brief discussion of algorithmic complexity and a summary of the basic dynamic programming techniques that make many parts of the system feasible.

Various symbols and notation are needed for talking about the vision algorithms and their complexity. The special symbols are summarized in Appendix A. The set of discrete image pixels is denoted $\mathbf{P}$. Pixels are usually considered as atomic elements that are associated with a 2-D grid, but sometimes pixels are coordinate pairs, or vectors $\langle x, y \rangle$. An intensity is a natural number in the range $\{0, \ldots, 2^k - 1\}$ for a constant k, almost invariably 8. A gray-level image is a map from $\mathbf{P}$ to intensities, written $I : \mathbf{P} \longrightarrow \{0, \ldots, 2^k - 1\}$. A gray-level image is thus an array of eight bit numbers. The set of gray-level images is denoted $\mathcal{I}$. A boolean image maps from $\mathbf{P}$ to $\{0, 1\}$. A boolean image is thus a bitmap, or array of bits. Sometimes boolean images are treated as sets of pixels or unary relations on pixels. The set of boolean images is denoted $\mathcal{B}$.

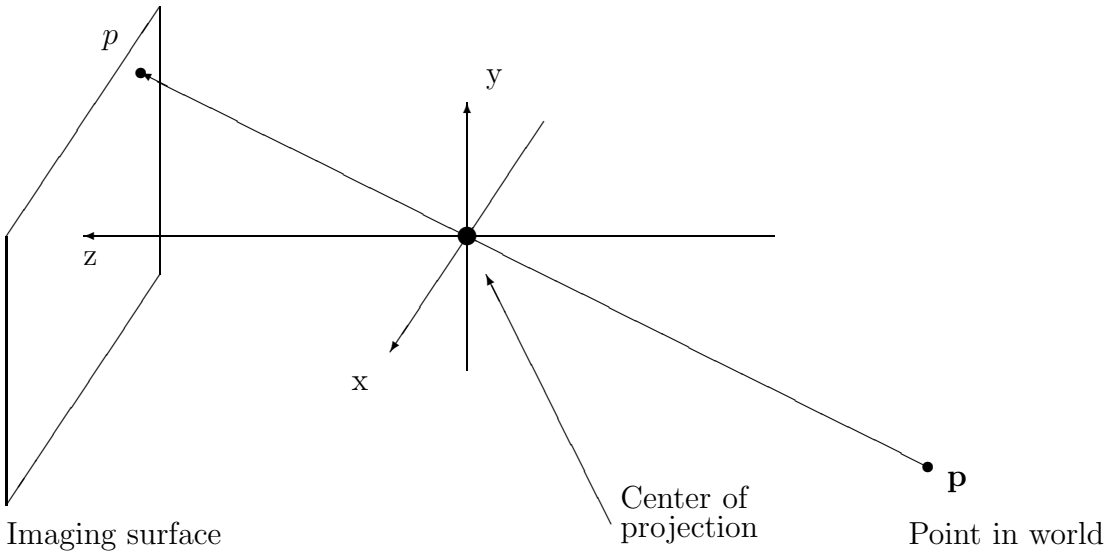


Figure 2.1: The perspective projection camera model

2.1 Camera model

The analysis in subsequent chapters assumes a standard perspective projection model of cameras. Figure 2.1 depicts this camera model, and its corresponding coordinate system. The x and y axes are the x and y axes of the image. The z axis is oriented from the world in front of the camera toward the imaging surface. A ray running from the object at point $\mathbf{p}$ passing through the center of projection, projects to a point p on the imaging surface.

The vertical and horizontal fields of view, fov_v, and fov_h, are important angular measures of the lens and camera system. The vertical (horizontal) field of view is the angle between the top and bottom (left and right) edges of the imaging surface measured from the center of projection. The fields of view determine the approximate vertical and horizontal visual angles subtended by pixels on the imaging surface.

Figure 2.2: Effects of a flat imaging surface

2.2 Camera motion

Arbitrary camera motion can result in arbitrary changes on the imaging surface. Camera motion relative to a fixed environment can be described in terms of translations and rotations about the three axes. Parallax resulting from translating the camera along the x axis, for example, causes points on the imaging surface corresponding to nearby objects to shift further than points corresponding to distant objects. The human eye when pivoting in its socket performs a particularly simple kind of movement about the center of projection: namely, rotations about the x and y axes. Several of the algorithms described in later chapters rely on just such camera motions (see Sections 5.8 and 6.2.5).

Since imaging surfaces are flat, camera rotations about the x and y axes result in different projected motions at the center of the imaging surface and at its edge. Luckily, for points within a fairly narrow field of view, these variations are not great. Figure 2.2 depicts this situation. Consider a camera with a 1-D imaging surface, and a point in the world whose projection on the imaging surface subtends an angle of θ_1 with the principal axis of the camera. When the camera rotates an angle of $\theta_2 - \theta_1$, the projected point on the imaging surface now subtends an angle θ_2 with the

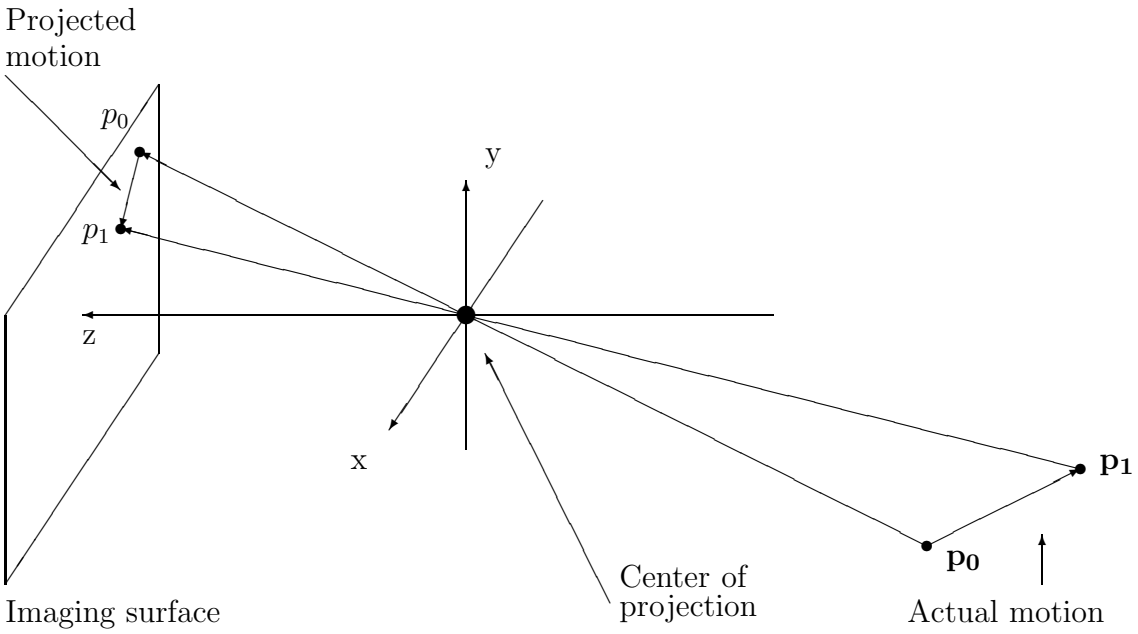


Figure 2.3: Motion on the imaging surface

z axis. The distance that the projected point has traveled on the imaging surface d, is $(\tan\theta_2 - \tan\theta_1)f$, where f is the focal length of the camera. Clearly, as the initial angle θ_1 gets larger, for the same rotation angle $\theta_2 - \theta_1$, d gets larger. The derivative of $\frac{d}{f}$ with respect to θ_1 as θ_2 goes to zero is the derivative of $\tan\theta_1$, $\frac{1}{\cos^2\theta_1}$. For small angles θ_1, $\cos^2\theta_1 \approx 1$. Hence, for relatively narrow fields of view, camera rotations about the x and y axis result in approximately uniform motion across the imaging surface.

2.3 Motion fields and optical flow

When an object in the field of view of the camera moves, there can be a corresponding change on the imaging surface of the camera. This corresponding change on the imaging surface has the potential of being related to the actual motion by the projection relation. Consider the case shown in Figure 2.3 in which an object has moved left and away from the camera. At time t_0, a point in the world, $\mathbf{p_0}$, picks out a

point $\mathbf{p}$ on the surface of the object, while p_0 is the point on the imaging surface that corresponds to $\mathbf{p_0}$ as projected through the lens of the camera. At time t_1, after the object has moved, $\mathbf{p_1}$ picks out the same point $\mathbf{p}$ on the surface of the object, and p_1 marks the point on the imaging surface that now corresponds to $\mathbf{p_1}$. Although the motion from $\mathbf{p_0}$ to $\mathbf{p_1}$ involves a change in depth (along the z axis), vectors on the imaging surface can only represent the component of the motion parallel to the imaging surface. Nevertheless, the motion of each point in the field of view of the camera is mapped by the projection relation to a displacement vector on the imaging surface. Since displacements on the imaging surface do not reflect changes in depth, displacements measure motions in terms of visual angles, not distances in the world.

A *motion field* is a field of displacement vectors. Each displacement vector is associated with a point on the imaging surface. The displacement vector at each point corresponds to the 2-D projection of a 3-D motion occurring at that point in the field of view between times t_0 and t_1. Equivalently, a motion field can be a mapping between points on the imaging surface at time t_0 and points on the imaging surface at time t_1. If the camera is moving in such a way that the imaging surface contains points at time t_0 for which there are no corresponding points at t_1, these points are mapped to $\perp$ by the motion field. Similarly, when an object is moving against a background, points that have been occluded by the object are also mapped to $\perp$. A motion field can be thought of as representing the projection of true visible motion; it maps each point on the imaging surface that is the projection of a visible point in the world at time t_0 to a point on the imaging surface that corresponds to the projection of the same visible point in the world at time t_1.

Discrete motion and optical flow

The ideal notion of motion fields can be adapted to the discrete case. A discrete motion field maps each pixel on the imaging surface to another pixel on the imaging surface, or $\perp$. The motion of each pixel is intuitively defined to be the predominant motion of the points within the pixel. Since motion fields are only an expository device for introducing the idea of scene motion, sampling issues will not be discussed. From now on, motion fields will be discrete maps and will be written $\tilde{M} : \mathbf{P} \longrightarrow \mathbf{P} \cup \{\perp\}$.

Motion fields are a useful abstraction, but the only evidence that computer vision can access is arrays of gray-levels corresponding to changes in light intensity. The motion field that arises when a sheet of white paper slides across a white paper background represents the motion that is occurring. However, from images of the scene, no changes are apparent. The difference between what is apparent in the images and the actual projected motions leads to the distinction between *optical flow* fields and motion fields. Horn describes optical flow as the "apparent motion of brightness patterns observed when a camera is moving relative to the objects being imaged" [46, P. 278]. The apparent motion of a brightness pattern is an elusive concept. The famous illusion that a barber pole appears to be moving upwards when it is merely rotating argues that optical flow cannot have a simple, unambiguous definition. The term "optical flow" is used to denote approximations to the motion field, $\tilde{M}$. Given two images I_i and I_{i+1}, captured at times and t_0 and t_1 respectively, denote an optical flow field $M_i : \mathbf{P} \longrightarrow \mathbf{P} \cup \{\perp\}$. The best such optical flow field is identical to the discrete motion field arising from motions between times t_0 and t_1. The set of optical flow fields is written $\mathcal{M}$. When pixels are viewed as vectors, the set of differences of pixels, i.e., displacement vectors, is denoted $\mathbf{D}$. Occasionally, a different concept of an optical flow field that maps a pixel to a displacement vector is used: $M_i : \mathbf{P} \longrightarrow \mathbf{D} \cup \{\perp\}$. Multivalued flow fields that map pixels to sets of pixels are denoted $M_i^* : \mathbf{P} \longrightarrow 2^{\mathbf{P}} \cup \{\perp\}$, or $M_i^* : \mathbf{P} \longrightarrow 2^{\mathbf{D}} \cup \{\perp\}$. The set of such multivalued flow fields is denoted $\mathcal{M}^*$.

2.4 Objects

For ease of expression, it will often be useful to consider the case where there is exactly one moving object in the field of view, and that delineating the extent of the object is unproblematic. Given that there is one moving object of indisputable outline, an abstract boolean image $\tilde{O}_i : \mathbf{P} \longrightarrow \{0, 1\}$ can be defined that corresponds to the actual projection of the moving object on the imaging surface when an image was taken. Just as an optical flow field is an approximation to the motion field, a boolean image O_i is an approximation to $\tilde{O}_i$ on image I_i.

2.5 Mathematical shorthand

Several of the described algorithms involve finding the element of a set $\mathcal{S}$ for which the value of some function f is maximal, i.e., the element a of $\mathcal{S}$ such that $f(a)$ is maximal. This maximal element is denoted $\sup_f \mathcal{S}$. In the case that the extremum is not unique, a random element from the set of extremal elements is chosen. If $\mathcal{S}$ is empty, the result is $\perp$, i.e., *random-element*$(\{\}) = \perp$.

$$\sup_f \mathcal{S} \quad \equiv \quad \textit{random-element}(\{\, a \in \mathcal{S} \mid f(a) = \max_{a' \in \mathcal{S}} f(a') \,\})$$

It is often the case that a function f maps from $\mathcal{S}_0$ to $\mathcal{S}_1 \cup \perp$. Such a function is *injective* if

$$\forall a \in \mathcal{S}_0 \quad f(a) = \perp \quad \vee \quad \forall a' \in \mathcal{S}_0 \quad f(a') = f(a) \Rightarrow a = a'.$$

2.6 Complexity analyses

One prominent lacuna in much low-level vision research is any mention of algorithmic complexity[1]. When the goal of vision research is, to use a phrase of John McCarthy's, "epistemological adequacy," [65] that is, to see what can in principle be extracted from an image, ignoring issues of cost is reasonable. However, when the goal of vision research is to develop algorithms to support the performance of some task in a dynamic environment, such analyses become crucial.

One way to analyze vision algorithms is to treat images as being of constant size — say 512×512. However, this stance forces one to conclude that practically all low-level vision algorithms take constant time — i.e., some multiple of the constant image size. Image size is thus included as one parameter in cost analyses. For example, with an $N \times N$ image, a connected components algorithm might have complexity $\mathcal{O}(N^2 + 2 \lg N)$.

Most of the algorithms discussed in this thesis perform the same operation on each pixel. The algorithms generally operate over a subregion surrounding a pixel. The radii of these local regions are also parameters in complexity analyses.

[1] Some researchers such as Tsotsos [94] have emphasized complexity analysis for higher-level vision.

It might seem that all of these local algorithms would have the same complexity $\mathcal{O}(N^2r^2)$ where r is the local radius. However, what matters in many of these algorithms, is how dynamic programming, i.e., the ordered caching of partial results, can be used to bring the complexity closer to $\mathcal{O}(N^2)$. Sadly, when real-time performance is required, big-oh analysis is not sufficient; computation per image must be achievable within a fixed number of target machine operations and instructions-per-pixel becomes the unit of interest.

Since the algorithms have been implemented for both serial[2] and parallel machines, analyses are presented for both serial and parallel versions of the algorithms. The parallel algorithms were designed for a Connection Machine (CM), a massively parallel *single instruction multiple data* (SIMD) computer with hypercube connectivity [44].

Performing vision tasks on a SIMD machine is modeled so that there is one pixel per processor[3]. Since every instruction must be executed on each processor on a SIMD machine the N^2 term corresponding to the image size is omitted from SIMD complexity measures. The performance of an algorithm running on a SIMD machine depends not only on the number of local operations performed, but also on how much communication is performed. The comparative costs of local computation and communication differ between machine architectures. It is therefore expedient to maintain distinct analyses for computation and communication.

As there is no generally accepted model for CM complexity analysis, a simple three component model is used. The three cost components include: local, arithmetic operations; local, uniform, hypercube communications; and global, non-uniform communications. A local communication, such as "for each processor (pixel), send a datum to its neighbor to the north," is much cheaper than a global, non-uniform communication, such as "for each processor, send a datum to some other arbitrary processor that

[2]In fact the "serial" version of the tracking system runs on a *multiple instruction multiple data* (MIMD), five Intel i860 processor system. It is serial in the sense that communication is minimal, and that the same algorithms would be used to implement the same system on a single, five times faster truly serial machine.

[3]The assumption of one pixel per processor is justified since even if multiple pixels are handled by a single processor only an approximately constant multiplicative factor is introduced to the computation. The tracking system at Xerox PARC processes 128×128 images and the PARC CM has 128^2 processors.

it specifies." On the CM, the cost of an operation increases approximately linearly with the number of bits processed. However, since there are significant start-up costs associated with all operations, in most cases the length of operands is ignored and only the number of operations is counted. Each SIMD algorithm is assigned three cost expressions, one for each of the three modeled components.

2.7 Local-area dynamic programming

When a value, $F(p)$, is associated with each pixel p, it is often useful to compute a local summary of values of F in a region surrounding each pixel. If the desired summary is of the right form, then particularly efficient algorithmic techniques can be used to compute the summaries at each pixel. When the techniques apply, since the local areas surrounding adjacent pixels overlap, the problem of computing summaries can be decomposed into subproblems such that partial results can be cached and shared across pixels. As noted by Aho *et al.*, "the filling-in of a table of subproblems to get a solution to a given problem has been termed *dynamic programming*" [4, P. 311][4]. The remainder of this section describes bounds on the cost of performing these local summarization operations. Details of the dynamic programming techniques used to achieve these bounds are described in Appendix B.

If the local region surrounding each pixel is rectangular and unvarying in size, and if the summary desired at each pixel is the result of combining the local values of F with an associative binary operator $\otimes$ then the dynamic programming techniques apply. The situation at a pixel p is shown in Figure 2.4. The local area is w pixels wide and h pixels high. If values of F are integers, and the desired summary of the local area is the sum of values of F, then $\otimes$ is addition. In this case, the desired result for a pixel p, $\sigma(p, w, h)$, can be written as a summation.

$$\sigma(p, w, h) \quad \equiv \quad \sum_{[\frac{-w}{2} < i \leq \frac{w}{2}]} \sum_{[\frac{-h}{2} < j \leq \frac{h}{2}]} F(p + \langle i, j \rangle)$$

[4]Note that this notion of dynamic programming is more general than that of dynamic-programming algorithms that cache partial results in order to find optimal solutions to a certain class of problems.

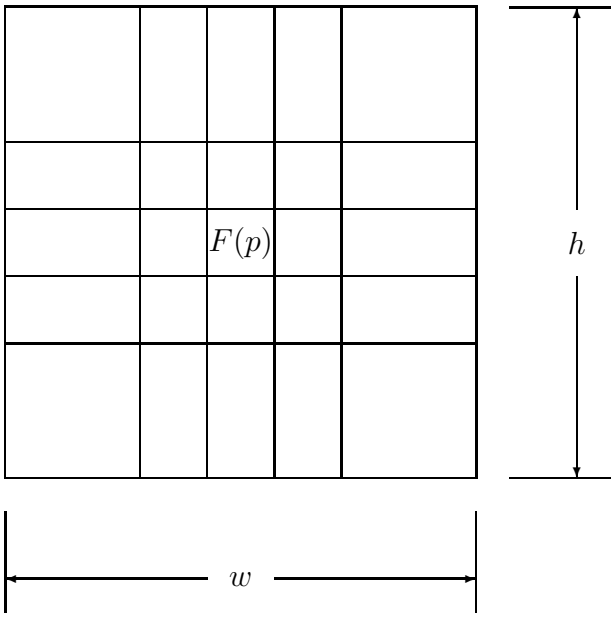


Figure 2.4: F in the local area surrounding pixel p

In the general case of an associative operator $\otimes$, $\sigma(p, w, h)$ can be written using $\bigotimes$.

$$\sigma(p, w, h) \quad \equiv \quad \bigotimes_{[\frac{-w}{2} < i \leq \frac{w}{2}]} \bigotimes_{[\frac{-h}{2} < j \leq \frac{h}{2}]} F(p + \langle i, j \rangle)$$

Interestingly, the dynamic programming techniques used for local-area computations differ for serial and parallel implementation, and indeed have distinct computational cost. When used for serial computation in which pixels are processed sequentially, dynamic programming can take advantage of full results generated for previous rows, and earlier columns. Benefits arise from not repeating these earlier computations. These techniques can be used on serial computers for forming sums of integers and vectors over local areas.

When used for parallel computation, processing happens simultaneously over the image, thus only partial results from earlier operations can be shared. Savings arise both from shared computation and reduced communication. Besides local-area summation, concatenation proves to be a cost-effective local-area operation for accessing neighborhood information in parallel implementation.

If the operator $\otimes$ has an inverse, $\sigma(p, w, h)$ can be computed for each pixel p

on an $N \times N$ image on a serial machine in $\mathcal{O}(4N^2)$ $\otimes$-operations. If $\otimes$ has no additive inverse, $\sigma(p, w, h)$ can computed on a serial machine in $\mathcal{O}((\lg w + \lg h)N^2)$ $\otimes$-operations. There is a fixed overhead that occurs at the edges of the matrix F in computing $\otimes$ on a serial computer. This overhead is not included in the complexity figures, as it is independent of image size. However, if the serial computation were split across many serial processors, this overhead could become significant. In the extreme, running the serial algorithm on each processor of the Connection Machine would incur a great deal of communication and would cost approximately $\mathcal{O}(wh)$ $\otimes$-operations per pixel.

On a SIMD computer, computing $\sigma(p, w, h)$ requires $\mathcal{O}(\lg w + \lg h)$ $\otimes$-operations per processor, regardless of whether $\otimes$ has an inverse. Computing $\sigma(p, w, h)$ on a SIMD machine with hypercube connectivity also requires $\mathcal{O}(\lg w + \lg h)$ hypercube communications per processor.

In general, if $\otimes$ is an operation whose output is larger than its input, e.g., addition on the natural numbers, there is a logarithmic complexity term related to the size of the output. However, for moderate local areas, this logarithmic factor stays below the word size of a machine, and is ignored.

Chapter 3

Architecture

This chapter presents the architecture of the tracking service itself as well as the architectures of two systems that have used the tracking service to implement camera pursuit. This discussion of architecture serves to explain the roles of the submodules of the service and the role of the tracking service in the implemented systems.

3.1 Tracking service architecture

The overall architecture of the tracking service is shown in Figure 3.1. Arrows signifying data paths show that information flows primarily bottom to top. Images enter the motion measurement module at the bottom of the service. There are two alternate top level interfaces to the tracking service. The more general interface is delineated by the dashed line below the box labeled "control." Outputs and inputs at this level are boolean representations of objects. This interface would be used to connect the tracking service to more general vision systems. The other interface to the tracking service is the arrow at the top of the diagram. At this level, the service only emits information about the current image-relative position of an object that is being tracked.

The modules of the tracking service will be described in the order they appear in the path of data: motion measurement, segmentation, and tracking. Issues related to the more general interface are discussed before the box labeled "control" is considered.

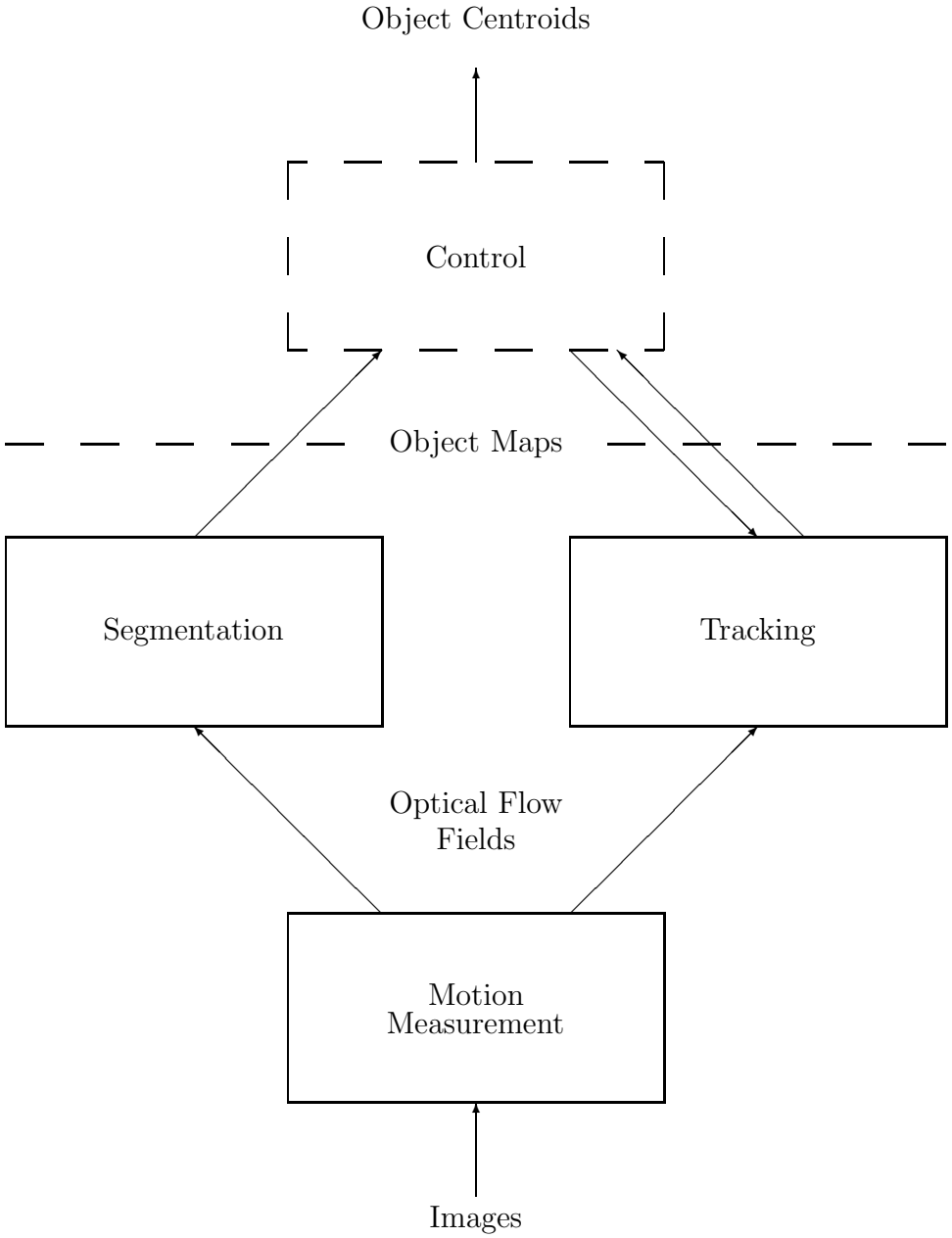


Figure 3.1: Tracking service architecture

3.1.1 Motion measurement

The motion measurement module takes in a pair of images I_i, and I_{i+1}, and produces an optical flow field M_i, that approximates the motion that is happening in the field of view during the interval between the two images. It can be viewed as a function from a pair of images to an optical flow field, $motion : \mathcal{I} \times \mathcal{I} \longrightarrow \mathcal{M}$, and is described in detail in Chapter 4. The motion measurement module needs no control information; it computes a new optical flow field as each successive image arrives.

3.1.2 Segmentation

The segmentation module takes as input a small number, n_s, of successive optical flow fields and produces a bounded set of boolean images that are approximations to the current image-relative positions of the objects that it has picked out as having moved during the period that is covered by the n_s flow fields. Formally, segmentation can be written as a function mapping sequences of optical flow fields to sets of boolean images $segment : \mathcal{M}^{n_s} \longrightarrow 2^{\mathcal{B}}$. The algorithm used for segmentation is described in detail in Chapter 5. The segmentation module expects control information in the form of requests to perform segmentation.

3.1.3 Tracking

The tracking module works on two optical flow fields M_i, and M_{i+1}, and an initial boolean image O_i. The two flow fields are the results of processing successive pairs from the three images, I_i, I_{i+1}, and I_{i+2}. The boolean image O_i approximates the position of an object in image I_i. The tracking module produces a new approximation O_{i+1} that represents the position of the object in image I_{i+1}. Formally, $track : \mathcal{M} \times \mathcal{M} \times \mathcal{B} \longrightarrow \mathcal{B}$. Across time, the tracking module uses a succession of n optical flow fields $M_1, \cdots, M_{n+1}$, starting from an initial boolean image O_1 that is an approximation to $\tilde{O}_1$. The output at image I_n, O_n, is intended to be an approximation to $\tilde{O}_n$. The algorithms that support this functionality are described in Chapter 6.

In normal operation, the tracking module performs in lock-step with the motion measurement module. As each new optical flow field arrives, another tracking step is made, and the resulting boolean object map is passed up. When a higher level module wants to start tracking a new object, the tracking module receives control information in the form of a request to start tracking a new object, along with a boolean object map that picks out the new object. The boolean object map must be current in the sense that it must pick out the position of an object in the image I_i, that was used for determining the current flow field M_i.

3.1.4 General interface

The more general interface to the tracking service is marked by the dashed line above the segmentation and tracking modules in Figure 3.1. This interface is intended to provide functionality somewhat like that of Pylyshyn and Storm's model of FINST's [78] and Agre and Chapman's [3] notion of markers. The term marker is adopted from the latter source. At this level, the service provides a small, fixed number of markers. Each marker can be associated with a region that is being tracked. If a region is associated with a marker, the marker is said to be bound, otherwise it is free. The system can track as many distinct moving objects as there are markers. The interface allows a controlling system to request that markers be bound to regions on the imaging surface. Similarly, the controlling system can expect to be told the current whereabouts of the region bound to each marker. Along with this support for markers, the tracking service can be asked to look for moving objects.

The interface can be summarized by its control signals, inputs and outputs.

Control signals:	• Find moving regions!
	• Bind marker j to a region!
Input:	• Regions to which to bind markers.
Output:	• Current positions of regions bound to markers.
	• Regions resulting from segmentation.

This interface is difficult to use in that a controller must respond to the service's output, issue control signals, and pass in data, all at frame rates of 10 to 15 frames per second. When a request is made to track a region of the image, that region must reflect the current approximate position of an object; tracking a region that no longer corresponds to an object is useless.

The interface is made still more difficult to use by imperfections in the tracking and segmentation modules. Over the course of many frames, the tracked region associated with a moving object may dwindle, or disappear. This degradation of markers can arise when an object stops, when it is occluded, or for many other reasons. Segmentation is not perfect either. It picks out parts of the imaging surface that are moving with similar trajectories. Multiple objects can move in unison and be picked out as one object. If only part of an object moves during the segmentation interval, only the moving part will be picked out.

3.1.5 Control

The control component that provides the object centroid interface is a client of the more general interface. Its goal is to keep track of the largest, independently moving object in the field of view, given the real-time requirements and the slightly unreliable segmentation and tracking operations provided from below.

This goal is achieved with a fairly complex, slightly heuristic, two part strategy: initially, choose a large object that has been picked out, or segmented, at least twice; and once an object has been chosen, keep track of the same thing as long as possible. This functionality is achieved by maintaining two markers for tracked objects.

One marker, the primary marker, is used for keeping track of an object that has been chosen to be tracked. The other marker, the secondary marker, is used for choosing objects. In normal processing, if a marker is bound to a region, it continues to be bound to the region, unless tracking loses the object altogether. If the primary marker is bound, the control component reports the centroid of the region currently associated with the primary marker at each frame. Periodically the control component requests that a segmentation be performed.

The segmentation module emits a small, bounded number of regions or *segmentations* that are considered to be sets of pixels that are moving together. Often one of these segmentations corresponds to the background. This segmentation should never be considered an independently moving object, and so is filtered out by testing whether most of the periphery of the image is included in the segmentation, or whether the segmentation consists of some large fraction of the image.

Occasionally, anomalous effects cause a scatter of like-moving small groups of pixels to be segmented as one moving object. In order to eliminate these fragmented segmentations, any segmentation, i.e. boolean image, that has too many isolated patches is discarded.

To see how to detect segmentations consisting of scattered pixels, consider their opposite: a single circular region. A measure that captures the coherence of a circular region is the ratio of square of the perimeter of the region to its area. For a circular region of any radius, this quantity is 4π. The measure increases for any less coherent patch than a single circle. A segmentation is discarded if the measure exceeds a threshold.

When a segmentation occurs, the control module attempts to match up new segmentations with the current bindings of its markers. This matching up of newly segmented regions and the current bindings of the two markers is done based on which markers are bound, and whether there is any correspondence between the segmented patches and the current bindings of the markers.

If (1) the primary marker is free, if (2) the secondary marker is in use, and if (3) a new segmentation that corresponds to the current binding of the secondary marker is found, then since the object has now been segmented at least twice, it is chosen as the new object to be tracked by the primary marker. That is, the primary marker is bound to the new segmentation corresponding to the old binding of the secondary marker, and the secondary marker is freed or bound to another new segmentation.

If (1) the primary marker is bound and if (2) some segmentation corresponds to the current binding of the primary marker, the primary marker is bound to the new segmentation.

Similarly, if (1) both the primary and the secondary marker are bound, and if (2)

some segmentation corresponds to the current binding of the secondary marker, the secondary marker is now bound to the new segmentation.

Finally, if the secondary marker is free, it is bound to the largest new segmentation that does not correspond to the primary marker.

This complex control strategy provides a transparent vision service that emits the centroid of a large moving object in the field of view at each frame. The shortcomings of the segmentation and tracking components are attenuated by the redundancy of two markers, as well as by repeated segmentations.

3.2 Camera pursuit systems

The tracking service has been used to implement two systems that pan a camera to follow a moving object. The basic control strategy is very simple. If the tracking service reports that an object is moving away from the center of the image, the camera is panned in the appropriate direction. The tracking service makes use of no information about how the robot is moving. Similarly, all panning commands are based on current measurements. No extrapolation of future object positions are computed or used.

The first system, constructed at Xerox PARC, makes use of a 16 thousand processor Connection Machine for vision computation, and a Heathkit Hero 2000 mobile robot to rotate the camera. The second system, constructed at Stanford University, performs vision computation on five Intel i860 processors, and pans and tilts a camera with a PUMA robot arm.

3.2.1 Latency

A fairly fundamental problem with doing real-time vision with contemporary digitizers and general-purpose computers is one of data transmission and latency. At some time t, an image is captured in the video camera and ends up in digitizer memory perhaps one frame time (one thirtieth of a second) later. The digital image then needs to be shipped into the main memory of processors that are going to perform vision

computations on them. Perhaps this second transmission takes another frame time. Once digital images are in main memory, the vision computation must be performed. Vision computation takes at least another frame time. If there is serial decomposition of vision processing such that one processor must finish before another begins, additional frame times can be lost. In the best of worlds, the results of vision processing match the world as it was a tenth of a second ago. For the CM system, the latency is almost half a second. For the Stanford system, the latency is perhaps a third of a second.

The problem with such latency, is that 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 third or a half 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.

3.2.2 The PARC system

The tracking system constructed at Xerox PARC is diagrammed in Figure 3.2. Video signal from the camera is passed to a digitizer mounted on the VME bus of a Sun-4. Digitized images are shipped into the Connection Machine where optical flow is computed, and segmentation and tracking occur. Centroids of tracked objects are sent up from the Connection Machine to the Sun. The Sun converts the centroid information into rotation commands that are transmitted to the Hero robot over a serial line.

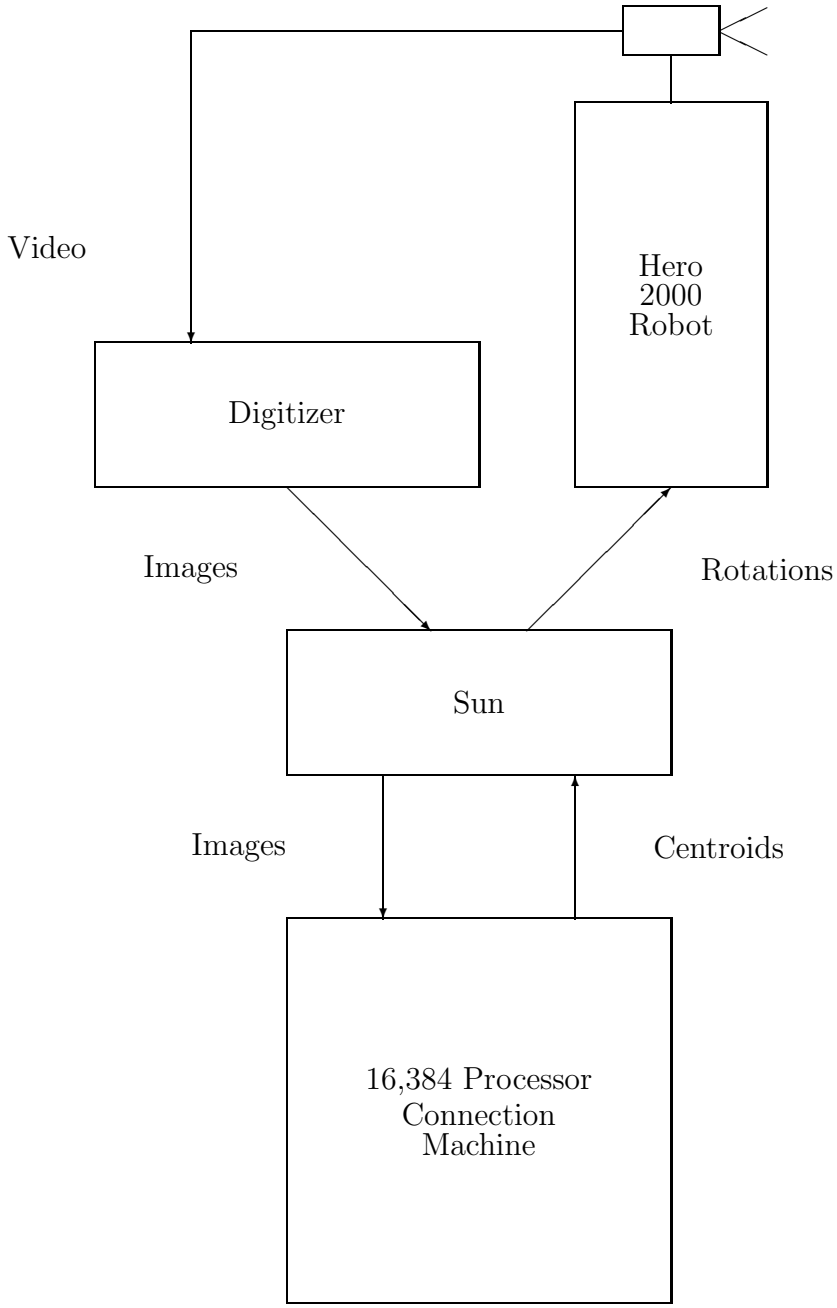


Figure 3.2: Connection Machine camera pursuit system

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 the processing time.

Second, the more images that are processed at once on the Connection Machine, the more efficient processing becomes. This economy of scale results from the fact that a serial front-end processor (the Sun-4) sends instructions to the Connection Machine. The resulting per-instruction overhead can be amortized by processing multiple images simultaneously. The 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 the system processes 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.

The Hero robot has a central pivoting joint that is used to rotate the camera. The camera is mounted so that its center of projection of the camera is directly over the pivoting joint to ensure that fairly uniform flow fields result from robot rotations as discussed in Section 2.2.

3.2.3 The Stanford system

The camera pursuit system built at Stanford is depicted in Figure 3.3. The second system is not massively parallel, but instead runs on five Intel i860 processors, mounted on two boards within the VME cage of a Sun. The Intel processors run at approximately 40 million instructions per second (MIPS), providing a total of about 200 MIPS. The CM can perform orders of magnitude more operations per second than 200 MIPS, yet the Stanford system only performs a little slower than the PARC system (ten hertz versus 15 hertz) on slightly smaller images (128×114 versus 128×128). The fact that the same computation can be performed with fewer MIPS is accounted for by the elimination of most inter-processor communication, and the differences between the techniques used for dynamic programming on serial and SIMD machines.

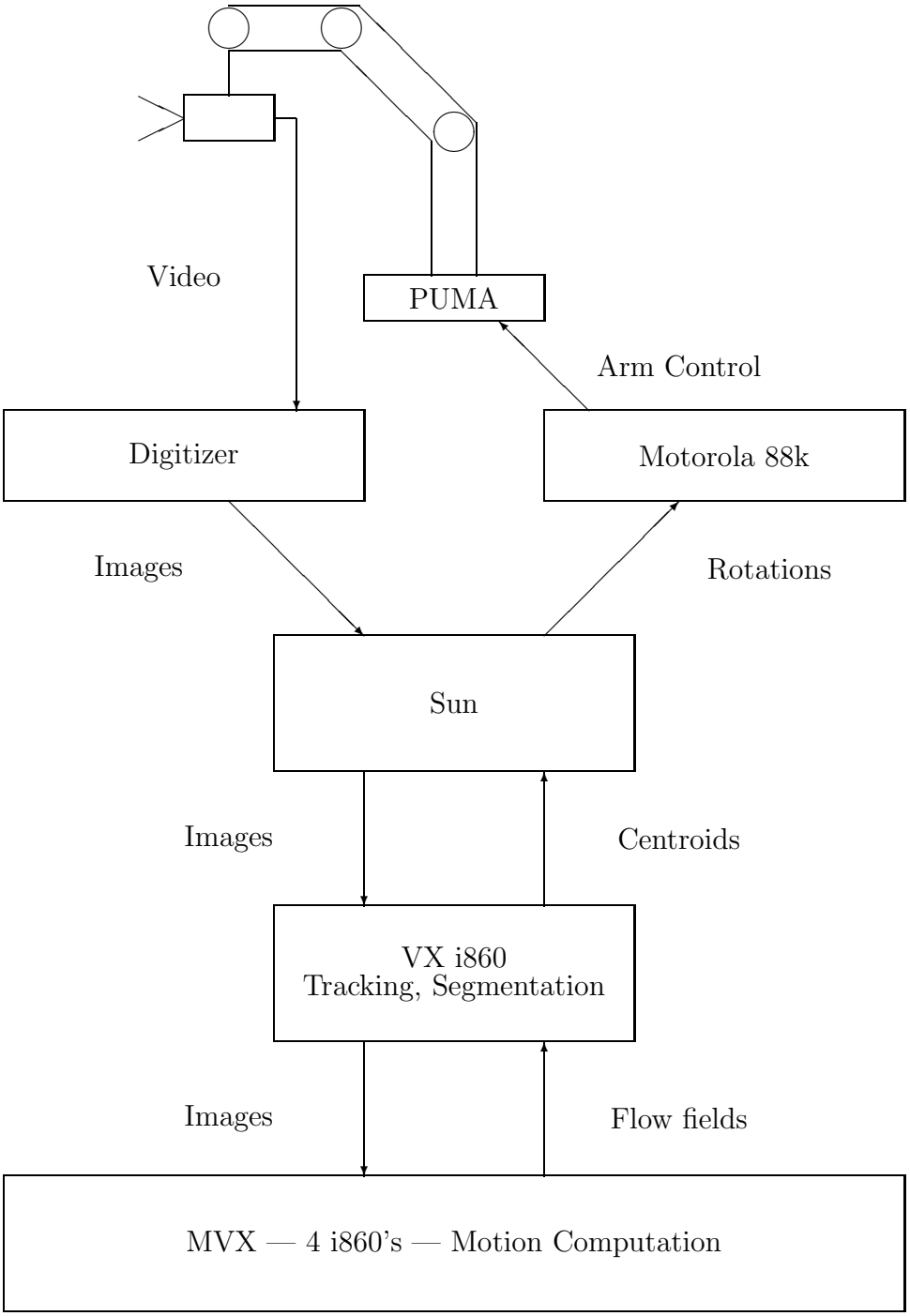


Figure 3.3: Intel i860 based camera pursuit system

Again, video signal from the camera is passed into a digitizer on the VME bus of a Sun. Digitized images are passed from the Sun to a VX board[1] containing a single i860 processor. The VX divides the image into four slightly overlapping horizontal slices. One slice is distributed to each of the four i860 processors on the MVX board. Each processor on the MVX computes one fourth of an optical flow field. The flow field is passed back to the VX where segmentation and tracking are performed. The VX passes centroid information to the Sun. The Sun in turn, passes rotation information to a Motorolla 88k processor that performs computation relating to controlling the joint torques of the Puma arm.

A lot of time is devoted to shipping data between processors, yet the whole system runs at about ten frames per second, on 128×114 images. Controlling a Puma arm with five degrees of freedom is considerably more difficult than controlling a single rotating joint on the Hero 2000. However, using Khatib's COSMOS system [55], the manipulator control problem is transparently specified in operational space so that no effort need be expended on geometric or kinematic issues. Rotation is specified about a single point in operational space, namely, the center of projection of the camera.

Figures 3.4-3.6 show a sequence of images captured while the pursuit system was operating. The white line shows the outline of the tracked object.

[1]The two boards containing the i860 chips were originally designed by Sun as a graphics accelerator. The boards are currently manufactured by a company called Vicom. One board, called a VX has one i860 processor on it. The other board, an MVX has four i860's on it.

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 3.4: Camera pursuit sequence part 1

Frame 21 Frame 22 Frame 23 Frame 24

Frame 25 Frame 26 Frame 27 Frame 28

Frame 29 Frame 30 Frame 31 Frame 32

Frame 33 Frame 34 Frame 35 Frame 36

Frame 37 Frame 38 Frame 39 Frame 40

Figure 3.5: Camera pursuit sequence part 2

Frame 41 Frame 42 Frame 43 Frame 44

Frame 45 Frame 46 Frame 47 Frame 48

Frame 49 Frame 50 Frame 51 Frame 52

Frame 53 Frame 54 Frame 55 Frame 56

Frame 57 Frame 58 Frame 59 Frame 60

Figure 3.6: Camera pursuit sequence part 3

Chapter 4

Motion measurement

This chapter describes the algorithm used for computing optical flow fields from a pair of images. As discussed in Section 2.3, if two images, I_i and I_{i+1} are captured at times t_0 and t_1 respectively, then the motion field $\tilde{M}_i$ represents the motions that occurred between t_0 and t_1. The purpose of the motion algorithm is to compute an optical flow field M_i that is an approximation to $\tilde{M}_i$ given the two images I_i and I_{i+1}.

The requirements for what the optical flow algorithm must be able to do arise from both the tracking system as a whole, and the tracking algorithm itself. It must function on the kinds of images and image sequences that the tracking system might encounter in indoor or outdoor environments. It must produce optical flow fields that are suitable for the segmentation and tracking algorithms. Finally, it must be suitable for real-time implementation.

Generalizing over sets of images is a rash thing to do, however, the images captured at ten to fifteen frames per second by the panning robot systems have tended to have several common characteristics. Objects moving in the scene are non-rigid, and hence tend to display multiple regions of distinct motion. Things in the field of view move fairly slowly, but displacements are very often more than one pixel. Both objects in the scene, and the camera itself tend to shift directions of motion rapidly. Since the tracking system is intended for use in unstructured environments, images can come from either indoor or outdoor scenes.

Algorithms for segmentation and tracking rely on the cross temporal behavior of regions of the image. The behavior of regions is determined by the behavior of individual pixels. The optical flow field must be dense; each pixel needs to have an estimate as to where it has been displaced.

Optical flow fields do not need to be terribly precise approximations to motion fields, since discretized motion displacements are used. Indeed, Verri and Poggio [98] argue that the underlying evidence, the optical flow field associated with variation in image brightness patterns, rarely agrees with the motion field. Although the flow field does not need to be quantitatively precise, two qualitative properties must be preserved. Where there is a discontinuity in the motion field, there ought to be a discontinuity in the flow field somewhere nearby. Similarly, where there is a region of relatively uniform displacement vectors in the motion field, the flow field ought to consist of homogeneous regions of flow vectors.

Two general constraints on motion in the visible world make computing optical flow possible. Motions in the world result in changing intensity patterns. Individual estimates of motion can be made based on these changing patterns. Parts of objects moving in the world tend to move coherently and hence result in a locally smooth motion field. The coherence of motion supports the assumption of a locally smooth flow field and allows individual motion estimates to be combined to achieve credible measurements.

Making use of these two constraints, the optical flow algorithm has two steps: an initial motion measurement step and a smoothing step. For each pixel p on the first image, the initial motion measurement step determines a best match in the area surrounding p on the second image. Since images are noisy, and since the initial estimation step is not perfect, the second phase of the algorithm locally filters the initial estimates to produce a better approximation to the motion field.

The output of any locally computed optical flow algorithm is likely to contain colliding flow vectors, i.e., multiple pixels that are mapped to the same pixel. In the case of a receding object, such collisions make sense since part of an object that projects to two pixels may shrink to one. More commonly, collisions arise from occlusion. Parts of the segmentation and tracking algorithms are more simply defined if the optical

flow field is injective (has no collisions). In a procedure called rectification, an injective flow field can be generated by locally prioritizing the estimated displacement for each pixel.

First, the initial motion measurement step is presented. The smoothing step is discussed in Section 4.2. Following the description of the basic motion algorithm, the rectification step is described. Section 4.4 presents two practical difficulties found in images taken with a moving camera, along with partial solutions. Next, other approaches to computing optical flow fields are laid out. Finally, the overall optical flow algorithm is discussed.

4.1 Initial motion measurements

The task of the initial motion measurement phase is to determine for each pixel on one image, without considering neighboring pixels' opinions, the most likely corresponding pixel on the next image. As input, the initial motion measurement algorithm takes two images, I_i and I_{i+1}. As output, it produces a dense, multivalued flow field $M_i^* \in \mathcal{M}^*$.

The time interval between the two images is assumed to be quite small, and hence things in the image are expected to travel only short distances. As movements are small, the part of the world that corresponds to pixel p on image I_i will appear within a small radius of p on image I_{i+1}. Thus, the most likely displacement for a pixel p on image I_i can be determined by searching for the most similar (least dissimilar) pixel on I_{i+1}. The area that is explored for a best displacement is termed the *search window*. Rather than attempting to determine the best displacement on the basis of the gray-level of a single pixel, the best displacement is determined by matching the gray-levels of a patch of pixels. This patch of pixels is known as the *correlation window*. The measure of dissimilarity used for comparing patches pixels is the *sum of squared differences* (SSD).

Both computational cost and the quality of the results demand that the search window be limited in size. The cost of the search goes up with the square of the search radius. Similarly, the chances of a spurious best match increase with the size

		(-2,2)	(-1,2)	(0,2)	(1,2)	(2,2)		
(-4,1)	(-3,1)	(-2,1)	(-1,1)	(0,1)	(1,1)	(2,1)	(3,1)	(4,1)
(-4,0)	(-3,0)	(-2,0)	(-1,0)	(0,0)	(1,0)	(2,0)	(3,0)	(4,0)
(-4,-1)	(-3,-1)	(-2,-1)	(-1,-1)	(0,-1)	(1,-1)	(2,-1)	(3,-1)	(4,-1)
		(-2,-2)	(-1,-2)	(0,-2)	(1,-2)	(2,-2)		

Figure 4.1: The search window used in the implemented systems

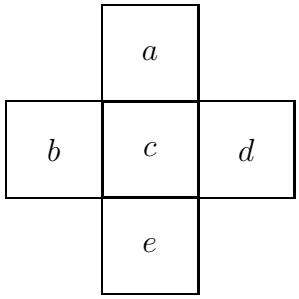


Figure 4.2: The correlation window

of the search window. Hence, a small fixed search window, or set of displacements, is determined. For a circular search window, the set of displacements is $\mathbf{D} = \{ d \in \mathcal{Z} \times \mathcal{Z} \mid \|d\| < r_v \}$, for r_v a small integer. In practice, the search window is not circular, but rather elliptical. Figure 4.1 depicts the set of 37 displacements used in the implemented systems.

The correlation window used consists of five cells: the pixel and its four four-connected neighboring pixels (see Figure 4.2). Given a pair of images I_i and I_{i+1}, define the degree of mismatch for a pixel $p \in \mathbf{P}$, and displacement $d \in \mathbf{D}$ to be

$$E(p,d) \equiv \sum_{\|\delta\| \leq 1} (I_i(p+\delta) - I_{i+1}(p+\delta+d))^2.$$

The result of the initial motion measurement phase, M^* for each pixel p, is the set

of displacements that produce the minimal error term:

$$M^*(p) \quad \equiv \quad \{\, d \in \mathbf{D} \mid E(p,d) = \min_{d' \in \mathbf{D}} E(p,d') \,\}.$$

It would be preferable for the initial estimate phase to produce a single valued flow field, rather than a multivalued one, but since ties can occur, the result must be multi-valued. In practice, a random minimal element or even an arbitrary minimal element can be chosen for each pixel.

4.1.1 Justifying SSD correlation

Why use the sum of squared differences of intensities as a dissimilarity measure? It is a common technique for the estimation of stereo and motion disparity, it is fairly inexpensive to compute and has reasonable behavior in the presence of noise. One way to think about a correlation window containing k cells is as if it were a point in k-space. Then two pixels are more or less similar depending on the Euclidean distance between them as characterized by their surrounding windows.

This view of SSD correlation is particularly attractive when thinking about noise in image formation. A reasonable model of the differences between two images taken of the same unchanging scene at different times, is that the second image will differ from the first by some amount of Gaussian noise. When viewed as points in a k-dimensional space, the points that result from changes induced by Gaussian noise will cluster about the original point. The Euclidean distance between the new point and the old point will provide a good gauge of the likelihood that one local area is the result of Gaussian noise on the other local area.

Kass [52] suggests using multiple local representations of local image properties as the basis for more robust matching. This approach, although promising, would be expensive to compute, and would compound difficulties at motion discontinuities. Local image properties other than intensities tend to be aggregations of information from the surrounding area. Aggregation of image information across motion discontinuities results in measures that do not recur in subsequent images.

a	b	c
d	e	f
g	h	i

g	d	a
h	e	b
i	f	c

Figure 4.3: A correlation window before and after a $-90°$ rotation

Burt *et al.* [23] examine several correlation measures other than SSD's, including cross-correlation and normalized cross-correlation, and the effects of varying correlation window sizes. These researchers discount SSD correlation since it can be drastically affected by overall changes in image intensity. However, given a suitably fast frame rate such that changes in lighting are not significant, SSD correlation can extract information both from the local texture and the local mean intensity.

4.1.2 Why this window size?

Several constraints bear on the choice of correlation window sizes. Camera-relative rotation and discontinuities in the motion field argue for a small window. Balanced against demands for a small window is the need for a measure with a reasonable amount of discrimination.

To see that camera-relative rotation in a scene argues for a small correlation window, note that what matters in correlation is getting the individual cells of the window on one image to line up, or overlap with the appropriate cells on the second image.

Given a grid with uniquely labeled cells, define the *degree of overlap* between the grid and itself translated by up to one half a cell in the vertical and horizontal axes, and rotated an arbitrary amount to be the fraction of overlap of identically labeled cells. Figure 4.3 shows a grid, and the same grid rotated $-90°$. With no rotation or

translation, the grid achieves a degree of overlap of 1 since each cell fully overlaps a cell with the same label. Rotated $-90°$, the grid achieves a maximal degree of overlap of $\frac{1}{9}$ since only one cell overlaps another with the same label.

Without rotations, no matter what size the SSD window, the worst degree of overlap possible is $\frac{1}{4}$ at displacement $\langle .5, .5 \rangle$. Similarly, with a single-celled correlation window, and arbitrary translation and rotation, the least possible degree of overlap is $\frac{1}{4}$.

Consider the case of a rotation with no translation, and a large correlation window. For a given rotation θ, cells further out from the center of rotation will traverse greater distances. If the center of a cell shifts more than $\sqrt{2}$ pixels then it cannot possibly overlap at all with its corresponding cell[1]. For a cell, the center of which is r pixels away from the center of rotation, an angle $\theta = arcsin\frac{\sqrt{2}}{2r}$, will preclude any overlap with the corresponding cell. Thus, given a fixed rotation θ, a larger correlation window will have a proportionally smaller degree of overlap. If camera-relative rotations are likely, the correlation window ought to be as small as possible. Burt *et al.* [23], studying the effects of window size on correlation given image dilation, empirically conclude that the correlation window should be small.

Discontinuities in the motion field arising from independently moving scene elements, or flexing objects also argue for small correlation windows. On the perimeter of an object that is translating relative to a background, for a correlation window of diameter d_c, the degree of overlap is at best approximately $\frac{d_c}{2}$, because even when object pixels from the two images line up perfectly, cells corresponding to the background all differ. The degree of overlap improves as the correlation window is moved away from the perimeter of the object. As the correlation window increases in size, there are more pixels where the correlation window overlaps the motion boundary, and the worst case degree of overlap gets smaller. Researchers have attempted to ameliorate the effects of motion boundaries on SSD correlation by examining the local SSD surface [7, 15], but the methods for characterizing SSD surfaces and ways of using these characterizations do not yet appear practical.

On the other hand, arguing for a large correlation window, there is a need for

[1]Two unit squares whose centers are separated by $\sqrt{2}$.

a certain degree of discrimination among pixels. Consider a single-celled correlation window with eight-bit gray-levels. For a set of 37 displacements, on a random image, the odds are approximately $\frac{9}{64}$ that a pixel other than the correct one will have the same gray-level. Taking into account noise, subpixel displacements, and the fact that intensities tend to vary smoothly, the odds of there being an ambiguity, or worse, a false best displacement, become even greater.

A small, 5 cell correlation window is used. It is intended as a good middle point in the space of correlation windows. With respect to rotation, after a single celled window, the 5 cell window is the best vertically and horizontally symmetric, pixel-centered correlation window. The worst case overlap for a rotation of 180° and a translation of $\langle .5, .5 \rangle$ is $\frac{1}{20}$. At object boundaries of square objects, ignoring translations and rotations, the 5 cell window has a worst case degree of overlap of $\frac{4}{5}$ except at corners where it is $\frac{3}{5}$. Using the 5 cell correlation window there are 2^{40} distinct SSD patches dramatically reducing the odds of ambiguity. For displacement vectors centered between pixels, a four cell window would be best.

Okutomi and Kanade [73] have developed methods for adaptively varying window sizes to allow stable, precise estimation. They have analyzed the one-dimensional case, and extended the analysis to the two-dimensional case of stereo. Neither one-dimensional matching, nor stereo displacements involve rotation, and hence rotation is not modeled. Adaptive sizing techniques also incur additional expense in that the computation is no longer uniform across the image.

4.1.3 Limitations

SSD correlation with a fixed set of discrete displacements has several weaknesses. The primary ones result from motions that are larger than the maximum motion detected, or smaller than one pixel.

No matter what motion occurs, SSD correlation will always produce an answer, namely, the displacement with the least degree of mismatch. In the case of an object moving further than the largest measured displacement, correlation cannot produce a correct answer. However, since objects rarely appear as uniform intensities, but rather as increasing and decreasing intensity slopes, in the case of large motions,

SSD correlation generally produces the longest displacement vector that points in the same direction as the correct displacement. Despite the correct patch not being in the search window, the general ramp-like nature of images causes the patch in the search window that is closest to the patch being sought to be the most similar.

Producing an answer that points in the right direction is better than a random answer, but it leaves open the question of how to deal properly with large motions. One cannot go on extending the local search window indefinitely, as each increase in size also increases the ambiguity of the match. Researchers such as Burt [24], Glazer [39] and Anandan [7] have proposed performing correlation using hierarchical structures across scale in order to measure large motions. Such proposals are discussed in Section 4.5. Assuming that computational costs can be handled, increasing the speed of image capture appears to be a promising avenue. For this discrete approach to computing motion, increasing the frame rate has the disadvantage that more motions will appear as displacements of less than one pixel.

Projected motions that are smaller than one pixel pose a slightly different problem. If an object is moving sufficiently slowly, such that its motion is always less than half a pixel, the correct discretized displacement is always stationarity. Thus a sloth could crawl past a computer that was making discrete motion measurements and never register a non-zero displacement.

4.1.4 Complexity

Any algorithm for computing SSD correlation over a local search window must perform one difference and determine one squared term for each displacement in the search window. In parallel, forming a squared difference for each displacement is complicated by the need to get image data from neighboring processors. The parallel computation can be efficiently performed at pixel p by making a local copy of the gray-levels corresponding to the search window surrounding p on $p^{'}$s processor. This local copy can be formed using the dynamic programming techniques described in Section 2.7, with $\otimes$ instantiated as concatenation. Accessing the right image data for SSD correlation in parallel costs $\mathcal{O}(\lg |\mathbf{D}|)$ hypercube communications. Computing squared differences for each displacement costs $\mathcal{O}(2|\mathbf{D}|)$ operations per pixel both in

serial and parallel.

Shared results can be used in computing the sums of these squared differences. For a square SSD template, local dynamic programming techniques apply. Matching an unweighted, square SSD window across the image can be performed in serial, in time independent of the window size. In parallel, the same computation can be done in time dependent on the log of the correlation window size. However, the template used is not square, and is very small. If the squared differences of the SSD template are labeled a, b, c, d, e as in Figure 4.2, then the sum can be decomposed as $(a + b) + c + (d + e)$. Note that for one pixel, $a + b$ is the same quantity as $d + e$ for another pixel one up and one to the left. Using these shared results, the sums for a five pixel template at $|\mathbf{D}|$ displacements can be computed with $\mathcal{O}(3|\mathbf{D}|)$ additions per pixel. In parallel, this computation incurs three hypercube communications to bring the various sums to the local processor. The final step of forming an initial estimate involves finding the best displacements in time $\mathcal{O}(|\mathbf{D}|)$. In summary, the complexity of performing SSD correlation at $|\mathbf{D}|$ displacements with a five pixel correlation window is:

	Computation		Communication
Phase	Serial	Parallel	Hypercube
Initial measurement	$\mathcal{O}(6\|\mathbf{D}\|N^2)$	$\mathcal{O}(6\|\mathbf{D}\|)$	$\mathcal{O}(\lg \|\mathbf{D}\| + 3)$

4.2 Motion smoothing

The results of the initial motion measurement step can be quite noisy. The second half of the motion computation is a filtering step intended to remove spurious motion estimates. Motion fields have the distinctive property that they vary smoothly in regions of the imaging surface that correspond to the interior of objects, and are discontinuous at the perimeters of independently moving objects.

Traditional filtering techniques such as mean filtering only make sense when the data to be filtered varies continuously. At the edge of a translating object, taking the mean of the neighboring displacement estimates would result in a strange displacement half way between the displacement of the object and that of the background.

Many optical flow algorithms [6, 47] use relaxation to find an optimally smooth flow field, or some form of least squares fit [54, 61] to enforce local smoothness. However, enforcing smoothness across discontinuities in the motion field introduces many displacement vectors that are only artifacts of the smoothing process. Spoerri and Ullman [87] look at the problem of detecting discontinuities in the motion field before attempting to find consistency. Their approach involves looking for bimodality in histograms of initial motion measurements. The key observation in their algorithm is that the modes in a histogram of local measurements indicate the predominant local motions.

Because the underlying motion field can contain discontinuities the filtering operation used is *mode filtering*. The result of mode filtering at each pixel is the most popular displacement in the region surrounding the pixel. As input, mode filtering takes an initial motion estimate M^*, and produces an optical flow field M, and a field of modes V.

Formally, let r_m denote the radius of the mode filter region. Define the votes, V for a displacement d at a pixel p,

$$V(p,d) \quad \equiv \quad |\{\, \langle x,y\rangle \mid |x| \le r_m \wedge |y| \le r_m \wedge d \in M^*(p + \langle x,y\rangle)\,\}|.$$

By abuse of notation, the maximum number of votes for a pixel is denoted:

$$V(p) \quad = \quad \max_{d \in \mathbf{D}} V(p,d).$$

A slight complication arises from the fact that there may not be a unique mode of a distribution. The problem can be side-stepped by choosing one of the most popular displacements at random. In practice, ties occur rarely. The result of the mode filtering step is the displacement with the most votes.

$$M(p) \quad \equiv \quad \sup_{\lambda d \cdot V(p,d)} \mathbf{D}$$

Or equivalently, for an optical flow field mapping pixels to pixels,

$$M(p) \quad \equiv \quad p + \sup_{\lambda d \cdot V(p,d)} \mathbf{D}.$$

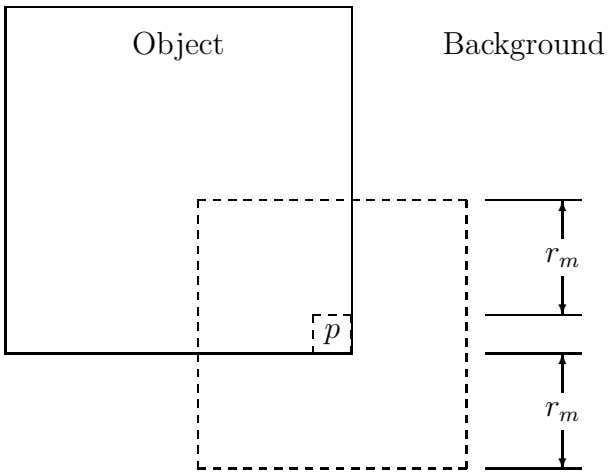


Figure 4.4: Mode filtering pixel p

Mode filtering does not appear to be a commonly used technique. Coleman *et al.* [27] appear to have named the technique and used it for removing very small regions of distinct gray-levels in images prior to attempting segmentation. Davies [30, 31] suggests using mode filtering for gray-level image enhancement.

4.2.1 Limitations

Mode filtering has several weaknesses resulting from its expectation of finding coherence across a local area. When mode filtering across a discontinuity, odd effects can occur. Since these effects only become more complicated when the underlying data consists of motion estimates, the case of mode filtering a binary image is considered. The two valued image data is assumed to be noise free. If the underlying data is randomly perturbed, approximately the same distributions of mode filter votes will arise. If on the other hand, the underlying input data are skewed in some way, mode filtering may well mismeasure. See Section 4.4.1 for an example of particularly pernicious skew.

Mode filtering has distorting properties at corners. Figure 4.4 depicts a square

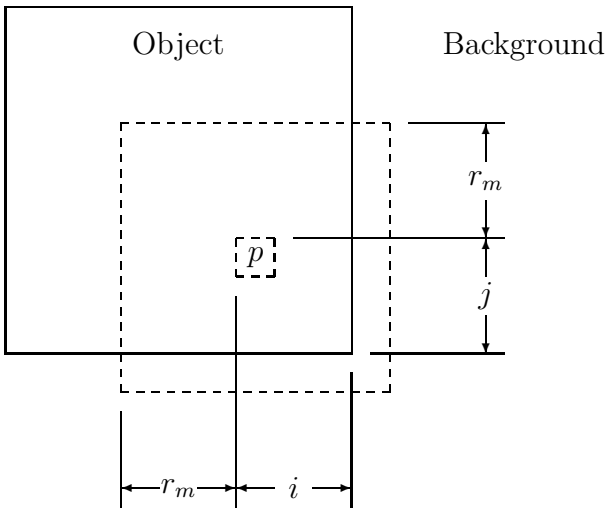


Figure 4.5: Position of pixel p

object on a distinctly labeled background. The corner pixel p of the square object must be misclassified as part of the background by mode filtering; the mode filter region centered about p, indicated by the dashed line, covers only $(r_m + 1)^2$ pixels of the square object, while the mode filter region as a whole contains $(r_m + 1 + r_m)^2$ pixels. For any radius $r_m \geq 1$, $(r_m + 1 + r_m)^2 > (r_m + 1)^2$ and so the most popular labeling at p must be that of the background region.

The corner pixel p is not the only misclassified pixel. The number of pixels that get misclassified near the corner of a square object is a function of the radius of the mode filter. Whether an individual pixel near the corner of an object gets misclassified depends on its position relative to the corner. The corner relative position of a pixel can be indicated by how far it is from the edges of the square object. In Figure 4.5 the distance from the center pixel of the mode filter to the right edge of the square object is labeled i, and the distance from the center pixel of the mode filter to the bottom of the square object is labeled j. The area of the whole mode filter is $(2r_m + 1)^2$. From the figure, it is easy to see that the area of the part of the mode filter that overlaps the square object is $(r_m + i)(r_m + j)$. A pixel at position $\langle i, j \rangle$ for $1 \leq i, j \leq r_m + 1$ is

misclassified when the area of overlap between the mode filter region and the square object is less than half the area of the mode filter region, i.e., when

$$(r_m + i)(r_m + j) \quad < \quad \frac{1}{2}(2r_m + 1)^2.$$

The number of pixels erroneously classified when mode filtering with radius r_m at the corner of a square object, written $e(r_m)$, is the cardinality of the set of such pixels.

$$e(r_m) \quad \equiv \quad |\{\, \langle i, j\rangle \mid 1 \le i, j \le r_m + 1 \wedge (r_m + i)(r_m + j) < \frac{1}{2}(2r_m + 1)^2 \,\}|$$

Rewriting this in terms of j,

$$e(r_m) \quad = \quad |\{\, \langle i, j\rangle \mid 1 \le i, j \le r_m + 1 \wedge j < \frac{1}{2}\frac{(2r_m + 1)^2}{r_m + i} - r_m \,\}|.$$

A continuous approximation to the cardinality can be computed by finding the area under the curve, for $1 \le x \le r_m + 1$ of

$$y \quad = \quad \frac{1}{2}\frac{(2r_m + 1)^2}{r_m + x} - r_m.$$

This area can be found by integration.

$$e(r_m) \quad \approx \quad \int_1^{r_m+1} \frac{1}{2}\frac{(2r_m + 1)^2}{r_m + x} - r_m dx$$

By simplification and change of variable, the integral becomes

$$e(r_m) \quad \approx \quad \frac{1}{2}(2r_m + 1)^2 \int_{r_m+1}^{2r_m+1} \frac{1}{z} dz - \int_1^{r_m+1} r_m dx.$$

Using the integral of $\frac{1}{z}$, a closed form approximation can be found.

$$e(r_m) \quad \approx \quad \frac{1}{2}(2r_m + 1)^2[\ln(2r_m + 1) - \ln(r_m + 1)] - r_m^2$$

For the mode filter used in practice, where $r_m = 3$, the 5 pixels shown in Figure 4.6 are misclassified (the approximation yields 4.7). For $r_m = 7$, 23 pixels are actually misclassified, while the approximation yields 21.7.

Another shortcoming of mode filtering, even assuming perfect initial estimates, is that mode filtering will hide sufficiently small objects. Consider a square object of

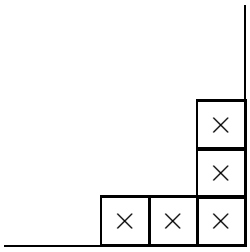


Figure 4.6: The five misclassified pixels for $r_m = 3$

size $k \times k$. If $k^2 < 2r_m + 1$ no trace of the object will appear in the mode filtered result. Thus for the mode filter radius of three used in practice, an object that is smaller than five by five will disappear.

If the results of the initial motion measurement phase were not discrete, but instead each pixel had a precise floating point estimate of its motion, the applicability of mode filtering comes into question. There are standard methods for finding the mode of real-valued data [77], but it is an open issue as to whether these techniques would be feasible for efficient implementation.

4.2.2 Complexity

Mode filtering involves two steps: histogramming the initial measurements in the local region, and finding the mode of this histogram. For simplicity, rather than speaking in terms of the mode filter radius r_m, the mode filter diameter d_m, is used in the analysis. In serial, $d_m = 2r_m + 1$. In parallel, since a a binary recursive composition of partial sums is desired, $d_m = 2^{\lceil \lg(2r_m+1) \rceil}$.

Finding the mode is straightforward. If one ignores the possibility of ties, and is willing to choose the first or last most popular displacement, computing the mode is a one-pass maximum computation involving $|\mathbf{D}| - 1$ comparisons per pixel. If some sort of randomization is included, the cost goes up. It can require two passes over the histogram, once to find the maximum number of votes and count the number of pixels having this number of votes, and once to select the random element.

The output of the initial motion estimate computation $M^*(p)$ can be represented as a vector of ones and zeros of length $|\mathbf{D}|$. An entry in such a vector will be one

if the corresponding displacement is a displacement with the least SSD score, zero otherwise. Viewing initial estimates as vectors makes computing the histogram of initial estimates amenable to the dynamic programming techniques introduced in Section 2.7. The group operation $\otimes$ is, in this case, vector addition. The width w and height h are both d_m.

For the parallel case, as predicted by the analysis in Section 2.7, generating local histograms of initial estimates can be performed with $\mathcal{O}(2 \lg d_m)$ vector additions and $\mathcal{O}(2 \lg d_m)$ hypercube communications. Since each vector is of length $|\mathbf{D}|$, computing local histograms costs $\mathcal{O}(2 \lg d_m |\mathbf{D}|)$ additions. Combined with the $|\mathbf{D}| - 1$ comparison operations, mode filtering costs $\mathcal{O}((2 \lg d_m + 1)|\mathbf{D}| - 1)$ operations on a SIMD machine.

The analysis in Section 2.7 would suggest that computing the motion histograms in serial would cost four vector additions per pixel, where each vector addition would involve $|\mathbf{D}|$ scalar additions. However, by choosing exactly one displacement in the initial estimate step this figure can be reduced. Choosing one displacement for each pixel allows the individual results of the initial estimate phase to be encoded as indices rather than $|\mathbf{D}|$ element vectors. As noted in Appendix B, forming the aggregation in a matrix F for a single pixel requires the application of $\otimes$ and $\otimes^{-1}$ to a single element of F each. Since these single elements of F can be encoded as indices, the operations on single elements of F can be performed with one scalar addition per element, rather than $|\mathbf{D}|$ scalar additions. Thus the local histogram can be generated in $\mathcal{O}(2|\mathbf{D}| + 2)$ scalar additions per pixel. The whole mode filtering step including finding the mode can be performed in $\mathcal{O}(3|\mathbf{D}| + 1)$ operations per pixel on a serial computer.

The following table summarizes the complexity of mode filtering in serial and parallel.

	Computation		Communication
Phase	Serial	Parallel	Hypercube
Mode filter	$\mathcal{O}((3\|\mathbf{D}\| + 1)N^2)$	$\mathcal{O}((2 \lg d_m + 1)\|\mathbf{D}\| - 1)$	$\mathcal{O}(2 \lg d_m \|\mathbf{D}\|)$

4.3 Rectifying optical flow fields

The motion computation results in a displacement vector for every pixel on the image. For some pixels such as pixels that have been occluded, having a motion estimate makes little sense. There is no information available as to where a pixel that has been occluded has moved. A consequence of assigning a displacement vector to each pixel is that in general, the optical flow field may map several pixels to the same point p; the pixels collide. Both the segmentation and tracking algorithms can be simplified if the optical flow field is injective. Since some of the displacement vectors that map to the same point are artifacts resulting from occlusion, and since injective optical flow fields simplify the computation, it is desirable to *rectify* the optical flow field M and produce an injective flow field $\overline{M}$.

If the optical flow field is to be made injective, obviously only one pixel can be allowed to map to the point p. A reasonable choice for which pixel should map to p would be the pixel with the most credible motion estimate. In the case of a mode filtered motion computation, the number of votes $V(p)$ (see Section 4.2) that the winning displacement received at pixel p is a good estimator of the credibility of this motion estimate. Thus one can produce an injective optical flow field by ensuring that for each pixel p' in the range of the flow field, of the pixels that originally map to p' only the pixel p with the highest number of votes maps to p'.

To see that the number of votes the winning displacement received ought to be a useful measure of credibility near occlusion boundaries, consider pixels that are occluded and those around them. Occluded pixels will tend to have relatively random initial estimates, since there is no correct answer. Pixels on either side of the occluded region will have much more consistent estimates. Thus the mode filtering region over an occluded pixel will tend to be composed of a set of random estimates and two or more neighboring distributions. The number of votes for the most popular displacement at occluded pixels will tend to be lower than the number of votes at neighboring unoccluded pixels. Since rectification could be performed with any credibility estimate, let $L : \mathbf{P} \longrightarrow \mathcal{N}$ be a function that maps a pixel to a natural number indicating how likely its motion estimate is. In the current implementation,

L is instantiated by the vote count V from the mode filtering step.

The rectified inverse of M, $\overline{M}^{-1}(p)$ can be defined as the pixel p' that M maps to p with the highest credibility.

$$\overline{M}^{-1}(p) \quad \equiv \quad \sup_{L} \{\, p' \mid M(p') = p \,\}$$

Using this definition of an injective inverse to the optical flow field, an injective *rectified* optical flow field $\overline{M} : \mathbf{P} \longrightarrow \mathbf{P} \cup \{\perp\}$ is easily defined.

$$\overline{M}(p) \quad \equiv \quad \begin{cases} \perp & \text{if } \neg\exists p' \quad \overline{M}^{-1}(p') = p \\ p' & \text{where } \overline{M}^{-1}(p') = p, \text{ otherwise.} \end{cases}$$

In serial, rectification involves two address computations and at most one comparison per pixel. In parallel, rectification involves two non-hypercube communications per pixel.

	Computation		Communication	
Phase	Serial	Parallel	Hypercube	Non-hypercube
Rectification	$\mathcal{O}(3N^2)$	—	—	$\mathcal{O}(2)$

4.4 Complications and improvements

Several practical difficulties arise when computing motion from real image sequences. This section describes these difficulties and the solutions that have been used to mitigate their effects.

4.4.1 Motion aliasing

Straight, high contrast elements of a scene can result in highly textured cyclic image patterns that are classed as a form of spatial aliasing. When these straight, high contrast edges project onto the imaging surface at certain angles, small camera motions can result in pronounced apparent shifts of texture along these edges. Often, such high contrast elements occur in regions of low texture; the illusory texture shifts can present more compelling evidence of motion than actual camera-relative motion. The

Figure 4.7: A straight, high contrast edge with slope $-1:8$

Figure 4.8: The high contrast edge after sampling

initial motion measurement phase of the motion algorithm may detect the illusory motion. Since the apparent shift can occur uniformly along a straight line, most of the estimates along the line will observe this illusion. This mass hallucination can influence the mode filtering step and result in lines of illusory motion in the resulting flow field. This problem is referred to as the motion aliasing problem. First, the formation of these spatial aliasing patterns is discussed. Next, the conditions under which the illusory motions occur are described. Finally, an explanation of how these effects interfere with segmentation and tracking and a description of the solution used to sidestep the problem are presented.

Spatial aliasing

Consider the situation shown in Figure 4.7 of a high contrast diagonal edge of downward slope, one in eight. Figure 4.8 shows the same edge after it has been sampled. Notice that the pattern of gray-levels along the edge is cyclic with a periodicity of approximately 8. Without loss of generality, only slopes between 0 and 1 (45°) will

Figure 4.9: The high contrast edge shifted down and sampled.

be considered. The periodicity of the cyclic step function that corresponds to a line with a slope of the form $1 : j$ for integers j, is j. For other angles θ, the apparent periodicity is approximated by $\cot\theta$. Rosenfeld and Kak [81] touch on this form of spatial aliasing, and Legault [57] relates it to the display of images.

Spatial aliasing in motion

The effects of small camera motions on patterns of spatial aliasing are considerable. Translations can have dramatic effects. Consider shifting the edge in Figure 4.7 down by one sampling pixel. As the edge moves down, the cyclic pattern of the sampled image shifts along the edge. Figure 4.9 shows the result of sampling Figure 4.7 shifted down by less than one half pixel.

These shifting patterns of spatial aliasing can strongly effect the motion computation. If the high contrast edge has a fairly steep slope relative to a line perpendicular to the direction of movement (i.e., short periodicity), then any camera motion is likely to produce a measurable shift in aliasing pattern. If the slope is say one in six, then a camera movement can shift the pattern by three pixels in either direction along the high contrast edge. SSD correlation will likely find a best match displaced along the high contrast edge in accordance with the shift in pattern.

If the high contrast edge has a small slope relative to the direction of camera movement, and some small motion (e.g., jitter) perpendicular to the direction of camera movement occurs, measurable pattern shifts can also occur. Here the period of the pattern is large, but small camera motion can result in shifts along the spatial aliasing pattern that can be detected in low-texture regions.

Motion aliasing ameliorated

The problem of illusory motions resulting from shifting spatial aliasing has not been solved. It is a particularly difficult problem in that once spatial aliasing is introduced into the image, there is no way to distinguish it from real data. An approach that greatly reduces the problem involves detecting high contrast linear features and suppressing SSD estimates in those regions. High contrast edges are detected using the zero-crossings of the image convolved with a Laplacian where $\sigma = 1$. Linearity is detected when the majority of pixels in a seven pixel line are zero-crossing pixels. SSD estimates are suppressed in regions showing linear features by zeroing their contribution to the mode filtering step.

4.4.2 Stationary bias

A problem that has occurred in practice is a tendency to measure stationarity when the camera is actually rotating. Two common, but unmodeled factors cause this phenomenon: glare and dirt on the camera lens. This illusion is easier to deal with than motion aliasing in that the resulting bias is for a single displacement, i.e., stationarity.

Presumably such lens effects ought to result in some fairly constant bias in the SSD scores. By rotating the camera in front of a low texture, stationary scene, it is possible to determine both the actual pixel displacement, and pixels that are detecting illusory stationarity. At pixels that mistakenly choose the stationary motion, one can compare the stationary SSD score with the SSD score of the actual displacement. Empirically, the difference in scores hovers around 1. Thus, simply introducing a weak anti-stationarity bias by incrementing the SSD score for the stationary displacement by 1 tends to alleviate this effect.

4.5 Related work

All optical flow algorithms share the goal of determining the motion of scene elements. There have been several comparisons and critiques of these optical flow algorithms that cover many of the important points of similarity and difference between the

various approaches to determining scene motion [14, 59, 85]. This section mentions many approaches to computing optical flow, but emphasizes their suitability for the tracking system.

The issues that determine an algorithm's suitability for optical flow-based segmentation and tracking were mentioned at the beginning of the chapter:

- Must produce a dense flow field.
- Must handle general indoor and outdoor scenes.
- Must handle multi-pixel displacements.
- Must handle spatially non-uniform motions.
- Must handle rapid changes in motion direction.
- Flow fields must agree with motion fields at discontinuities.
- Flow field must be homogeneous where motion field is homogeneous.
- Must be computationally cheap.

Most optical flow algorithms, including the one described in this chapter have two phases: a motion measurement phase and a smoothing phase. The discussion of approaches shares this bipartite structure with the flow algorithms. A discussion of techniques for making initial motion estimates is followed by an examination of various methods for applying smoothness constraints to flow fields. Pyramid schemes for computing optical flow are discussed following the analysis of motion measurement and smoothing algorithms.

4.5.1 Motion estimation

The motion estimation phases of optical flow algorithms fall into three classes: local matching, gradient-based techniques and techniques using spatiotemporal filters. Local matching techniques tend to produce individual displacement vectors at each

point, or sets of displacement vectors combined with evaluation scores. Gradient-based techniques usually produce estimates of the component of the local velocity that lies parallel to the local intensity gradient. Spatiotemporal filter algorithms produce distributions of velocities or component velocities.

Local matching

Perhaps the earliest techniques used for measurement of image motion involve local matching. Local matching approaches find the displacement that produces the best fit according to some similarity measure between one image and the next. Leese *et al.* [56] and Smith and Phillips [86] used cross-correlation of large image patches on satellite imagery for detecting cloud motion.

A theme that has been recurrent in discussions of matching is that "it is computationally impractical to estimate matches for a large number of points" [54, P. 229]. To avoid matching at many points in the image, Moravec [66] and Barnard and Thompson [13] used an interest operator to pick out distinctive points as good candidates for matching.

Several authors [7, 24, 66] have proposed using a coarse to fine, or pyramid structure to reduce the cost of matching approaches. Such methods are described below in Section 4.5.3.

Nishihara [71] uses the sign of the Laplacian of images as the basis for matching. For pairs of boolean images, SSD correlation can be performed by counting pixels in the correlation window that are mapped to one by exclusive-or. A large correlation window is used to compensate for the low discriminability of boolean images.

For noisy images, or imagery in which there is little spatial and temporal smoothness, local matching is probably the motion estimation technique of choice.

Gradient-based motion

Gradient-based optical flow algorithms [33, 39, 47] tend to assume a smooth spatial intensity surface and that overall image intensities remain constant across time. The spatial and temporal derivatives of images are used to estimate velocity in the direction of the spatial intensity gradient. First-order derivatives of a single pixel

only determine a component velocity, or constraint line on possible motion vectors. The techniques used to combine individual constraint lines are discussed as forms of smoothing.

Nagel [69] makes use of constraints on the second derivatives of image intensity to determine unique velocity vectors at corner-like points in the image. Hildreth [43] measures motion normal at zero crossing edges of the Laplacian of the Gaussian.

The difficulties of determining image derivatives from sampled data mean that input data must be smoothed over a fairly large region. The initial step of most gradient-based algorithms involves smoothing the input images with a Gaussian filter to minimize the effects of noise. When performing numerical differentiation large regions of support in space and time are used since differentiation amplifies noise and is susceptible to local effects. This blurring of image data from regions of distinct velocities tends to produce very poor estimates of normal displacement in the vicinity of discontinuities in the spatiotemporal motion field.

Kearney *et al.* [54] analyze gradient-based flow methods and cite three classes of difficulty for such methods, homogeneous regions in which image gradient is poorly defined, regions of high texture, and regions with motion discontinuities where the temporal gradient is hard to measure. Little and Verri [59] emphasize the errors that arise from the necessary initial smoothing step.

From the point of view of the tracking and segmentation algorithms, weaknesses of gradient-based motion estimates include poor behavior at motion discontinuities and in highly textured regions. The problems at motion discontinuities would tend to be exacerbated by the rapid changes in camera and object motion. Gradient-based approaches often produce poor results for displacements of more than one or two pixels.

Spatiotemporal filter-based motion

The use of spatiotemporal, velocity-tuned linear filters originated from studies of human motion perception [1, 42]. The basic intuition behind these approaches is that image motion can be determined from orientation in space-time. A standard example involves an object translating over time. In a 3-D, space-time volume, the

object appears as a ramp. The slope of the ramp with respect to time is tied to the velocity of the object, while the direction of the slope indicates the direction of motion.

Heeger [42] uses 12 distinct spatiotemporal filters tuned to different spatial orientations. The output of his initial motion estimation step is a collection of energies associated with the 12 filters.

Fleet and Jepson [34] use phase information in the output of a collection of spatiotemporal filters to compute component velocities. Component velocities are computed based on the gradient at surfaces of constant phase.

Barron *et al.* [14] show very promising results from Fleet and Jepson's phase-based algorithm. However, the motion sequences used appear to demonstrate great temporal smoothness in the motion field. The use of spatiotemporal filters requires significant spatiotemporal support. Many of the image sequences that the tracking system deals with involve rapid shifts in motion direction that would tend to corrupt the outputs of temporal filters.

4.5.2 Motion smoothing

Motion smoothing is motivated by the fact that individual initial motion measurements may be poor, or even completely ambiguous in the case of component velocities. Motion smoothing is justified by the observation that the motion field tends to be locally smooth except at regions corresponding to disparately moving objects.

Motion smoothing algorithms attempt to find a consistent interpretation to initial motion measurements, either locally or globally. When the motion field is discontinuous, motion smoothing algorithms tend to attempt to find consistency among measurements that represent distinct motions.

Local smoothing

Lucas and Kanade [61] and more recently many others [34, 84] make use of the fact that gathering together many component velocities in a local area overdetermines the local displacement. A least squares fit of the local constraints is used to determine

the best local displacement.

Heeger [42] smoothes the outputs of his spatiotemporal filters with a Gaussian filter and then applies a least squares technique to determine a best local displacement.

In regions of images with uniform gradient, collections of local component velocities may still not determine a unique displacement. At discontinuities in motion fields, least squares techniques have an effect similar to averaging the distinct velocities. Thus, spurious displacements are introduced to smooth the gap between regions of distinct motion.

Another approach to local motion smoothing is voting. The idea of voting is that all of the pixels in the local area cast votes expressing their opinions about possible displacements. The mode filtering technique described in this chapter can be thought of as unweighted local voting. If the initial motion estimation stage produces multiple displacements in cases of ties, the voting procedure is a single vote one, otherwise it is a multiple vote scheme.

Little *et al.* [20, 58] describe an optical flow algorithm based on weighted, multiple vote, local voting. In their approach each pixel determines an estimate of how likely each displacement is. The pixel then votes this entire vector of preferences in the form of multiple, weighted votes. The filtered estimate at each pixel is the displacement with the highest sum of weighted votes. Although their weighted voting scheme allows any match evaluation function, squared differences of intensity is the principal evaluation function used in their experiments.

When squared differences are used as the local dissimilarity estimation function, the scheme of Little *et al.* looks very much like the initial motion estimation phase of the algorithm described in this chapter. Their algorithm determines the displacement with the least sum of squared differences at each pixel. As discussed in Section 4.1, using a large correlation window has several drawbacks. Near motion discontinuities, and in the case of rotation, using a large correlation window can preclude finding the correct displacement. Mode filtering initial measurements based on a small correlation window gathers support from a larger region, but avoids some of the difficulties of a large correlation window.

Global smoothing

Global smoothing attempts to find a consistent interpretation for all motion estimates made across the image. The principle argument in favor of global smoothing is that it allows motions to be computed in regions that contain little or no local evidence about motion. Arguments against global smoothing include the arbitrary propagation of the effects of discontinuities in the motion field, and the significant communication required to propagate global constraints in massively parallel implementation.

Barnard and Thompson [13] perform iterative optimization on an initial network of likelihoods for potential matches between initially detected high interest points. The optimization attempts to find a maximally consistent interpretation of matches between two images.

Horn and Schunk [47] and others [43, 69] combine a constraint on the smoothness of the flow field with the constraints from the spatial and temporal derivatives. These constraints are formulated as a functional that can be iteratively optimized to find a flow field that is maximally consistent with both local component velocities and the global smoothness constraint. Anandan [7, 6] applies a global smoothness constraint to initial measurements determined by SSD correlation.

Global smoothing can be quite expensive, since it involves many iterations consisting of communication followed by local arithmetic. Horn and Schunk's algorithm, for example, requires approximately 20 arithmetic operations and 8 hypercube communications per pixel on each iteration. Results from the algorithm are often shown after 50 or more iterations.

4.5.3 Pyramid schemes

Many researchers have proposed using a multiscale representation to get around the problem of displacements that are too large for a gradient-based flow algorithm, or too expensive for a correlation-based algorithm. A hierarchy of band-pass or low-pass filtered, or even intensity averaged images is formed. Since lower frequency information can be sampled less often, resolution is reduced at each level.

Moravec [66],and Lucas and Kanade [61] attempt to determine one displacement in

a coarse-to-fine procedure. A displacement is determined at the lowest frequency level, or smallest image using correlation (in the case of Moravec) or a gradient computation (in the case of Lucas and Kanade). The resulting displacement picks out a small area on the next higher frequency level in which to determine a more precise displacement. The process is repeated until the highest resolution level is reached.

If one assumes that there is one predominant displacement that is to be detected at a point, this coarse to fine processing makes sense both from the point of view of the accuracy of the results and from the standpoint of computational complexity. Attempting to measure large displacements of small scale features can produce very bad estimates due to aliasing. Coarse to fine analysis can avoid these effects by measuring small motions at each scale. The savings in computational complexity are clearest in the case of local matching. If the search window for local search is $k \times k$, then evaluations must be made at k^2 points. If a coarse to fine strategy is followed, 4 or 9 points can be evaluated at $\lg k$ levels, reducing the computation from k^2 to $\lg k$.

Burt *et al.* [24] and Heeger [42] generalize the approach to full optical flow computation but do not use a coarse to fine processing order. Instead, the output of their algorithms is a set of optical flow fields, one for each level of the multiscale image structure. Burt justifies this approach with the argument that "velocity estimates for rapidly moving objects need not be as accurate as estimates for slowly moving objects" [24, P. 246].

Glazer [39] and Anandan [7, 6] use a coarse to fine approach for processing a hierarchical multiscale image structure to generate full optical flow fields. At the coarsest level, a displacement in a 3 by 3 region is determined for each pixel. At each subsequent level of the hierarchy, for each pixel the estimate made for the pixel's parent at the coarser level is used to select a local matching region.

These approaches suffer from poor localization of motion discontinuities, and minimal computational savings due to lack of uniform processing. The difficulties will be described in the context of Anandan's framework.

The problem of poor localization can be seen by considering a pixel at the coarsest level that is centered on (or even near) a motion discontinuity at the finest level. The initial motion estimation step will determine two velocities in the neighborhood.

Anandan's algorithm uses global smoothing at each level, thus the result at the top level will be a displacement somewhere between the two actual displacements in the vicinity. At the next lower level, the search window for some pixels may not include the correct displacement. Anandan's algorithm uses an overlapped pyramid projection scheme [22] to attenuate the effects of motion discontinuities. However, this scheme in which each pixel at a lower level can receive four displacements from above, does not solve the problem. It can still be the case that the correct displacement is not one of the four passed down from above. The pervasiveness of the problem can be seen by considering that the 5×5 SSD correlation window and the global smoothing step at the coarsest level will tend to distort motion estimates at motion discontinuities. These same distortions can be magnified and augmented at each descending level of the processing hierarchy.

The computational savings of pyramid approaches are minimized by the overlapped pyramid projection scheme and the non-uniformity of the computation. Since the search windows at a level are generated by local warping, the dynamic programming techniques described in Section 2.7 cannot be used for the SSD correlation step and the full cost of the 5×5 correlation window must be born.

The intuition that motion should be measured across scale is well motivated; however, it is still unclear how to combine measures made at different scales to provide accurate information in scenes containing non-uniform motion.

4.6 Conclusion

The optical flow algorithm described in this chapter performs unweighted SSD correlation matching, mode filtering and rectification for every pixel on the image. Thus it produces a dense optical flow field. Figures 4.10– 4.12 show the various stages of the optical flow algorithm. Figure 4.10 shows two successive images taken of a swinging ceramic mug. The mug is swinging to the left. Figure 4.11 shows needle diagrams of the output of three stages of the flow algorithm. The needle diagram in the upper left shows the output of the SSD correlation phase of the algorithm. Considering the stationary background, it can be seen that many pixels have erroneous motion

Figure 4.10: Two images of a swinging mug

estimates. The needle diagram in the upper right shows the result of mode filtering the initial estimates. The lower needle diagram shows the result of rectifying the mode filtered motion estimates. Solid 'x's indicate pixels that have been mapped to $\perp$. Figure 4.12 shows the effects of rectification. The left hand side of the figure shows discontinuities in the flow field generated by mode filtering. Notice that the white line on the left side of the mug is quite far from the mug itself. The right side of the figure shows discontinuities in the rectified flow field. Notice that a region of pixels on the left side of the mug have now been labeled with $\perp$. This region on the left side of the mug is part of a region that is occluded in the next frame.

4.6.1 Discussion

Performing initial motion estimation with SSD correlation does not depend on a smooth spatial intensity gradient, so is suitable for highly textured natural objects and outdoor environments. Local correlation does not rely on local derivatives. Not depending on local derivatives, but instead actually matching local patches makes it possible to reliably measure discretized, multi-pixel displacements.

Not having to compute local derivatives obviates the need to perform temporal

Initial flow field

Mode filtered flow field

Rectified flow field

Figure 4.11: Three stages of optical flow computation

Discontinuities in filtered result Discontinuities in rectified result

Figure 4.12: The effects of rectification

convolutions that would degrade in the presence of rapid changes in motion direction. Similarly, no initial spatial smoothing needs to be done, meaning that information is not initially smeared across motion discontinuities.

Mode filtering is well suited for multiple non-uniform motions, since it does not average measures from opposite sides of motion field discontinuities. This discrete, step-edge property of mode filtering means that flow fields tend to agree with motion fields at discontinuities. Although mode filtering introduces distorting effects at corners, the effects are probably less pernicious than the effects of averaging techniques used in other algorithms.

Similarly, since mode filtering does not introduce, or attempt to measure, fractional displacements, regions of the motion field that are homogeneous tend to result in homogeneous regions in the flow field.

4.6.2 Complexity

The optical flow algorithm described in this chapter is cheap enough to be computed on images that are 128×128 at more than 15 cycles per second on four i860 processors with a clock speed of 20 megahertz. The optical flow computation for one pixel takes about 300 instructions. The complexity of computing a flow field for a pair of images is summarized below.

Computation:

Phase	Serial	Parallel
Initial estimate	$\mathcal{O}(6\|\mathbf{D}\|N^2)$	$\mathcal{O}(6\|\mathbf{D}\|)$
Mode filter	$\mathcal{O}((3\|\mathbf{D}\|+1)N^2)$	$\mathcal{O}((2\lg d_m+1)\|\mathbf{D}\|-1)$
Rectification	$\mathcal{O}(3N^2)$	—
Total	$\mathcal{O}((9\|\mathbf{D}\|+5)N^2)$	$\mathcal{O}((2\lg d_m+7)\|\mathbf{D}\|)$

Communication:

Phase	Hypercube	Non-hypercube
Initial estimate	$\mathcal{O}(\lg\|\mathbf{D}\|+3)$	—
Mode filter	$\mathcal{O}(2\lg d_m\|\mathbf{D}\|)$	—
Rectification	—	$\mathcal{O}(2)$
Total	$\mathcal{O}(\lg\|\mathbf{D}\|+\lg d_m\|\mathbf{D}\|+3)$	$\mathcal{O}(2)$

Chapter 5

Motion Segmentation

This chapter presents the algorithm for picking out the moving objects that are to be tracked. The task of splitting up an image into subregions or segments is known as *segmentation*. Without a purpose in mind, the reasons for segmenting and the way in which segmentation should be done are nebulous. To see the fundamental nature of the problem, consider the following excerpt from a 1985 image segmentation survey from *Computer Vision, Graphics, and Image Processing*:

> What should a good image segmentation be? Regions of an image segmentation should be uniform and homogeneous with respect to some characteristic such as tone or texture. Region interiors should be simple and without many small holes. Adjacent regions of a segmentation should have significantly different values with respect to the characteristic on which they are uniform. Boundaries of each segment should be simple, not ragged and must be spatially accurate [41].

Picking out independently moving objects on the basis of their motion is a more circumscribed task. The intuition behind motion segmentation is that motion measurements in a region corresponding to a moving object differ from those of the background.

Much early image segmentation work was done with histograms [72]. In histogram-based segmentation, the image intensities (or other measures such as texture) of all

pixels in the image (or in a subregion of an image) are plotted on a histogram. If one is lucky and an object of interest differs in the histogrammed modality from the background, a distinct peak corresponding to the object will appear in the histogram. The pixels corresponding to the object can then be picked out based on the fact that they exhibit one of the values that formed the selected peak. This approach of histogramming static image properties such as intensity or texture suffers from the difficulty that such properties do not necessarily distinguish the objects in the scene; i.e., there may be no partitioning of the histogram that picks out an object. In terms of survival, any predator or prey of a system that uses gray-level histogramming need only sport a range of gray-levels to avoid detection.

The approach used for picking out moving objects is based on histogramming motion vectors. The idea is that since there will be a motion vector associated with each pixel in the image, these motion vectors, like other modalities, can be histogrammed, and a moving object can be detected as a peak in the histogram. However, not all of a non-rigid object may move at once. Fields of composite motions, or *trajectories*, across several frames provide better separation between the objects and the background than do single optical flow fields. Once trajectories have been collected across several frames, the trajectories are histogrammed. The resulting histogram is partitioned into clusters of motion. Finally, the result of the segmentation phase is generated by partitioning the image into sets of pixels that exhibit trajectories that belong in the same histogram cluster.

A common, second stage of gray-level histogramming makes use of state-space search to compensate for the likelihood that no single pair of gray-level thresholds pick out an object. This search involves splitting and merging regions of clustered pixels to achieve better segmentations based on spatial proximity [16, 51]. These splitting and merging techniques appear unsuitable for general purpose, real-time systems for two reasons. First, regions must be split and merged based on arbitrary criteria comparing global similarity and coherence. The arbitrary criteria used for deciding how to merge and split make the approach unsuitable for unstructured environments. Second, state space search among segmentations appears prohibitively expensive in that it involves search through a large space of segmentations, with global evaluation

steps at each branch point.

This motion histogramming technique relies on the restriction of camera motion to pivoting about the center of projection (i.e.rotations about the x and y axes; see Section 2.2). This restriction on motion ensures that objects which are not moving independently will have very similar camera-relative motions, and hence form a single cluster in the histogram.

The chapter begins with a description of how trajectories are formed and histogrammed. Methods for partitioning the resulting histogram are presented. Next, the definition of the result of segmentation is given. Finally, limitations and related work are discussed.

5.1 Forming trajectories

The trajectory of a pixel represents the motion of the pixel across the previous n_s frames. Trajectories are formed by composing the optical flow fields across the n_s frames. Optical flow fields with non-uniform motion are in general not *surjective* and hence, the resulting field of trajectories may not assign a trajectory to each pixel. However, extrapolated trajectories can be computed at gaps in the trajectory field, based on the most recent motions that the pixel has undergone.

Recall from Section 4.3 that the optical flow field M_i is used as the basis for producing an injective inverse to the optical flow field, $\overline{M}_i^{-1}$. Using this definition of $\overline{M}_i^{-1}$ the notion of the k-th predecessor, $P_i(p,k)$, can be defined recursively:

$$\begin{aligned} P_i(p,0) &\equiv p \\ P_i(p,1) &\equiv \overline{M}_{i-1}^{-1}(p) \\ P_i(p,k) &\equiv P_{i-1}(\overline{M}_{i-1}^{-1}(p), k-1) \end{aligned}$$

A field of trajectories, $T_i(p,k)$, over k frames for pixels p ending at frame i can be defined as the vector field difference,

$$T_i(p,k) \equiv p - P_i(p,k)$$

where '$-$' takes $\perp$ to $\perp$.

Because the optical flow fields are non-uniform, the trajectories for many pixels may be $\perp$. When the image is partitioned, each pixel is assigned to a partition based on its trajectory. Pixels mapped to $\perp$ will not be classifiable. However, trajectories can be extrapolated based on motions that are known for some tail sequence of the k frames.

$$T_i'(p,k) \equiv \begin{cases} \perp & \text{if for all } j, 1 \le j \le k, P_i(p,j) = \perp \\ \frac{k}{j} T_i(p,j) & \text{for the largest } j, 1 \le j \le k, P_i(p,j) \neq \perp \end{cases}$$

5.2 Histogramming trajectories

Forming the histogram, $H_i(v, n_s)$, of the computed trajectories is straightforward. For each possible trajectory, v (x,y displacement pair), count the number of pixels that are mapped to the trajectory.

$$H_i(v, n_s) \equiv |\{\, p \mid T_i(p, n_s) = v \,\}|$$

It is essential that the histogramming step not include the extrapolated trajectories, as they can skew the distribution of the histogram. Any pixel that is mapped to $\perp$ by T will be mapped to $\perp$ or some multiple of $\frac{n_s}{j}$ for an integer $j < n_s$ by T'. These multiples are not uniformly distributed over the range of trajectories.

5.3 Partitioning the histogram

The trajectory histogram can be viewed as a topography. The idea of partitioning the histogram is thus to split the topography into independent peaks. The partitioning step produces a map $Q_i : \mathbf{T} \longrightarrow 2^{\mathbf{P}}$ where $\mathbf{T}$ is the set of possible trajectory vectors.

Partitioning a one-dimensional histogram is intuitively simple. One merely needs to look for minima in the histogram and split the histogram at these points. Since the trajectory measurements are subject to noise, and since the form of the peaks of the histogram depends on the distribution of motion in the underlying objects, convolving the histogram with a Gaussian operator alleviates problems with anomalous local minima. If the peaks in a histogram are viewed as representing the noisy

measurements of unique actual velocities, then pixels with velocities near a minimum between two peaks are probably a mix from both underlying actual velocities. To avoid velocities with such mixed distributions, one can partition peaks at inflection points rather than minima by convolving with a Laplacian operator. Both of these approaches work quite well on histograms of the x-component of trajectories.

Finding peaks in a two-dimensional histogram is slightly more complicated. An algorithm for directly finding such peaks can proceed by growing local maxima outward. By sorting the histogram entries by height and judicious use of doubly linked lists, peaks can be found in time $\mathcal{O}(|\mathbf{T}| \lg |\mathbf{T}|)$ where $\mathbf{T}$ is the set of trajectories. As in the case of the one-dimensional histograms, the two-dimensional histogram can be convolved with a two-dimensional Gaussian operator to minimize problems with anomalous local minima.

Experiments with using a two-dimensional Laplacian operator for partitioning the histogram provided fairly poor results. Using the decomposition of the Laplacian into the sum of two separable convolutions, following the approximation suggested by Huertas and Medioni [49], peaks ought to be outlined by a connected zero-crossing edge at the inflection point. However, the close proximity of neighboring peaks makes the detection of zero-crossings unreliable and results in inferior segmentations.

5.4 Partitioning the image

Once the trajectories have been partitioned into distinct clusters, a segmentation $S_i(p)$ can be formed, assigning each pixel p to the cluster to which its trajectory belongs.

$$S_i(p) \equiv Q_i(T'_t(p, n_s))$$

Note that here the extrapolated trajectories are used for classification, so that as many pixels as possible can be classified.

Figures 5.1, 5.2, and 5.3 show the various stages of the segmentation process. Figure 5.1 shows the first and last image of n_s images used for generating trajectories. Note that the person has moved slightly to the right in the image, while the

Figure 5.1: First and last images used for segmentation

Figure 5.2: Trajectory histogram before and after smoothing

background has shifted significantly to the left. Figure 5.2 shows the histogram of trajectories both before and after smoothing. The small peak on the right corresponds to the person, while the large peak on the left corresponds to the background. Figure 5.3 shows the outline of the resulting segmented object.

5.5 Limitations

This motion segmentation algorithm has difficulty picking out objects in which the predominant motion is not a translation. For example, a disk spinning about the z-axis would show up as a cloud in the trajectory histogram.

Figure 5.3: Outline of resulting segmentation

No connectivity is required or enforced in the output of segmentation. Two objects that display the same motion will be lumped in the same cluster. This policy is justified by the inherent non-locality of connected component computation (see Section 6.2). A reasonable approach to distinguishing distinct, but similarly moving objects might follow the work of Mahoney [62], using local maxima of neighborhood homogeneity as centers of distinct objects.

One might view using the motions across n_s frames as similar to the conceptually simpler idea of computing optical flow between the first and last frame. Each trajectory represents where a pixel has moved from the first to the last frame. However, computing optical flow between more widely spaced frames has two severe drawbacks. First, as the time interval between frames increases the likelihood of encountering large changes in scene appearance that would prevent good motion measurements also increases. Second, as the maximum possible displacement increases, the area of the local search increases quadratically. This increase in search area results in corresponding quadratic increases in both the amount of ambiguity in motion estimates, and the cost of computing the flow field.

5.6 Complexity

Forming trajectories takes two operations per pixel per frame. In parallel, forming trajectories also involves one non-local communication per pixel per frame. Histogramming the trajectories takes two operations per pixel, and involves global communication in parallel. Since the trajectory histogram is not associated with any particular pixel and since it is much smaller than the number of processors on a SIMD machine, histogram partitioning is best done on a serial, front-end computer. Smoothing the histogram by means of two one-dimensional convolutions with a Gaussian filter with a support width of d_g takes $\mathcal{O}(3d_g|\mathbf{T}|)$ operations. Partitioning the histogram takes time $\mathcal{O}(|\mathbf{T}| \lg |\mathbf{T}|)$.

In summary, segmentation over n_s frames takes:

Computation:

Phase	Serial	Parallel
Segmentation	$\mathcal{O}((2n_s+2)N^2 + (3d_g + \lg\|\mathbf{T}\|)\|\mathbf{T}\|)$	$2n_s + 2 + (3d_g + \lg\|\mathbf{T}\|)\|\mathbf{T}\|$

Communication:

Phase	Non-hypercube communication
Segmentation	$\mathcal{O}(n_s + 1)$

5.7 Related work

All motion segmentation techniques share the basic intuition that a moving object in a scene may be picked out on the basis of its distinct motion. Early attempts at motion segmentation suffered from the misapprehension that computing a dense motion field is an inconceivably expensive task. Another common weakness in attempts at motion segmentation stems from only considering the motions arising from one pair of images. Only two groups appear to have used motion information from more than a single pair of images [36, 83].

An early approach by Potter [76] computes motion using a rather special purpose edge template matching scheme. Images are primarily segmented based on regions

of identical motion. Nearby points are grouped when mathematical continuity in the underlying motion appears likely.

Fennema and Thompson [33] present an elegant method for using a form of Hough transform of normal flow estimates from the entire image to perform segmentation. Normal flow estimates are mapped to sets of points in direction and magnitude space consistent with them. These direction and magnitude space points are histogrammed. The predominant line in the direction and magnitude space is assumed to correspond to a moving object. Points corresponding to this image motion are declared to be one object. The process is then repeated on the portions of the image not classified by the previous pass. Thompson [90] elaborated on this technique by combining intensity information with the normal-flow-based technique. Difficulties with this normal flow approach include dependence on unfiltered normal flow components that are highly susceptible to noise, the expense of plotting sets of points in the Hough transform space, and the basic premise of only using two frames for segmentation.

Bandopadhay [12] describes a global motion estimation and segmentation algorithm that is similar to the work of Fennema and Thompson. Initial motion estimates are formed by pairing selected interest points, rather than normal flow vectors. Initial estimates are globally histogrammed to provide for feed back to the initial motion estimation step. The algorithm and its effectiveness are not well expressed. However, the dependence on interest points and the motions between pairs of frames probably limits the utility of the approach.

Jain *et al.* [50] experimented extensively with using image differences to perform segmentation of moving objects. Although image differences are extremely cheap to perform, the cases in which differencing gives reasonable results vary widely, depending on the intensity contrast of object and background, and how far the object has moved. When the camera undergoes motion, differencing becomes still more problematic.

Another approach to motion-based segmentation involves attempting to find discontinuities in the motion field, and using these detected discontinuities as the borders of distinctly moving image regions. Thompson *et al.* [91] convolve the x and y components of a motion field with the Laplacian of the Gaussian. The two results of

this convolution are combined into a vector field. Vector reversals indicate motion boundaries. This work was extended in [92], and [68] with particular emphasis on detecting occluding surfaces.

Spoerri and Ullman [87] attempt to detect motion edges by examining the structure of local histograms of neighboring motion estimates. Significant bi-modality, for example, is evidence for the presence of a motion boundary. Black and Anandan [15] perform similar examinations of the SSD surface for evidence of motion boundaries.

Detecting motion boundaries can be useful for improving motion estimates near object boundaries. However, for detecting distinctly moving objects, edges must be aggregated to form distinct regions of motion. Methods that are based on the Laplacian may produce connected edges, but will often produce spurious zero crossings scattered about the image. Methods based on examining the structure of a locally computed motion surface suffer from the problem that the surface characterizations involved are essentially heuristic and cannot be expected to produce connected motion boundaries.

Adiv [2] uses a Hough transform to break an image into regions whose motion is consistent with rigid motion of a planar patch. These initial regions are iteratively combined to form object hypotheses that are consistent with the same 3-D translation and rotation parameters. This method may be time consuming and relies heavily on the rigidity of objects in the scene.

The idea of characterizing the motion segmentation problem as a regularization problem has been explored by several researchers. Murray and Buxton [67] formulate a criterion for a "best segmentation" based on the model that the scene is made up of a fixed number of moving planar surface patches. A search for an interpretation of the scene that is consistent with measured normal flow vectors is made using simulated annealing. François and Bouthemy [36] follow a somewhat similar tack, but allow the use of the output of one segmentation as the seed for the segmentation of the subsequent pair of images. The principal shortcomings of these regularization-based approaches are the prespecification of the number of objects expected, and the cost of optimization techniques.

Thompson and Pong [93] discuss various techniques for detecting moving objects in

cases of restricted camera motion. They present a method for determining whether an object that is being tracked[1] is actually moving. The test involves histogramming the directions of optical flow vectors in a scene. If the histogram is bi-modal, the tracked object is moving. Another set of techniques involve knowing some camera motion parameter so that motions inconsistent with the known motion can be detected. Nelson [70] proposes a similar technique for detecting objects when the camera is known to be translating forward.

Burt *et al.* [21] propose a motion segmentation algorithm based on successively factoring out distinct components of motion and recomputing motion in regions inconsistent with previously discovered components. Peleg and Rom [75] propose a similar method that successively factors out 3-D motions. Both these approaches make use of an hierarchical motion computation so that they can handle great disparities in velocity. However, both require fairly rigid objects in order for the grouping of consistent motions. The 3-D approach further suffers from the extreme environmental constraints that current techniques for measuring 3-D motion require (e.g., the scene must have uniform depth).

Shio and Sklansky [83] present a system for picking out distinct moving people in a stationary camera scene. The segmentation is made based on a sequence of images rather than a single pair. Their method, however, relies on an *ad hoc* motion computation and camera stationarity.

5.8 Discussion

The motion segmentation algorithm described works well, as evidenced by the success of the two implemented camera-panning systems. Often spurious pixels are included in resulting segmentations. The adjustment phase of the tracking algorithm (Section 6.2), eliminates these pixels as they tend not to move with the segmented object.

If more than one object is moving with the same trajectory, all the pixels corresponding to the like-moving objects will form one cluster in the trajectory histogram. Since there is no global connectivity step in the segmentation algorithm, all of these

[1] "Tracked" in the sense of camera pursuit (See Section 1.3)

like-moving pixels will be grouped in one object map. Similarly, if only part of an object moves during the interval used for computing trajectories, only the part that moved will be selected in an object map.

Since images are taken either from a stationary camera or a camera that is undergoing rotation about the center of projection on the x and y axes, the effects of parallax are minimal. For more general camera motions, parallax would cause the trajectories corresponding to background motion to be less uniform. In extreme cases, large translations in a cluttered environment would cause a less homogeneous set of trajectories and might interfere with the clustering step.

Chapter 6

Motion tracking

This chapter describes the motion-based tracking algorithm. The algorithm is intended to maintain the approximate image extent of a tracked object from frame to frame. There are two cross-temporal goals to this tracking problem: maintaining correspondence and maintaining coherence. Maintaining correspondence involves determining where each part of the object that is being tracked has gone to. Maintaining coherence of the representation of the object being tracked requires dealing both with parts of the object that appear and disappear, and with errors in maintaining correspondence.

Formally, for each cycle, the tracking algorithm takes two rectified (i.e., injective) optical flow fields $\overline{M}_i$ and $\overline{M}_{i+1}$, and a boolean map, O_i, representing the extent of the object in image I_i as input. As output, it produces a new boolean map O_{i+1} representing the position of the object in image I_{i+1}.

For simplicity of explanation, only the situation in which one object is being tracked is described. Tracking multiple objects can be handled by running the same algorithm with distinct data structures for each object that is being tracked.

The primary intuition behind the tracking algorithm is focussed on maintaining correspondence. If some set of pixels corresponds to the position of an object in one frame, then mapping that set of pixels through the optical flow field ought to produce a new set of pixels corresponding to the new position of the object. Thus, by projecting the tracked object through successive flow fields, one ought to be able

to keep track of the position of an object across many images. Errors resulting from discretization, noise, occlusion and inclusion, however, tend to force the set of tracked pixels to diverge from those corresponding to the projected object.

The second intuition behind the tracking algorithm is focussed on maintaining coherence of the representation of the tracked object. Over time, the outline of a moving object tends to correspond to motion boundaries in the flow field. The existence of this correspondence between the outline of a moving object and the motion boundaries can be used to compensate for degradation in the tracked region resulting from projection or initial inaccuracy. In the ideal case of a rigid object translating against a stationary background, object boundaries precisely coincide with edges in the optical flow field. With non-rigid objects, and a shifting background, the claim, although more qualitative, remains true in most cases.

Consistent with these two ideas, the tracking algorithm has two steps for generating the best approximation to the image relative position of $\tilde{O}_{i+1}$: projecting the tracked object through the flow field $\overline{M}_i$, and adjusting the extent of the tracked region to agree with motion boundaries. Projection is described first. Next, the motivation for adjustment, and algorithms that perform it are presented. The chapter concludes with a discussion of related object tracking systems.

The tracking algorithm produces several intermediate steps in the generation of O_{i+1} from O_i. The various stages of intermediate object representation are denoted O_i, O^1_{i+1}, O^2_{i+1}, and O_{i+1}.

6.1 Projection

The projection step in which the tracked object is passed through the flow field has two phases. The first is the obvious application of the injective optical flow field to the boolean tracked image. The second step is a clean-up operation intended to resolve gaps introduced in the projected object as a consequence of the injective flow field not being surjective. The output of the first phase is denoted O^1_{i+1}. The output of the second phase, denoted O^2_{i+1}, is O^1_{i+1} with gaps filled. This projected and filled representation is the output of the projection step.

Recall that $\overline{M}_i$ is the injective optical flow field, constructed in Section 4.3, that maps pixels in image I_i to pixels in image I_{i+1}, and that $\overline{M}_i^{-1}$ is the inverse of $\overline{M}_i$. If $O_i(p)$ is true, that is if part of the object is assumed to be mapped by the projection relation to p, and if $\overline{M}_i(p) = p'$, then $O^1_{i+1}(p')$ is true as well. Thus, the first boolean result O^1_{i+1} is simply O_i mapped through $\overline{M}_i$ wherever $\overline{M}_i$ is defined.

$$O^1_{i+1}(p) \quad \equiv \quad \begin{cases} O_i(\overline{M}_i^{-1}(p)) & \text{if } \overline{M}_i^{-1}(p) \neq \perp \\ \perp & \text{otherwise} \end{cases}$$

This intermediate result maps pixels to $\perp$ that are not in the range of $\overline{M}_i$; i.e., if $\overline{M}_i$ maps no pixel to pixel p, then $O^1_{i+1}(p) = \perp$. The result of the second phase, O^2_{i+1} only differs from O^1_{i+1} at those points where $O^1_{i+1} = \perp$.

A pixel p that is mapped to $\perp$ by O^1_{i+1} is mapped to 1 by O^2_{i+1} if a majority of pixels in a local neighborhood of diameter d_p surrounding p are mapped to 1 by O^1_{i+1}, otherwise, pixel p is mapped to 0.

$$O^2_{i+1}(p) \quad \equiv \quad \begin{cases} O^1_{i+1}(p) & \text{if } O^1_{i+1}(p) \neq \perp \\ 1 & \text{if } \dfrac{{d_p}^2}{2} < \displaystyle\sum_{[\frac{-d_p}{2} < j \leq \frac{d_p}{2}]} \sum_{[\frac{-d_p}{2} < k \leq \frac{d_p}{2}]} \downarrow O^1_{i+1}(p + \langle j, k \rangle) \\ 0 & \text{otherwise,} \end{cases}$$

The function $\downarrow$ maps $\perp$ to 0 and is the identity function otherwise. In practice, d_p is seven or eight.

Since a region that is mapped to $\perp$ corresponds to a place where no pixel is estimated to have gone, the region is likely to represent an area of non-uniform motion. If such a region occurs inside an object, it may result from non-rigid motion, and most of the local neighborhood will be part of O^1_{i+1}. If such a region occurs at the edge of an object, it may result from pixels being uncovered, and at least half of the local neighborhood is probably not in O^1_{i+1}. This winner-takes-all scheme has the effect of filling holes in the tracked object introduced by non-rigid motion and disregarding uncovered pixels.

The first phase of projection requires two address computation operations in serial, and a non-hypercube communication in parallel. The winner-takes-all second phase can be handled using the dynamic programming techniques from Section 2.7 for

computing local sums. Thus, in serial, the second phase takes four operations per pixel, while it takes $2 \lg d_p$ operations and hypercube communications in parallel. The cost of projecting is summarized below.

	Computation		Communication	
Phase	Serial	Parallel	Hypercube	Non-hypercube
Projection	$\mathcal{O}(6N^2)$	$\mathcal{O}(2 \lg d_p)$	$\mathcal{O}(2 \lg d_p)$	$\mathcal{O}(1)$

6.2 Adjustment

The optical flow algorithm described in Chapter 4 produces a discrete approximation to the continuous motion field. Discretization introduces some inaccuracy into the flow field. Noise, local effects near discontinuities in the motion field, and other phenomena can result in more severe inaccuracies in the flow field. Thus, over many frames, merely composing the results of the projection phase of tracking would tend to compound these inaccuracies. The compounded inaccuracies would, in turn, cause the approximation O_n to diverge from the actual projected image extent of the tracked object, $\tilde{O}_n$.

The goal of the adjustment step of the tracking algorithm is to maintain the coherence of the representation of the tracked object. If the motion of an object between times t_1 and t_2 is captured on images I_{i+1} and I_{i+2}, some set of pixels on I_{i+1} that moved cohesively ought to correspond to the projected object $\tilde{O}_{i+1}$. The adjustment phase of the tracking algorithm attempts to take advantage of this localization of the motion of an object to improve the estimate of the position of the tracked object. The adjustment phase takes as input O^2_{i+1}, the output of projection, and a rectified optical flow field $\overline{M}_{i+1}$, and produces O_{i+1}, the final estimate of the image position of $\tilde{O}_{i+1}$.

The idea of adjustment is first presented as a global algorithm based on connected components of motion. The global algorithm is shown to have shortcomings relating to its global nature. Next, a local version of adjustment is described. Several local approximations to connected components are analyzed. Finally, the computational complexity of adjustment is presented.

6.2.1 Global adjustment

Suppose one had a boolean image A that approximates the actual image location $\tilde{O}_{i+1}$ of an object and a flow field $\tilde{M}_{i+1}$ specifying the motion between image I_{i+1} and I_{i+2}. The image A would be approximate in the sense that some pixels in A are not in $\tilde{O}_{i+1}$ and some pixels in $\tilde{O}_{i+1}$ are not in A. The goal of adjustment is to produce a better estimate A' of the image location of the object $\tilde{O}_{i+1}$ using information in the motion field $\tilde{M}_{i+1}$. An estimate A' is better than another estimate A if it more closely approximates $\tilde{O}_{i+1}$, ie. if

$$|(\tilde{O}_{i+1} - A') \cup (A' - \tilde{O}_{i+1})| \leq |(\tilde{O}_{i+1} - A) \cup (A - \tilde{O}_{i+1})|.$$

Imagine that a rigid object is translating in the image plane at two pixels to the right per frame. The motion field $\tilde{M}_{i+1}$ will have the property that pixels corresponding to $\tilde{O}_{i+1}$ will be moving two pixels to the right, while the remaining pixels will move with the background. Knowing that the object is translating in this fashion would allow one to specify an improved A instantly; simply declare A' to be the set of pixels that are moving at two pixels to the right. In general, one does not have any such accurate prior knowledge of the object's motion.

If the input estimate A is sufficiently close to $\tilde{O}_{i+1}$, one can still produce a better estimate A' given the motion field $\tilde{M}_{i+1}$, without prior knowledge of the object's motion. The key observation is that the majority of pixels that are moving right two are in A and similarly, the majority of pixels moving with the background are not in A.

This observation can be generalized into an algorithm based on the consensus of connected components of motion. The image can be broken up into regions of uniform motion. If the majority of such a region is part of A, then it can be assumed that the region represents a part of the moving object and all of the region can be made part of A'. Similarly, if the majority of a region is not in A, then it can be assumed that the region does not correspond to the moving object and none of the region is part of A'.

Formally, the set of connected components of motion $\mathcal{C} = \{C_1, \ldots, C_k\}$ for a

motion field $\tilde{M}$ is a partition of the set of pixels in which each subset C_j is a four-connected set of pixels all manifesting the same displacement. As a function $\mathcal{C}(p)$ maps a pixel p to the subset $C_j \in \mathcal{C}$ that contains p. Notice that the area of the motion field corresponding to a translating, unoccluded, rigid object is approximately uniform, and hence forms a connected component of motion. In general, a moving object will correspond to one or more connected components of motion in the motion field.

An improved estimate A' is the union of connected components whose majority is in A. The majority of a component C is in A if the number of pixels in C is less than twice the number of pixels that are both in C and A. The improved estimate A' is thus composed of the set of components that are mostly moving with A.

$$A'(p) \quad \equiv \quad |\mathcal{C}_{i+1}(p)| < 2|\mathcal{C}_{i+1}(p) \cap A|$$

If the motion field $\tilde{M}_{i+1}$ and approximation A meet certain criteria, this consensus algorithm can be shown to produce a solution A' that is identical to $\tilde{O}_{i+1}$. Suppose (1) that the object is moving distinctly from the background, so that the boundaries of $\tilde{O}_{i+1}$ coincide with motion boundaries. Further, suppose (2) that the approximate guess A is good enough in the sense that when a pixel p is in $\tilde{O}_{i+1}$ and not in A, it is in a component of motion that is mostly part of A, and when p is in A and not in $\tilde{O}_{i+1}$, it is in a component of motion that is mostly not in A. Given these two hypotheses, it is easy to see that any superfluous pixel in A will not appear in A', since the majority of its component is not in A, and a pixel missing in A will appear in A', since the majority of its component is in A. A' will coincide with $\tilde{O}_{i+1}$ because the boundaries of $\tilde{O}_{i+1}$ correspond to boundaries of the connected components of motion. The left side of Figure 6.1 shows the projection O, of a stylized round object translating right against a stationary background. The flow field has two connected components, the right-going circle and the stationary background. The right side of Figure 6.1 shows the original approximation to the position of the object A. The left side of Figure 6.2 shows the approximation A overlaid with the motion edges resulting from the translating object. Clearly any pixel inside the circle lies in a connected component that is mostly in A. Similarly, any pixel outside the circle lies

Figure 6.1: Moving object $\tilde{O}$ and approximation A

Figure 6.2: Motion edges over A and the adjusted object A'

in a connected component that is mostly not in A. The right side of Figure 6.2 shows the resulting improved boolean map A'.

This global algorithm based on determining the majority of pixels in connected components of motion has two principal drawbacks: its effects can be too drastic, and computing connected components on a parallel machine necessarily requires significant amounts of communication, and is hence expensive. The algorithm can be seen to have dire effects by considering the case of an object that stops for a frame. There will be exactly one connected component of motion. If the initial approximate object A is less than half of the image, the new approximation A' will be the empty set. If A is more than half of the image, A' will constitute the whole image.

To see that computing connected components involves significant communication, consider a region of uniform motion that is spiral. The whole length of the spiral must end up in the same component, with the same label, and hence the label must be communicated from one end of the spiral to the other. Cypher *et al.* [29] have described an algorithm that computes connected components on a SIMD computer with $\mathcal{O}(N \lg N)$ communications for an $N \times N$ image.

6.2.2 Local adjustment

Some local form of adjustment is needed to sidestep the drawbacks of global adjustment. By maintaining the principle of the consensus of a homogeneous region and localizing the idea of connected components, local adjustment can be achieved. The intuition is to somehow determine for the neighborhood surrounding a pixel, a local connected component, and then to compute whether the majority of this local component is in A. Such a local scheme reduces the problem of drastic change, since it can only have effects at the perimeter of the approximation to the object's position, A. A local adjustment algorithm also has the potential of reducing computational cost by reducing the amount of communication needed.

A local notion of connected component can be defined with respect to a local neighborhood, $R(p)$, a set of pixels in a square region surrounding p of width and

height d_a.

$$R(p) \equiv \bigcup_{[\frac{-d_a}{2}<j\leq\frac{d_a}{2}]} \bigcup_{[\frac{-d_a}{2}<k\leq\frac{d_a}{2}]} p + \langle j, k\rangle$$

A local component $\mathcal{C}^l(p)$ is a subset of $R(p)$ that satisfies some definition of componenthood. Given such a sketch of a definition of a local component, the adjustment algorithm can be redefined locally.

$$A'(p) \equiv |\mathcal{C}^l(p)| < 2|A \cap \mathcal{C}^l(p)|$$

The question remains how to define $\mathcal{C}^l(p)$. Four approaches seem to be more or less plausible.

The first approach would be to compute the connected components of motion on each pixel's local neighborhood. This would be impractically expensive in serial due to duplication of effort. On a SIMD machine, the computation would also be prohibitive, as connected component computations require branching and/or address indirection.

The second approach is similar to the first, but instead of computing connected components for each pixel, the connected components for the whole image would be computed. The local pixel's subregion $R(p)$ intersected with the global connected component labelling would be taken to be the local component. This approach might be practical in serial, but in parallel, it would require computing global connectedness, and hence at least as much communication as the global solution.

A third approach involves noting that for small regions $R(p)$, the subset of pixels of $R(p)$ with the same displacement as p is a close approximation to the connected component containing p intersected with $R(p)$. Instead of requiring that pixels be connected, consider all pixels in $R(p)$ that exhibit the same motion to be the local component. This definition for local components that ignores connectivity is used in the serial implementation and is denoted $\mathcal{C}^=$. In serial, O_{i+1} is defined from O^2_{i+1} to be:

$$O_{i+1}(p) \equiv |\mathcal{C}^=(p)| < 2|O^2_{i+1} \cap \mathcal{C}^=(p)|.$$

The final and by far the most difficult to describe approach involves computing a local, radial approximation to connectedness at each pixel. This local approximation,

written $\mathcal{C}^r$ is the most effective SIMD algorithm, and hence is used for the Connection Machine implementation. In parallel, O_{i+1} is defined from O^2_{i+1} to be:

$$O_{i+1}(p) \quad \equiv \quad |\mathcal{C}^r(p)| < 2|O^2_{i+1} \cap \mathcal{C}^r(p)|.$$

6.2.3 Serial local components

The definition of a local component for a pixel p that is used in the serial implementation is the set of pixels in a local region that have the same displacement vector as p.

$$\mathcal{C}^=(p) \quad \equiv \quad \{\, p' \in R(p) \mid \overline{M}(p) = \overline{M}(p') \,\}$$

Two values $|O^2_{i+1} \cap \mathcal{C}^=(p)|$, and $|\mathcal{C}^=(p)|$ are needed to compute $O_{i+1}(p)$ for each pixel. Perhaps the most straightforward implementation of this algorithm would involve counting these two quantities for each pixel. The cost would be approximately $\mathcal{O}(2d_a^2N^2)$. In the implemented systems, d_a is 15, so the cost of this computation, $\mathcal{O}(450N^2)$ would swamp the cost of computing optical flow.

Another approach is to use the dynamic programming techniques from Section 2.7. For each pixel, the result of the dynamic programming computation is a vector of counts with $2|\mathbf{D}|$ entries. For each displacement $d \in \mathbf{D}$, there are two entries. One consists of the count of pixels in the local neighborhood surrounding p that are moving with the displacement. The other is the number of these similarly moving pixels that are also tracked. The appropriate counts for a given pixel p can be accessed by looking at the pair of entries for displacement $d = \overline{M}(p)$.

In this case, the basic operation, $\otimes$, is vector addition. The individual elements that are added and subtracted to form a new vector, are vectors with single, unit entries. Computing a new column vector sum takes only two operations for the count of similarly moving pixels, and two operations for the count of similarly moving pixels that are tracked. As noted in Appendix B, forming a new area sum involves adding and subtracting full vectors of length $2|\mathbf{D}|$ The total cost is then $\mathcal{O}((4 + 4|\mathbf{D}|)N^2)$. In practice, $|\mathbf{D}|$ is 37, so the cost is $\mathcal{O}(152N^2)$.

A single column of d_a pixels can have at most d_a pixels with distinct displacements. In fact, since the optical flow field tends to be smooth, each column will generally have

far fewer than d_a distinct motions represented. These observations argue that a sparse vector representation should be used to represent the various counts in question. The single entry additions and subtractions that produce column sums can no longer be computed in unit time, since the appropriate entries must be found by searching. Let the number of distinct motions in a column be denoted n_a. In the worst case, $n_a = d_a$. In practice an average value for n_a is less than 3. Finding the two necessary entries can be done in $2n_a$ operations, while the individual additions and subtractions have unit cost. The cost of producing a new column vector sum is $(2n_a + 4)N^2$. The area sum requires adding $2n_a$ entries and subtracting $2n_a$ entries from the previous area sum. The total cost for each pixel is $(6n_a + 4)N^2$. In the worst case n_a would be d_a, so the algorithm has complexity $\mathcal{O}(6d_a + 4N^2)$. The worst case cost is approximately 94 operations per pixel, for $d_a = 15$, and the empirical average case cost, for $d_a = 3$, is approximately 22 operations per pixel.

The SIMD cost of computing $\mathcal{C}^=$ is quite high. To facilitate comparison with the actual computation used for adjustment in parallel, the SIMD cost is expressed in terms of the number of hypercube communications, the number of bits communicated, and the number of bits processed.

Recall that $\mathcal{C}^=$ is just the set of pixels in the local neighborhood that have the same displacement vector. Probably the most efficient algorithm for computing $\mathcal{C}^=$ involves making a local copy of all the neighboring displacement vectors and identifying those that have the same displacement as the center pixel. The creation of the local copy can be done using the local dynamic programming techniques of Section 2.7, with concatenation as the operator $\otimes$. For a local neighborhood of diameter d_a, the local copy requires $\mathcal{O}(2 \lg d_a)$ communications. Each displacement vector requires $\lg |\mathbf{D}|$ bits to represent, so $\mathcal{O}(1 + \lg |\mathbf{D}| d_a^2)$ bits must be transmitted per processor. Finally, identifying the neighboring pixels with the same displacement requires $d_a^2 - 1$ comparisons of length $\lg |\mathbf{D}|$, for a total of $\mathcal{O}((d_a^2 - 1) \lg |\mathbf{D}|)$ bit operations.

6.2.4 Parallel local components

This section describes a very efficient algorithm for computing a local approximation to connected components on each processor of a SIMD computer. The definition of

1	2	3	4	5	
a	b	a	b	a	1
a	b	a	b	a	2
a	b	b	b	a	3
a	a	a	a	a	4

Figure 6.3: Recalcitrant connected components

radially connected components $\mathcal{C}^r$ is an example of a formal notion arising from a combination of partial constraint on what is to be computed, and partial constraint on computational cost. The constraint on what is to be computed is that the result should be a local approximation to the connected component computation, with the relaxation that the component containing the central element is the only one of interest. The constraint on computational cost is that the algorithm be suitable for SIMD implementation and be cheap. Pixels will be described as being labeled with a *color*, rather than a displacement, as the algorithm has application to other problems than connectivity of flow fields.

One property of the connected components problem makes it expensive to compute. Connected components are traditionally computed in slightly more than one pass by scanning left to right, top to bottom. Whenever a pixel with a distinct color is encountered, a new label is introduced. The drawback to the algorithm is that during processing two pixels that will eventually end up with the same label may be temporarily marked with distinct labels. To avoid backtracking, an additional data-structure is used to record equivalences between the labels of such pixels, and a final pass marks each pixel with the correct label. An example which suggests that there is no way to avoid these temporary labelings is shown in Figure 6.3. The 'a's and 'b's are the colors that determine pixel equivalence. Processing top to bottom, left to right, there is no way of knowing while working on row one, that columns one and five ought to have the same label. Only when processing the bottom row does it become clear that columns one and five are in the same connected component.

If connected components were being computed locally on each processor, this final step of updating pixel labelings would require non-uniform processing on each

processor. Some processors might require no re-labeling, while other processors might require extensive re-labeling.

The observations that motivate the notion of radially connected components are that each pixel only cares about the connected component that is centered about itself, and that only degenerate regions such as spirals require more than one pass. The definition of a *radially connected component*, $\mathcal{C}^r$, combines these two observations, specifying a component that is "grown out" from the center pixel, or origin.

To precisely specify the notion of a radially connected component, a connectivity graph is defined to represent the pixel grid on which the connected component is to be computed. Given this graph, the notions of a *legal transition* on the graph, and *legal reachability in k steps* are defined. Finally *radially connected components* is defined in terms of legal reachability in $r_a = \lfloor \frac{d_a}{2} \rfloor$ steps:

$$\mathcal{C}^r(p) \quad \equiv \quad \{\, p' \mid \textit{legally-reachable}(p', p, r_a) \,\}.$$

The connectivity graph

Define the directed connectivity graph $\mathbf{G} = \langle \mathbf{E}, \mathbf{V} \rangle$. The vertices from the set $\mathbf{V}$ are labeled with triples: $\langle \textit{color}, x, y \rangle$. For each pair of integers, x, y, there is exactly one vertex with these x and y coordinates. The edges $\mathbf{E}$ represent 4-connectedness among vertices labeled with the same color. An edge (v_1, v_2) is an element of $\mathbf{E}$ if $\textit{color}(v_1) = \textit{color}(v_2)$ and the vertices are 4-connected, that is, either

$$x(v_1) = x(v_2) \wedge |y(v_1) - y(v_2)| = 1.$$

or

$$y(v_1) = y(v_2) \wedge |x(v_1) - x(v_2)| = 1$$

Figure 6.4 shows such a connectivity graph. Vertices are labeled with a's and b's to indicate their colors. The labels on the left and bottom edges denote the y and x coordinates of the vertices. The vertex at position $(0, 0)$, the center pixel, is called the *origin*.

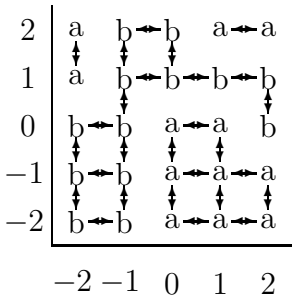


Figure 6.4: The connectivity graph for a small image

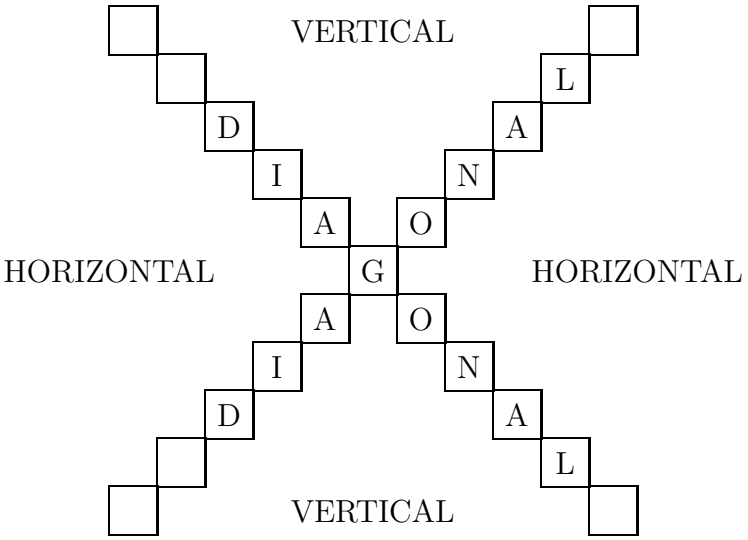


Figure 6.5: The three sets of vertices

Legal transitions

In order to define legal transitions on the connectivity graph, partition the vertices into three sets with respect to the origin: the vertical vertices in which $|x| < |y|$, the horizontal vertices in which $|x| > |y|$, and the diagonal vertices in which $|x| = |y|$. These three sets of vertices are shown in Figure 6.5.

Given these three sets of vertices, two classes of edges are defined on the connectivity graph: *radial* and *lateral* edges. The notions of radial and lateral edges are intended to make sense in terms of the origin (the vertex with 0 as its x and y labels). A radial edge goes away from the origin, while a lateral edge lies on a square centered at the origin.

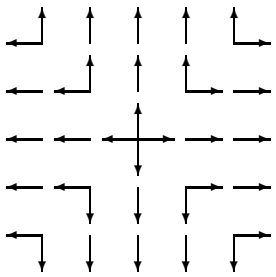


Figure 6.6: The radial edges restricted to 4-connectivity

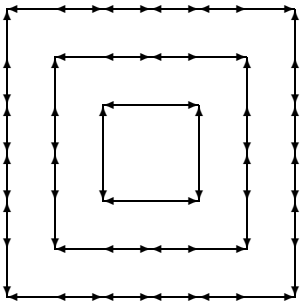


Figure 6.7: The lateral edges restricted to 4-connectivity

An edge (v_1, v_2) is a radial edge if v_1 is either a vertical vertex or a diagonal vertex and $|\mathrm{y}(v_1)| < |\mathrm{y}(v_2)|$, or if v_1 is either an horizontal vertex or a diagonal vertex and $|\mathrm{x}(v_1)| < |\mathrm{x}(v_2)|$. This definition is a very weak constraint, but in conjunction with the 4-connectivity requirement specified by the connectivity graph, only transitions that grow out radially from the origin are admitted. Figure 6.6 shows the radial edges restricted to 4-connectivity.

An edge (v_1, v_2) is a lateral edge if v_1 is a vertical vertex and $\mathrm{x}(v_1) \neq \mathrm{x}(v_2)$, or if v_1 is an horizontal vertex and $\mathrm{y}(v_1) \neq \mathrm{y}(v_2)$. This definition is similarly weak, but in conjunction with the 4-connectivity requirement restricts transitions to those edges lying on concentric squares centered about the origin. Figure 6.7 depicts the lateral moves restricted to 4-connectivity.

A legal transition involves traversing either a single radial edge from the starting vertex, or traversing a radial edge followed by a lateral edge. More formally, a

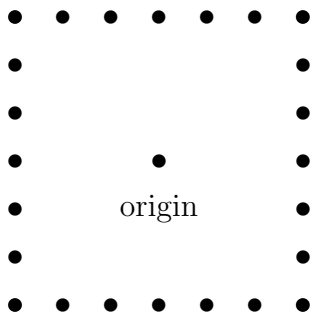


Figure 6.8: Vertices legally reachable in 3 steps

transition from one vertex v_1 in $\mathbf{V}$ to another vertex v_2 in $\mathbf{V}$ is a legal transition, if

1. $(v_1, v_2) \in \mathbf{E}$ and (v_1, v_2) is a *radial edge*, or
2. There is a third vertex $v_3 \in \mathbf{V}$ such that $(v_1, v_3) \in \mathbf{E}$, $(v_3, v_2) \in \mathbf{E}$, (v_1, v_3) is a *radial edge*, and (v_3, v_2) is a *lateral edge*.

Legal reachability and radial connectedness

A vertex v_2 is legally reachable in 1 step from vertex v_1 if (v_1, v_2) is a legal transition. A vertex v_2 is legally reachable in k steps from vertex v_1, *legally-reachable*(v_2, v_1, k), if there is another vertex v_3 such that (v_1, v_3) is a legal transition, and v_2 is legally reachable from v_3 in $k-1$ steps. Figure 6.8 shows those vertices reachable in 3 steps from the origin in an image of uniform color. A radially connected component of radius k is all those vertices legally reachable in k or fewer steps from the origin.

The key properties of legal reachability in conjunction with the 4-connectedness graph are that the sets of legally reachable vertices are disjoint for each k, and that the set of legally reachable vertices for $k+1$ steps is determined by the set of legally reachable vertices for k steps. Thus to compute a radially connected component of radius k, one can serially compute the concentric rings of legally reachable vertices for $j = 1, 2, \ldots, k$.

By mapping these notions about graphs back to images, the definition of legal reachability can be applied to pixels. In this way, for $k = r_a = \lfloor \frac{d_a}{2} \rfloor$,

$$\mathcal{C}^r(p) \quad \equiv \quad \{\, p' \mid \textit{legally-reachable}(p', p, r_a) \,\}.$$

The cost of radial components

Having defined locally connected components, it is worth analyzing their computational cost, and comparing this cost with that of the alternate idea of counting local pixels with the same displacement vector. Locally connected components can be computed using a small quantity of initial communication followed by r_a phases of outward growth at each pixel.

Each pixel determines whether it has a distinct color from its north and east neighbors. A local copy of this information is collected at each pixel in the form of two boolean maps centered at the local pixel, that demarcate region boundaries. Once these local region boundaries are known, radial connectivity with the central pixel can be determined in r_a steps. The initial connected component is represented as a boolean map including only the central pixel itself. Each step involves a radial expansion, followed by two lateral expansions, one clockwise, and one counterclockwise. These phases can be most easily explained in terms of the initial step of growing from the central pixel to the three by three square surrounding the central pixel. The radial expansion in this case consists of adding any of the four pixels to the north, east, south and west, that do not lie across color region discontinuities. The clockwise lateral phase involves adding the northeast pixel if the north pixel has been added, and if there is no region boundary between the north pixel and the northeast pixel, and similarly for each of the other three corners. In the counterclockwise lateral phase, the northeast pixel is added if the east pixel is included, and there is no region boundary between the east pixel and the northeast pixel, and so on for the other three corners. Each of the subsequent steps for radii $2, 3, \ldots, r_a$ is similar, but requires expansion of the whole perimeter of the previous step.

Determining the initial north-south and east-west discontinuities requires two local communications and two comparisons of $\lg |\mathbf{D}|$ bits each. The local copy of these discontinuities can be made using the standard dynamic programming techniques in $\mathcal{O}(2 \lg d_a)$ communications, involving $2d_a^2$ bits. For each concentric ring, steps $i = 0, 1, \ldots, r_a - 1$, a radial expansion requires considering $1 + 2i$ additional cells for each of four directions. Similarly, each lateral expansion requires considering $1 + 2i$ additional cells for each of four directions. The total cost for computing the radial

component given the local connectivity information is thus:

$$\sum_{0 \le i < r_a} 4 \cdot 3 \cdot (1 + 2i) \quad = \quad 12 r_a^2.$$

Since $r_a = \lfloor \frac{d_a}{2} \rfloor$, $12 r_a^2 \le 3 d_a^2$. The comparison of the costs for computing radially connected components in parallel versus counting neighboring pixels with the same label are presented below.

Method	Computation	Communications	Bits communicated
$\mathcal{C}^r$	$\mathcal{O}(2 \lg \lvert\mathbf{D}\rvert + 3 d_a^2)$	$\mathcal{O}(2 + 2 \lg d_a)$	$\mathcal{O}(2 \lg \lvert\mathbf{D}\rvert + 2 d_a^2)$
($d_a = 15$, $\lg \lvert\mathbf{D}\rvert = 6$)	687	10	462
$\mathcal{C}^=$	$\mathcal{O}((d_a^2 - 1) \lg \lvert\mathbf{D}\rvert)$	$\mathcal{O}(2 \lg d_a)$	$\mathcal{O}(1 + \lg \lvert\mathbf{D}\rvert d_a^2)$
($d_a = 15$, $\lg \lvert\mathbf{D}\rvert = 6$)	1344	8	1575

The final step to performing the adjustment involves counting the number of pixels included in the local region, counting the number of pixels included that are also tracked, and including or excluding the local pixel based on the result. This final computation requires approximately $2 d_a^2$ operations.

6.2.5 Discussion

Adjustment is a fairly expensive operation, as summarized in the following table.

Phase	Serial	Parallel	Hypercube
Adjustment	$\mathcal{O}((6 d_a + 4) N^2)$	$\mathcal{O}(2 \lg \lvert\mathbf{D}\rvert + 3 d_a^2)$	$\mathcal{O}(2 + 2 \lg d_a)$
(average case)	$22 N^2$	—	—

The algorithms as described support adjustment on one patch. Adjusting additional patches requires a small amount of additional work. To adjust multiple patches, for each tracked patch, a count must be made of the number of pixels in the local connected component that are part of the patch.

Adjustment relies on the assumption that object boundaries coincide with motion boundaries. However, it also relies on the assumption that most motion boundaries

in the vicinity of object boundaries coincide with object boundaries. If, for example, the camera is undergoing a translation in a cluttered environment, there may be a plethora of motion boundaries.

When an object stops moving, local adjustment does not have the drastic effects of global adjustment. Yet, if the object's stopping results in a uniform optical flow field, local adjustment will have no local motion boundary to which to adjust. Thus, a convex tracked object will be gradually eaten away by local adjustment, since all pixels at the edge of the object will be moving with the same displacement and a majority of them will not be part of the tracked object.

Adjusting the output of the projection phase greatly improves the quality of the match between O_{i+1} and $\tilde{O}_{i+1}$. Nevertheless, inaccuracies in the optical flow field, motion boundaries internal to moving objects, and local effects from stop-and-go motion all contribute to a gradual degradation in the projected and adjusted tracked object over time.

6.3 Related work

The goal of tracking, as the term is used in this thesis, is to maintain information about the image-relative position of an object across time. There are three principal approaches that have been used to attack the problem. In 3-D model-based tracking, a 3-D model of the object that is being tracked is used to determine the object position across time. Special-purpose tracking takes advantage of distinctive visual properties of the objects that are to be tracked in order to simplify tracking. Generic object tracking algorithms, including the one described in this chapter, rely on visual properties that are expected to exist over wide classes of objects.

6.3.1 3-D model-based approaches

Systems that rely on a known three dimensional model of the object to be tracked tend to work in one of two ways. In one approach, the motions of individual features are used to determine the overall change in pose of the object. In the other approach,

a generate-and-test search in pose space is used to determine the new pose of an object. Both approaches assume an initial pose is known. These approaches often produce much more information than the image extent of the tracked object, but depend critically on a known object model.

Gennery [37] assumes a surface model and known initial position. Predictions of future positions of the object are generated from previous object motion and are used to facilitate the search for object features. Measurements of feature positions are used to improve estimates of object position. Thompson and Mundy [89] use a predicted object position to seed a heuristic search in object pose space. Verghese *et al.* [97] describe two algorithms. Their first algorithm assumes small changes in object pose in order to perform a local search in pose space. Each pose is evaluated by comparing predicted appearance with actual image. Their second algorithm assumes that image features are tracked across time. Knowing the cross-temporal correspondence between features makes it possible to determine the change in pose of the tracked object. Lowe [60] extends the general approach by introducing models with articulations.

Several researchers have attempted to use 3-D models to track flexible objects such as people. O'Rourke and Badler [74], and Hogg [45] have built systems that predict poses and hence feature positions to estimate the position and pose of people in image sequences. Poses and positions are determined by searching for configurations that are consistent with measured features. O'Rourke and Badler [74], use distinctive features such as hands and feet as the basis for pose determination, while Hogg [45] uses edge correspondences and image differencing as initial information.

These approaches assume a previously known object model, limiting their use to problems in which a fixed class of objects are to be tracked. Since object models are either rigid or assume limited articulation, only restricted classes of objects can be tracked. The search involved in determining the pose of multiply articulated objects such as people tends to make analysis extremely costly.

Building up 3-D models by tracking individual features promises to avoid the problem of a previously specified, fixed class of known objects. Roach and Aggarwal [80] use heuristics to generate wire frame models of objects in each scene, and attempt to determine a correspondence between models from image to image. Crowley *et al.* [28]

track edge segments from frame to frame in order to build up an object model across time. Although these systems seem to track previously unspecified objects, the work tends to assume that the objects are easily separable from the background, and is limited to rigid objects.

6.3.2 Special-purpose tracking

If some distinctive property of an object is known, e.g., that it has a plume that will appear in a specified frequency range, tracking can be performed by repeatedly picking out the object manifesting the distinctive property. The key to success with this class of algorithms is the existence of a robust mechanism for picking out the object that is being tracked in each frame.

Gilbert *et al.* [38] describe a system used to keep a camera aimed at and focussed on a plane or missile. The part of the system that keeps track of an object's image relative position relies on the assumption that "the target image has some video intensities not contained in the immediate background." It uses dynamically learned gray-level distributions to pick out the object in each frame. Schalkoff and McVey [82] propose a general architecture for tracking that relies upon being able to separate the object to be tracked from the background in each frame. Horswill [48] uses significant variation in gray-level intensity as a cue for picking out and tracking objects in his prey-following robot. Andersson [8] relies on the contrast between a white ping-pong ball and a dark background in the construction of his ping-pong playing robot. Similarly, Yamauchi and Nelson [103] make use of black balloons against a light background to implement a juggling robot.

6.3.3 Generic object tracking

Approaches to tracking that require neither a pre-specified object model nor continuous separability tend to rely on local independent motion measurements to keep track of the object. There are generally two cross-temporal goals to these algorithms: maintaining correspondence and maintaining coherence. Maintaining correspondence involves determining where each part of the object that is being tracked has gone

to. Maintaining coherence of the representation of the object being tracked requires dealing both with parts of the object that appear and disappear, and with errors in maintaining correspondence.

Early work on tracking clouds [56, 86] dealt primarily with maintaining correspondence; work consisted of computing correlation-based optical flow fields. Similarly, Chien and Jones [26] although arguing in favor of the general idea of an object tracking modularity, focus on the issue of tracking features across time.

Tsuji *et al.* [95] work on binarized cartoon imagery in tracking flexible objects. Their technique relies on repeatable segmentation of the image into distinct semantic units. Segmented regions are sought in each subsequent image to provide cross-temporal tracking. In real imagery, repeatable, meaningful segmentation is extremely difficult to achieve.

Kass and Witkin [53] use energy minimizing-splines, or *snakes*, to track moving objects. The energy functional used for tracking with snakes incorporates both the goal of maintaining correspondence and the goal of maintaining object coherence into one equation. The approach relies on the existence of local energy minima at object boundaries. If such minima do not exist or disappear, or if the object moves too far, the tracked contour will diverge from the object.

Baker and Garvey [10], extending their work on constructing spatiotemporal surfaces, track independently moving objects on the constructed surfaces. This work has the advantage that the spatiotemporal surfaces capture cross-temporal coherence in the motion of image features. Only preliminary work has been done on working out the correspondence between the space-time manifold and individual objects.

6.4 Discussion

Tracking is a useful visual modularity for systems that must function in dynamic environments. Most tracking research has assumed the existence of an object model, or the existence of some simple, repeatable object segmentation technique. The tracking algorithm described in this chapter provides tracking that is not object specific.

The algorithm has two phases: projection through the optical flow field which

maintains correspondence from frame to frame, and adjustment to motion boundaries which maintains the coherence of the tracked object. Figures 6.9, and 6.10 show the results of these two phases on actual images. The upper left frame in Figure 6.9 shows the outline of an input boolean object. A rectified flow field is shown in the upper right. The lower left of Figure 6.9 shows the results of projection through the rectified flow field. The lower right shows the results of filling holes in the projected patch.

Figure 6.10 shows the effects of adjustment. In the upper left hand corner, discontinuities in the rectified flow field $\overline{M}_{i+1}$ are overlaid on O^2_{i+1}, the result of projection and smoothing. The upper right hand frame shows the same discontinuities overlaid on image I_{i+1}. The lower left hand frame shows the discontinuities overlaid on the adjusted patch. The lower right hand frame shows the outline of the adjusted object on image I_{i+1}.

Input tracked object (O_i) Rectified flow field ($\overline{M}_i$)

Unsmoothed projected object (O^1_{i+1}) Smoothed projected object (O^2_{i+1})

Figure 6.9: Projection in action

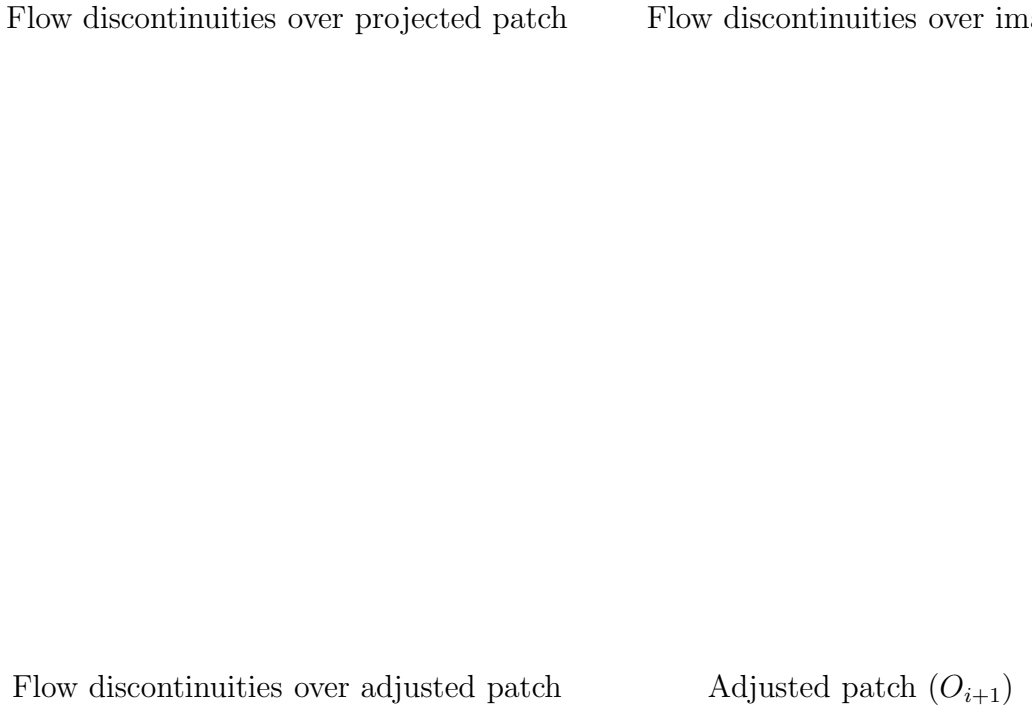


Figure 6.10: Adjustment in action

Chapter 7

Conclusion

This thesis has explored the problems of vision for robots in dynamic, unstructured environments. Exploring vision problems with a general class of tasks in mind has led to a non-traditional approach to vision research in which usability and computational efficiency take precedence over purity of results.

The thesis introduces the idea of *vision services* as an approach to building vision systems for robots in dynamic, unstructured environments. A vision service must summarize camera data in a way that directly supports action; it must be able to function without prior knowledge of a particular environment or the kinds of objects that appear in it; and it must be able to continue to provide useful summarizations without constant guidance.

A system that picks out and tracks moving objects is described. The system is shown to be an example of a vision service in that it continually provides object-position information without recourse to prior information or external guidance.

Since vision services operate in real-time and are suitable for directly controlling action, they permit immediate empirical validation, but make analytic validation difficult.

As vision services need to report on events in the world in an immediately usable fashion, the issue of how to simply characterize a scene comes to the fore. To a degree, the task of designing vision services is an exercise in practical reification. A vision service must be able to carve the perceived world into a small number of elements

such that descriptions in terms of these elements are useful for action.

Carving up the visual world on the basis of distinct visual motion is one instance of such a practical reification scheme. It appears that scene depth as computed from stereo imagery can be the basis for other approaches to usefully individuating scene elements in an unstructured environment. Other visual modalities such as color or texture may require more domain dependent information in order to support reification.

7.1 Lessons

The principal, unexpected lesson from this work is the benefit of adhering to uniform representations and uniform computation. One advantage is the computational benefit achieved from shared local computations. Another advantage is the implementational benefit arising from the fact that uniform algorithms operating on uniformly represented data end up being intrinsically simple.

Another, perhaps obvious, lesson is that attempting to build systems that work on real images uncovers many lurking issues. An example of such an issue is the motion aliasing problem discussed in Section 4.4.1 that appears unrecognized in the vision literature.

7.2 Complexity

If the proposal that researchers ought to implement vision services that can be coupled to effectors is taken seriously, then the computational complexity of different vision techniques becomes of prime importance. To this end, the complexity of the various algorithms described throughout the thesis are summarized below. Recall that the definitions of various symbols are summarized in Appendix A.

Computation:

Phase	Serial	Parallel
Initial estimate	$\mathcal{O}(6\|\mathbf{D}\|N^2)$	$\mathcal{O}(6\|\mathbf{D}\|)$
Mode filter	$\mathcal{O}((3\|\mathbf{D}\|+1)N^2)$	$\mathcal{O}((2\lg d_m+1)\|\mathbf{D}\|-1)$
Rectification	$\mathcal{O}(3N^2)$	—
Segmentation	$\mathcal{O}((2n_s+2)N^2+(3d_g+\lg\|\mathbf{T}\|)\|\mathbf{T}\|)$	$\mathcal{O}(2n_s+2+(3d_g\ \lg\|\mathbf{T}\|)\|\mathbf{T}\|)$
Projection	$\mathcal{O}(6N^2)$	$\mathcal{O}(2\lg d_p)$
Adjustment	$\mathcal{O}((6d_a+4)N^2)$	$\mathcal{O}(2\lg\|\mathbf{D}\|+3d_a^2)$
Average case	$22N^2$	—

Communication:

Phase	Hypercube	Non-hypercube
Initial estimate	$\mathcal{O}(\lg\|\mathbf{D}\|+3)$	—
Mode filter	$\mathcal{O}(2\lg d_m\|\mathbf{D}\|)$	—
Rectification	—	$\mathcal{O}(2)$
Segmentation	—	$\mathcal{O}(n_s+1)$
Projection	$\mathcal{O}(2\lg d_p)$	$\mathcal{O}(1)$
Adjustment	$\mathcal{O}(2+2\lg d_a)$	—

7.3 Open issues and limitations

Many issues are left open by this work. At a broad level, there is the question of whether the idea of vision services will allow additional progress to be made in machine vision. What services can be defined that are both useful and workable?

The presented segmentation and tracking techniques work well for rotations about the x and y axes of the camera. Will these techniques work for more general camera motions? If not, what techniques will?

The presented optical flow algorithm has limits on velocities that are too great, and velocities that are too small. What techniques will allow large motions to be measured? Can the approach be extended to subpixel motions?

The problem of motion aliasing discussed in Section 4.4.1 has been worked around in the current system. What general approaches could be taken to avoid the problem?

7.4 Future work

Cheap, general purpose computers are only now becoming powerful enough to perform real-time motion and stereo depth computations. The time is ripe for explorations in the space of vision services that couple cameras and effectors in real environments.

The notion of vision services must be tested by introducing vision services that are suitable for navigation and object manipulation in dynamic, unstructured environments. Various combinations of depth and motion measurements can be the basis for such vision services.

7.5 Summary of contributions

The contributions of this thesis include several algorithms to support an optical flow-based tracking system, an approach to building vision systems for dynamic, unstructured environments, and several implemented systems that validate the algorithms and indicate the power of the approach.

The thesis introduced a tracking algorithm that makes use of optical flow to keep track of unmodeled, non-rigid moving objects across time. An algorithm for picking out moving objects on the basis of clustered cross-temporal behavior was presented. A novel, efficient, optical flow computation based on SSD correlation and mode filtering was described. The performance of these algorithms was analyzed in detail to allow comparison with other methods.

The idea of *vision services* as a model for vision systems that are to be used in dynamic, unstructured domains was introduced, and shown to subsume the tracking system. The tracking service and its component vision algorithms have been validated with two real-time robotic camera pursuit systems.

By introducing a real-time, optical flow-based tracking system, this thesis has demonstrated that optical flow can be a practical visual modality for real-time systems in dynamic, and unstructured environments.

Appendix A

Special notation and constants

A.1 Special notation

Symbol	Denotation	Page
$\otimes$	the operator used in dynamic programming	30
$\mathcal{B}$	the set of boolean images	22
d_a	the diameter of the local adjustment region	101
d_c	the diameter of the correlation window	54
d_g	the width of the Gaussian mask used in segmentation	88
d_m	the diameter of the mode filter	62
d_p	the diameter of the clean up region in projection	95
$\mathbf{D}$	the set of displacements	27
fov_h	the horizontal field of view	23
fov_v	the vertical field of view	23
$\mathcal{C}$	the set of connected components of motion	97
$\mathcal{C}^l(p)$	a local version of a component	101
$\mathcal{C}^=$	local component based on equality	101
$\mathcal{C}^r$	local component based on radial connectedness	104

Symbol	Denotation	Page
$H_i(v, n_s)$	the histogram of trajectories	84
I	a single gray-level image	22
$\mathcal{I}$	the set of gray-level images	22
$L(p)$	the likelihood of given motion estimate	64
$\tilde{M}$	a motion field, representing true motion	26
$\mathcal{M}$	the set of optical flow fields	27
M	an optical flow field approximating the motion field	27
$\overline{M}$	a rectified optical flow field	64
M^*	the initial measurement of motion	50
$\mathcal{M}^*$	the set of multivalued optical flow fields	50
n_a	number of distinct motions in adjustment column	103
n_s	the number of flow fields used for segmentation	83
N	the height and width of an image	28
$\tilde{O}$	a boolean image representing an object	27
O	an approximation to $\tilde{O}$	27
$\mathbf{P}$	the set of pixels	22
$P_i(p, k)$	the k-th predecessor	83
Q_i	the partition of trajectory vectors	84
r_a	the radius of the local adjustment region	105
r_m	the radius of the mode filter region	58
r_v	the radius of the search window	51
$R(p)$	the $d_a \times d_a$ square surrounding p	100
Si	the result of motion segmentation	85
σ	the result of applying $\otimes$ to a local area	30
$\sup_f \mathcal{S}$	the element of $\mathcal{S}$ for which f is greatest	28
$T_i(p, k)$	a field of trajectories	83
$\mathbf{T}$	the set of trajectories used in segmentation	84

A.2 Constants

$\|\mathbf{D}\|$	=	37
d_a	=	15
d_g	=	9
d_m	=	7 in serial, 8 in parallel
d_p	=	7 in serial, 8 in parallel
fov_h	=	38°
fov_v	=	30°
n_a	≈	3
n_o	=	2
n_s	=	4
N	=	128 in parallel, 128 and 114 in serial
r_a	=	7
r_m	=	3

Appendix B

Local dynamic programming

As mentioned in Section 2.7 many of the operations used in the tracking system involve uniform local computations in the area surrounding each pixel. These operations are often amenable to a decomposition into subproblems such that partial results can be cached and shared across pixels, in a form of dynamic programming. Significant computational savings are achieved through these local area dynamic programming techniques.

These dynamic programming techniques can be applied if the desired local result at each pixel is the application of a group operation $\otimes$ from a group $\langle G, \otimes, e, {}^{-1} \rangle$ to a rectangular local area surrounding each pixel in a matrix of elements of G. Let F be such a two-dimensional matrix of elements of G. The origin is in the upper left, x coordinates run left to right, and y coordinates run top to bottom. Let $\sigma(x, w, y, h)$ denote the result of applying $\otimes$ to the local area of width w and height h whose lower right hand corner is pixel $\langle x, y \rangle$[1].

$$\sigma(x, w, y, h) \quad \equiv \quad \bigotimes_{[x-w<i\leq x]} \bigotimes_{[y-h<j\leq y]} F(i, j)$$

The symbol "$\bigotimes$" denotes the ordered application of $\otimes$ to the local area, much as "$\sum$" denotes the application of $+$. The degenerate case of width and height 1, picks out

[1]In Section 2.7, $\sigma(p, w, h)$ denoted the local summary centered around p. In this appendix, $\sigma(x, w, y, h)$ denotes a local summary picked out by its lower right hand corner. The two notations are related as follows: $\sigma(x, w, y, h) = \sigma(\langle x - \lceil \frac{w}{2} \rceil, y - \lceil \frac{h}{2} \rceil \rangle$,w,h).

an element of F.

$$\sigma(x, 1, y, 1) \quad \equiv \quad F(x, y)$$

The application of $\otimes$ to a column of height h can be written as a region of width 1.

$$\sigma(x, 1, y, h) \quad \equiv \quad \bigotimes_{[y-h<j\leq y]} F(x, j)$$

As noted in Section 2.7, if $\otimes$ is an operation whose output is larger than its input, e.g., addition on the natural numbers, there is a logarithmic complexity term related to the size of the output. However, for reasonably sized local areas, this logarithmic factor stays below the word size of a machine, and is ignored.

B.1 Serial dynamic programming

Computing the desired result, $\sigma(x, w, y, h)$, in serial is performed in some fixed order: assume top to bottom, left to right. The aim of dynamic programming is to decompose the problem into partial results that can be shared. The left hand side of Figure B.1 shows such a decomposition where + instantiates $\otimes$. The lower left square depicts the local area of the desired result. The upper left square depicts the local area of the result for the previous pixel (the pixel directly to the left of the current pixel). The center left square shows that the desired area can be generated from the previous result by removing a column from the left and appending a column on the right. As shown in the figure, the local aggregation for a pixel $\langle x, y\rangle$ can be decomposed into the product of three terms: the complete result for the neighboring pixel to the left $\langle x-1, y\rangle$, $\sigma(x-1, w, y, h)$, and two partial results in the form of columns, the leftmost column of $\sigma(x-1, w, y, h)$, that is $\sigma(x-w, 1, y, h)$; and the rightmost column of the desired result $\sigma(x, w, y, h)$, namely, $\sigma(x, 1, y, h)$.

$$\sigma(x, w, y, h) \equiv \sigma(x-1, w, y, h) \otimes \sigma(x-w, 1, y, h)^{-1} \otimes \sigma(x, 1, y, h)$$

The right hand side of Figure B.1 shows a similar decomposition for a single pixel wide column. Each column can be decomposed into the product of three terms: the

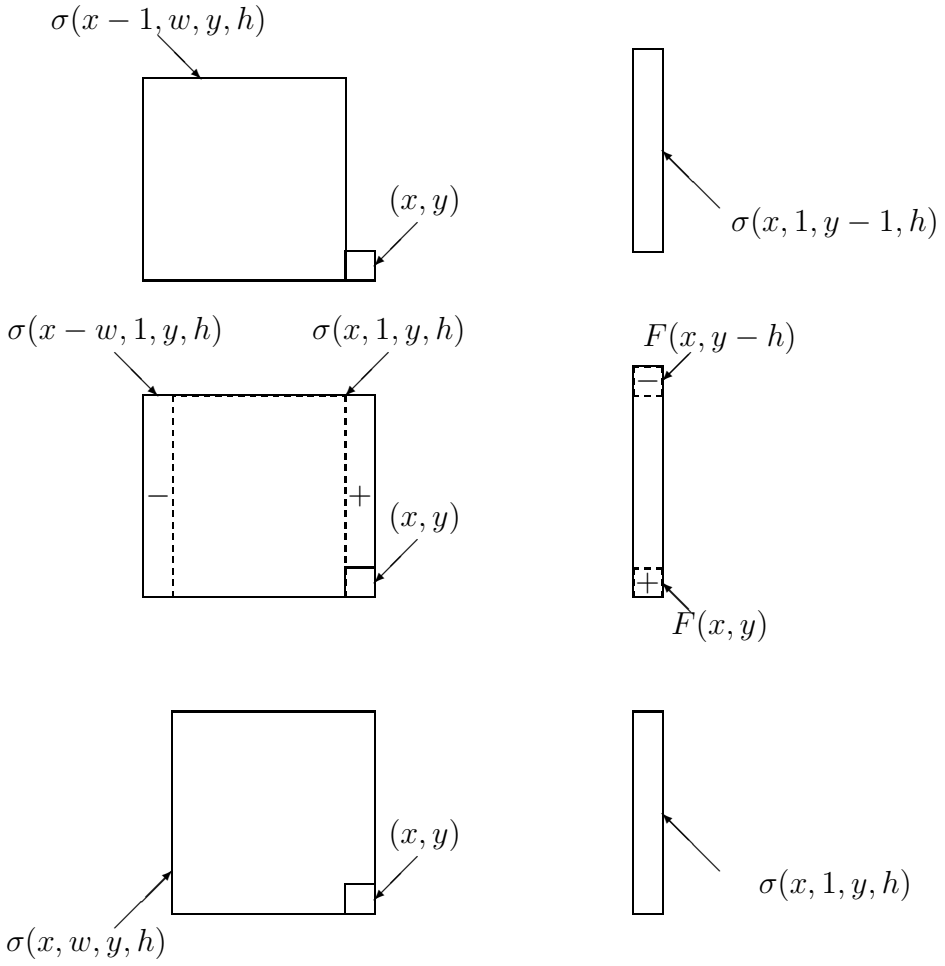


Figure B.1: Dynamic programming a local sum in serial

column from the previous row, and two of the basic elements from the matrix F, the top element of the old column, and the bottom element of the new column.

$$\sigma(x, 1, y, h) \equiv \sigma(x, 1, y - 1, h) \otimes F(x, y - h)^{-1} \otimes F(x, y)$$

Given this decomposition, as long as enough previous results have been preserved, the result for a local area of arbitrary size can be computed with four $\otimes$ operations per pixel. Two operations are needed to compute the right column of a new area, and two operations are needed to combine the left column, the old result and the right column. Applying $\otimes$ to a $w \times h$ area at each pixel on an $N \times N$ image takes

$$\mathcal{O}(4N^2)$$

applications of $\otimes$.

As noted in Section 2.7, there is a certain fixed overhead that occurs at the edges of the matrix F. This overhead is not included in the complexity figures, as it is independent of image size. However, if the serial computation were split across many serial processors, this overhead could become significant. In the extreme, running the serial algorithm on each processor of the Connection Machine would incur a great deal of communication and would cost $\mathcal{O}(wh)$ $\otimes$ operations per pixel.

B.2 SIMD dynamic programming

It is assumed that the matrix F of basic values that are being aggregated is distributed across processors, so that each processor is associated with one coordinate $\langle x, y \rangle$ and one value of $F(x, y)$. On a SIMD computer, results are completed simultaneously on all processors, hence decomposition techniques that depend on a neighboring processor's final results cannot be used. Instead, one can use a binary recursive decomposition of partial results.

Consider summing numbers over a local area. Initially, a processor only has its local input $F(x, y)$. By fetching a neighbor's value, and adding it to the initial value, each processor has the sum of a one by two area. By fetching a new sum from a neighbor on the other axis and adding it, each processor has the sum of a two by

two area. This process can be repeated until the desired sum is available at each processor.

For simplicity, assume that the width w and height h of local regions are powers of two. The aggregation for a region can thus be written recursively.

$$\begin{aligned}
\sigma(x,1,y,1) &\equiv F(x,y) \\
\sigma(x,w,y,h) &\equiv \sigma(\frac{x-w}{2},\frac{w}{2},y,h) \otimes \sigma(x,\frac{w}{2},y,h) \\
\sigma(x,w,y,h) &\equiv \sigma(x,w,\frac{y-h}{2},\frac{h}{2}) \otimes \sigma(x,w,y,\frac{h}{2})
\end{aligned}$$

For SIMD processing, one needs to consider where each partial result should be generated. In particular, it is important that the final result for a local area end up on the processor that corresponds to the center of the local area. Processor $\langle x,y\rangle$ thus ought to end up with the value $\sigma(\frac{x+w}{2},w,\frac{y+h}{2},h)$. Figure B.2 depicts the generation of $\sigma(x+4,8,y,h)$ on processor $\langle x,y\rangle$, starting from the column result $\sigma(x,1,y,h)$.

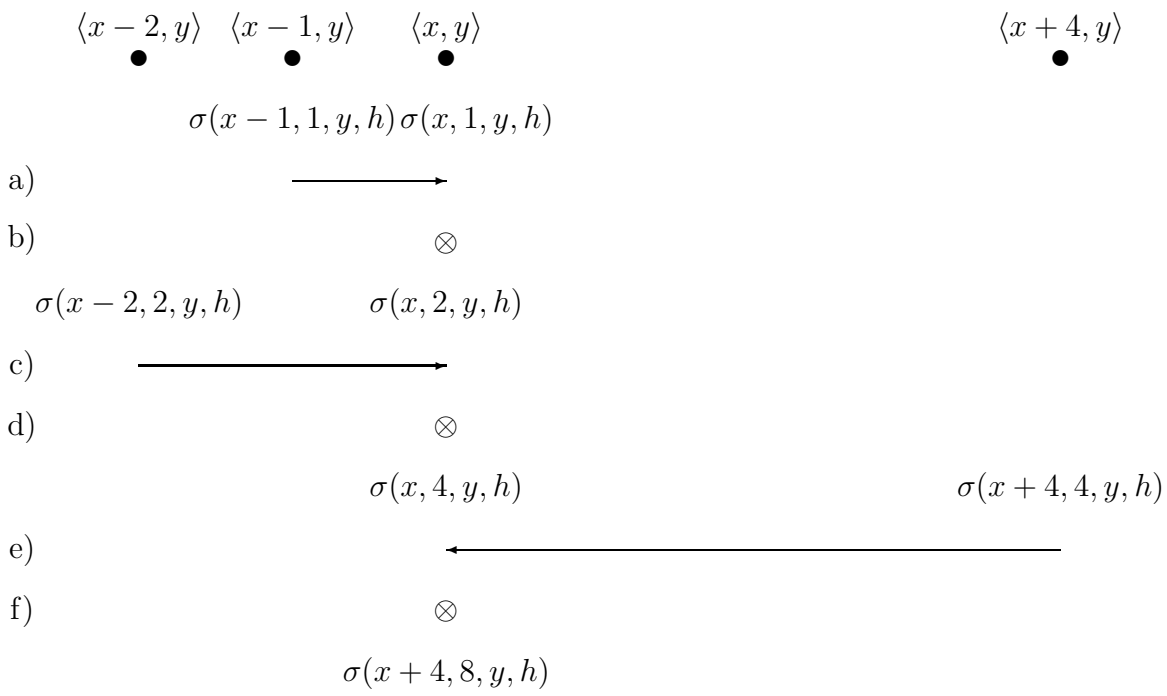
Since the area being worked on doubles at each step, the local aggregation of a rectangular region of width w and height h can be computed with $\lg w + \lg h$ $\otimes$ operations per pixel. For an $N \times N$ image, $\otimes$ can be computed over the whole image in

$$\mathcal{O}((\lg w + \lg h)N^2)$$

applications of $\otimes$. Since the communications depicted in Figure B.2 are all of power of two distances, the local aggregation can be performed with $\lg w + \lg h$ hypercube communications.

An interesting example of this parallel dynamic programming involves the use of concatenation. Many algorithms require values from all neighboring processors in some local region of width w and height h. To transmit each value independently would require wh uniform point to point communications, and still more hypercube communications. Using the above decomposition, the number of communications can be reduced to $\lg w + \lg h$.

This binary recursive version of the dynamic programming technique makes no use of the inverse, $^{-1}$, and can hence be performed for a monoid operation. The recursive dynamic programming technique can be used on serial computers to compute an



Black dots denote processors along the x axis. The dot labeled $\langle x, y\rangle$ is the processor whose result is being considered. Arrows indicate the transmission of partial results from neighboring processors.

Step (a) fetch a value $\sigma(x-1, 1, y, h)$ from processor on left $\langle x-1, y\rangle$.
Step (b) apply $\otimes$ to $\sigma(x-1, 1, y, h)$ and $\sigma(x, 1, y, h)$ to form $\sigma(x, 2, y, h)$.
Step (c) fetch a pair from processor at distance two.
Step (d) apply $\otimes$ to $\sigma(x-2, 2, y, h)$ and $\sigma(x, 2, y, h)$ to form $\sigma(x, 4, y, h)$.
Step (e) fetch a quadruple from processor at distance four.
Step (f) apply $\otimes$ to $\sigma(x, 4, y, h)$ and $\sigma(x+4, 4, y, h)$ to form $\sigma(x+4, 8, y, h)$.

Figure B.2: Dynamic programming $\otimes$ in parallel

aggregation of a monoid operation without an inverse, e.g., max. The number of computations done in serial is the same as in parallel, but no communication is required.

Appendix C

Motion and tracking algorithms in LISP

This LISP pseudo-code presents the motion and tracking algorithms specified in Chapter 4 and Chapter 6 in procedural form. The code is only intended to specify the desired results of the computation. The given procedures are not the dynamically programmed algorithms whose performance is analyzed. Most of the procedures ignore boundary conditions. The motion algorithm is followed by the tracking algorithm.

C.1 Motion algorithm

The motion algorithm takes as input two images and produces an optical flow field.

```
(defun compute-motion (i1 i2 optical-flow-field)
  (declare
    (type image i1 i2)
    (type flow-field optical-flow-field))
  (let (initial-flow-field votes-field filtered-flow-field)
    (declare
      (type flow-field initial-flow-field filtered-flow-field)
      (type votes-field votes-field))
    (compute-initial-motion-measurement i1 i2 initial-flow-field)
    (mode-filter initial-flow-field filtered-flow-field votes-field)
    (rectify filtered-flow-field votes-field optical-flow-field)))
```

The flow field produced by `compute-initial-estimates` can have multiple displacements for a single pixel, since ties can occur. Thus the resulting flow field is a matrix of lists of displacements.

```
(defun compute-initial-estimates (i1 i2 initial-flow-field)
   (declare
     (type image i1 i2)
     (type flow-field initial-flow-field))
   (dotimes (y (array-dimension i1 1))
      (dotimes (x (array-dimension i1 0))
         (setf (aref result x y)
               (initial-estimates i1 i2 x y)))))


(defvar *displacements* '((-2 2) (-1 2) ... (0 0) ... (2 2)))

(defun displacement-x (displacement)
  (first displacement))

(defun displacement-y (displacement)
  (second displacement))

(defun initial-estimates (i1 i2 x y)
  (declare
    (type image i1 i2)
    (type fixnum x y))
  (let* ((best-displacements (list (first *displacements*)))
         (least-ssd-score (ssd-score i1 i2 x y (first best-displacements))))
    (dolist (displacement (cdr *displacements*))
      (let ((ssd-score (ssd-score i1 i2 x y displacement))
            (cond ((< ssd-score least-ssd-score)
                   (setq best-displacements (list displacement))
                   (setq least-ssd-score ssd-score))
                  ;; a tie
                  ((= ssd-score least-ssd-score)
                   (push displacement best-displacements))))))
    (return-from initial-estimates best-displacements)))
```

```
(defun ssd-score (i1 i2 x y displacement)
  (let ((sum-of-squared-differences 0))
    (loop for (xx yy) in '((0 -1) (-1 0) (0 0) (1 0) (0 1)) do
      (let* ((old-pixel (aref i1 (+ x xx) (+ y yy)))
             (new-pixel (aref i2
                              (+ x xx (displacement-x displacement))
                              (+ y yy (displacement-y displacement))))
             (difference (- old-pixel new-pixel))
             (squared-difference (* difference difference)))
        (setq sum-of-squared-differences
              (+ sum-of-squared-differences squared-difference))))
    (return-from ssd-score sum-of-squared-differences)))



(defvar *mode-filter-radius* 3)

(defun mode-filter (initial-flow-field filtered-flow-field votes-field)
  (declare
    (type flow-field initial-flow-field filtered-flow-field)
    (type votes-field votes-field))
  (dotimes (y (array-dimension initial-flow-field 1))
    (dotimes (x (array-dimension initial-flow-field 0))
      (multiple-value-bind (displacement votes)
          (mode-filter-one-pixel initial-flow-field x y)
        (setf (aref filtered-flow-field y) displacement)
        (setf (aref votes-field x y) votes)))))



(defun mode-filter-one-pixel (initial-flow-field x y)
  (declare
    (values best-displacement most-votes))
  (let* ((best-displacements (list (first *displacements*)))
         (most-votes (count-votes (first best-displacement) initial-flow-field x y)))
    (dolist (displacement (cdr *displacements*))
      (let ((vote-count (count-votes displacement initial-flow-field x y)))
        (cond ((> vote-count most-votes)
               (setq best-displacements (list displacement))
               (setq most-votes vote-count))
              ;; a tie
              ((= vote-count most-votes)
               (push displacement best-displacements)))))
    (return-from mode-filter-one-pixel
      (values (choose-random-element best-displacements)
              most-votes))))
```

```
(defun count-votes (displacement initial-flow-field x y)
  (let ((vote-count 0))
    (loop for yy from (- *mode-filter-radius*) to *mode-filter-radius* do
      (loop for xx from (- *mode-filter-radius*) to *mode-filter-radius* do
        (when (member displacement
                      (aref initial-estimate (+ x xx) (+ y yy)))
          (setq vote-count (+ vote-count 1)))))
    (return-from count-votes vote-count)))


(defun rectify (filtered-flow-field votes-field surjective-optical-flow-field)
  (declare
    (type flow-field filtered-flow-field surjective-optical-flow-field)
    (type votes-field votes-field))
  (let (temp-dest-array)
    (setq temp-dest-array (make-array (array-dimensions filtered-flow-field)))
    (initialize-nil surjective-optical-flow-field)
    (initialize-nil temp-dest-array)
    (dotimes (y (array-dimension filtered-flow-field 1))
      (dotimes (x (array-dimension filtered-flow-field 0))
        (let ((displacement (aref filtered-flow-field x y))
              (vote-count (aref votes-field x y)))
          (push (make-competitor :vote-count vote-count :source-x x :source-y y)
                (aref temp-dest-array
                      (+ x (displacement-x displacement))
                      (+ y (displacement-y displacement)))))))
    ;;
    ;; temp-dest-array now contains at each pixel, a list of
    ;; entries corresponding to all pixels that think they were
    ;; displaced to this pixel.
    ;;
    (dotimes (y (array-dimension filtered-flow-field 1))
      (dotimes (x (array-dimension filtered-flow-field 0))
        (let ((competitors (aref temp-dest-array x y)))
          (when competitors
            (let ((winner (choose-one-best-scoring-competitor competitors)))
              (setf (aref surjective-optical-flow-field
                          (competitor-source-x winner)
                          (competitor-source-y winner))
                    (aref filtered-flow-field
                          (competitor-source-x winner)
                          (competitor-source-y winner)))))))))
```

```
(defun choose-one-best-scoring-competitor (competitors)
  (let* ((first-competitor (first competitors))
         (best-competitors (list first-competitor))
         (most-votes (competitor-votes first-competitor)))
    (dolist (competitor (cdr competitors))
      (let ((vote-count (competitor-votes competitor)))
        (cond ((> vote-count most-votes)
               (setq most-votes vote-count)
               (setq best-competitors (list competitor)))
              ;; a tiel
              ((= vote-count most-votes)
               (push competitor best-competitors)))))
    (let ((winner (choose-random-element best-competitors)))
      (return-from choose-one-best-scoring-competitor winner))))
```

C.2 Tracking algorithm

The tracking algorithm takes a boolean image representing a tracked object, and two surjective flow fields as input and produces a new boolean image representing the new position of the object. Boolean images can represent one of three values per pixel: 0, 1, or `nil`.

```
(defun track (input-object-map first-flow-field second-flow-field output-object-map)
  (declare
    (type object-map input-object-map output-object-map)
    (type flow-field first-flow-field second-flow-field))
  (let (intermediate-object-map)
    (declare
      (type object-map intermediate-object-map))
    (project input-object-map first-flow-field intermediate-object-map)
    (adjust intermediate-object-map second-flow-field output-object-map)))
```

```
(defvar *projection-smooth-diameter* 7)

(defun project (input-object-map surjective-flow-field output-object-map)
   (declare
     (type object-map input-object-map output-object-map)
     (type flow-field surjective-flow-field))
   (let (intermediate-object-map)
     (declare
      (type object-map intermediate-object-map))
     (initialize-nil intermediate-object-map)
     (dotimes (y (array-dimension input-object-map 1))
       (dotimes (x (array-dimension input-object-map 0))
         (let ((displacement (aref surjective-flow-field x y))
               (object? (aref input-object-map x y)))
           (when displacement
             (setf (aref intermediate-object-map
                         (+ x (displacement-x displacement))
                         (+ y (displacement-y displacement)))
                   object?)))))
     ;;
     ;; intermediate-object-map now is nil only at pixels to
     ;; which no pixel is mapped by surjective-flow-field.
     ;;

     (boolean-smooth intermediate-object-map *projection-smooth-diameter*
                     output-object-map)))


(defun boolean-smooth (input-object-map diameter output-object-map)
  (declare
    (type object-map input-object-map output-object-map)
    (type fixnum diameter))
  (let* ((lower-bound (floor (/ diameter 2)))
         (upper-bound (- diameter lower-bound))
         (half-area (/ (* diameter diameter) 2)))
    (dotimes (y (array-dimension input-object-map 1))
      (dotimes (x (array-dimension input-object-map 0))
        (unless (numberp (aref input-object-map x y))
          (let ((sum 0))
            (loop for yy from (- lower-bound) to upper-bound do
              (loop for xx from (- lower-bound) to upper-bound do
                (setq sum (+ sum (down (aref input-object-map (+ x xx) (+ y yy)))))))
            (setf (aref output-object-map x y) (> sum half-area)))))))))
```

Function `down` maps 0 and 1 to themselves and `nil` to 0.

```
(defun down (val)
  (if (numberp val) val 0))


(defvar *adjustment-radius* 7)

(defun adjust (input-object-map optical-flow-field output-object-map)
  (declare
    (type object-map input-object-map output-object-map)
    (type flow-field optical-flow-field))
  (dotimes (y (array-dimension input-object-map 1))
    (dotimes (x (array-dimension input-object-map 0))
      (setf (aref output-object-map x y)
            (adjust-one-pixel input-object-map optical-flow-field x y)))))


(defun adjust-one-pixel (input-object-map optical-flow-field x y)
  (declare
    (type object-map input-object-map)
    (type flow-field optical-flow-field))
  (let ((same-component
          (make-array (list (+ *adjustment-radius* 1 *adjustment-radius*)
                            (+ *adjustment-radius* 1 *adjustment-radius*))))
        (same-component-count 0)
        (same-component-and-tracked-count 0)
        ratio)
      (compute-same-component optical-flow-field x y same-component)
      (loop for xx from (- *adjustment-radius*) to (+ *adjustment-radius*) do
         (loop for yy from (- *adjustment-radius*) to (+ *adjustment-radius*) do
            (when (aref same-component (+ xx *adjustment-radius*)
                                       (+ yy *adjustment-radius*))
               (setq same-component-count (+ 1 same-component-count))
               (when (aref input-object-map (+ x xx) (+ y yy))
                  (setq same-component-and-tracked-count
                     (+ 1 same-component-and-tracked-count))))))
      (setq ratio (/ same-component-and-tracked-count same-component-count))
      (return-from adjust-one-pixel (> ratio (/ 1 2)))))
```

The computation done by the procedure `compute-same-component` differs in the SIMD and serial versions.

```
(defun compute-same-component (optical-flow-field x y same-component)
   (if (not *SIMD*)
       (locally-connected-same-component optical-flow-field x y same-component)
       (locally-counting-same-component final-estimate x y same-component)))
```

The non SIMD algorithm for computing a local component collects all pixels in the local neighborhood that have the same displacement as the local pixel.

```
(defun locally-counting-same-component (final-estimate x y same-component)
   (let ((my-displacement (aref final-estimate x y)))
      (loop for xx from (- *adjustment-radius*) to (+ *adjustment-radius*) do
         (loop for yy from (- *adjustment-radius*) to (+ *adjustment-radius*) do
             (setf (aref same-component (+ xx *adjustment-radius*)
                                        (+ yy *adjustment-radius*))
                   (equal (aref final-estimate (+ x xx) (+ y yy))
                          my-displacement))))))
```

The SIMD algorithm for computing a local component is fairly complex.

```
(defun locally-connected-same-component (optical-flow-field x y same-component)
  (initialize-nil same-component)
  ;; center is by definition same component
  (setf (aref same-component *adjustment-radius* *adjustment-radius*) t)
  (let ((local-optical-flow-field
          (make-array (list (+ *adjustment-radius* 1 *adjustment-radius*)
                            (+ *adjustment-radius* 1 *adjustment-radius*))))
        ;; copy final estimate into local
        (copy-into-local optical-flow-field x y local-optical-flow-field)
        (loop for radius from 1 below *adjustment-radius* do
          (traverse-radially radius local-optical-flow-field same-component)
          (traverse-laterally radius local-optical-flow-field same-component)))))
```

The procedure `copy-into-local` makes a local copy of a subregion of the optical flow field.

```
(defun copy-into-local (optical-flow-field x y local-optical-flow-field)
  (loop for xx from (- *adjustment-radius*) to (+ *adjustment-radius*) do
    (loop for yy from (- *adjustment-radius*) to (+ *adjustment-radius*) do
      (setf (aref local-optical-flow-field (+ xx *adjustment-radius*)
                  (+ yy *adjustment-radius*))
            (aref optical-flow-field (+ x xx) (+ yyy))))))
```

```
(defun traverse-radially (radius local-optical-flow-field same-component)
  (let ((copy-radius (- radius 1)))
    (loop for sign in '(1 -1) do
      (let ((inner-x (* sign copy-radius))
            (inner-y (* sign copy-radius)))
        (loop for index from (- copy-radius) to (+ copy-radius) do
          ;; do the horizontal first
          (possibly-move-to local-optical-flow-field
                            inner-x index sign 0 :radial)

          ;; do vertical next
          (possibly-move-to local-optical-flow-field
                            index inner-y 0 sign :radial))))))
```

The procedure `possibly-move-to` tests whether the neighboring pixel specified by `x-diff` and `y-diff` is reachable. If the pixel is reachable, it is labeled with the kind of traversal that is being done, radial or lateral.

```
(defun possibly-move-to (local-optical-flow-field x y x-diff y-diff label)
  (when (reachable local-optical-flow-field
                   x y (+ x x-diff) (+ y y-diff))
    (setf (aref same-component (+ x x-diff) (+ y y-diff)) label)))
```

The function `reachable` tests whether two adjacent pixels have the same displacement vector.

```
(defun reachable (local-optical-flow-field old-x old-y new-x new-y)
  (equal (aref local-optical-flow-field old-x old-y)
         (aref local-optical-flow-field new-x new-y)))
```

The procedure `traverse-laterally` attempts to grow the local region outward along a square of radius `radius`. It can only add cells that are adjacent to cells that were added by the previous radial growth step and have the same displacement vector.

```
(defun traverse-laterally (radius local-optical-flow-field same-component)
  (loop for sign in '(1 -1) do
    (let ((signed-radius (* radius sign)))
      (loop for index from (- 1 radius) to (- radius 1) do
        (loop for radial-sign in '(1 -1) do
          ;; do horizontal first
          (when (eq (aref same-component index signed-radius) :radial)
            (unless (aref same-component (+ index radial-sign) signed-radius)
              (possibly-move-to local-optical-flow-field
                                index signed-radius radial-sign 0
                                :lateral)))
          ;; do vertical next
          (when (eq (aref same-component signed-radius index) :radial)
            (unless (aref same-component signed-radius
                          (+ index radial-sign))
              (possibly-move-to local-optical-flow-field
                                signed-radius index 0 radial-sign
                                :lateral)))))))))
```

Bibliography

[1] Edward Adelson and James Bergen. The extraction of spatio-temporal energy in human and machine vision. In *IEEE Motion Workshop*, pages 151–155, 1986.

[2] Gilad Adiv. Determining three-dimensional motion and structure from optical flow generated by several moving objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7(4):384–401, July 1985.

[3] Philip Agre and David Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of AAAI-87*, 1987.

[4] Alfred Aho, John Hopcroft, and Jeffrey Ullman. *Data Structures and Algorithms*. Addison–Wesley, 1983.

[5] John Aloimonos, Isaac Weiss, and Amit Bandopadhay. Active vision. *International Journal of Computer Vision*, 1:333–356, 1988.

[6] Padmanabhan Anandan. *Measuring Visual Motion from Image Sequences*. PhD thesis, University of Massachusetts, Amherst, March 1987.

[7] Padmanabhan Anandan. A computational framework and an algorithm for the measurement of visual motion. *International Journal of Computer Vision*, 2:283–310, 1989.

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

[9] Ruzena Bajcsy. Active perception. *Proceedings of IEEE*, 76(8):996–1005, 1988.

[10] Harlyn Baker and Tom Garvey. Motion tracking on the spatiotemporal surface. In *Image Understanding Workshop*, pages 451–457, 1992.

[11] Dana Ballard. Animate vision. *Artificial Intelligence*, 48:57–86, 1991.

[12] Amit Bandopadhay. *A Computational Study of Rigid Motion Perception*. PhD thesis, University of Rochester, 1986.

[13] Stephen Barnard and William Thompson. Disparity analysis of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2(4), 1980.

[14] J. L. Barron, D. J. Fleet, S. S. Beauchemin, and T. A. Burkitt. Performance of optical flow techniques. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 236–242, 1992.

[15] Michael Black and Padmanabhan Anandan. Constraints for the early detection of discontinuity from motion. In *Proceedings of AAAI-90, Boston, MA*, pages 1060–1066, 1990.

[16] Claude Brice and Claude Fennema. Scene analysis using regions. *Artificial Intelligence*, 1:205–226, 1970.

[17] Rod Brooks. Achieving artificial intelligence through building robots. AI Memo 899, MIT, May 1986.

[18] Rod Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, March 1986.

[19] Rod Brooks. Intelligence without reason. In *Proceedings of IJCAI-91, Australia*, 1991.

[20] Heinrich Bulthoff, James Little, and Tomaso Poggio. A parallel algorithm for real-time computation of optical flow. *Nature*, 337:549–553, 1989.

[21] Peter Burt, James Bergen, Rajesh Hingorani, R. Kolczynski, W. Lee, A. Leung, J. Lubin, and H. Shvaytser. Object tracking with a moving camera. In *Proceedings of IEEE Workshop on Visual Motion*, pages 2–12, 1989.

[22] Peter Burt, Tsai-Hong Hong, and Azriel Rosenfeld. Segmentation and estimation of image region properties through cooperative hierarchical computation. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(12), 1981.

[23] Peter Burt, Chihsung Yen, and Xinping Xu. Local correlation measures for motion analysis, a comparative study. In *Proceedings of IEEE Pattern Recognition and Image Processing Conference*, pages 269–274, 1982.

[24] Peter Burt, Chihsung Yen, and Xinping Xu. Multi-resolution flow-through motion analysis. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 246–252, 1983.

[25] David Chapman. *Vision, Instruction and Action.* PhD thesis, MIT, 1990.

[26] R. Chien and V. Jones. Acquisition of moving objects and hand-eye coordination. In *Proceedings of IJCAI-75*, pages 737–741, 1975.

[27] Guy Coleman and Harry Andrews. Image segmentation by clustering. *Proceedings of the IEEE*, 67:773–785, 1979.

[28] James Crowley, Patrick Stelmaszyk, Thomas Skordas, and Pierre Puget. Measurement and integration of 3-d structures byte tracking edge lines. *International Journal of Computer Vision*, 1992.

[29] R. Cypher, J. Sanz, and L. Snyder. Algorithms for image component labeling on SIMD mesh-connected computers. *IEEE Transactions on Computers*, 39(2):276–281, February 1990.

[30] E. Davies. On the noise suppression and image enhancement characteristics of the median, truncated median, and mode filters. *Pattern Recognition Letters*, 7:87–97, 1988.

[31] E. Davies. *Machine Vision.* Academic Press, 1990.

[32] Rene Descartes. *Meditations on First Philosophy.* Bobbs-Merrill, 1960.

[33] Claude Fennema and William Thompson. Velocity determination in scenes containing several moving objects. *Computer Graphics and Image Processing*, 9:301–315, 1979.

[34] David Fleet and Allan Jepson. Computation of component image velocity from local phase information. *International Journal of Computer Vision*, 5(1):77–104, 1990.

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

[36] Edouard François and Patrick Bouthemy. Multiframe-based identification of mobile components of a scene with a moving camera. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 166–172, 1991.

[37] Donald Gennery. Tracking known three-dimensional objects. In *Proceedings of AAAI-82, Pittsburgh, PA*, pages 13–17, 1982.

[38] A. Gilbert, M. Giles, G. Flachs, R. Rogers, and Y. U. A real-time video tracking system. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2(1):47–56, January 1980.

[39] Frank Glazer. *Hierarchical Motion Detection*. PhD thesis, University of Massachusetts, Amherst, February 1987.

[40] A. Guzman. Decomposition of a visual scene into three-dimensional objects in a visual scene. In *AFIPS Fall Joint Conference*, volume 33, pages 291–304, 1968.

[41] Robert Haralick and Linda Shapiro. Image segmentation techniques. *Computer Vision, Graphics, and Image Processing*, 29:100–132, 1985.

[42] David Heeger. Optical flow using spatiotemporal filters. *International Journal of Computer Vision*, pages 279–302, 1988.

[43] Ellen Hildreth. *The Measurement of Visual Motion*. MIT Press, 1983.

[44] Daniel Hillis. *The Connection Machine*. The MIT Press, 1985.

[45] David Hogg. Model-based vision: a program to see a walking person. *Image and vision computing*, 1(1):5 – 20, February 1983.

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

[47] Berthold Horn and Brian Schunk. Determining optical flow. *Artificial Intelligence*, 17:185–203, 1981.

[48] Ian Horswill. Reactive navigation for mobile robots. Master's thesis, MIT, May 1988.

[49] Andres Huertas and Gerard Medioni. Detection of intensity changes with subpixel accuracy using laplacian-gaussian masks. *Transactions on Pattern Analysis and Machine Intelligence*, 8(5), September 1986.

[50] Ramesh Jain, W. Martin, and J. Aggarwal. Segmentation through the detection of changes due to motion. *Computer Graphics and Image Processing*, 11:13–34, 1979.

[51] Takeo Kanade. Region segmentation: Signal vs semantics. *Computer Graphics and Image Processing*, 13(4):279–297, 1980.

[52] Michael Kass. Computing visual correspondence. In Alex Pentland, editor, *From Pixels to Predicates*, pages 78–92. Ablex, 1986.

[53] Michael Kass, Andrew Witkin, and Dmitri Terzopolous. Snakes: Active contour models for machine vision. In *Proceedings of International Conference on Computer Vision*, pages 259–268. IEEE, 1987.

[54] Joseph Kearney, William Thompson, and Daniel Boley. Optical flow estimation: An error analysis of gradient-based methods with local optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 229–244, 1987.

[55] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1), 1986.

[56] John Leese, Charles Novak, and Ray Taylor. The determination of cloud pattern motions from geosynchronous satellite image data. *Pattern Recognition*, 2:279–292, 1970.

[57] Richard Legault. The aliasing problems in two-dimensional sampled imagery. In Lucien Biberman, editor, *Perception of Displayed Information*. Plenum Press, 1973.

[58] James Little, Heinrich Bulthoff, and Tomaso Poggio. Parallel optical flow using local voting. In *Proceedings of International Conference on Computer Vision*, pages 454–459, 1988.

[59] James Little and Alessandro Verri. Analysis of differential and matching methods for optical flow. In *IEEE Motion Workshop*, pages 173–180, 1989.

[60] 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.

[61] Bruce Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *DARPA Image Understanding Workshop*, pages 121–130, 1981.

[62] James Mahoney. Exhaustive hierarchical computations for labeling every pixel in a binary image with topology and geometry across scales. Technical report, Xerox PARC, 1990.

[63] Greil Marcus. *LIPSTICK TRACES: A Secret History of the Twentieth Century*. Harvard University Press, 1989.

[64] David Marr. *Vision*. W. H. Freeman and Company, New York, 1982.

[65] John McCarthy and Patrick Hayes. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence*, volume 4, pages 463–502. Edinburgh University Press, 1969.

[66] Hans Moravec. Towards automatic visual obstacle avoidance. In *Proceedings of IJCAI-77*, page 584, 1977.

[67] David Murray and Bernard Buxton. Scene segmentation from visual motion using global optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(2), 1987.

[68] Kathleen Mutch and William Thompson. Analysis of accretion and deletion at boundaries in dynamic scenes. In W. Richards, editor, *Natural Computation*, pages 44–54. The MIT Press, 1988.

[69] Hans-Hellmut Nagel. Displacement vectors derived from second-order intensity variations in image sequences. *Computer Vision, Graphics, and Image Processing*, 21:85–117, 1983.

[70] Randal Nelson. Qualitative detection of motion by a moving observer. Technical Report 341, Computer Science Department, University of Rochester, April 1990.

[71] Keith Nishihara. Prism: a practical real-time image stereo matcher. AIM 780, MIT, May 1984.

[72] Ron Ohlander, Keith Price, and Raj Reddy. Picture segmentation using a recursive region splitting method. *Computer Graphics and Image Processing*, 8(3):313–333, 1978.

[73] Masatoshi Okutomi and Takeo Kanade. A locally adaptive window for signal matching. In *Proceedings of International Conference on Computer Vision*, pages 190–199, 1990.

[74] Joseph O'Rourke and Norman Badler. Model-based image analysis of human motion using constraint propagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2(6):522–536, 1980.

[75] Shmuel Peleg and Hillel Rom. Motion based segmentation. In *International Conference on Pattern Recognition*, pages 109–113, 1990.

[76] Jerry Potter. Scene segmentation using motion information. *Computer Graphics and Image Processing*, 6:558–581, 1977.

[77] William Press, Brian Flannery, Saul Teukolsky, and William Vetterling. *Numerical Recipes in C*. Cambridge University Press, 1988.

[78] Zenon Pylyshyn and Ron Storm. Tracking multiple independent targets: Evidence for a parallel tracking mechanism. *Spatial Vision*, 3(3):179–197, 1988.

[79] Willard Van Orman Quine. Things and their place in theories. In *Theories and Things*. The Belknap Press, 1981.

[80] J. Roach and J. Aggarwal. Computer tracking of objects moving in space. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1(2):127–135, 1979.

[81] Azriel Rosenfeld and Avinash Kak. *Digital Picture Processing*. Academic Press, 2nd edition, 1981,1982.

[82] R. Schalkoff and E. McVey. A model and tracking algorithm for a class of video targets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 4(1), 1982.

[83] Akio Shio and Jack Sklansky. Segmentation of people in motion. In *Proceedings of IEEE Workshop on Visual Motion*, pages 325–332, 1991.

[84] Eero Simoncelli, Edward Adelson, and David Heeger. Probability distributions of optical flow. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 310–315, 1991.

[85] Ajit Singh. *Optic Flow Computation*. IEEE Computer Society Press, 1991.

[86] Eric Smith and Dennis Phillips. Automated cloud tracking using precisely aligned digital ATS pictures. *IEEE Transactions on Computers*, 21(7):715–729, July 1972.

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

[88] Michael Swain and Markus Stricker. Directions promising in active vision. CS 91–27, University of Chicago, November 1991.

[89] D. W. Thompson and J. L. Mundy. Motion-based motion analysis: Motion from motion. In Robert Bolles and Bernard Roth, editors, *Robotics Research: The Fourth International Symposium*, pages 299–309. MIT press, 1988.

[90] William Thompson. Combining motion and contrast for segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2(6), November 1980.

[91] William Thompson, Kathleen Mutch, and Valdis Berzins. Edge detection in optical flow fields. In *Proceedings of AAAI-82, Pittsburgh, PA*, pages 26–29, 1982.

[92] William Thompson, Kathleen Mutch, and Valdis Berzins. Dynamic occlusion analysis in optical flow fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7(4):374–383, 1985.

[93] William Thompson and Ting-Chuen Pong. Detecting moving objects. In *Proceedings of International Conference on Computer Vision*, pages 201–208, 1987.

[94] John Tsotsos. A 'complexity level' analysis of vision. In *Proceedings of International Conference on Computer Vision*, pages 346–355, 1987.

[95] Saburo Tsuji, Michiharu Osada, and Masahiko Yachida. Tracking and segmentation of moving objects in dynamic line images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2(6):516–522, 1980.

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

[97] Gilbert Verghese, Karey Gale, and Charles Dyer. Real-time, parallel motion tracking of three dimensional objects from spatiotemporal sequences. In Vipin

Kumar, editor, *Parallel Algorithms for Machine Intelligence and Vision*, pages 310–339. Springer-Verlag, 1990.

[98] Alessandro Verri and Tomaso Poggio. Against quantitative optical flow. In *Proceedings of International Conference on Computer Vision*, pages 171–180, 1987.

[99] Terry Winograd and Fernando Flores. *Understanding Computers and Cognition: A New Foundation for Design*. Addison-Wesley, 1986.

[100] 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.

[101] John Woodfill and Ramin Zabih. A real-time vision system for robots in unstructured domains. In *Proceedings SPIE conference, Sensor Fusion IV*, 1991.

[102] John Woodfill and Ramin Zabih. Using motion vision for a simple robotic task. In *AAAI Fall Symposium on Sensory Aspects of Robotic Intelligence*. AAAI, 1991.

[103] 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.