Real-Time Optical Flow

PhD Thesis

Ted Camus
Department of Computer Science
Brown University, Providence, RI 02912, USA.

September 24, 1994

Real-Time Optical Flow

by

Theodore Armand Camus

B.S., Rensselaer Polytechnic Institute, 1988

M.S., Brown University, 1991

Thesis

Submitted in Partial Fulfillment of the requirements for

the Degree of Doctor of Philosophy

in the Department of Computer Science at Brown University.

May 1995

© Copyright
by
Theodore Armand Camus
1994

Abstract

Currently two major limitations to applying vision in real tasks are robustness in real-world, uncontrolled environments, and the computational resources required for real-time operation. In particular, many current robotic visual motion detection algorithms (optical flow) are not suited for practical applications such as segmentation and structure-from-motion because they either require highly specialized hardware or up to several minutes on a scientific workstation. In addition, many such algorithms depend on the computation of first and in some cases higher numerical derivatives, which are notoriously sensitive to noise. In fact the current trend in optical flow research is to stress accuracy under ideal conditions and not to consider computational resource requirements or resistance to noise, which are essential for real-time robotics. As a result robotic vision researchers are frustrated by an inability to obtain reliable optical flow estimates in real-world conditions, and practical applications for optical flow algorithms remain scarce. Algorithms based on the correlation of image patches have been shown to be robust in practice but are in general infeasible due to their computational complexity. This thesis describes a space-time tradeoff to this algorithm which converts a quadratic-time algorithm into a linear-time one, as well as a method for dealing with the resulting problem of temporal aliasing. One limitation of this algorithm is that it does not give truly real-valued image velocity measurements. Therefore, it is not obvious that it can be used for a wide range of robotics vision tasks. One particular application for optical flow is time-to-contact: based on the equations for the expansion of the optical flow field it is possible to compute the number of frames remaining before contact with an observed object. Although the individual motion measurements of this algorithm are of limited precision, they can be *combined* in such a manner as to produce remarkably accurate time-to-contact measurements, which can be produced at real-time rates.

Vita

Theodore (Ted) Armand Camus was born in Lowell, MA on July 11, 1967. He graduated with honors with a B.S. in Computer Science from Rensselaer Polytechnic Institute in 1988. He received a Master's degree in Computer Science from Brown University in 1991. He will be a Research Associate in the Intelligent Systems Division of the National Institute of Standards and Technology (NIST) beginning November 1994. He is a member of the AAAI, IEEE, IEEE Computer, SPIE, and MVA/SME.

Acknowledgments

I am grateful for the guidance and support I have received from my advisors Heinrich Bülthoff and Tom Dean over the years. It may be a cliche, but without them this thesis could not have been done. Their faith in me in my early years, and the freedom they gave me, is beyond words of thanks. My third committee member, Ben Kimia, has taught me that there will always be more wonderful things to learn. Few people are able to have such a diverse committee, consisting of a Cognitive Scientist, a Computer Scientist, and an Engineer. Despite the risk of being a servant to too many masters, I hope I have pleased them all. I can never pay them back for what they've given me, so instead I will pay them forward by doing what I can for others.

Had I not met Andrew Duchon and Bill Warren, this thesis would have been very different. Thanks to them, I found an excellent application for my research. I wish them the best of luck with their own efforts, and I look forward to working with them in the future.

In my first years at Brown, I am thankful to have had Ken Bayse, Jak Kirman, Moises Lejter, and Solomon Shimony to look up to, and to answer the questions of a clueless first-year AI student. I'm glad that my first and only TA assignment was in Dan Lopresti's assembly language course; this started my love of hacking and fascination with computer technology. I am especially glad that I chose to finish my thesis at the Max-Plank Institute in Tübingen, Germany this summer. Although I was too busy finishing my thesis to do much of anything except work, I will always remember the warm people there. Above all,

I thank my parents, Dorothy and Armand Camus, for giving me all the love a child could want.

Jim Little first described the straightforward implementation of the constant-time box filter. Marian Nodine first suggested using temporal coherence to help solve the temporal aliasing problem. Susi Huber provided some synthetic image sequences during the course of this work. The translating tree sequence was taken at SRI and was taken from the 1991 IEEE Workshop on Visual Motion image database. The diverging tree images were produced by David Fleet at Toronto. All other image sequences in this paper were taken at the Artificial Intelligence Lab in the Computer Science Department at Brown University. Any errors are of course the author's own responsibility.

This work was supported in part by a National Science Foundation Presidential Young Investigator Award IRI-8957601 to Tom Dean, by the Air Force and the Advanced Research Projects Agency of the Department of Defense under Contract No. F30602-91-C-0041, and by the National Science foundation in conjunction with the Advanced Research Projects Agency of the Department of Defense under Contract No. IRI-8905436. Study at the Max-Planck-Institut für biologische Kybernetik at Tübingen during the summer of 1994 was supported by a grant from the Deutscher Akademischer Austauschdienst.

Contents

1 Introduction **1**

2 Optical Flow **6**

2.1 Gradient-Based Optical Flow 9

2.2 Velocity-Tuned Filter Optical Flow 14

2.3 Correlation-based Optical Flow 15

3 Linear-Time Optical Flow **21**

3.1 Time-Space Dimensional Reduction 21

3.2 Temporal Aperture Problem 26

3.3 Temporal Aliasing 27

3.4 Harmonic Search Intervals 29

3.5 Summary 33

4 Time-to-Contact **34**

4.1 Calculating Time-to-Contact 34

4.2 Selection of Velocities Detected 56

4.2.1 Imminent Collision results 56

4.2.2 Slow Sequence 57

4.2.3 Velocity-subset results 58

4.3 Robustness Versus Noise 61

4.4 Effects of Heuristic Temporal Antialiasing 69
4.5 Rotation examples 77
4.6 Summary 80

5 Real-Time Performance and Hardware Considerations 83

5.1 Box Filter Implementation 84
5.2 Personal Computers and Workstations 88
5.3 Parallel Implementations 93
5.4 Pyramid Architectures 97
5.5 Summary 102

6 Conclusions and Future Work 103

6.1 Timeline 112

A Robust M-estimation 114

Bibliography 118

List of Figures

2.1 Example of optical flow "needles". 7

2.2 The Aperture Problem. 11

2.3 Gradient-based Optical Flow 12

2.4 Correlation-based optical flow : search space 16

2.5 Correlation-based optical flow : patch matching 17

3.1 Correlation-based optical flow : geometry of search space 22

3.2 Linear-time optical flow : frame delay = 1 23

3.3 Linear-time optical flow : frame delay = 2 24

3.4 Linear-time optical flow : pixel aliasing 26

3.5 The Temporal Aperture Problem 27

3.6 Temporal Aliasing 28

3.7 Harmonic Search Intervals 30

3.8 Motion Parallax : geometry 31

3.9 Motion Parallax : example 32

4.1 Time-to-contact experiment 35

4.2 Time-to-contact optical geometry 36

4.3 Images from collision sequence 37

4.4 Optical flow of the collision sequence at frames 12 and 41 41

4.5 Optical flow of the collision sequence at frames 96 and 126 42

4.6 Optical flow of the collision sequence at frames 134 and 139 43
4.7 Weighted average of optical flow vectors 45
4.8 Histograms of optical flow vectors 46
4.9 Histograms of averaged motion measurements 48
4.10 Time-to-contact estimates : no lower-bound threshold 51
4.11 Time-to-contact estimates : with lower-bound threshold 52
4.12 Results of time-to-contact experiment 53
4.13 Results of imminent collision experiment 56
4.14 Results of slow collision experiment 57
4.15 Results of slow collision experiment : alternate parameters 58
4.16 Motion log-plots : without and with temporal antialiasing 59
4.17 Time-to-contact experiment : skipping 1/2 pixels/frame 60
4.18 Motion log-plots : skipping 1/2 pixels/frame 60
4.19 Time-to-contact experiment : skipping 1/3, 1/4 pixels/frame 61
4.20 Time-to-contact experiment : 6-bit, 5-bit, 4-bit, 2-bit precision 62
4.21 Images from collision sequence : 2-bit precision 64
4.22 Optical flow at frames 12 and 41 : 2-bit precision 65
4.23 Optical flow at frames 96 and 126 : 2-bit precision 66
4.24 Optical flow at frames 134 and 139 : 2-bit precision 67
4.25 Time-to-contact experiment : plus additive Gaussian noise 70
4.26 Time-to-contact experiment : plus additive random noise 71
4.27 Grey-coded map of temporal aliasing 73
4.28 Time-to-contact experiment : with and without heuristic antialiasing 74
4.29 Optical flow at frame 41 with and without temporal antialiasing 75
4.30 Motion log-plots with and without temporal antialiasing 76
4.31 Time-to-contact experiment with fastest-first, temporal continuity rules . . . 76

4.32 Relation of expansion and rotation 78

4.33 Conversion of expansion motion vector to rotation 79

4.34 Rotating the SRI tree sequence 80

4.35 Optical flow superimposed on rotated tree 82

5.1 Demonstration of a 3x3 box filter. 86

5.2 Direct updating of best motion without conditionals. 88

5.3 Magic numbers for various computers/CPU's 92

5.4 A pyramid representation of an image 98

5.5 Scales of a pyramid representation 99

5.6 An example demonstrating the limits of pyramids 100

6.1 Sum-of-squared-differences plot of best-matching optical flow 106

6.2 Boundary detection using depth from motion parallax 108

6.3 Timeline of linear-time optical flow. 113

A.1 Asymptotic Relative Efficiency for given error probabilities. 115

Chapter 1

Introduction

There are many motivations for developing autonomous mobile robots. Robots in general are desirable for replacing humans in many situations. Some tasks may be impossible for unaided humans, such as very heavy lifting or microscopic visual inspection. Some tasks may be immediately hazardous to human beings, such as fire fighting, handling toxic chemicals, operating in areas of high radiation, or combat. Tasks which are unhealthy in the long term are also suitable candidates for automation, such as performing repetitive motions or working with industrial chemicals. Tedious jobs such as assembly line quality control may be performed more accurately by an automated system. Despite these motivations, applications for robotic systems remain relatively scare. Some difficulties include restricted mobility of current systems, restricted sensing capability (including sensitivity to noise in the sensing process), and the limited computational power available for such systems.

A robot that has true mobility must be able to sense its surroundings and other objects in its environment. Using computer vision to solve robotics related tasks has many advantages over more traditional sensing modalities such as sonar, active infrared and structured light. First, given the proper optics, vision essentially has no distance limitation, possibly making use of wide-angle lenses at one extreme to telephoto lenses at the other. In contrast, a laser light striper's range might be limited to one or two meters. Second, the information available in an image is both spatially dense (a quarter-million pixels is standard) and potentially of

very high resolution (8 bits is standard, up to 12 bits is available). Third, it is a totally passive sensor, and need not be excluded from sensitive environments where the emission of sonic or infrared pulses might be intrusive. Finally, decades of scientific research on human vision and the human visual system is available to researchers in computer vision.

Despite these strong incentives to use vision as a powerful means of perception on mobile robots, and extensive mathematical analysis of computer vision, practical real-time robotic vision algorithms remain elusive, as recently noted by D. Touretzky et.al. ([TWR94]) :

> "But the real problem with video cameras is that image processing is computationally expensive. Even something as simple as calculating real-time optic flow requires more processing power than is practical for a mobile robot. Yet optic flow is known to be computed in the early stages of mammalian vision. Such observations underscore the tremendous gulf that remains between today's digital computers and real nervous systems."

It is undeniably true that biological organisms are able to perform such tasks as motion detection, and most evidence from the field of computer vision (even to the present day) would tend to confirm the above statement that it might not seem possible to be able to compute optical flow, or point-wise visual motion detection, in real-time using computing power that was practical for a mobile robot. Nonetheless, [C93] proposed the thesis that optical flow could indeed be be computed in real-time using computing power that is appropriate and practical for a mobile robot, that the optical flow thus produced would be robust, and that it would be sufficiently accurate to be used for certain robotic vision tasks. Here "real-time" is defined in the loose sense of being fast enough to be calculated on-line and be usable in some reactive system; on the order of 4-5 frames per second at a minimum. "Practical for a mobile robot" implies something on the order of a current low-end workstation or possibly a mid- to high-end personal computer. Unfortunately, in the

past this has proven very difficult, since such the calculation of optical flow is traditionally computationally intensive, very sensitive to noise, and often inaccurate.

In fact currently the two major limitations to applying vision in real tasks are robustness in real-world, uncontrolled environments, and the computational resources required for real-time operation. In particular, many current optical flow algorithms are not suited for practical applications such as segmentation and structure-from-motion because they either require highly specialized hardware or up to several minutes on a scientific workstation. For example, [WM93] quotes 4 minutes on a Sun workstation and 10 seconds on a 128-processor Thinking Machines CM5 supercomputer. If computer processing power continues to grow at approximately 54% per year [PH94], then such an algorithm would take approximately 16 years for the 1000-fold increase in performance necessary for real-time rates on an ordinary workstation. In addition, many such algorithms depend on the computation of first and in some cases higher numerical derivatives, which are notoriously sensitive to noise. In fact the current trend in optical flow research ([BFBB92], [WM93]) is to stress accuracy under ideal conditions (either no noise or at best, modeling the noise as Gaussian, a questionable assumption), and not to consider computational resource requirements or resistance to noise, which are essential for real-time robotics. As a result robotic vision researchers are frustrated by an inability to obtain reliable optical flow estimates in real-world conditions, and practical applications for optical flow algorithms remain scarce.

Algorithms based on the correlation of image patches (e.g. [BLP89a]) have been shown to be robust in practice but are in general infeasible due to their computational complexity. This thesis describes a space-time tradeoff to this algorithm which converts a quadratic-time algorithm into a linear-time one, as well as a heuristic for dealing with the resulting problem of temporal aliasing.

The success of this optical flow algorithm may be measureed by how well it fulfills the three basic requirements of robotic vision: such an algorithm should be robust, fast, and

accurate. If an algorithm is not robust, i.e. it cannot perform well in varied circumstances and in unpredictable environments, it is of little use for robotic vision regardless of how fast it runs or how well it can perform in certain limited and clearly specified conditions. For this reason, we avoid algorithms which depend on visually distinguishable markers or highly textured patterns, since it is not reasonable to expect that the environment will always be constrained in such a manner. Furthermore, such algorithm must be vigorously tested against very noisy and low quality, degraded images to ensure robustness against noise.

The second requirement for a practical robotic vision algorithm is that it be fast; clearly, any algorithm that takes several seconds or minutes cripples the usefulness of an autonomous mobile robot. In particular, such an algorithm should run in real-time when run on commonly available hardware, such as a low-end workstation. Although it may be possible to run a computer vision algorithm quickly on for example a massively parallel supercomputer, the practicality of such a solution for mobile robotics is questionable, due to the tremendous cost, weight, and power consumption of these devices.

Finally, robotic vision must be accurate, as appropriately defined for a given task. It is useful to remember that for some tasks, *qualitative* vision may be sufficient. Quantitatively accurate robotic vision is of course desirable, but under no circumstances may the first two criteria of robustness and real-time performance be sacrificed. If these first two criteria are satisfied however, accuracy may then be optimized.

This thesis has two main immediate contributions. A new, linear-time optical flow algorithm is developed which is very fast and robust (although not necessarily very precise), and has many uses in its current form. Second, an algorithm for calculating quantitatively accurate time-to-contact is developed, without sacrificing computational efficiency or robustness.

From a broader perspective, real-time optical flow is now available to virtually anyone with a camera and low-end workstation. Furthermore, it is argued that this algorithm is

very likely to always remain significantly faster than other optical flow algorithms (at least those that do not employ special hardware) on standard, commonly available computing hardware. This algorithm has demonstrated remarkably accurate quantitative results for at least some applications, and there is every reason to believe that such results could be extended to further applications using a similar methodology. Perhaps the most important lesson learned is that quantitative computer vision can be possible (and practical) without complicated models or extremely accurate low-level measurements.

Chapter 2

Optical Flow

The primary difficulty with computer vision is that a single 2-dimensional image can arise from effectively an infinite number of 3-dimensional scenes. A characteristic of image formation, one that must be well understood in order to attempt to understand the scene giving rise to a particular image, is the unavoidable loss of information in the imaging process. A single pixel is represented by a single intensity value, which itself it the result of many factors, such as the intensity, color, and location of the light source or sources, and the orientation, reflectance, and transparency of the objects in the scene, as well as the optical and electrical properties of the imaging device itself. Each of these effects may be understood individually, but they are mutually confounding; a single 2-dimensional image can correspond to an infinite number of 3-dimensional scenes.

Motion detection is an essential means of perception for an organism for determining the organism's own movement, the structure of the organism's environment, as well the motions of other organism's in its environment. Movement in images can be divided into *real* motion across the retina, and *apparent* motion, the perception of motion that arises when an object or pattern appears to instantaneously move from one place to another [RA86]. In the case of the latter, the *correspondence* problem must be solved [U79], that of matching points or features in one image with that of another. If motion across the retina is *real*, then it may be possible to detect motion using the spatial and temporal derivatives in an image, however,

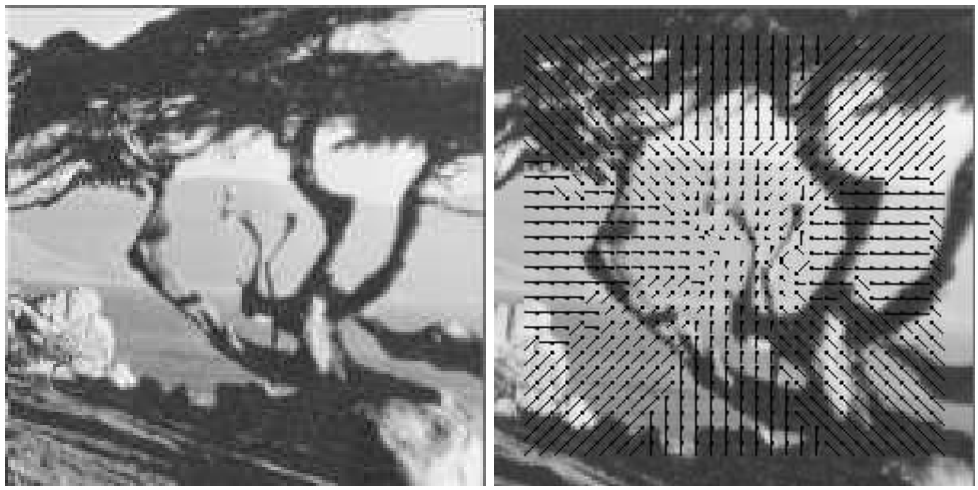


Figure 2.1: Optical flow "needles" indicate direction and magnitude of motion of pixels from the left image to the right image.

if the motion (or time delay) between successive frames is sufficiently small, then *apparent* motion begins to approximate *real* motion, thus it can be argued that studying the former is all that is necessary.

An object moving in 3-D space has a three dimensional velocity vector field $W(x, y, z)$. This 3-D motion field is projected onto the retina of an observer as the 2-D motion field $W_p(x, y)$. Unfortunately, this 2-D motion field may not be perceived directly since it is a purely geometrical concept; all that may be detected is some local measure of incident light at each point, $E(x, y)$. In a typical machine vision system, two images are taken by a camera separated by a discrete time interval Δt. What may be observed at each point is the changes in a point's intensity value $\Delta E(x, y)$ during this time interval. The optical flow is a vector field describing this intensity change by indicating the motion of features from one image to the other. An example of optical flow is shown in Figure 2.1.

By accurately computing this 2-D vector field, it is in principle possible to calculate three-dimensional properties of the environment, and quantities such as time-to-contact with an an observed object [Lee76]. Biological organisms make considerable use of optical flow, such as the detection of discontinuities [PRH81], and figure-ground discrimination

[B81]. Motion detection is useful for autonomous mobile robot navigation [PBF89] as well as undersea navigation [NSY91]. [K86], [KvD78] discusses the decomposition of optical flow into curl, divergence and shear for the purposes of determining local shape of an object, and [Reg86] presents evidence that the human visual system possesses specific detectors for similar types of basic motion. [TR93] describes an application for image coding. [MB90] covers experiments in segmentation, structure from motion, tracking and qualitative shape analysis. [NA89] discusses obstacle avoidance using only qualitative optical flow.

In general the optical flow will not be the same as the true 2-D projection of the 3-D motion field [VP87]. For example, a rotating, perfectly featureless sphere will not induce any optical flow, however the 2-D projection of its motion field is non-zero everywhere on the sphere except at the occluding boundaries. Conversely, if the sphere is stationary but a light source moves, the changes in shading will induce an optical flow field even though the motion field is zero everywhere ([Horn86]). However, for a sufficiently textured surface, the optical flow field will be arbitrarily close to the motion field.

Many techniques for optical flow exist (see [BFBB92] and [LV89] for reviews and discussions of several techniques). Although these techniques can perform very well for certain sequences of images, there are very few that are currently able to support real-time performance. Authors rarely report the computational time needed for their algorithms; when they do, it is on the order of many minutes per frame ([WM93]), or require specialized hardware such as a Connection Machine ([BLP89a], [DN93], [SU87], [WZ91], [L88]), Datacube ([LK93], [N91]), custom image processors [DW93], or PIPE [KSL85], [WWB88], [ATYM]. Techniques which can run in real-time often impose strict restrictions on the environment; [HB88] presents a technique that can segment in real-time for tracking purposes, but requires that a textured object be moving in front of a relative textureless background. We desire a technique that computes a more general-purpose optical flow field, in real-time on a standard desktop workstation, without imposing strict restrictions on the input.

Techniques for optical flow can be divided into *gradient-based*, *velocity-tuned filter based*, and correspondence-based motion, each described in turn.

2.1 Gradient-Based Optical Flow

A common technique for computing optical flow is to assume that the total spatial and temporal derivatives of the image brightness remain constant. For small motions, constant ambient illumination, these assumptions are more or less true except for pathological situations such as occluding boundaries. While occluding boundaries is an important application for optical flow, for the moment we will only consider the general case.

From [HS81], assume that the brightness of a given point in an image is constant :

$$\frac{dE}{dt} = 0. \tag{2.1}$$

And we wish to find the velocity $v = (u, w) = (\frac{dx}{dt}, \frac{dy}{dt})$. After expanding equation 2.1 we have :

$$\frac{\partial E}{\partial x}u + \frac{\partial E}{\partial y}w + \frac{\partial E}{\partial t} = 0. \tag{2.2}$$

(Note that equation 2.2 does not contain second and higher order terms; these vanish as $\delta t \to 0$. Since δt is often a significant fraction of a second, it is questionable to ignore these terms, thus yielding second-derivative methods such as [UGVT88], which we will mention later. For now we will ignore this issue.)

The spatial derivatives $\frac{\partial E}{\partial x}$ and $\frac{\partial E}{\partial y}$ and the temporal derivative at an image point $\frac{\partial E}{\partial t}$ can be estimated using two or more images by using for example finite or central difference methods [LV89]. This leaves two unknowns u and w, motion along the X and Y axes respectively, with only one constraint, equation 2.2. Only the motion along the direction of the gradient $(\frac{\partial E}{\partial x}, \frac{\partial E}{\partial y})$ is available. This is known as the *aperture problem* [MU81], [NS88] and is illustrated in Figure 2.2. The left of Figure 2.2 shows an instance of the *strong*

aperture problem [BLP89b]. If intensity variations are one-dimensional (i.e. no curvature of the intensity isocountours), and no endpoints or distinguishing markers are visible (as if the contour were viewed through a small aperture), then the motion of a point on a line is ambiguous; point A could have moved to point A or point A' for example. Under these conditions (when intensity variation is one-dimensional) the problem as stated is *ill-posed* without further assumptions and the correct motion cannot be determined. When dealing with spatial gradients, the aperture problem manifests itself in the fact that first derivatives are only a planar approximation to the curvature at a given point. Generally however natural image sequences do have sufficient intensity variations. The right of Figure 2.2 shows a translating curved contour. Here, the motion of point A may be disambiguated by comparing the motion of nearby points as well; points B and C match the correct motion of point A, but that of points D and E do not. If there are two-dimensional intensity variations, then the aperture problem is known as the *weak aperture problem*. However, this particular solution requires a non-local mechanism, and in this case an assumption of local rigidity, which is not necessarily true in general, but may be approximately true for most cases.

Given that nearby points on moving objects are likely to have similar three-dimensional velocities, their two-dimensional velocity projections should also have similar velocities. The problem can then be formulated as minimizing the sum of the errors in the equation for the rate of change of image brightness and the deviation from smoothness in the optical flow [HS81]. The optical flow field is then determined by minimizing a cost functional L ([HS81], [KWM89]):

$$L(\dot{x},\dot{y}) = \int\int \{[E_x\dot{x} + E_y\dot{y} + E_t]^2 + \lambda[(\frac{\partial\dot{x}}{\partial x})^2 + (\frac{\partial\dot{x}}{\partial y})^2 + (\frac{\partial\dot{y}}{\partial x})^2 + (\frac{\partial\dot{y}}{\partial y})^2]\}dx\,dy$$

[KWM89] proposes implementing such a smoothness constraint in a neural network and relates it to the primate's visual system.

In contrast to finding dense flow using a smoothness constraint, [Hil84] proposes calcu-

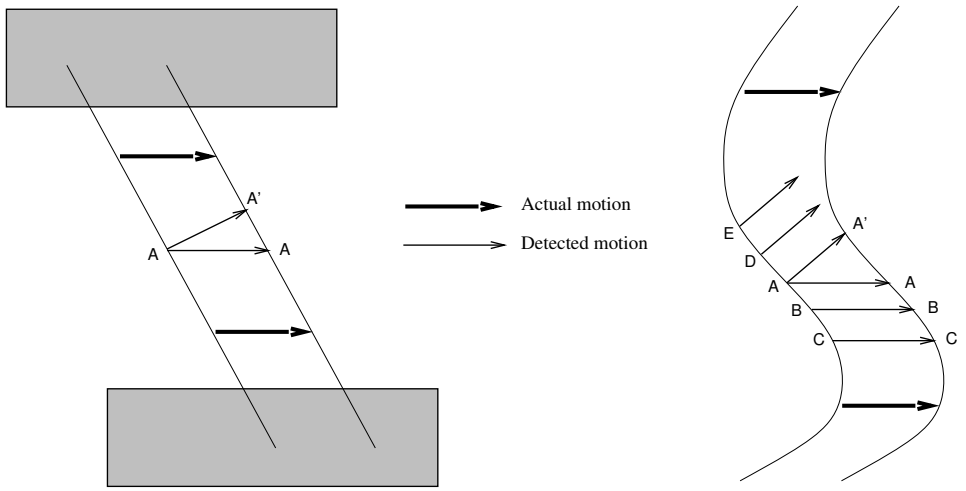


Figure 2.2: The *strong aperture problem* is illustrated on the left; only motion *normal* to a translating straight contour may be determined. If there are curvature or intensity variations, then we have the *weak aperture problem*, which can be satisfactorily solved if certain assumptions are enforced.

lating optical flow at points along the contours in the image. Here contours are extracted from the image using Marr and Hildreth's edge detector [MH80] and required that the velocity field be smooth only along a contour and not across it. This method does not blur the optical flow field at discontinuities, but only gives sparse measurements (only along the contours) and may confuse object boundary contours with other kinds of edges, such as those due to changes in shape or reflectance.

[SAH91] represents the problem probabilistically to account for the confounding effects of image noise, low contrast regions, multiple motions, lighting changes. A two-dimensional probability distribution is used that reflects directional uncertainty in the motion estimates. Areas of high contrast are treated as more reliable than areas of low contrast, subject to an asymptotic maximum, rather than effectively normalizing contrast when combining velocity information of a given point with that of nearby points.

A description of a general gradient-based method for a one-dimensional image is seen in Figure 2.3. Here a one-dimensional image intensity profile is shown in the vicinity of a given point p at coordinate (x). At time t_0 the image intensity at point p is E_0, and at time

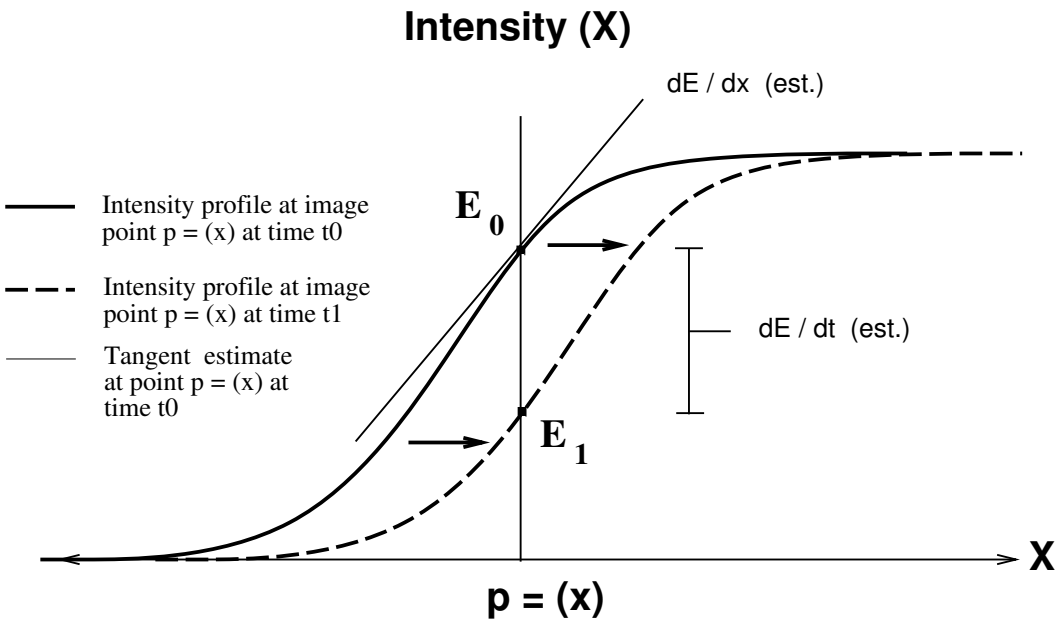


Figure 2.3: Gradient-based Optical Flow

t_1 at point p it is E_1. An estimate $\frac{\delta E}{\delta t}$ of the temporal derivative $\frac{\partial E}{\partial t}$ can be provided using a forward-difference method : $\frac{\delta E}{\delta t} = \frac{E_1 - E_0}{t_1 - t_0}$. The spatial derivative $\frac{\partial E}{\partial x}$ can be similarly estimated using the pixels adjacent to pixel p.

[DN93] calculates gradient-based optical flow using the multipoint technique for solving partial differential equations. Here the least-squares solution for equation 2.2 is solved in a local NxN neighborhood, under the assumption that local velocity is locally constant. This can run at several frames per second on a Connection-Machine 2 (depending on the size of the neighborhood N); see also [DNS92].

[BJ94] minimizes an objective function consisting of three terms : one, the standard optical flow intensity constraint equation; two, the differences with the optical flow in neighboring pixels; and three, the difference with a parametric planar patch estimate of the optical flow in regions of approximately constant intensity. Robust error norms are used to reject outliers in each error term.

[WWB88] calculates the motion of edges by differentiating their Gaussian-convolved

spatio-temporal activation profiles with the PIPE pipelined image processor at 15 frames per second. This method is very fast and can be robust, but only gives normal flow at edges, not the true flow.

One solution to the aperture problem of equation 2.2 is to assume that the total temporal derivative of the spatial gradient is zero :

$$\frac{d\nabla E}{dt} = 0$$

Expanding this equation leads to two constraints :

$$\frac{\partial^2 E}{\partial x^2} v_1 + \frac{\partial^2 E}{\partial x \partial y} v_2 + \frac{\partial^2 E}{\partial x \partial_t} = \frac{\partial^2 E}{\partial y \partial x} v_1 + \frac{\partial^2 E}{\partial y^2} v_2 + \frac{\partial^2 E}{\partial y \partial_t} = 0$$

This approach is taken in [Nag87], [UGVT88], and assumes that the displacements being measured are less than one half of a cycle of the highest spatial frequencies in the image, otherwise there will be aliasing. Although this approach can in theory give very precise measurements, one major problem is that numerical differentiation of this sort is very sensitive to noise in the input (for example cf. [KMB+91]). This is particularly true if the spatial derivative is small (i.e. the slope is flat) at the point in question, in which case a constant amount of noise has a greater detrimental effect on the numerical differentiation. Methods based on second or higher derivatives are potentially even more susceptible to noise as the problems associated with numerical differentiation are even worse. [To92a] calculates velocity at areas of sufficient contrast in the image using the technique of [UGVT88] at about 2-3 frames per second on a Sun 4/330. For his tracking application, dense measurements are not needed, nor is it necessary for highly accurate measurements. Any less than a full 8 bits of information per pixel may make second-derivative techniques unsuitable however [To92b]. In practice for robustness, [UGVT88] implements two variants of regularization techniques to create a smooth optical flow field. Smoothing the image can help attenuate the effects of noise, however smoothing the basic optical flow measurements by imposing a smoothness constraint or regularization such as in [HS81] can smooth over discontinuities in the motion

field and introduce inaccuracies, as well as increase the computational cost. One option for improving performance for these methods is to only calculate optical flow where the spatial derivatives are large (i.e. the gradient is steep), such as is discussed in [BFBB93], however this can result in a sparse motion field which may be inadequate for some problems. It is desirable to find a method of calculating optical flow that is less sensitive to noise in the imaging process, gives a dense output, and is computationally efficient.

2.2 Velocity-Tuned Filter Optical Flow

One class of optical flow techniques represents motion in terms of spatial and temporal frequencies. From [Heg87], two-dimensional patterns translating in the image plane occupy a plane in the spatio-temporal frequency domain :

$$\omega_t = u\omega_x + v\omega_y$$

where ω_t, ω_x, ω_y are the temporal and spatial frequencies of the motion respectively. This approach assumes that several points in spatio-temporal space can be detected in the moving surface. If, however, the aperture problem exists, then the points in spatiotemporal space occupy a line, and the plane is not fully determined. To solve this problem, [WA85] combines ten oriented sensors into opponent-motion pairs, discarding the smaller response of the pair. In theory, the output of any two remaining sensors (within 90 degrees of the actual motion) is sufficient to determine the actual motion, since that would provide two linearly independent components of the velocity vector. [SS85] describes *elaborated Reichardt detectors*, in which the point receptive fields of the original Reichardt detectors are replaced with spatial filters, and stresses the need for effective voting rules in differentiating between different sensors' outputs. [AB85] describes the construction of phase-independent separable spatiotemporally oriented opponent-motion filters, and suggests using the ratios between multiple sensors for contrast-invariance; although higher contrast will increase the *absolute*

outputs of the motion sensors, the *ratios* of their outputs should remain constant. [AB85] also shows the relationship between energy models and Reichardt detectors.

[WM93] first convolves the image with a set of linear, separable spatiotemporal filters and applies the brightness constancy equation (equation 2.1) to each. The resulting over-determined set of equations is then solved using a *total least squares* or *orthogonal regression* technique.

[Fl92] describes a phase-based method which describes component velocity in terms of the motion of level phase contours in the output of band-pass velocity-tuned filters. This technique is essentially a differential method applied to phase rather than intensity [BFBB92]. The full 2-dimensional velocity field is determined by fitting a linear velocity field to the component velocities. Although these methods can produce accurate results ([BFBB93], [WM93]), methods based on velocity-tuned or spatiotemporal filters can be extremely computationally intensive, requiring up to several minutes or even hours on a scientific workstation. Thus without some powerful special-purpose hardware, they are generally not suitable for real-time robotic vision.

2.3 Correlation-based Optical Flow

In general it is not possible to determine the correct optical flow field given a pair of successive image frames, due to the aperture problem. If certain assumptions are enforced, however, then the problem becomes well-posed, and can be satisfactorily solved in most cases. A relatively noise-resistant method of calculating optical flow is described in [BLP89a], [BLP89b] and summarized here. We will assume that the maximum possible displacement for any pixel is limited to η in any direction. The actual value of η depends on the expected velocities of the pixels in the image plane. This is shown in Figure 2.4.

Since we are generally concerned with the flow of rigid-bodied objects, it is usually the case that any given pixel has the same velocity as those of its neighbors. We assume that

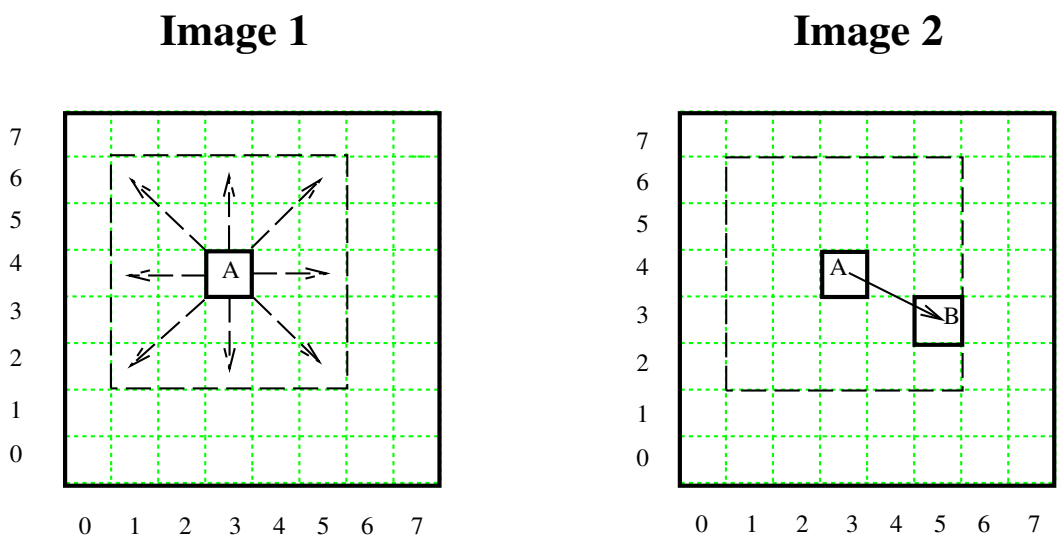


Figure 2.4: In image 1, the search space for Pixel A for $\eta = 2$ is shown. The correct motion from image 1 to image 2 can be displayed by the vector AB.

these pixels are in a square neighborhood, or window, of size ν centered around the given pixel. That is, it is assumed that the motion vectors for pixels adjacent to a given pixel will be similar, as shown in Figure 2.5. All examples in this paper use a value $\nu = 7$.

The motion for the pixel at [x,y] is defined to be the determined motion of the patch of $\nu * \nu$ pixels centered at [x,y], out of $(2\eta+1)*(2\eta+1)$ possible displacements. We determine the correct motion of the patch of pixels by simulating the motion of the patch for each possible displacement of [x,y] and considering a match strength for each displacement. If ϕ represents a matching function which returns a value proportional to the match of two given features, then the match strength M(x,y;u,w) for a point [x,y] and displacement (u,w) is calculated by taking the sum of the match values between each pixel in the displaced patch P_ν in the first image and the corresponding pixel in the actual patch in the second image :

$$\forall u, w : M(x, y; u, w) = \sum \phi(E_1(i, j) - E_2(i + u, j + w)), (i, j) \epsilon P_\nu \qquad (2.3)$$

A possible function for ϕ is the absolute difference between the two pixels' intensity values; another is the squared difference of their respective intensity values. In these cases

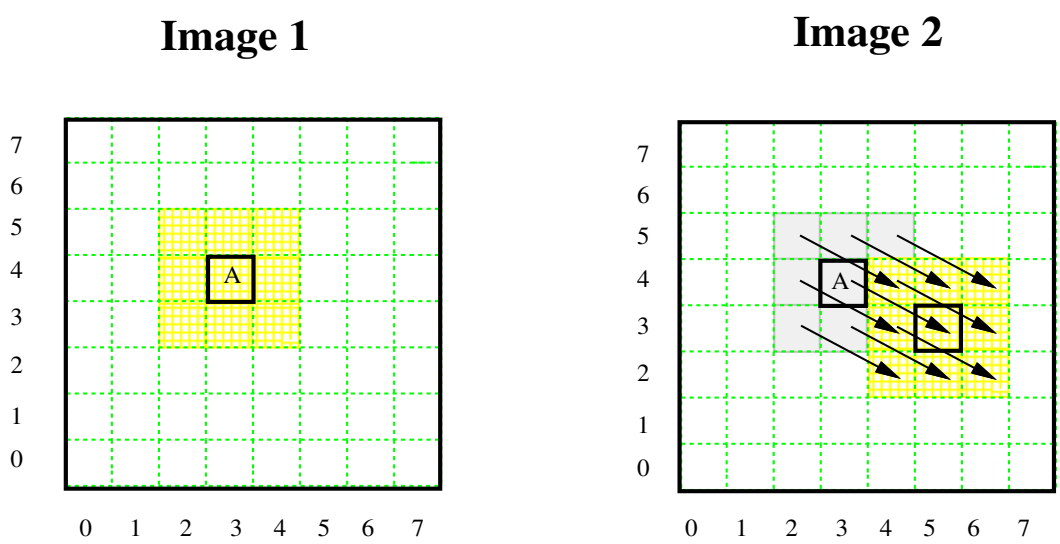


Figure 2.5: Optical flow for pixels adjacent to a given pixel is assumed constant due to the rigid-body assumption. In this figure, $\nu = 3$.

a lower value indicates a better match. Moravec [Mor77] uses a variant of normalized cross correlation in stereo matching. Although this has the advantage of insensitivity to contrast, the sum of absolute differences measure used in most examples in this thesis was found to be very insensitive to the image contrast. In addition, experiments using normalized cross-correlation did not produce good results.

In an ideal situation, where we have only fronto-parallel motion, no noise, constant ambient illumination, and a translation of an integer number of pixels (within our given range) we would expect the pixels of the two patches to match perfectly. The actual motion of the pixel is taken to be that of the particular displacement, out of $(2\eta+1)*(2\eta+1)$ possible displacements, with the maximum neighborhood match strength (equivalently minimum patch difference); thus this is called a "winner-take-all" algorithm. This is repeated for each pixel in the image, independently of the other pixels, with the exception that motion is not computed along a $\eta + \lfloor \nu/2 \rfloor$ border at the edges of the image since the matching operator would have to access points that are not available in the image itself. With a large neighborhood, ties are relatively rare and are decided arbitrarily.

This algorithm has many desirable properties. Due to the two-dimensional scope of the matching window, this algorithm generally does not suffer from the aperture problem except in extreme cases [BLP89b], and tends to be very resistant to random noise. In fact the algorithm's "winner-take-all" nature does not even require that the calculated match strengths have any relation whatsoever to what their values should theoretically be, it is only necessary that their relative ordering remains correct. For example, a change in illumination between frames would certainly affect the individual match strengths, but need not change the best matching pixel shift. Conversely, any noise in a gradient-based method usually directly results in errors in the basic optical flow measurements. In the case of a change in illumination, the image intensity constraint equation does not apply since total image intensity does not remain constant. In addition, since the patch of a given pixel largely overlaps with that of an adjacent pixel, match strengths for all displacements for adjacent pixels tend to be similar, and so the resultant optical flow field tends to be relatively smooth, without requiring an additional smoothing step. Finally, since one optical flow vector is produced for each pixel of input (excepting the small $\eta + \lfloor \nu/2 \rfloor$ border), optical flow measurement density is 100 percent. For these reasons, the basic correlation-based algorithm tends to be robust in practice, thus satisfying the first criteria for practical robotic vision.

[A87a] and [G87] prefilter the images with band-pass filters and employ a coarse-to-fine matching strategy in which initial coarse estimates at one spatial scale are fed into the next higher resolution level. Although sub-pixel motions can be calculated by finding the minima of a sum-of-squared-difference with a quadratic approximation to the intensity surface, this usually requires an iterative procedure which increases computation time. Furthermore, although matching methods such as these can do well with translating images, they often have difficulty with diverging images [BFBB93], such as will be analyzed later in this thesis (Chapter 4). Pyramid structures are discussed further in Section 5.4.

[AP93] computes optical flow only along a single dimension; often, the vector sum of two one-dimensional optical flow vectors is sufficiently *qualitatively* similar to the two-dimensional optical flow to be used in certain robotic vision tasks. Using Green's theorems, the divergence of the optical flow field can be estimated by intergrating the optical flow normal to a closed contour. By using the estimated divergence of the flow field, time-to-contact is calculated with errors reported on the order of 10% [AP93]. The simplicity of this approach makes it a plausible candidate for biological implementations.

[Hub94a] uses the correlation of 2-dimensional image patches along a single dimension ([AP93], [BLP89a]) to simulate the navigation of a fly in a synthetic maze using optical flow, at a rate of about one frame per second using 512x512 images [Hub94b]. A genetic algorithm is used to train the artificial fly's optomotor response to turbulent air conditions to prevent collisions with the maze's walls.

The independence of one pixel's chosen displacement from all other pixels' displacements motivates massive parallel implementations such as the one implemented on the connection machine [BLP89a]. But even these massively parallel implementations are only "close-to-real-time", on the order of seconds per frame. Here we define "real-time" in the loose sense of being fast enough to be calculated on-line and be usable in some reactive system; on the order of 4-5 frames per second at a minimum.

It is possible to perform the bulk of the computations in customized silicon chips; one such (partial) implementation has been done in CMOS [C91]. [DW93] calculates structure-from-motion using the same basic shift-match-winner-take-all algorithm implemented on the Image Understanding Architecture simulator; they report an estimated 0.54 seconds per frame using a maximum possible displacement $\eta = 20$. [LK93] calculates optical flow (for the purposes of tracking) within a limited radius (+/- 2 pixels vertically, +/- 3 pixels horizontally) using a 7x7 correlation window at 10 Hz on 128x120 images using the Datacube MaxVideo 200.

Computational issues are discussed in detail in Chapter 5. Certainly, without the luxury of custom image processors, other techniques must be used for practical operation with conventional serial computers due to the search-based nature of the optical flow algorithm itself, which is described next.

Chapter 3

Linear-Time Optical Flow

Since correlation-based optical flow algorithms (e.g. [BLP89a]) satisfy the first criteria for practical robotic vision, that of robustness, we will use it as a starting point. The second criteria for practical robotic vision is computational efficiency. This chapter will analyze the computational complexity of the traditional correlation-based optical flow algorithm and develop a space-time tradeoff to create a very fast, linear-time optical flow algorithm.

3.1 Time-Space Dimensional Reduction

One limitation with the traditional correlation-based algorithm described in Section 2.3 is that its time complexity grows quadratically with the maximum possible displacement allowed for the pixel; see the left of Figure 3.1. Intuitively, as the speed of the object being tracked doubles, the time taken to search for its motion quadruples, because the area over which we have to search is equal to a circle centered at the pixel with a radius equal to the maximum speed we wish to detect.

However, note the simple relationship between velocity, distance and time :

$$velocity = \frac{\Delta distance}{\Delta time}.$$

Normally, in order to search for variable velocities, we keep the inter-frame delay δt constant and search over variable distances (pixel shifts) :

$$\Delta v = \frac{\Delta d}{\delta t}.$$

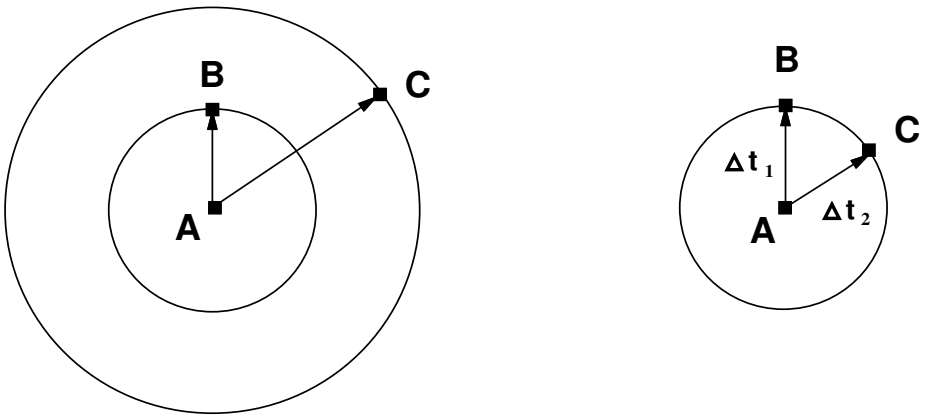


Figure 3.1: As the maximum pixel shift increases linearly, the search area increases quadratically. However with a constant shift distance and variable discrete time delays, search over time is linear.

However, we can easily see from Figure 3.1 that doing so results in an algorithm that is quadratic in the range of velocities present. Alternatively, we can keep the shift distance δd constant and search over variable time delays :

$$\Delta v = \frac{\delta d}{\Delta t}. \tag{3.1}$$

In this case, we generally prefer to keep δd as small as possible in order to avoid the quadratic increase in search area. Thus, in all examples δd is fixed to be a single pixel. (Note however, there is nothing preventing an algorithm based on both variable Δd and Δt's). Since the frame rate is generally constant, we implement "variable time delays" by integral multiples of a single frame delay. Thus, we search for a fixed pixel shift distance $\delta d = 1$ pixel over variable integral frame delays of $\Delta t \in \{1, 2, 3, ...S\}$. S is the maximum time delay allowed and results in the slowest motion calculated, $1/S$ pixel/frames. For example, a $1/k$ pixel/frames motion is checked by searching for a 1-pixel motion between the current frame t and frame $t - k$. Thus our pixel-shift search space is fixed in the 2-D space of the current image, but has been extended linearly in time. As before, the chosen motion for a given pixel is that motion which yields the best match value of all possible shifts.

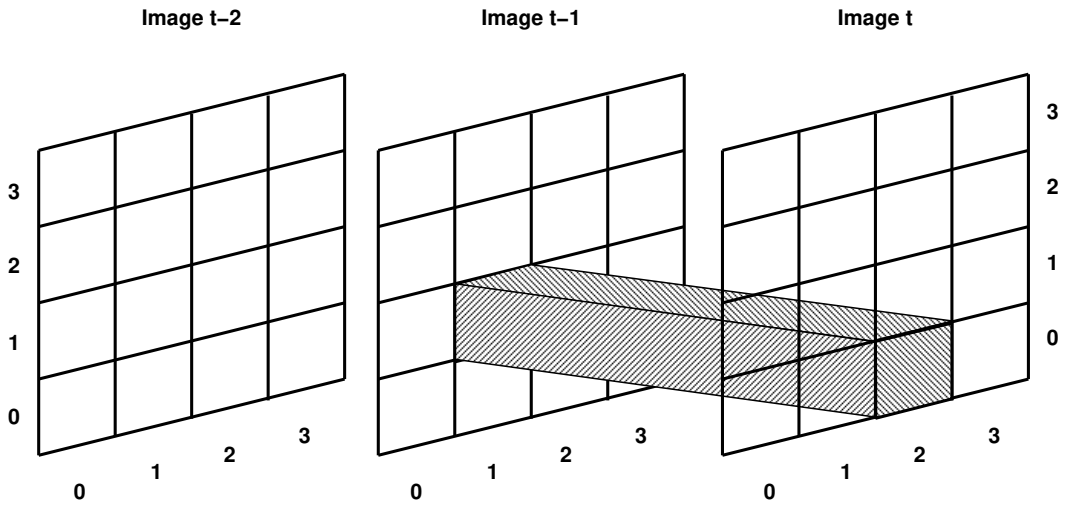


Figure 3.2: Visualization of motion from image $T-1$: (1,1) to image T: (2,0). This would be an optical flow of (1,-1) pixel/frames motion for pixel image T: (1,1).

For example consider Figure 3.2 and Figure 3.3. Here we are trying to calculate the optical flow for pixel (1,1) at the current frame, image T. In Figure 3.2 the optimal optical flow for pixel (1,1) from image $T-1$ to image T is calculated to be a pixel shift of (1,-1). This is only a temporally local measurement however; it may not be the final chosen motion. In Figure 3.3 the same search is performed, except using image $T-2$ as the first image. In this case the calculated motion happens to be a pixel shift of (0,1) pixels over 2 frames, equivalently (0,1/2) pixel/frames motion. If the maximum time delay $S=2$, then the procedure would stop here, otherwise we would continue processing frames until finally the optical flow from image $T-S$ to image T was calculated as well. The best of all these motions is taken to be the actual motion.

For a given velocity of $1/\delta t$ pixel/frames, we assume that motion is constant for t frames in order to register a cumulative motion of one pixel. Failure of this assumption can result in temporal aliasing, discussed in more detail later. [P90] calculates the normal velocity of edges using temporal-delay sensors which are summed in a later stage to determine actual motion. The images are first smoothed with a Gaussian to reduce the chances of aliasing due to multiple edges appearing at a single sensor.

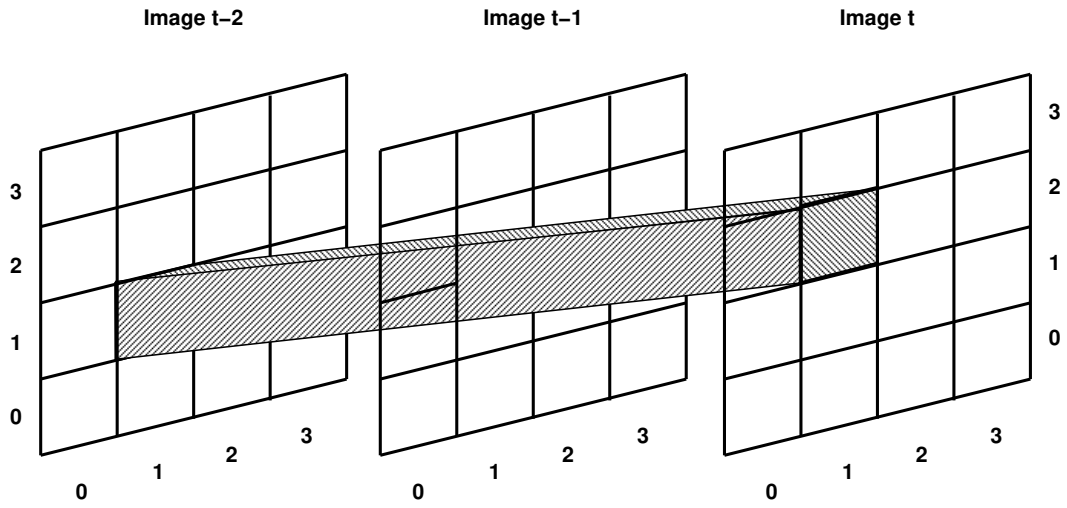


Figure 3.3: Visualization of motion from image $T-2$: (1,1) to image T: (1,2). This would be an optical flow of (0,1/2) pixel/frames motion for pixel image T: (1,1).

This time-space tradeoff reduces a quadratic search in space into a linear one in time, resulting in a very fast algorithm for computing optical flow. This partially satisfies the second criteria for practical robotic vision, that of real-time performance. Performance issues, particularly hardware considerations, are discussed in detail in Chapter 5.

One factor in producing real-time performance is that the original images are generally subsampled from 256x256 pixels to 64x64 pixels in size. Ideally, some optimal filtering procedure would be used to produce these reduced-resolution images, however this is generally not possible in a real-world situation. This is because the original images must be subsampled on the framebuffer itself in order to reduce the framebuffer to host computer memory bandwidth requirements (for example, the author's own system can support a maximum continuous image rate of about 12 64x64 images per second). The method of subsampling is not known, but due to limited hardware on the framebuffer itself it is likely to be a simple technique such as a box filter which performs an unweighted average of the pixels within the filter area, i.e., most likely a 256x256 to 64x64 reduction is performed by an unweighted averaging of each subsequent 16x16 block of pixels. Since this system is over five years old, normal advances in framebuffer technology would be expected to greatly increase this rate.

For the moment however, we will simply have to deal with this loss in resolution. For this reason as well, all images in this paper that are originally greater than 64x64 in resolution are subsampled to 64x64 using a box filter, since it is likely the technique used by simple framebuffers (to use a more sophisticated technique would give an unrealistic advantage to synthetic and high-resolution images).

When viewing such a subsampled image sequence, it is clear that considerable aliasing is occuring (see [FvD+90] for details on the problem of aliasing). In fact the aliasing is so bad one wonders how the algorithm can work at all. Figure 3.4 provides an example of this problem. On the left, a dark pixel moves between two CCD sensors. In particular, this dark pixel is split between both sensors during frame 1. This split cannot be represented in an actual image of the given resolution; however, the CCD sensor must provide a unique intensity for each pixel location. Thus in the actual image the pixel's intensity is "split" between both pixel locations. An image sequence $\{0, 1, 2\}$ would result in a flickering effect as the dark pixel passed between the two positions. For an algorithm measuring spatial derivatives, this could result in inaccurate measurements. The linear-time correlation algorithm, however, does not measure these types of spatial derivatives. Instead, it performs pixel matching across space and time. In this example, a perfect pixel match can be made between frame 0 and frame 2 (a motion of 1/2 pixel/frames), completely avoiding the difficulties associated with frame 1.

This algorithm has been successively used on many real and synthetic image sequences for a variety of real-time robotic vision tasks ([CB91], [C93], [C94b]; see also Section 6.1). [Du94a] reports being able to use this optical flow algorithm to instantiate some fly-like control laws ([War88]) in a small mobile robot. With the camera mounted on top of the robot, the image information from the camera is fed through the optical flow routine and the motion information is then used to directly control the robot. With a frame rate of 4-5 frames per second and the robot moving at 4-5 cm per second, the robot is able to

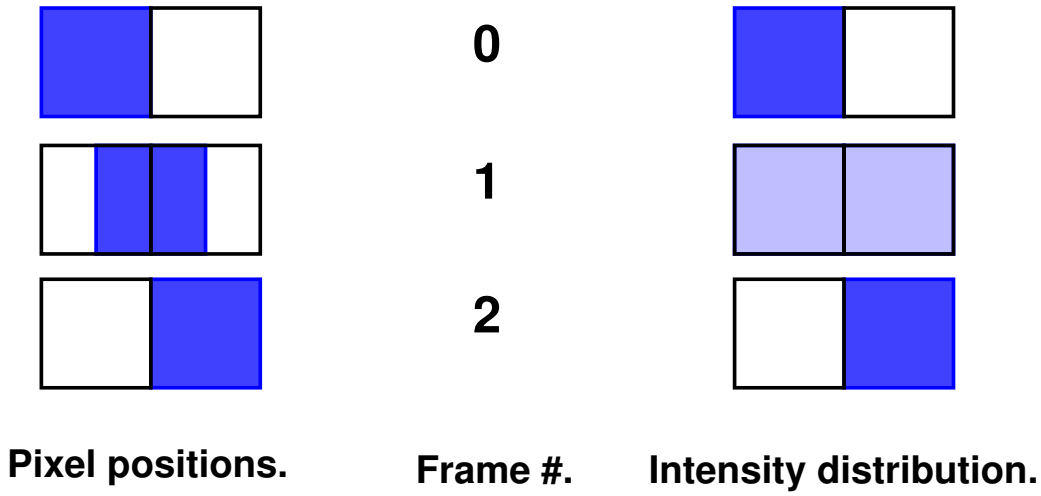


Figure 3.4: A dark square moves across a light background. As the square crosses a pixel boundary, its intensity is "split" between the two adjacent pixels in the image.

successfully maneuver through an unmodified office environment. Recently, [Du94b] has also been able to use the algorithm to play the game of tag with a cardboard target, and even with a slow-moving person, using only optical flow for pursuit, docking, and escape.

3.2 Temporal Aperture Problem

The standard aperture problem has already been discussed in Section 2.1. In the temporal domain, it takes on a slightly different form. In Figure 3.5 there is a contour translating to the lower-right at a certain speed, such that it requires $\delta t = 2$ time units to translate one pixel. We expect a match between points A at time T and point B at time $T-2$ to be very close. However, we get another "close" match between points A at time T and point C at time $T-1$. Globally, this motion is not correct, however motion is calculated independently at each time delay, and locally this false motion has a better match than no motion for that time delay.

If the contour shown extended infinitely with no end markers or distinguishing illumination changes, then we would have an instance of the *strong aperture problem* [BLP89b], which in principle cannot be solved. If there is at least some variation of the isocontour

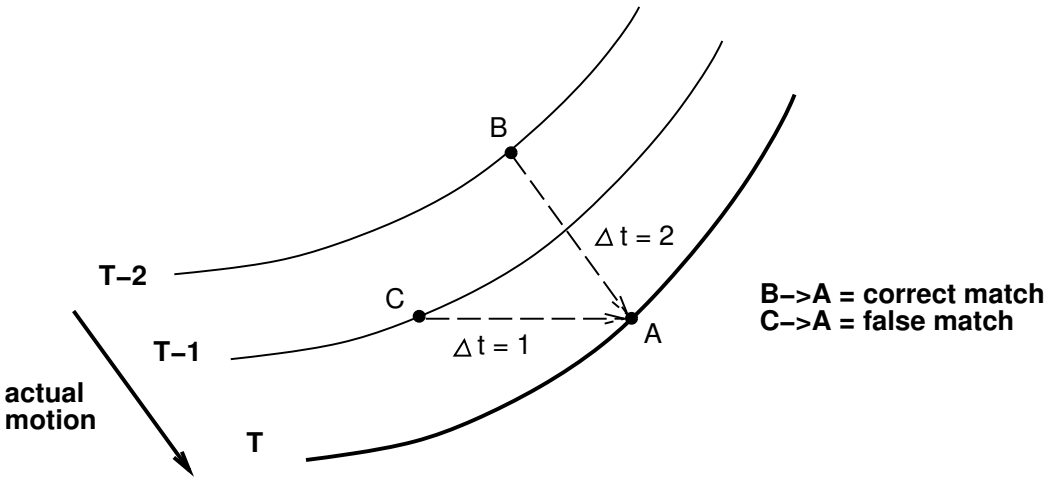


Figure 3.5: The Temporal Aperture Problem. The correct motion is $B \rightarrow A$, however if our window in time is too narrow, an incorrect motion $C \rightarrow A$ may be detected, because the contours are similar at points C and A.

and/or textural differences, then we have an instance of the *weak aperture problem*, which can be solved but requires a non-local mechanism. In our case of an area-based correlation algorithm, our non-local mechanism is the matching window, which has a finite (but usually sufficient) spatial extent.

In the case of the standard aperture problem, the motion of a contour may be ambiguous when viewed through a small aperture (or equivalently when our matching window is too small). In the case discussed above, the incorrect motion of the contour may be detected when our search "window" through time is too narrow. Thus we call this the *temporal aperture problem*. By sufficiently extending our search through time and picking the best match across time as well as space, we can in general avoid this problem. However, there is one special circumstance where this approach fails due to temporal aliasing, discussed next.

3.3 Temporal Aliasing

Previously, we have seen that we may solve the aperture problem for most cases by taking the best match value for a translating pixel in both time as well as space. Unfortunately, this does not always give the correct result. Examine Figure 3.6. Pixel A translates to pixel

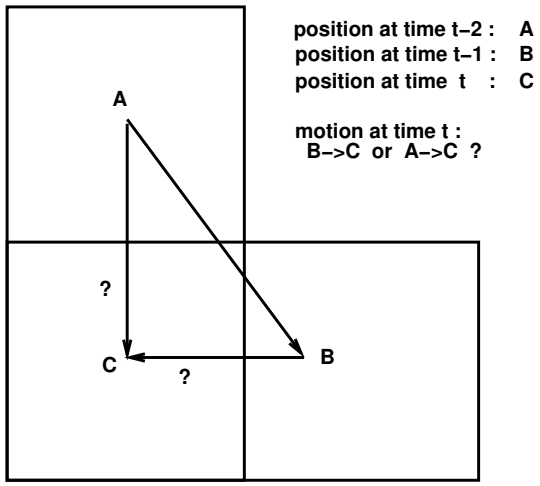


Figure 3.6: Temporal Aliasing. The correct motion is $B \rightarrow C$, however $A \rightarrow C$ *may* give a better match. Our solution to the Temporal Aperture Problem would fail for such a case.

B and then to pixel C, as if some object was executing a tight turn. In this diagram, it might be entirely possible that the A $\Rightarrow$ C motion may yield a better match than B $\Rightarrow$ C. If this is true, then based on a best-match policy, we would erroneously choose A $\Rightarrow$ C as the correct motion.

One "obvious" solution is to assume that the faster motions are correct.

It works for this example, and one can argue that the potential for aliasing is only limited by the sampling rate. Therefore, it might seem that the motion B $\Rightarrow$ C was correct, based on temporal aliasing arguments. This solution was presented in [CB91], and was a reasonable solution at the time, because available computational power limited practical motion detection to a small number of temporal delays. Now however, delays of as much as $\delta t = 20$ frames are practical and have been successfully tested. In these cases, the "fastest first" rule [CB91] too often selects an incorrect faster motion over a slower, better matching motion. The Temporal Aperture Problem is an example of this phenomenon.

Our solution to this problem is the following: in cases where the best velocity match is not in the same direction as the fastest motion for that time frame, then we have an inconsistency. Either 1) the slower motion is due to temporal aliasing or 2) the faster

motion is due to the temporal aperture problem. To resolve the inconsistency, we perform a second search over the multiple independent measurements, starting from the best matching velocity and continuing until the fastest velocity. Among these, we select as the correct motion the velocity with the best matching value subject to either a temporal consistency constraint, or the motion satisfies the intent of the "fastest-first" rule. In the former case, we assume that a motion has temporal support if it the motion at time t is "similar" to that at time $t+1$. To do this, we perform a "look-ahead" of one frame, and we define "similar" to be motion in the same direction (i.e. we allow for slight speed variations). In the latter case, if the motion is in the same direction as the fastest motion in the image, then we accept it.

Although not guaranteed to be correct, this heuristic has been thoroughly tested on many natural and synthetic image sequences and seems to perform quite well. Slow motion is accommodated by the temporal consistency constraint, and faster motions are supported by the modified "fastest-first" rule. Note that this additional search is limited to the number of velocities searched over, i.e. it is still a strictly linear algorithm. In addition, we limit our look-ahead to only one frame. Although techniques that use considerable temporal support can perform quite well [BFBB92], in real-time robotics it is not acceptable to impose a latency of a large number of images before a result is produced. The lookahead of a single image used by this algorithm is a relatively modest requirement. Section 4.4 examines the effect of the temporal antialiasing heuristic when performing time-to-contact calculations.

3.4 Harmonic Search Intervals

We have seen that by performing searches in time instead of space we can, in theory, convert a quadratic time algorithm into a linear one. In this section we will more closely examine this point.

One disadvantage of the traditional algorithm is that it computes image velocities that

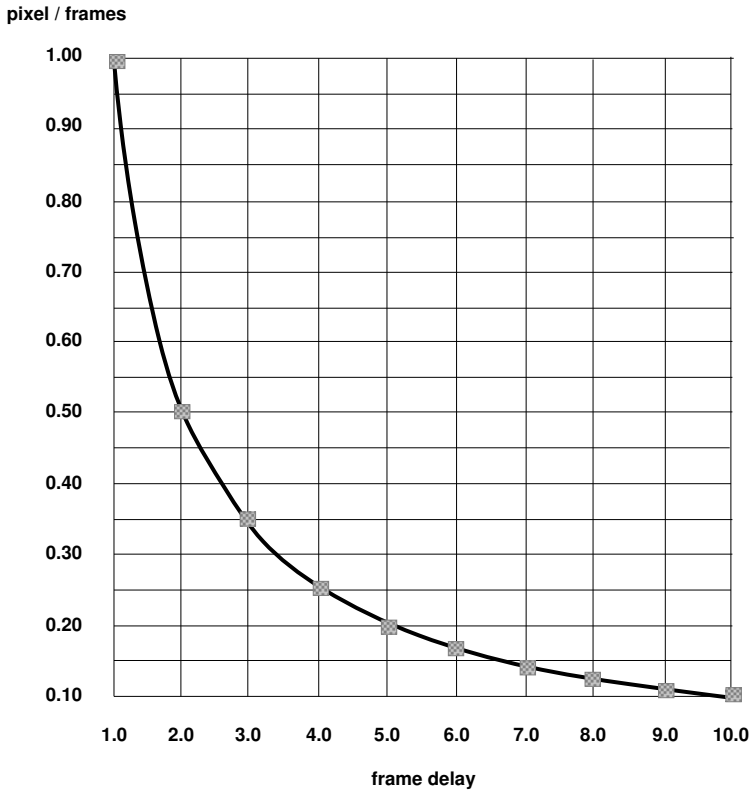


Figure 3.7: Only 1/(frame delay) velocity measurements can be detected in the current implementation.

are integral multiples of pixel shifts [BLP89a,b] : {1,2,3,...,η} pixels per frame. Although the algorithm discussed in this paper does calculate sub-pixel motions, it still computes velocities that are basically a ratio of integers (generally with the numerator equal to one pixel), not a truly real-valued measurement. Given $\delta d = 1$ and discrete frame delays $\Delta t = \{1, 2, 3, ..., S\}$ equation 3.1 yields $\{1/1, 1/2, 1/3, ..., 1/S\}$ pixels per frame equivalent motion, Figure 3.7. An immediate consequence of this is that sub-pixel motions are now detectable, correcting a deficiency of the original algorithm. Although the linear set of velocities may seem more intuitive than the harmonic series, in fact the latter is often much more suited to the types of motion found in real vision problems.

One example is shown in Figure 3.8 [Nal93]. If the velocities in the image are small, we can approximate

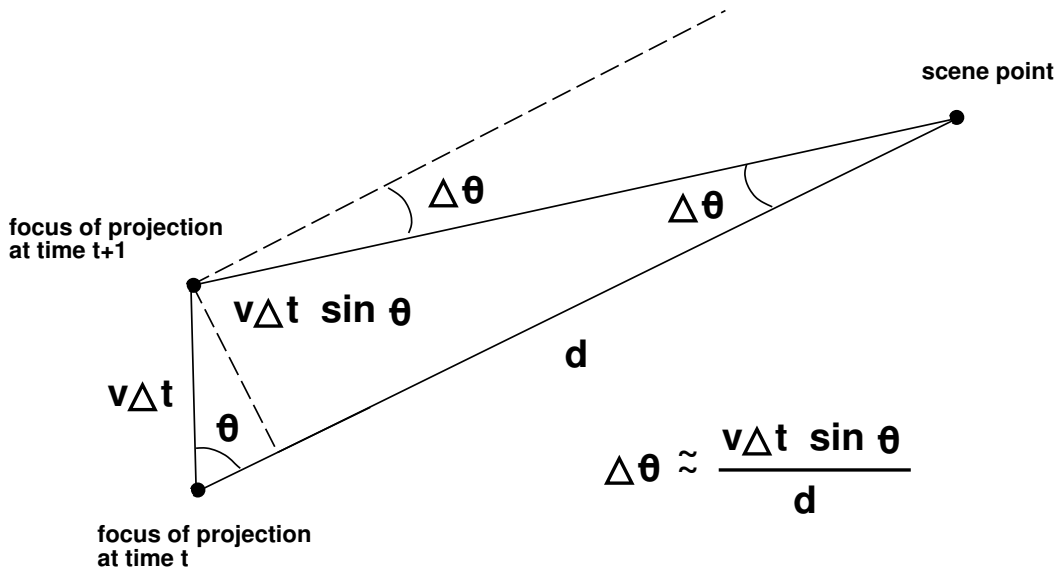


Figure 3.8: [From Nalwa 93] Change in visual angle is inversely related to depth.

$$\Delta\theta \approx \frac{v\Delta t \sin\theta}{d} \tag{3.2}$$

where θ is the visual angle (although visual angle is a measure used with spherical perspective projection, rather than planar perspective projection which is more appropriate for a machine vision system, they are approximately the same near the line of sight where $\tan\theta \approx \theta$. Although the former has the advantage of being independent of any coordinate system, the latter more accurately characterizes a machine vision system. See also [T91], Appendix).

The motion of a point in an image for a moving camera with line of sight orthogonal to the direction of motion is inversely proportional to a point's distance from the focus of projection, and can be used to determine depth; for example see Figure 3.9. This algorithm, which computes velocities inversely proportional to the discrete frame delays $\Delta t = \{1, 2, 3, ...S\}$ is therefore very well suited to computing depth via motion parallax. It would be much more awkward to attempt to calculate motion parallax using a linear set of motions $\{1, 2, 3, ..., \eta\}$ pixels per frame. Although there is a sine factor in equation 3.2, note that sine "degrades

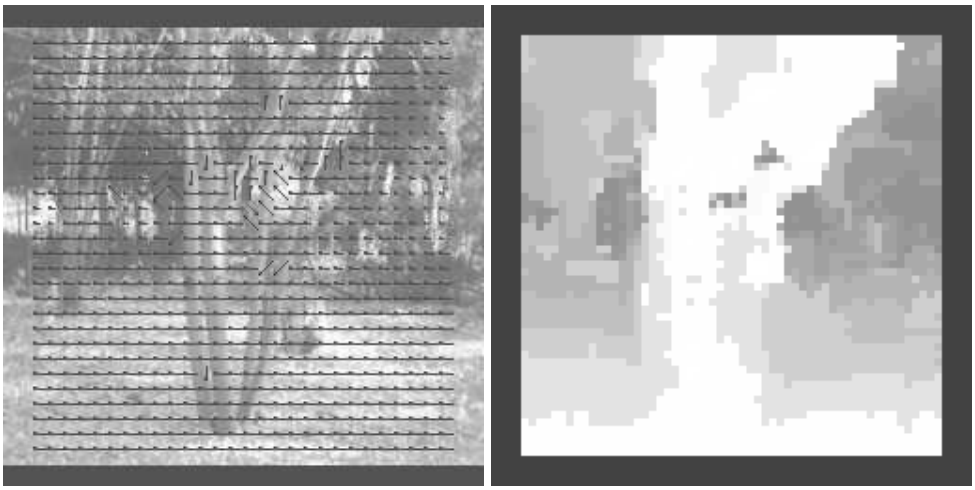


Figure 3.9: Depth from motion parallax. Camera is translating left, with the resulting optical flow and grey-level shaded relative depth map.

gracefully" with small deviations from 90 degrees, e.g : a 10 degree deviation from orthogonal motion to the line of sight is a factor of 0.985, and a 20 degree deviation (quite large for even a moderately calibrated mobile robot) still yields 0.94. Even a very large 30 degree deviation from the orthogonal yields only a .866 factor. To complete the argument, even if the deviation was a ridiculous 45 degrees (i.e. the robot was heading as much parallel to the line of sight as orthogonal to it!) the factor is still only 0.707. Thus, we do not need anywhere near perfect orthogonal motion to make practical use of depth from motion parallax.

It should be noted that it is not necessary to limit the measured velocities to a strict harmonic sequence. Given that the motion to be detected lies within our slowest and fastest measurements (1/1 and $1/S$ pixels per frame), and in the absence of bad spatial aliasing, we need only make measurements as precisely as we need. For example, we could detect a range of velocities $v = \{1/1, 1/2, 1/4, 1/8, .., 1/(2^{\lfloor log_2 S \rfloor})\}$ pixels per frame. In effect, this means that we have the option of detecting up to a quadratic range of velocities, again at only a linear-time cost. Of course, the resolution within these ranges is reduced by the same factor, but this may be a useful trade-off for a particular application. Conversely, it is

generally not possible for the traditional algorithm to "skip" pixels, since the actual motion may be missed. Section 4.2.3 discusses these issues in more detail.

3.5 Summary

The space-time tradeoff discussed in Section 3.1 allowed the development of an optical flow algorithm that was linear in the range of velocities detected, rather than quadratic as in the traditional algorithm. Unfortunately, dealing with time in this new manner introduces the problems of temporal aliasing and the temporal aperture problem, which were addressed by adding heuristic temporal antialiasing to the linear-time optical flow algorithm. Adding time to the search space also had some unexpected beneficial side effects, as noted in Section 3.4.

One disadvantage of the patch-matching approach is that the basic motion measurements are integer multiples of pixel-shifts. Although the linear-time algorithm discussed in this thesis does calculate sub-pixel motions, it still computes velocities that are basically a ratio of integers (generally with the numerator equal to one pixel), not a truly real-valued measurement. Although calculating real-valued optical flow measurements may be possible by using interpolation, this remains future work. In addition, the angles computed in the current implementation are only the eight nearest-neighbor pixels for eight possible angles of motion (plus the possibility of no motion). Increasing angular accuracy may also be possible by interpolating pixels, however this too remains as future work. Although it is not always necessary to have accurate optical flow to perform such functions as obstacle avoidance ([NA89]), Chapter 4 demonstrates that despite these deficiences remarkable accuracy may still be achieved in the context of calculating time-to-collision.

Chapter 4

Time-to-Contact

One application of optical flow is time-to-contact, sometimes pessimistically referred to as time-to-collision or time-to-crash [Lee76], [T90]. Using only optical measurements, and without knowing one's own velocity or distance from a surface, it is possible to determine when contact with the surface will be made. Time-to-contact provides an excellent test for an optical flow algorithm, and will be the primary example used throughout this thesis.

4.1 Calculating Time-to-Contact

The primary example of quantitative computer vision used throughout this thesis will be the calculation of time-to-contact during a collision sequence of a small mobile robot with a sweatshirt draped over a chair as in Figure 4.1 (many qualitative examples also appear in [C93]). The sweatshirt is used so that there is large enough of a surface area, and so that the robot's camera can actually make contact with the sweatshirt safely without risk to the camera or robot. The sequence begins with frame 0 on the right of Figure 4.1.

The robot began about a meter and a half away from the chair, and translates at a rate of approximately 5 cm per second. The frame rate is about 5 frames per second, so each frame represents about 1 cm of actual translation. (All measurements are subject to the limits of the accuracy of the robot's gears and the reliability of UNIX's timing commands.) The camera's lens has a field of view of 60 degrees, however only the central 256x256 pixels

Figure 4.1: Time-to-contact experiment. Sweatshirt is draped over a chair on left. Frame 0 of sequence shown on the right.

of an original 512x512 image were used in subsampling to 64x64. Figure 4.3 shows the image of the chair at frames 12, 41, 96, 126, 134, and 139. Note the very low contrast of the sequence; gradient and especially second-derivative techniques such as [Nag87] which suggest calculating flow at grey-level "corners" would not find any such high-contrast locations in these images, once the chair fills the field of view. Although contrast-sensitive elementary motion detectors may be sufficient for very small (and lightweight) insects such as the fly ([Bor90], [EB93]) they are clearly insufficient for mobile robots. In addition, note that the image is badly out of focus close to the target.

Actual contact with the sweatshirt was made somewhere between frames 141 and 142 (nominally 141.5). Thus we would want our time-to-contact algorithm to determine, at every point during the sequence, that the contact point is around frame 141.5 (i.e. the current frame number plus the current time-to-contact in frames). For this sequence the slowest velocity searched for, $1/S$, was set to 1/10 pixel/frames.

Figure 4.2 describes the optical geometry (the classic reference for the image-plane coordinate system is [LP80]). A point of interest P at coordinates (X,Y,Z) is projected through the focus of projection centered at the origin of the coordinate system (0,0,0). P is fixed in

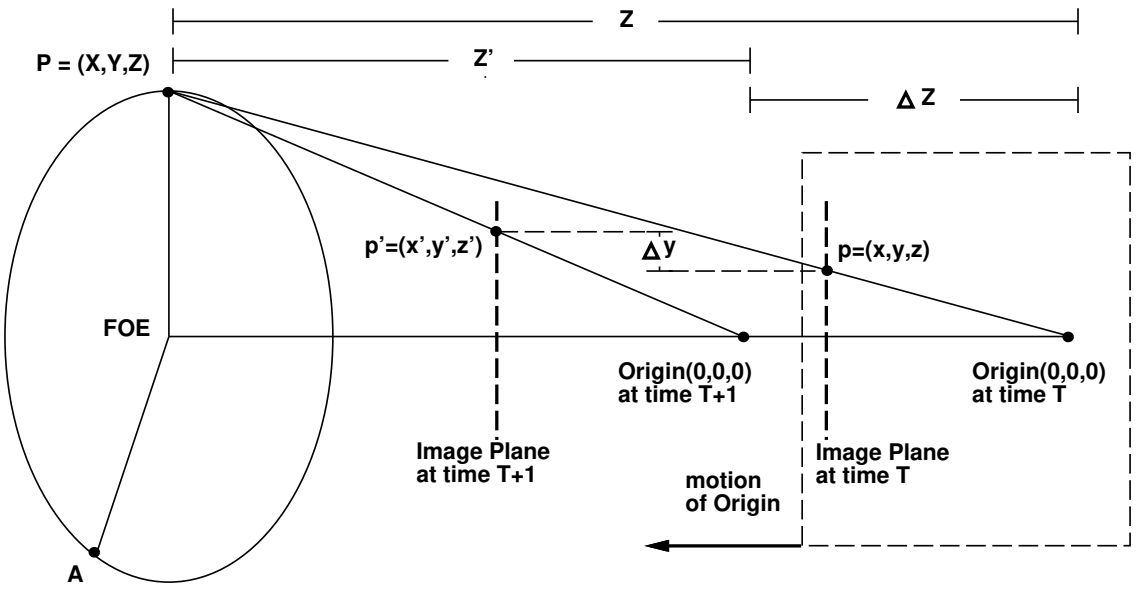


Figure 4.2: Optical Geometry for time-to-contact. See text for details.

physical space and does not move. The origin/focus of projection, however move forward with a velocity $\frac{dZ}{dt}$. If the camera is facing the same direction as the direction of motion, then this direction is what is commonly known as the focus of expansion (FOE), since it is the point from which the optical flow diverges. (In the case of mobile robotics, it is quite reasonable to assume that the camera is in the same direction as the direction of translation; we do not have to solve the general case of egomotion in order to exhibit useful behaviors.) The image plane is fixed at a distance z in front of the origin; for convenience, we set $z = 1$. (The actual value of z would depend on factors such as the focal length of the camera.) This image plane moves along with the origin. P projects onto point p in this plane. As the image plane moves closer to P, the position of p in the image plane changes as well. Using similar triangles :

$$\frac{y}{z} = \frac{y}{1} = \frac{Y}{Z}$$

Differentiating with respect to time [where $\dot{x}$ represents $\frac{dx}{dt}$] :

$$\dot{y} = \frac{\dot{Y}}{Z} - Y(\frac{\dot{Z}}{Z^2})$$

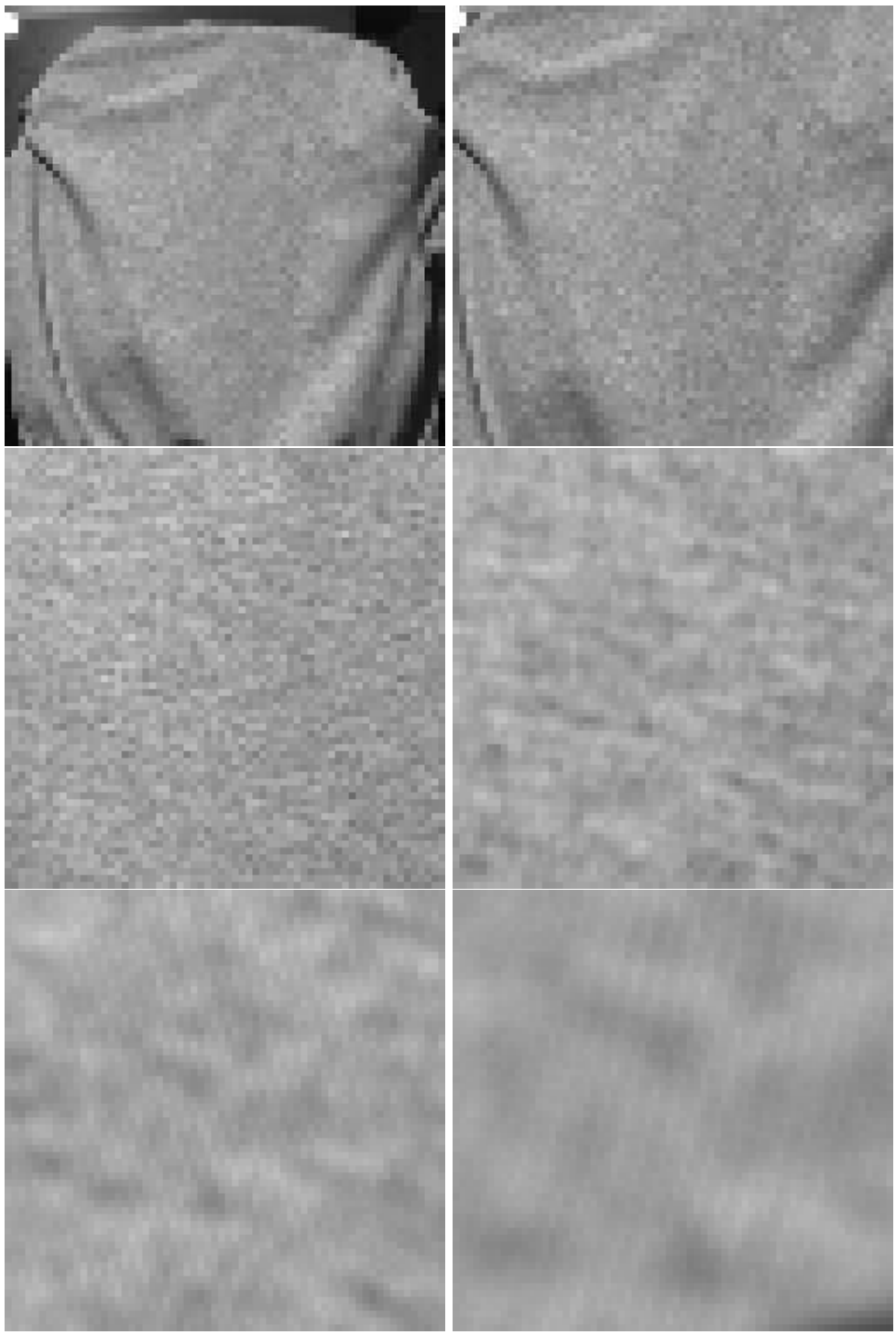


Figure 4.3: Images of the chair (in row-major order) at frames 12, 41, 96, 126, 134, and 139. Note the extremely low contrast of the sequence.

Since P is immobile, set $\dot{Y} = 0$ and substituting (yZ) for Y :

$$\dot{y} = -y(\frac{\dot{Z}}{Z})$$

Finally divide by y and take the reciprocals of both sides :

$$\frac{y}{\dot{y}} = -\frac{Z}{\dot{Z}} = \tau \tag{4.1}$$

The quantity τ is known as the time-to-contact [Lee76]. Note that the left hand side contains purely optical quantities, and that knowledge of τ does not give any information about distance or velocity per se, but only of their ratio. The left-hand side of equation 4.1 gives us a method for calculating time-to-contact : for a camera heading in the same direction as the FOE, pick a point in the image, and divide its distance from the FOE by its divergence from the FOE.

This requires first calculating the FOE. Ideally, we could simply take the intersection of any two optical flow vectors, but due to measurement errors this would be inaccurate, and in the case of this particular algorithm it certainly would not work, since the velocity measurements are limited to only eight directions (and zero). [TGS91] uses a least-squares approximation to the intersection of all the available optical flow vectors. [P92] uses several direction-selective and speed-tuned sensors without assuming that accurate velocity information is available. Since the optical flow vectors at various points in the image can be treated independently, heading determination can be calculated with a two-layer linear neural network [HW91]. [RL85] computes the FOE in cases where there is a rotational component present by calculating the difference vectors $\parallel \Delta_{ij}\mathbf{u} \parallel$ of the optical flow vectors in a local area to a given vector. The largest of these vectors should correspond to depth discontinuities, where the vector differences should be greatest. A least-squares fit is performed for those vector differences above a certain threshold. This is true regardless of rotational motion, since the latter adds the same motion component regardless of the depth

variations in the image [LP80]. There is psychophysical evidence supporting this *differential motion* method of separating the rotational and translational components of the flow field [WH89]. [Hil91] extends the approach of [RL85] by considering moving objects in the image as well. Since the chair in Figure 4.1 used in the time-to-contact experiments represents a flat surface, techniques based on differential motion cannot be used for this example.

It is possible to calculate the FOE directly without computing the intervening optical flow. [NG92] discusses a method for calculating the FOE using the assumption that calculated depth should always be positive. The image brightness constraint equation proposed by [NH87] reduces in the purely translational motion case to :

$$c + \frac{1}{Z}(s \cdot t) = 0 \tag{4.2}$$

where c is the temporal derivative of the image intensity function $E(x, y, t)$, $t = (t_1, t_2, t_3)$ is the instantaneous translational velocity of the camera relative to a stationary scene, $Z(x, y)$ is the depth map of points in the scene, and $s = r \times E_r$ where $E_r = (E_x, E_y, 0)$ is the spatial gradient of E at image point $r = (x, y, z) = (x, y, 1)$ at time t. (As usual, z is set to 1 without loss of generality.)

For each estimated motion $\hat{t} = (\hat{t_1}, \hat{t_2}, \hat{t_3})$ equation 4.2 yields a depth map :

$$\hat{Z} = -\frac{1}{c}(s \cdot \hat{t}) \tag{4.3}$$

except where c vanishes. Thus are an infinite number of $\{\hat{t}, \hat{Z}\}$ solutions to equation 4.2. [NH87] shows that for an incorrectly calculated FOE, there is a corresponding cluster of mostly negative depth values along the "FOE constraint line" which connects the actual FOE with the computed FOE. In addition, clusters of positive depth values are expected on either side of the negative cluster along the FOE constraint line. A vote for the actual FOE may be calculated by computing several candidate FOE's, computing depth estimates within a small radius, and constructing a line that intersects the centroids of the positive and

negative depth clusters; this line is ideally coincident with the FOE constraint line. Several such FOE estimates can be made, with the actual FOE taken to be their intersection. For robustness, [NG92] uses only the sign of the depth values and not their magnitudes.

This algorithm can take advantage of the X-Y component representation of the optical flow vectors. In the case of translational motion and a single object filling the field of view, the FOE may be calculated by averaging the X and Y components of all the optical flow vectors, treated as unit vectors, and treating the average as an offset from the line of sight, which is assumed to be straight forward. The vectors are treated as unit vectors for robustness, *i.e.*, vectors close to the FOE (or far away from the FOE) are not penalized by some factor related to their magnitudes. This gives an initial estimate for the FOE. Then the estimate is refined by averaging the flow vectors within a window of half size on each edge until the window size is only 4x4 pixels. This method is very simple and extremely fast, and produces visually accurate results for most cases, especially with images with low noise. Unlike gradient-based approaches (such as [NG92]), the patch-matching nature of this algorithm is largely insensitive to local contrast. The use of the entire dense optical flow field is supported by [WH89] which presents psychophysical evidence that observer performance in heading estimation decreases with a decrease in the density of the optical flow information available.

The optical flow for at several points in the collision sequence are shown in Figures 4.4-4.6 along with the calculated FOE.

From equation 4.1 and Figure 4.2 we can see that the expected divergence of any point along a circle of a given radius should be equal; e.g., given the assumption of a flat surface and a direct approach the divergence of any point A should be equal to that of point P. Therefore, it is justifiable to average the optical flow measurements along the circumference of any circle of a given radius, centered at the FOE, to get a single real-valued measurement for $\dot{y}$ in equation 4.1. The number of individual measurements used in this thesis is 4 times

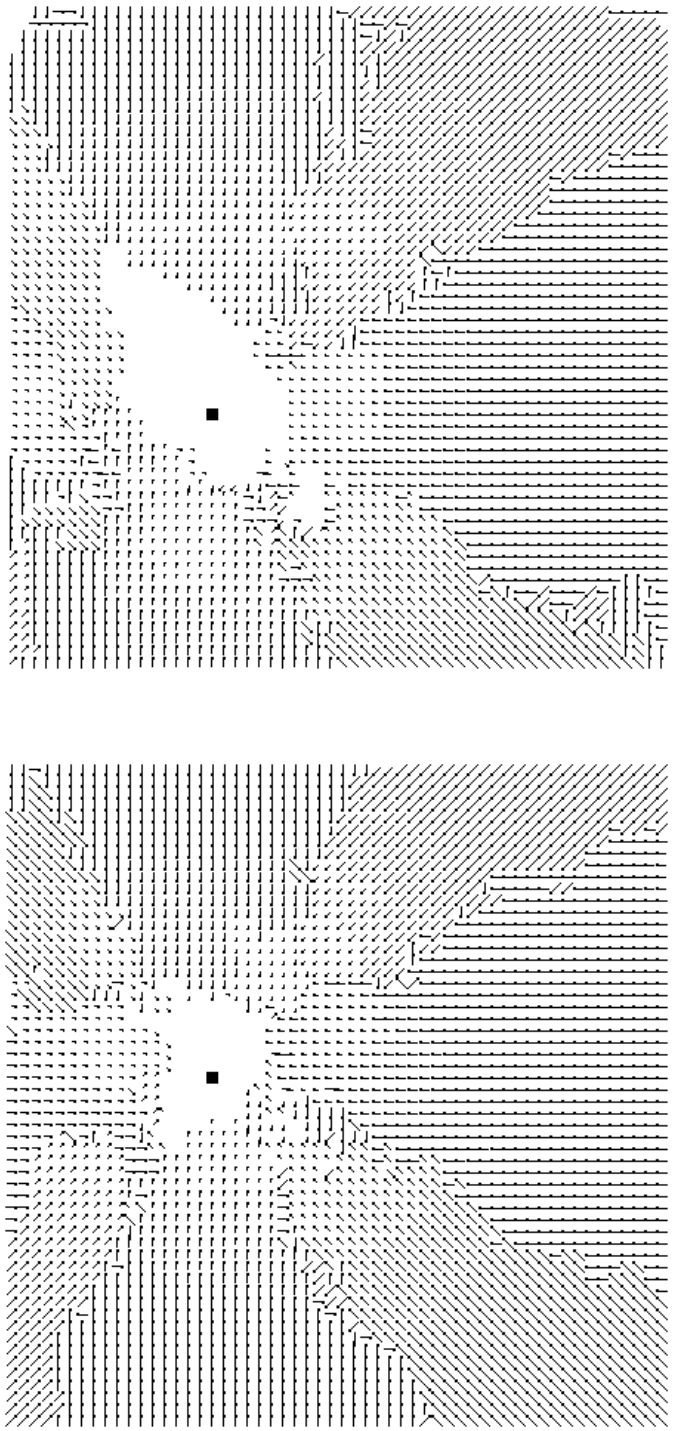


Figure 4.4: Optical flow of the collision sequence at frames 12 and 41.

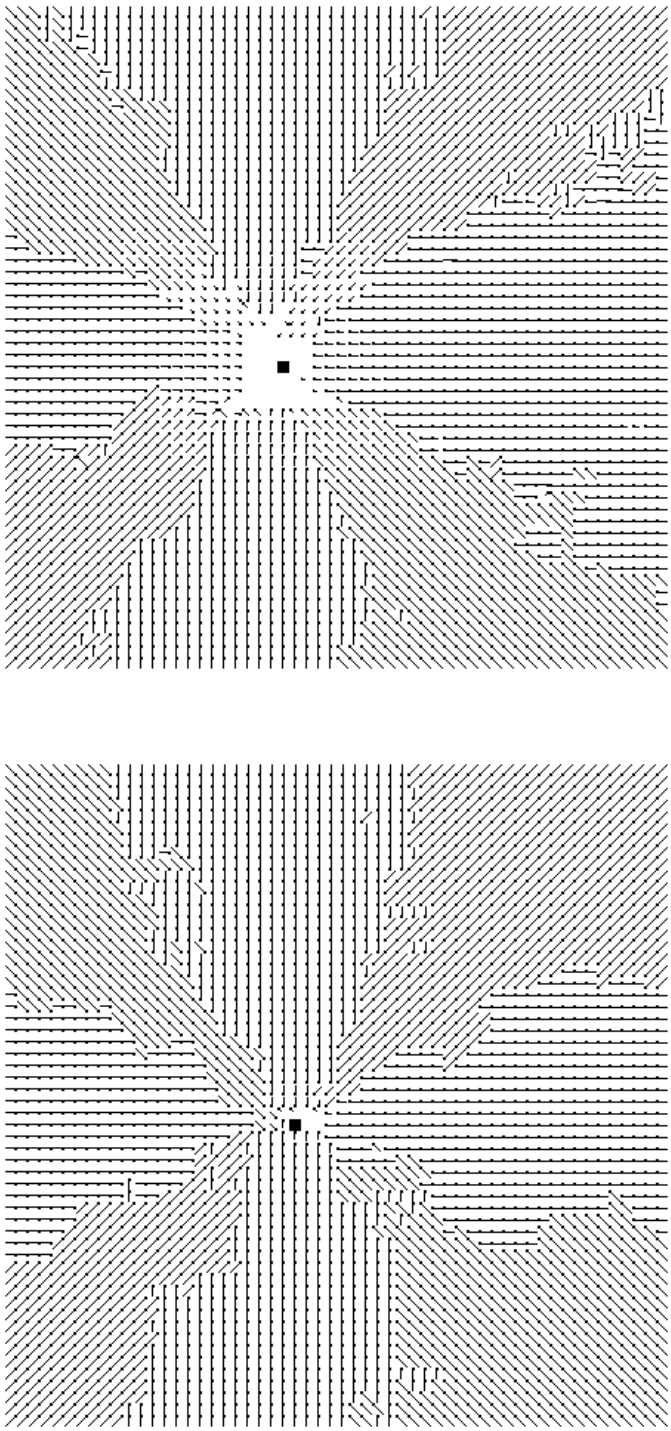


Figure 4.5: Optical flow of the collision sequence at frames 96 and 126.

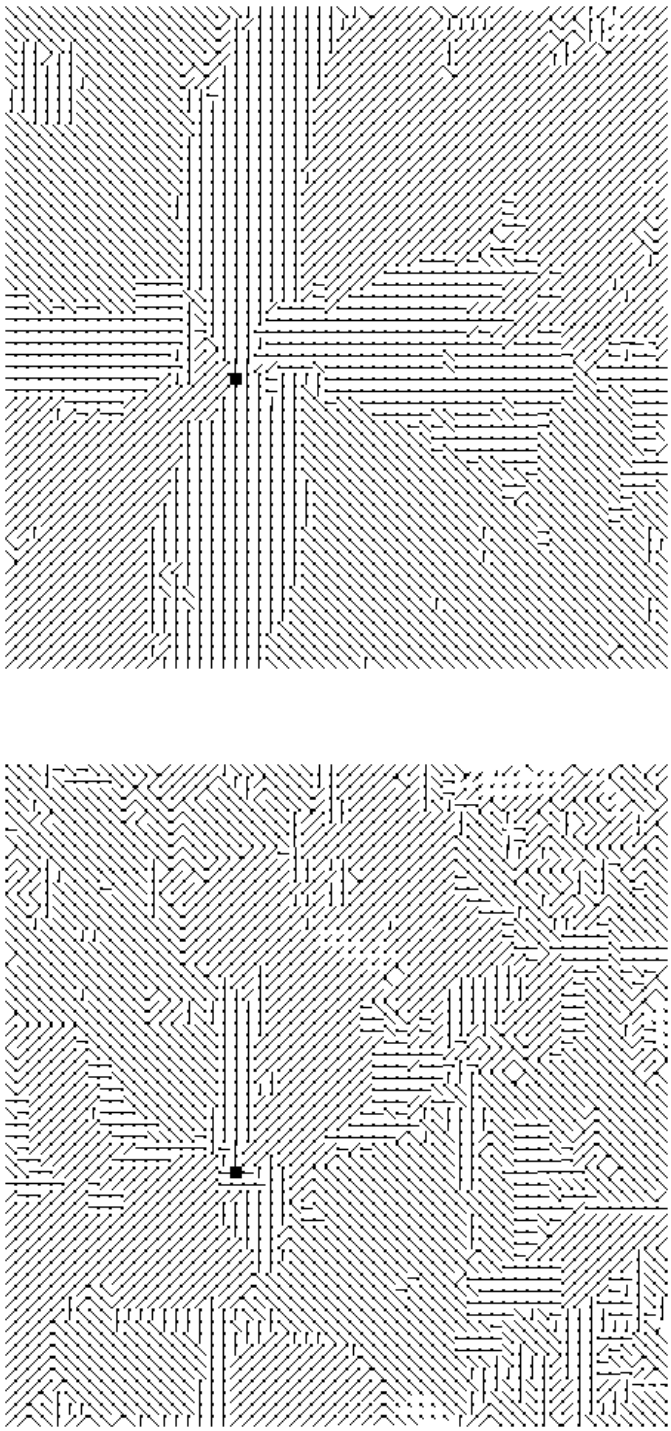


Figure 4.6: Optical flow of the collision sequence at frames 134 and 139.

the radius in pixels, i.e. 4 measurements for a one-pixel radius, 8 for a 2 pixel radius, up to 124 for a maximum 31-pixel radius. In this case, each "individual" measurement is in fact the weighted averaged of the four nearest pixels to that actual real-coordinate point in the image (Figure 4.7. This weighted averaging method has many of the desirable properties of the "weighted area with overlap" sampling filter described in [FvD+90] (p.132 and p.617) but has the advantage of being extremely fast to compute (only a few floating point subtracts, adds and conversions to integer). In addition, any motion along a diagonal must be considered to be $\sqrt{2}$ times the normal magnitude value, since images are generally in a square tessellation. Quantity y in equation 4.1 is then simply the radius of that particular circle, and then finally a single measurement of time-to-contact is calculated by $\tau = y/\dot{y}$. The number of independent time-to-contact measurements available is only limited by the image size, and the position of the FOE. If the FOE is roughly centered, then for a X*X size image we might expect X/2 measurements to be available. If the FOE is not centered, then there may be fewer measurements which can be made in image itself. Experiments with synthetic images have shown the algorithm to be robust with respect to non-centered FOE's, as long as the FOE is located in the image itself.

Figure 4.8 shows grey-level shaded histograms of the distribution of velocities for each given radius, for frames 12, 41, 96, 126, 134, and 139 of the collision sequence. Ideally, this would be a straight line intersecting at the origin of the graph. Of course the algorithm only computes motions that are $1/T$ pixel/frames, so this is not the case. However each individual motion measurement actually consists of the weighted average of the four nearest pixels, so there is a distribution of motion measurements rather than just the $1/T$ or $\sqrt{2}/T$ pixel/frames quantizations. Not surprisingly however motions corresponding to "pure" harmonic velocities (i.e. all four pixels were the same velocity) dominate (the largest of which are noted in the figure).

For frame 134 in Figure 4.6 the maximum motion is about 4 pixels/frame at a radius of

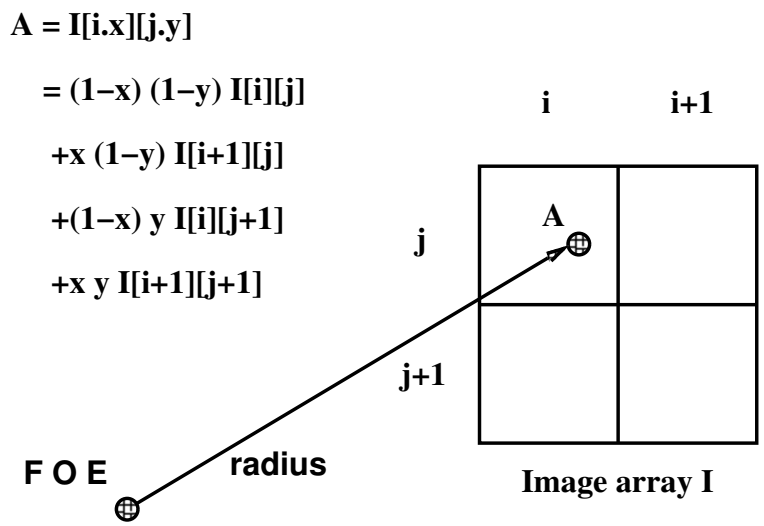


Figure 4.7: Calculation of weighted average of the four pixels closest to a real-valued coordinate. Note that a pixel's coordinate refers to the *center* of that pixel's physical area.

30 (given a time-to-contact of 7.5 frames). Although this motion is much greater than can be detected with the current implementation, the *direction* of motion is still visibly correct (to the nearest 45 degree quantization), even for this very fast motion. Thus it is entirely conceivable to have an extension to this algorithm which calculates greater-than-one-pixel motion using the 1-pixel motion as an initial guess to limit the search space and resulting quadratic running time. In this case, we would limit the motion search to a "cone" of fixed-pixel width along the direction predicted by the 1-pixel motion search. This would keep the linear-time and constant-angular-precision properties of the original algorithm.

The current implementation however is limited to a maximum velocity of one pixel per frame along any of the four cardinal directions (NEWS). Along the diagonals, the maximum velocity is $\sqrt{2}$ pixels/frame, however when the actual motion for points along a circle of a given radius exceeds one pixel per frame (for the current implementation), the optical flow in the NEWS directions saturates, yielding slower than actual optical flow, and thus a longer than actual time-to-contact. For the current problem, we will simply not consider measuring the time-to-contact for radii whose averaged motion results in greater than one

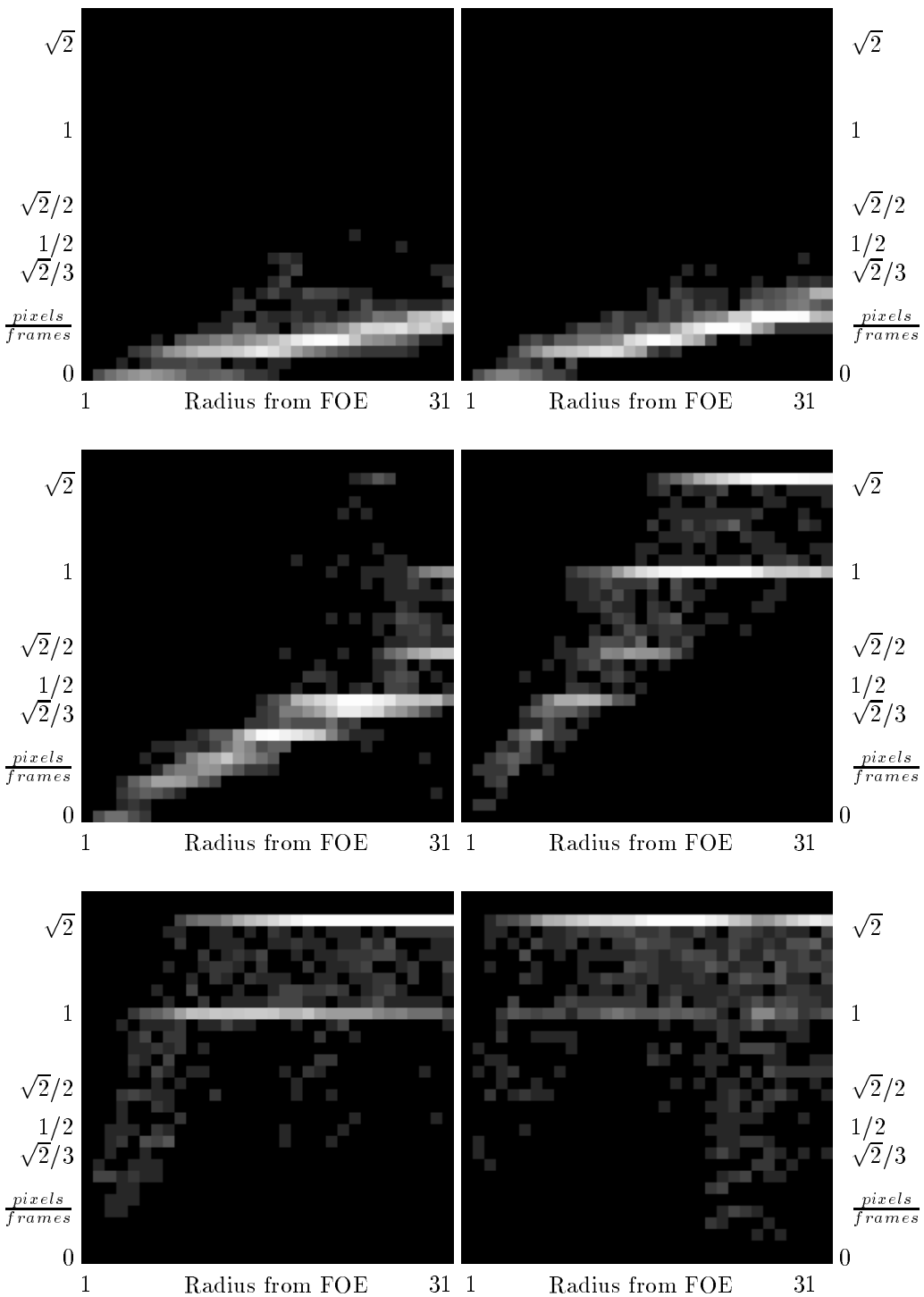


Figure 4.8: Grey-level histograms (in row-major order) of individual optical flow measurements for frames 12, 41, 96, 126, 134, and 139. The histograms have been brightened with gamma factor 2 to enhance visibility.

pixel per frame. Since we are using the averaged motion, it is possible that the distribution of velocities are such that some 1.0-pixel/frames motion are saturated before an average of 1.0 pixel/frames motion is reached, however the effect seems small enough to safely ignore. Figure 4.9 shows the averaged motion measurements as a function of the radius, or distance from the FOE. The horizontal line represents a motion measurement of 1.0. Where these motion measurements cross this line places an upper bound on the radius of the circles used to measure time-to-contact; this upper bound is drawn as a vertical line drawn from above the time-to-contact best-fit line. Since the expected TTC is naturally reduced by 1 frame per frame, the distance at which 1.0 pixel/frames motion is reduced by one pixel per frame according to equation 4.1, and thus this threshold cutoff point is itself expected to be reduced by one frame per frame. Even when only 2.5 frames away from collision, there are enough valid measurements for an accurate time-to-contact calculation.

The low (left) end of the chart represents very slow velocities near the FOE. Since in practice we limit the velocity search to some lower bound, these slow velocities may be slower than can be detected. However, the slowest velocity that is detected may give a better match than no motion, thus these pixels' velocities may be rounded up to $1/S$ pixels/frame, yielding a lower than actual time-to-contact for that radius. Thus it is often useful to set some lower velocity threshold below which time-to-contact measurements will not be considered, since they are likely to be inaccurate. Rather than setting a threshold on the individual measurements, we set a threshold on the averaged measurement for the entire circle. This allows the slowest velocities to participate in the distribution of velocities for a circle of a given radius, as in Figure 4.8. Although there currently is no automatic mechanism for setting this threshold, values from 1.6 to 1.8 times the slowest velocity checked for in the image have been successful. In addition, experiments show that there is a point where increasing the threshold has little effect on the measurements (for those frames where the object fills the field of view). Thus it could be possible to adaptively set this threshold as

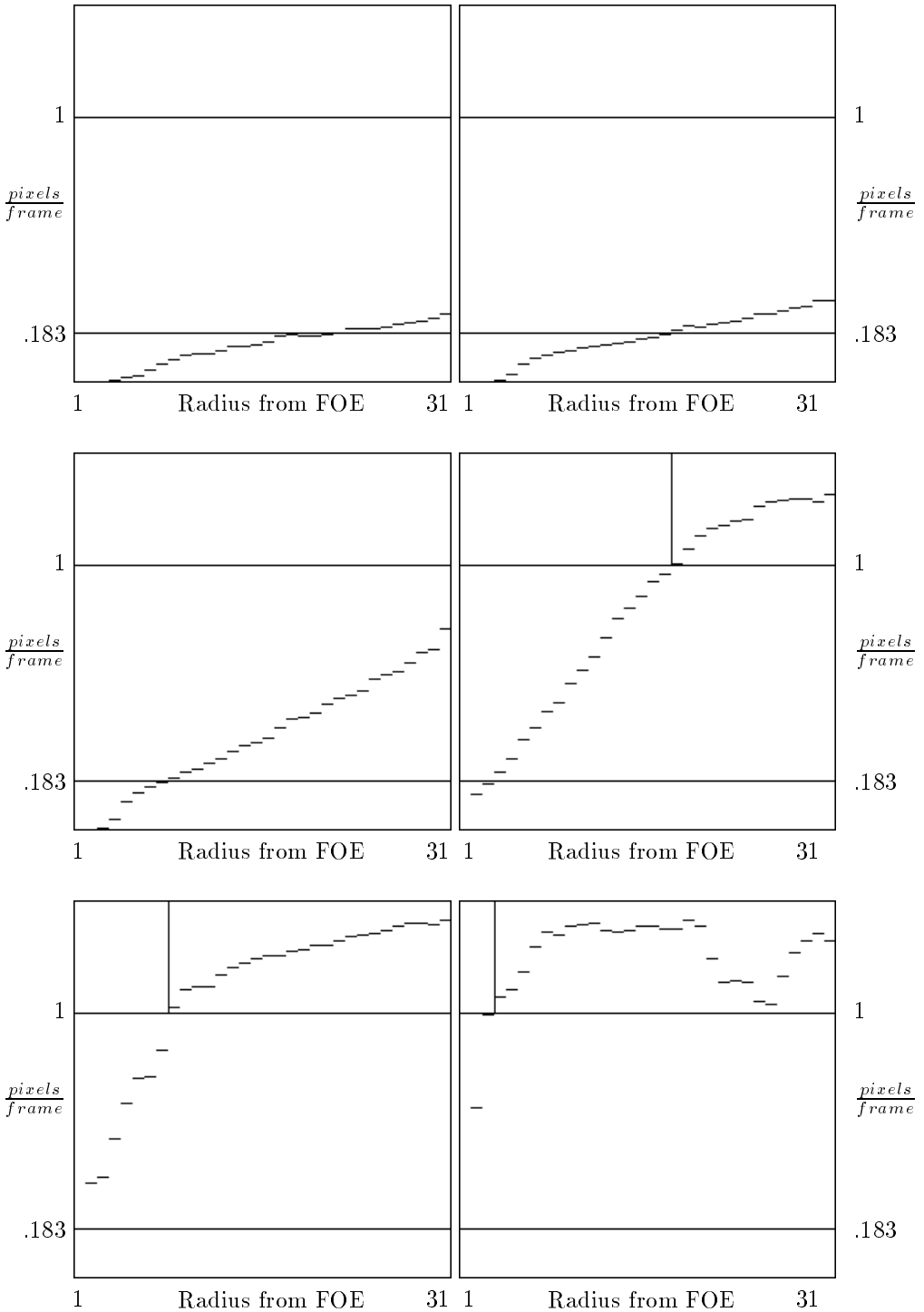


Figure 4.9: Histograms of averaged optical flow measurements (in row-major order) for frames 12, 41, 96, 126, 134, and 139. Each measurement is the weighted average of the measurements shown in Figure 4.8.

well. Figure 4.9 shows this cutoff point as the lower horizontal line. Even at the beginning of the sequence when only a few measurements are above threshold, the time-to-contact measurement is quite reliable. Together these two thresholds eliminate measurements that are outside the optimal range of $\{1/1, 1/2, ..., 1/S\}$ pixel/frames.

By calculating an estimate of time-to-contact along the circumferences of circles of multiple radii, we can compute several independent estimates, and take some best-fit measure among them. In this case a robust maximum-likelihood estimate is used [H81]. This form of regression, unlike the standard least-squares estimate case, has the desirable property that outliers have only a minimal (non-linear) affect the final result. For standard statistical measurements, it might be unwise or unscientific to treat suspect data with anything other than total equality with the rest of the data set, but in this type of problem it is a simple fact that there are measurements which simply "deserve" to be thrown out. One way to approach this problem is in two separate steps, outlier rejection followed by a classical estimation technique. [H81] argues against this approach, noting 1) it can be difficult to cleanly separate these two steps, given that the definition of outliers depends on some initial estimate in the first place, 2) the cleaned data will not be normal (statistically, some good data will have been eliminated and some bad data retained), and 3) empirically, the best two-step estimators do not perform as well as the best robust estimators. This author's own experience with both approaches agrees completely with this argument; in fact, the latter approach was adpoted for precisely this reason!

A detailed explanation of the implementation of robust maximum-likelihood estimation can be found in Appendix A. For the moment, it is important to note that the implementation is an iterative procedure requiring an initial estimate. In this case an excellent initial estimate is the geometric mean of the 31 (or less) measurements (one per radius length). The procedure generally converges after only 4-6 iterations, and since there are only 31 measurements, time to convergence is negligible. Although the geometric mean

itself does tend to attenuate the effects of outliers, it alone does not yield the performance of the robust maximum-likelihood regression procedure. In general, a slightly better initial estimate could be provided by noting that the measurements close to the FOE tend to be higher-than-actual optical flow due to a rounding-up of velocity measurements, yielding a shorter-than-actual time-to-contact. However, the extremely fast rate of convergence given the geometric-average initial estimate does not justify any additional problem-specific information at this particular stage.

Figure 4.10 shows the time-to-contact measurements for each radius for several frames in the collision sequence. Each measurement results from applying equation 4.1 to Figure 4.9. The vertical line represents the 1.0-pixel/frames upper threshold, the dotted horizontal line represents the initial estimate, and the solid horizontal line represents the final maximum-likelihood estimate for time-to-contact. No lower-bound thresholds were used. The plots for frames 134 and 139 have only a few valid measurements less than 1.0-pixel/frames, so for enhanced visibility only the lower left-hand quarter is shown, zoomed in by a factor of two.

Figure 4.11 includes the lower-bound threshold of .183 pixel/frames, shown by the vertical line meeting the time-to-contact estimate from below. In many of these cases the initial estimate is within a frame of the final maximum-likelihood estimate and thus is represented by the same horizontal line. Examining this figure, there is generally a drop in the time-to-contact measurements corresponding to motions below the lower-bound threshold, due to the "rounding up" of the slowest velocities near the FOE. The location of this sudden drop could possibly be used to adaptively set value of the lower-bound threshold as well. (Although [H81] argues against an explicit outlier rejection step before maximum-likelihood estimation, this situation differs somewhat in that there is prior knowledge of the nature of the outliers in this case.)

Figure 4.12 left shows a plot of these best-fit time-to-contact measurements against the

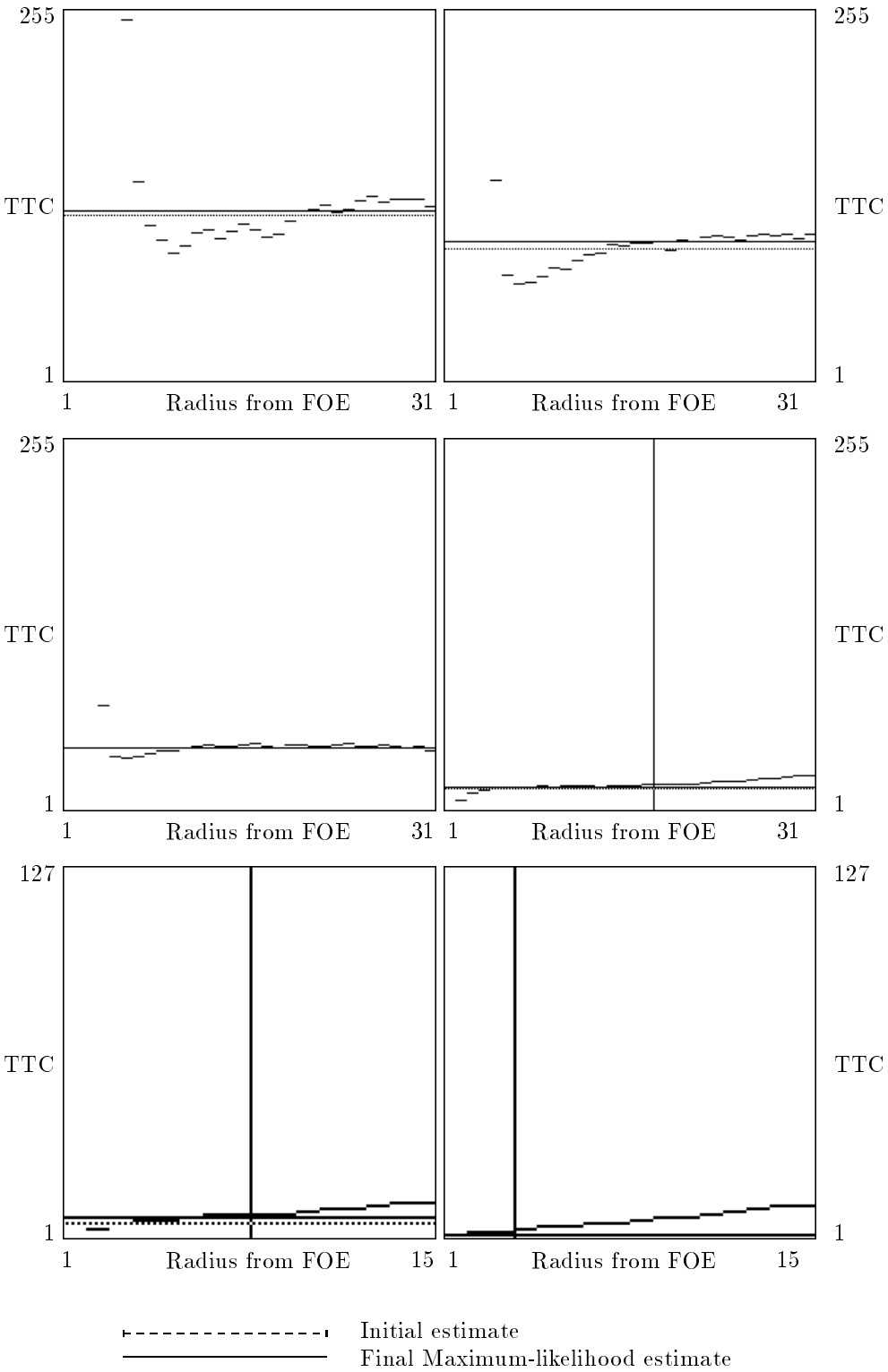


Figure 4.10: Time-to-contact estimates (in row-major order) for frames 12, 41, 96, 126, 134, and 139. No lower-bound thresholds were used.

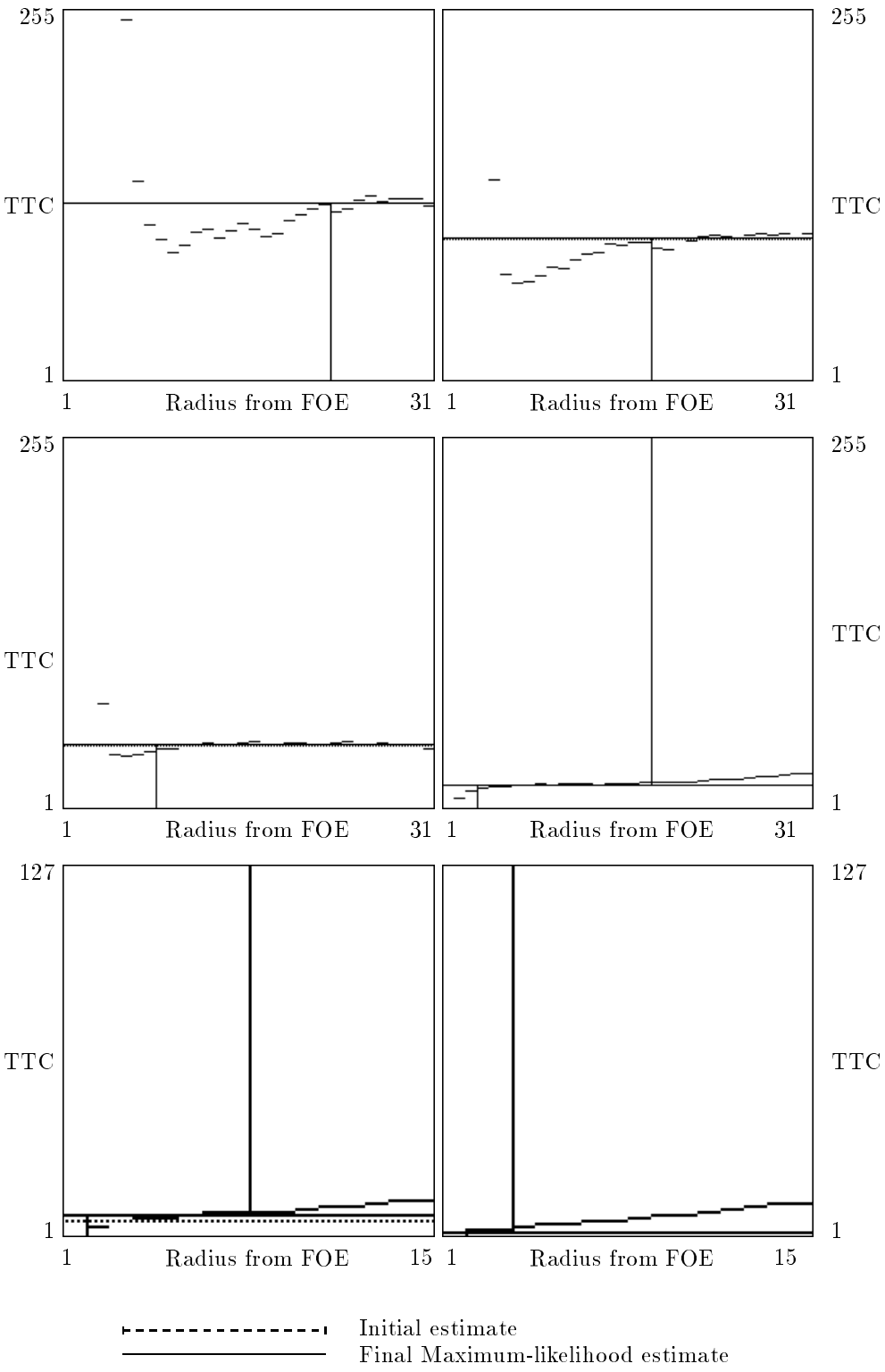


Figure 4.11: Time-to-contact estimates (in row-major order) for frames 12, 41, 96, 126, 134, and 139. Lower-bound threshold is .183 pixel/frames.

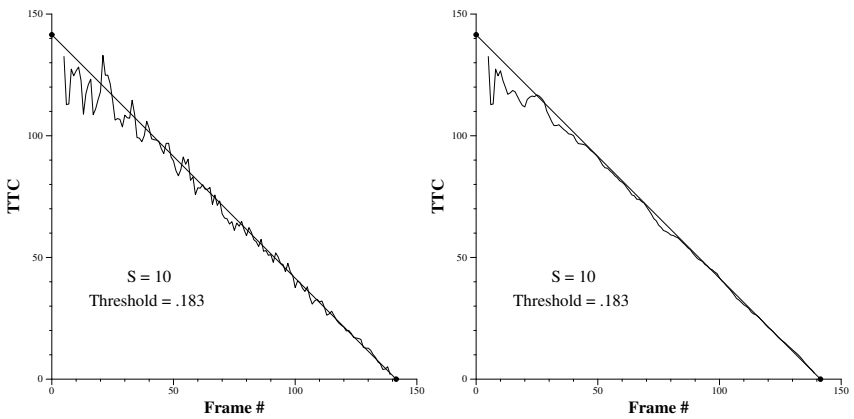


Figure 4.12: Results of time-to-contact experiment for "fast" collision sequence. Left figure is a plot of instantaneous time-to-contact vs. frame number. Right figure is similar except that each value represents the average of the current and previous 7 instantaneous time-to-contact measurements.

current frame number. The actual contact point is somewhere between frames 141 and 142, so for the purposes of this graph was arbitrarily set to 141.5, represented by the solid line from (0,141.5) and (141.5,0). The jagged nature of this plot suggests aliasing due to "jumps" of the velocity measurements at various positions in the image from $1/S$ (or $\sqrt{2}/S$) pixels/frames to $1/(S-1)$ (or $\sqrt{2}/(S-1)$) pixels/frames as the target is approached, but this has not been confirmed. In any event, we can take an average of the last n time-to-contact measurements as the final measure of time-to-contact. For examples in this paper, $n = 8$, and the final result is shown in Figure 4.12 right. The time-to-contact calculations were computed offline and can be computed at 4 frames per second on a Sun Sparcstation 10/41 (all the time-to-contact calculations are roughly equivalent in CPU time to calculating a single speed of optical flow).

Most of the measurements in the last 100 frames are within a frame of this value, and the average is 140.86 with standard deviation 0.91 frames. The average for the last 50 frames is 141.43, with standard deviation 0.52 frames. In addition, measurements are valid up until 2 frames before contact (invalid measurements are clearly labeled as such by the algorithm). Clearly, integrating across space and time more than compensates for the non-real-valued

measurements of the linear-time optical flow algorithm.

These results compare favorably with other techniques reported in the literature. [UGVT88] calculates time-to-contact in a real image sequence using a second-derivative method and reports an error of 2.2 time steps at a distance of 31.1 time steps away from contact when measured over a 10x10 optical flow vectors, with a standard deviation of 2.8 time steps. [TS91] simplifies the calculation of time-to-impact by using velocity represented in a polar coordinate system, and computes a coarse hazard map using computational time on the order of one minute on a Sparcstation 1. [HG91] reports accuracy of better than 1% at distances of 15 meters down to 2 meters from the target using recursive filtering of subpixel measurements of features in the image, using a real-time image processor. Although good results such as these can be achieved using large (40cm by 40cm is quoted) and presumably high-contrast patterns, it may not be realistic to assume that subpixel measurements of image features are possible in the general case. It would be very difficult to locate any distinctive features in Figure 4.3, and extremely difficult if not impossible to localize them to subpixel resolution. [AP93] uses Green's theorems and a one-dimensional correlation-based optical flow technique to estimate the divergence of the optical flow field by integrating the optical flow normal to a closed contour; for the application of time-to-contact they report errors on the order of 10%.

There are two issues worth mentioning here. First, we assumed that the optical flow at each point along the circumference each circle of a given radii is normal to that circle, and pointed away from the FOE. Clearly, with only 8 directions available, this will not be true in general. We are hoping that "on average" the directions of the flow vectors are correct. Second, the magnitudes of the optical flow vectors are also quantized, and will themselves not be accurate. In addition, those that are directed along a diagonal count for $\sqrt{2}$ times as much motion as those directed along a cardinal direction. We are again hoping that the flow will be evenly distributed around the correct value. It is not obvious that the optical flow

measurements will be well-behaved in this manner, so it must be verified experimentally.

Some vectors will be totally incorrect, and point towards the FOE. In all cases however only the absolute magnitude of the vector is taken (and implicitly assumed to be normal to the circle of that radius). Since each "individual" measurement along the circle is actually the weighted average of the four nearest pixels, it is not very practical to disallow measurements based on one or two that point towards the FOE or are otherwise not close to normal to the circle at that point (although such a mechanism could be implemented).

This is not the only way to approach this problem. Another option is to average the motions across the entire image, rather than along individual circles of a given radius, and then a best fit among these measurements. Performing the averaging in two steps, however, enables us to set the thresholds we described above and conveniently eliminate points that yield measurements that fall outside the optimal range of $\{1/1, 1/2, ..., 1/S\}$ pixel/frames. It also allows us to visualize the optical flow algorithm as a function of distance from the radius. Finally, placing points that are a constant distance from the FOE into a single equivalence class yielding a single measurement results in a dimensional reduction which results in finding the maximum-likelihood estimate of only up to 31 measurements (the maximum radius used for a 64x64 image) instead of up to 4096 individual points (minus those in the border area). This makes the computation time for finding the maximum-likelihood estimate negligible, given the usual fast convergence.

Consider again equation 4.1. As a robot approaches an object with constant velocity and constant frame rate, the time to contact in frame units, τ, would be expected to decrease by 1 per frame : $\tau = k, k-1, k-2, \ldots, 2, 1$. If the image distance from the FOE of a given point is y, then the image motion at this point (and around a circle of this radius) $\dot{y}$ would follow a sequence of :

$$\frac{y}{k}, \frac{y}{k-1}, \frac{y}{k-2}, \ldots, \frac{y}{2}, \frac{y}{1}.$$

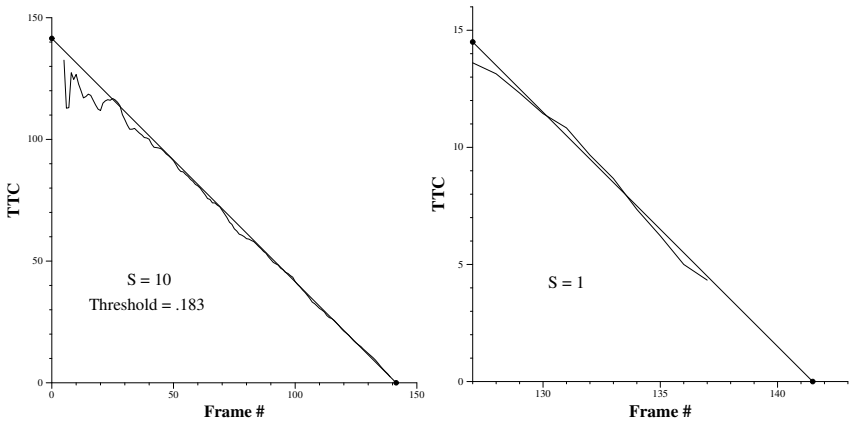


Figure 4.13: Results of imminent collision experiment. Full sequence is shown on left; note that the object did not fill the field of view until about frame 40. Right graph shows special "imminent collision" detector.

This is a harmonic sequence multiplied by the constant y. Thus, it may be possible to create a special-purpose time-to-contact detector which looks for this special sequence of motion at a given radius y from the FOE; such a sequence might result in more stable time-to-contact calculations.

4.2 Selection of Velocities Detected

Section 4.1 calculated time-to-contact in a given collision sequence of 142 frames, using a maximum S value of 10, yielding the velocities $\{1/1, 1/2, ..., 1/10\}$ pixel/frames. This section will examine the ramifications of choosing a different values of S.

4.2.1 Imminent Collision results

An accurate calculation of time-to-contact in Section 4.1 requires a relatively large S of 10 (i.e., motion as slow as 1/10 pixels/frame) to achieve high resolution far away from the target. This resolution is not required, however, if the robot starts out much closer to the target. As an exercise, the linear-time optical flow algorithm was run on the same data with $S = 1$. While only measuring one speed, and calculating time-to-contact, the algorithm can run at over 22 frames per second on a Sun Sparcstation 10/41. In addition, note that since

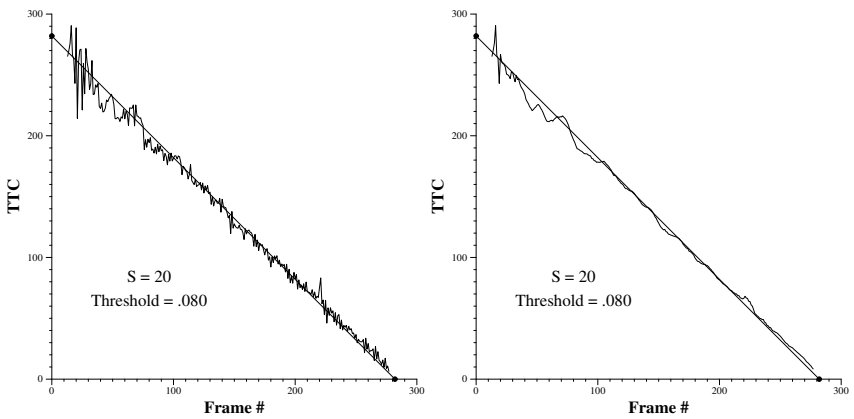


Figure 4.14: Results of time-to-contact experiment for "slow" collision sequence. Left figure is a plot of instantaneous time-to-contact vs. frame number. Right figure is similar except that each value represents the average of the current and previous 7 instantaneous time-to-contact measurements.

there is no temporal aliasing possible with only one detected speed, there need not be the one frame lookahead as is normally done, thus even this small latency is eliminated. The average for the last 10 valid time-to-contact measurements (up to 5 frames before contact) is 141.38, with standard deviation 0.26 frames. Thus this algorithm would appear to be an excellent candidate for an "imminent collision" detector.

4.2.2 Slow Sequence

The collision sequence used in Section 4.1 consisted of 142 frames, at a frame rate of 5 frames a second and 5 cm/sec motion, and a value of $S = 10$ was used to detect motion. A second sequence was taken with the velocity of the robot cut in half, providing a sequence of 283 frames before collision, with each image representing .5 cm of robot translation. S was set to 20, i.e. the slowest motion detected was 1/20 pixels/frame, or only .05 pixels/frame motion. The lower-bound threshold was arbitrarily set to .080. Figure 4.14 shows the instantaneous and averaged measurements for this sequence.

For comparison, the same sequence is tested in Figure 4.15 with $S = 16$ and $S = 20$, both with no threshold. Note that results improve with a S value of 20 over an S value of 16; i.e. this implies that the algorithm is valid for motions as slow as .05 pixels/frame.

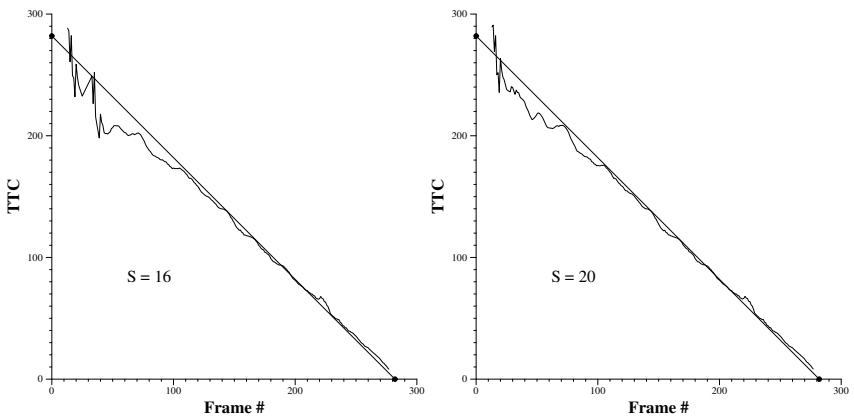


Figure 4.15: Results of time-to-contact experiment for "slow" collision sequence for $S = 16$ and $S = 20$, and no threshold.

(The lower-bound threshold still improves performance however). Note that the algorithm gives a time-to-contact result that is within 1-5% of actual well over 200 frames away from contact.

4.2.3 Velocity-subset results

As noted in Section 3.4, it is not necessary to calculate all velocities $v \in \{1/1, 1/2, 1/3, ..., 1/S\}$ pixels/frame. If a velocity $1/k$ is omitted, then ideally any pixel that would have been assigned velocity $1/k$ will be assigned a velocity $1/(k-1)$ or $1/(k+1)$ instead. Our basic assumption in performing matches is that the motion with the best match value is the correct motion; we do not make any assumptions about the remaining match values (in particular about any "next best" matches). The fact that the match values themselves are not required to be a correct reflection of what these values "should" be is a tremendous advantage of this algorithm. Gradient-based approaches employing the intensity constraint equation generally require that the measured grey-level intensity values of individual pixels are in fact accurate and stable from one frame to another, otherwise numerical differentiation will fail.

Conversely, the correlation-based algorithm only requires that the correct motion have the best matching shifted patch value. This approach basically assumes that there be a

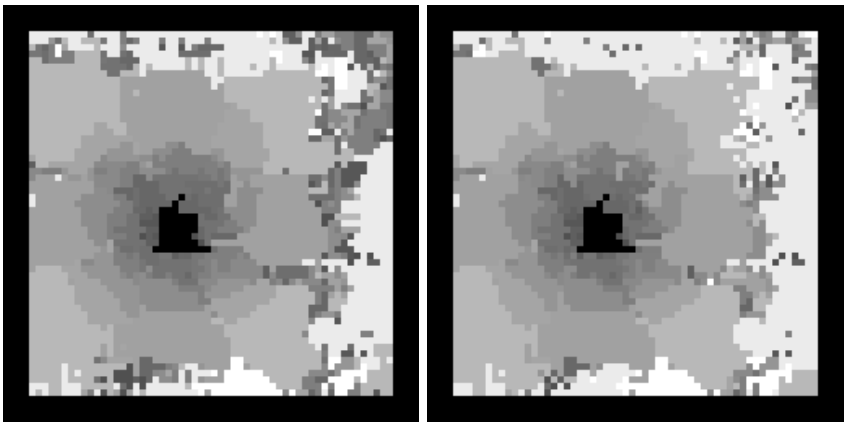


Figure 4.16: Greyscale log plots of the magnitude of the motion vectors for frame 126 using velocities $\{1/1, .., 1/10\}$ *except* for 1/2 pixels/frame. Left plot is without heuristic antialiasing, right is with.

tested motion somewhere near the actual motion. However, if the algorithm is run without attempting to calculate a given velocity $1/k$, a pixel with actual velocity near $1/k$ may have an incorrect velocity (in particular one that is neither $1/(k-1)$ or $1/(k+1)$) chosen instead. Figure 4.16 shows a greyscale log plot of the magnitudes of the motion vectors for frame 126 of the standard collision sequence of Section 4.1, using the normal parameters, except that velocity 1/2 pixels/frame is omitted. The left of Figure 4.16 is without temporal antialiasing, the right is with temporal antialiasing. Many of these slower, incorrect velocities fail the "fastest motion direction or continuity" check of the temporal antialiasing heuristic, and are corrected by the antialiasing. The effectiveness of the temporal antialiasing on the time-to-contact calculations is discussed in detail in Section 4.4.

Figure 4.17 shows the results for the entire collision sequence using the normal parameters, except that velocity 1/2 pixels/frame is omitted. The left of Figure 4.17 is without temporal antialiasing, the right is with temporal antialiasing. In both graphs, there is a crossover point where the time-to-contact changes from too high to too low, resulting from detected motion that is too slow or too fast respectively. Before the crossover point (earlier in the sequence), velocities lower than 1/2 pixels/frame are sometimes selected for an

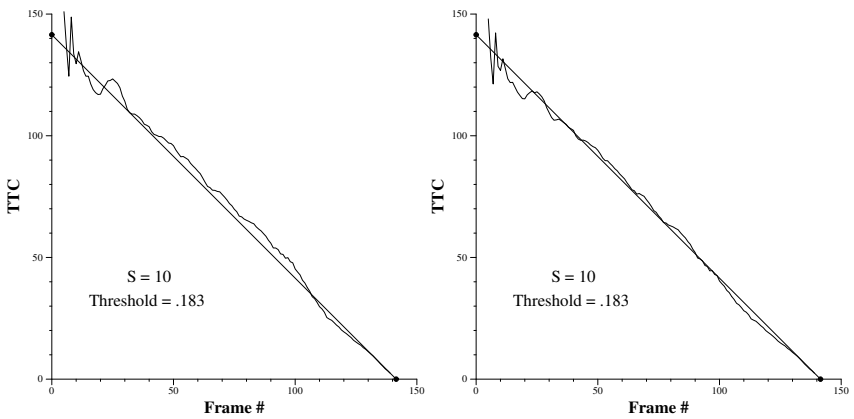


Figure 4.17: Time-to-contact experiment using velocities $\{1/1, .., 1/10\}$ *except* for 1/2 pixels/frame. Left is without heuristic antialiasing. Right graph is experiment with antialiasing.

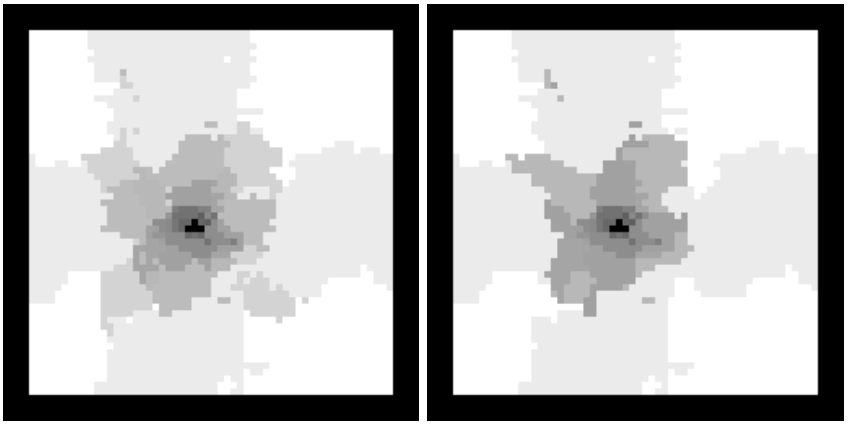


Figure 4.18: Greyscale log plots of the magnitude of the motion vectors for frame 126 using velocities $\{1/1, .., 1/10\}$ with 1/2 pixels/frame (left) and without (right).

actual velocity 1/2 pixels/frame since that velocity is not available for the algorithm. In particular, velocity 1/3 pixels/frame is often chosen. After the crossover point, velocity 1/1 pixels/frame in often chosen instead of 1/2 pixels/frame, resulting in a higher than actual motion and subsequent lower time-to-contact.

At higher actual velocities, the slower speeds do not give good matches, and are not chosen (even without the temporal antialiasing). At frame 126, the effects about cancel (Figure 4.18).

Figure 4.19 shows the results of the fast collision sequence with the normal parameters except that 1/3 pixels/frame is not calculated (left) and 1/4 is not calculated (right),

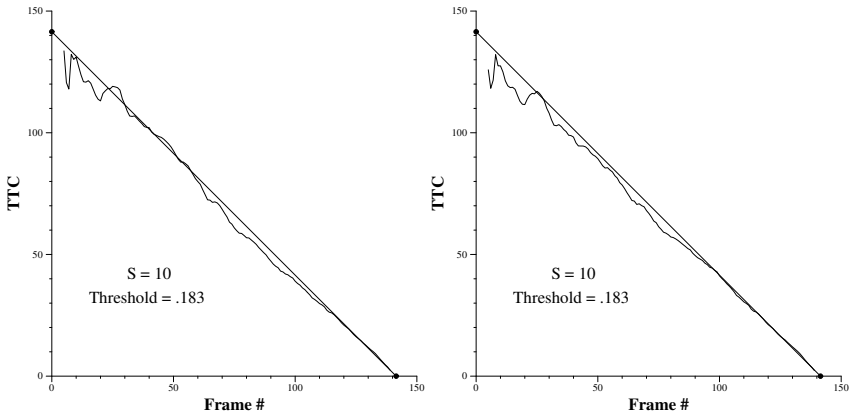


Figure 4.19: Time-to-contact experiment using velocities $\{1/1, .., 1/10\}$ *except* for 1/3 pixels/frame (left) and 1/4 pixels/frame (right).

respectively.

4.3 Robustness Versus Noise

Real-time robotic vision has two essential requirements, real-time performance and robustness in real-world situations. The former condition has been addressed by the development of a linear-time optical flow algorithm. The second condition demands that algorithms be extremely robust against noise. In particular optical flow methods based on numerical differentiation are notoriously sensitive to noise. Although high-resolution CCD cameras are available, they can be very expensive. Optical flow methods employing extensive filtering can be very accurate but are computationally intensive [BFBB93]. Even considering the yearly exponential growth in available computational power, practical applications may still be elusive considering the cost of a quality CCD camera and high-resolution framebuffer. The following examples demonstrate the algorithm's degradation of performance for the time-to-contact experiment of Figure 4.12 given dropped bits of precision, additive Gaussian noise, and additive, evenly distributed random noise. For reference, the noise distribution of the sequence was estimated by calculating the standard deviation of the pixel intensity differences between pairs of still images at the very beginning of the sequence (before motion

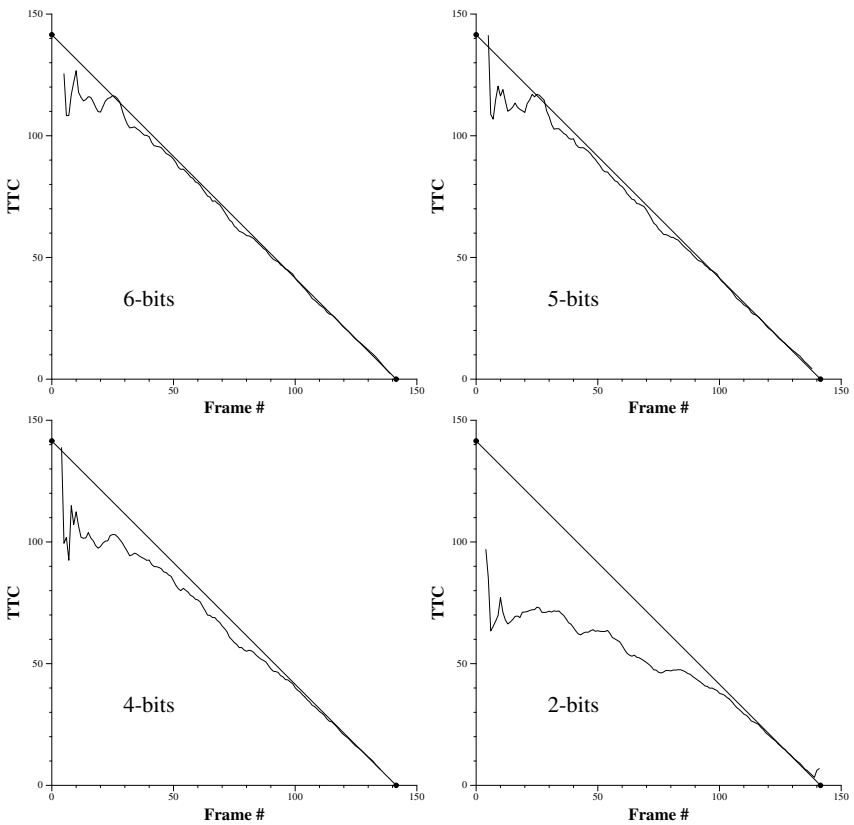


Figure 4.20: Time-to-contact experiment with "fast" sequence with 6-bit, 5-bit, 4-bit, and 2-bit precision images. S was set to 10, and lower threshold = .183.

began) using the identity :

$$S_d(X) = \sqrt{(S_d(X_1)^2 + S_d(X_2)^2)/2}$$

where $S_d(X_1)^2 + S_d(X_2)^2$ is the measured variance of the image pair differences (assuming two equal, independent noise distributions X_1 and X_2). For this sequence the noise appears to be approximately Gaussian distributed with a standard deviation of about 1.66σ; thus, the "actual" noise of the fast collision sequence is roughly an additional 1.66σ above any added noise and/or degradations.

Normally, each pixel's grey-level intensity is represented by a value in the range of $\{0,..,255\}$, which is representable by an 8-bit unsigned integer. Figure 4.20 demonstrates the algorithm's performance with the fast collision sequence reduced to 6, 5, 4, and 2 bits

of pixel data per frame (a maximum of 64, 32, 16, and 4 grey-level values respectively). Bits were "rounded" based on the value of the next most significant bit. There is virtually no degradation of performance at all at 7 bits precision (not shown), very little at 6 bits precision and the results degrade gracefully with further reduced precision. In general, only about half of the possible grey levels are actually used by a particular image in this sequence for a given bit precision.

It is particularly interesting to note the performance of the algorithm with the precision reduced to only 2 bits. The extremely limited resolution of these images is evident in Figure 4.21. In fact, the vast majority of the image sequence actually only uses 2 colors, corresponding to grey levels 128 and 192 respectively, and are thus effectively 1-bit bitmaps. Clearly, with only 2 grey levels to use, techniques based on numerical differentiation cannot be used (and are undesirable with less than 8 bits precision anyway [To92b]). However, the pixel-matching technique can still be successful based on the simple argument presented in Figure 3.4. Given that the images are effectively 1-bit, the algorithm could be viewed as performing a degenerate form of binary feature matching, as in [BLP89a]. [ZL93] notes many of the computational advantages in using a 1-bit feature map (in their case, an edge map) over full 8-bit grey-level images. The optical flow for these images is shown in Figures 4.22-4.24.

For the fast collision sequence, the time-to-contact value is at worst about within a factor of 2 of the correct value and becomes more accurate as contact approaches. At 90 frames away the algorithm produces a time-to-contact 70% of actual, and at 50 frames away it is 85% of actual. At 40 frames from contact the algorithm is within three frames of actual, and at 25 frames from contact it is within a single frame. The algorithm maintains this accuracy up until two frames before contact (same as that for the full-resolution sequence), at which point it diverges slightly.

If the object is known to be close, then the detection of slower motion may be omitted

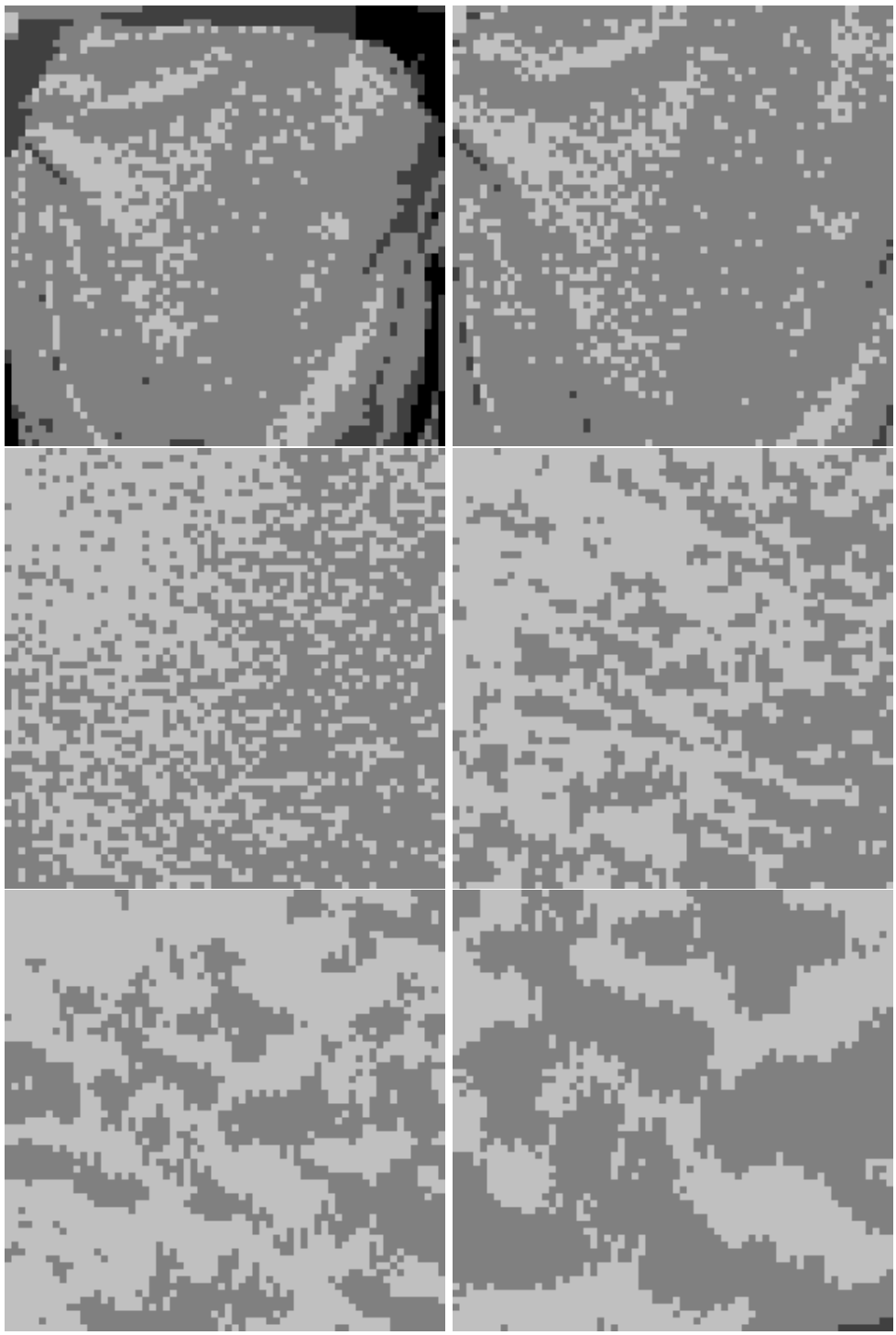


Figure 4.21: Image of the chair (in row-major order) at frames 12, 41, 96, 126, 134, and 139 reduced to only 2 bits of precision. The majority of the images actually only use 2 grey levels, and are effectively 1-bit bitmaps.

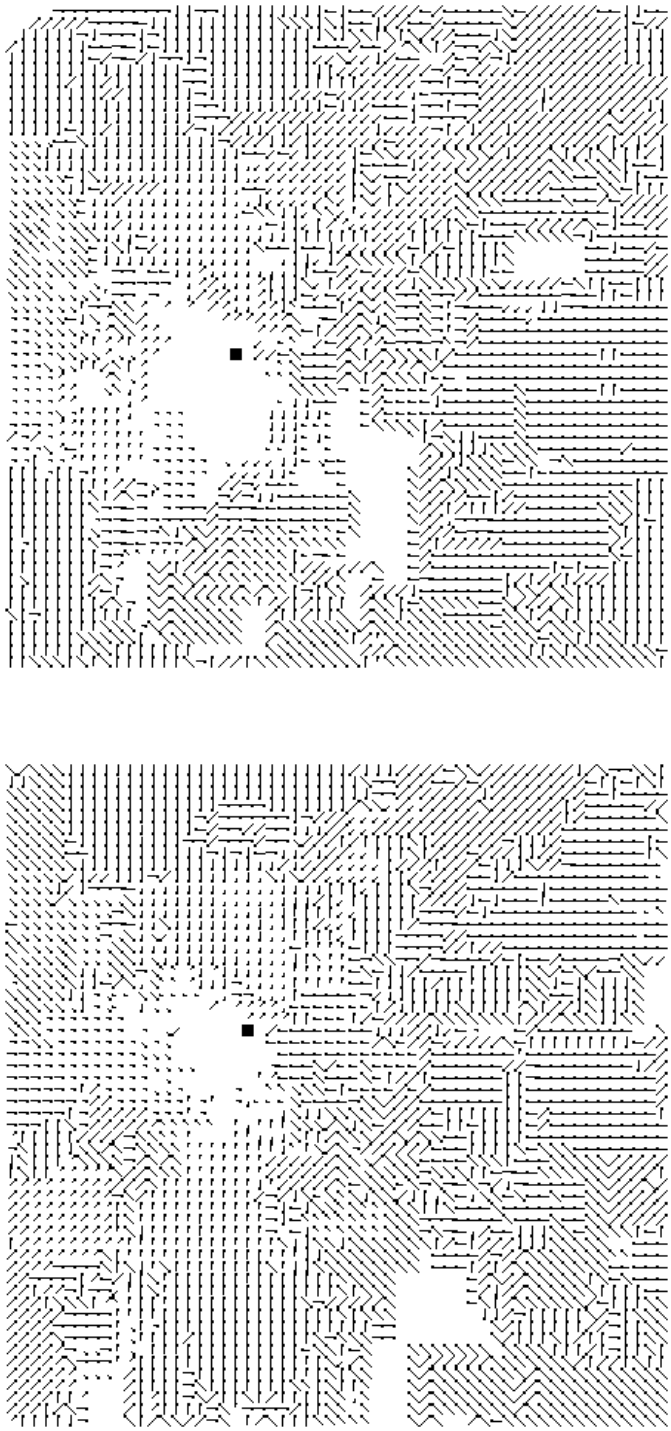


Figure 4.22: Optical flow of the collision sequence for frames 12 and 41 reduced to 2 bits precision.

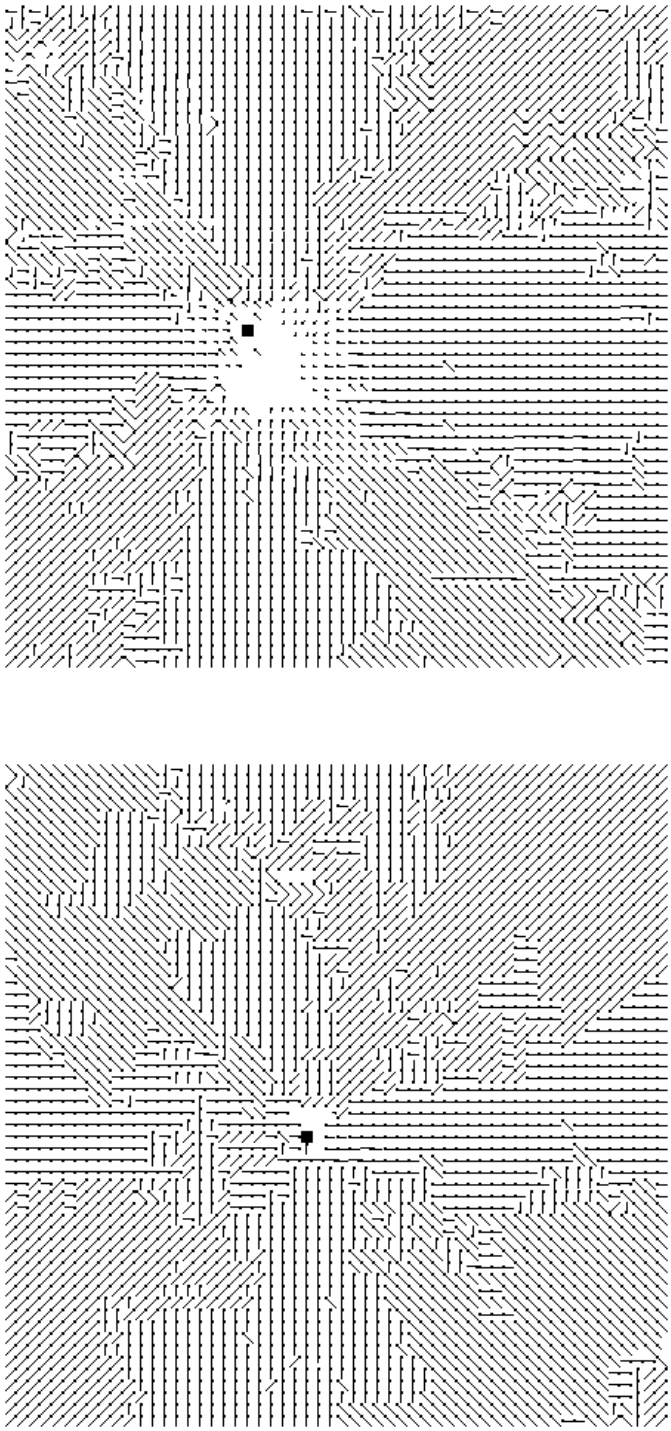


Figure 4.23: Optical flow of the collision sequence at frames 96 and 126 reduced to 2 bits precision.

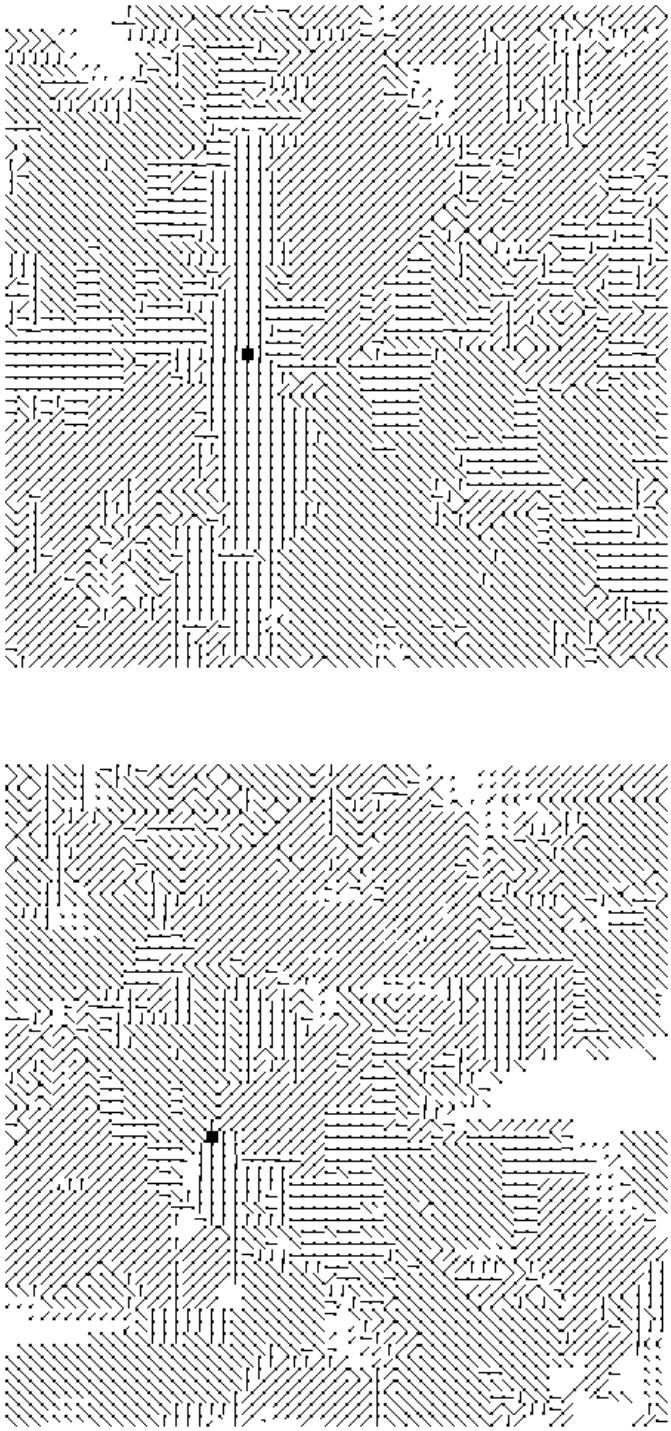


Figure 4.24: Optical flow of the collision sequence at frames 134 and 139 reduced to 2 bits precision.

with only a minimal loss of accuracy. For example, the algorithm run on the fast collision sequence reduced to only 2 bits precision (effectively 1-bit), calculating 4 speeds per frame, can run on a Sparcstation 10/41 at 10 frames per second, and produces subframe accuracy for the last 15 frames of the sequence (excepting the very last two frames before contact). The fact that such accuracy is possible given such poor input data, only four detected velocities $(1, 1/2, 1/3, 1/4)$ and only eight directions suggest relatively simple hardware implementations, in terms of modest computational power, low-quality CCD camera, and low-precision framebuffer requirements. An important point here is that we are not merely using some "quick and dirty trick" to determine time-to-contact, nor are we tracking the motion of feature points, which might be undesirable due to the difficulty of extracting reliable feature points (and assuming that there are enough such points in the image for subsequent processing). We are using the same exact optical flow algorithm, just on very poor data.

Figures 4.25, 4.26 demonstrates the algorithm's performance with 2, 4, 6, 8, 10 and 12 σ additive Gaussian and evenly distributed random noise, respectively. The Gaussian (normal) distribution was produced using the standard formula from [K81], with mean zero and variance σ^2. For evenly distributed random noise, a lookup table of standard deviations indexed by the bound x for a random variable evenly distributed between $\{-x, .., 0, .., x\}$ (inclusive) was computed using the standard formula for variance: $V(x) = E(x^2) - E(x)^2$, where $E(x^2)$ is twice the sum of squares from 0 to x inclusive divided by $2x + 1$, and $E(x)^2 = 0$. In all cases a pixel's grey-level value is clipped to lie between 0 and 255 (i.e. representable by 8 bits) before processing; the standard deviation of the added noise is that *before* clipping. There does not seem to be a significant difference between the effects of Gaussian and random noise for this algorithm.

Of course, these are only typical runs. Random noise, especially that with a high standard deviation, may produce slightly different results with different runs. In particular, the

FOE detector sometimes has trouble with the very last few frames of a sequence with very high noise. In some cases, it produces a FOE which is outside the actual image, which invalidates the time-to-contact estimate. In other cases, the FOE is placed in an area where all motion estimates are greater than or equal to 1 pixel per frame, which is above the upper-bound threshold.

For example, after adding 8σ of Gaussian-distributed noise (right center in Figure 4.25), the time-to-contact calculations are within 3 frames of actual 40 frames before contact, and are within a single frame of actual 25 frames before contact. The algorithm maintains this accuracy up until 8 frames before contact, after which point the measurements are invalid. The signal-to-noise ratio is 1.74 or less for the last 40 frames, and is 1.6 or less for the last 25 frames.

A lesson learned is that instead of relying on the accuracy of individual measurements, each one of which is very difficult to obtain and is sensitive to noise, it may be better to use some combination of individual measurements, each one of which is robust but not necessarily very precise. This philosophy seems quite consistent with biological implementations, such as the spatial integration of simple correlation-type motion detectors found in the fly [EB93] (although the fly does not appear to explicitly calculate time-to-contact [BB88]).

4.4 Effects of Heuristic Temporal Antialiasing

Section 3.3 discussed the temporal aliasing which can occur as a result of the space-time tradeoff of the linear-time optical flow algorithm. This section will demonstrate that for the application of time-to-contact, the heuristic described is effective in improving results. In this case, all motion is radially directed away from the FOE, so the type of temporal aliasing as described in Figure 3.6 cannot occur in general. Therefore one might expect that the heuristic would not help. However, in cases where the direction of the fastest possible motion disagrees with the motion with the best match, the temporal continuity constraint does

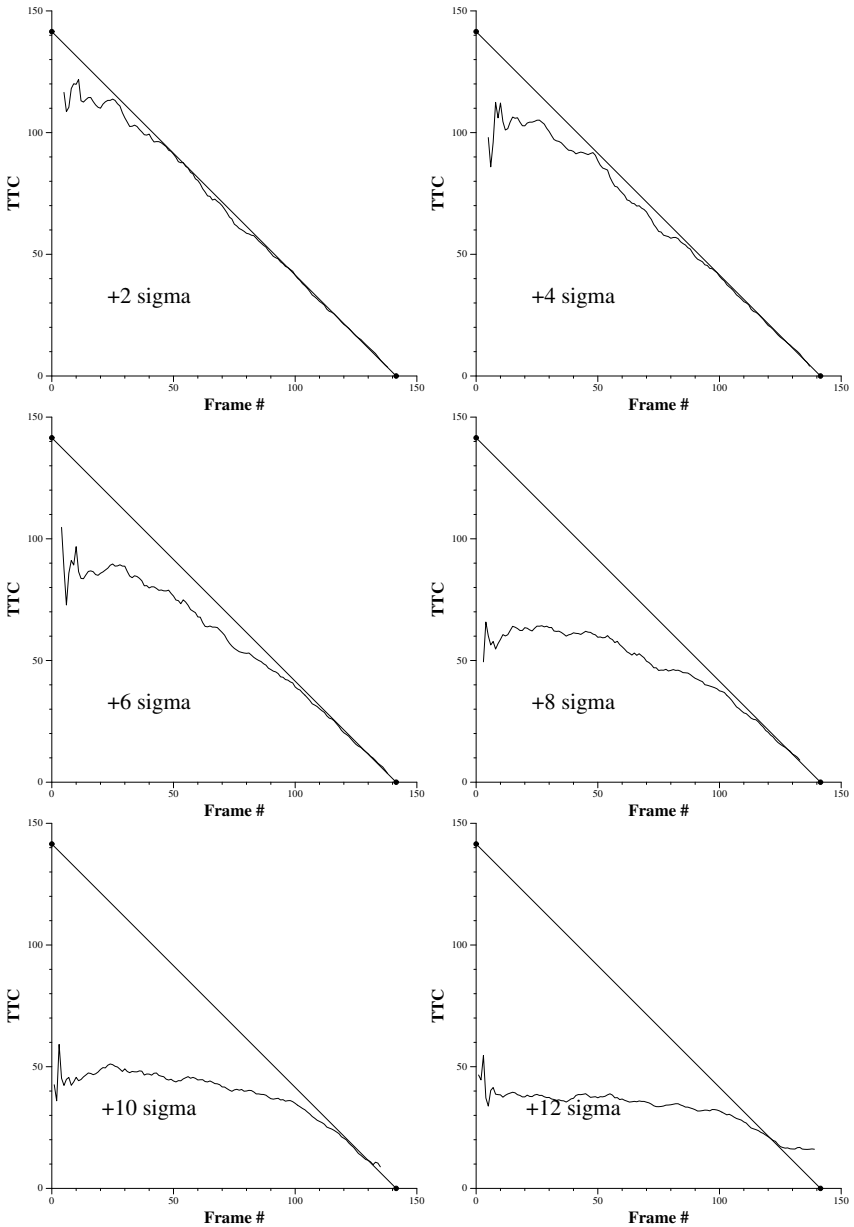


Figure 4.25: Time-to-contact experiment with "fast" sequence with additive Gaussian noise of 2, 4, 6, 8, 10 and 12 σ.

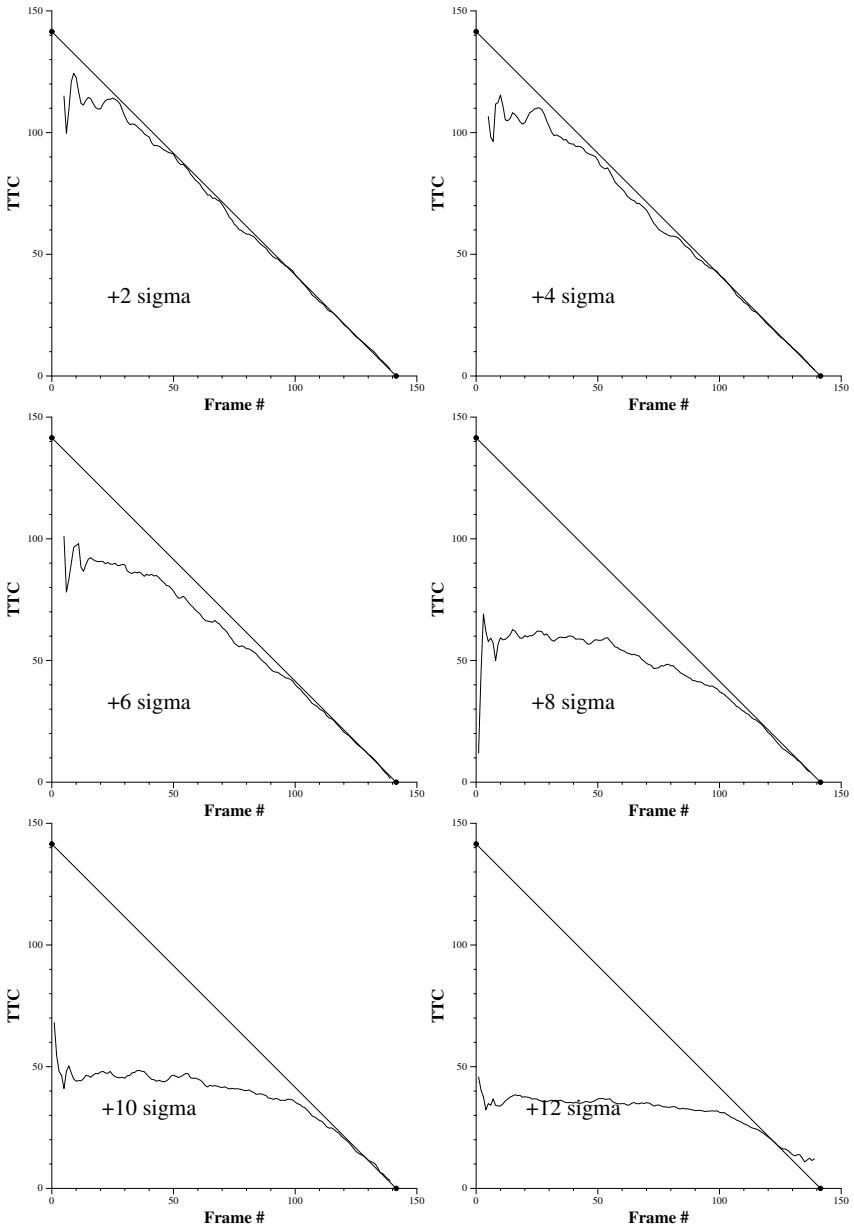


Figure 4.26: Time-to-contact experiment with "fast" sequence with additive, evenly distributed random noise of 2, 4, 6, 8, 10 and 12 σ.

provide additional information. In particular, nearly all motion is continuous across time in this particular sequence, although of course that cannot be guaranteed in general. The time-to-contact algorithm itself does not exploit the temporal continuity of the individual motions detected. The very last step of the time-to-contact algorithm does average the last 8 time-to-contact measurements, however this final averaging step operates only on the final time-to-contact values and not on the individual velocity measurements. Otherwise, the only other "temporal coherence" that is currently used is that which is implicit in the algorithm itself, i.e. that a motion of $1/n$ pixels/frame continue for n frames before being detected. But in this case the single frame of look-ahead is sufficient to exploit temporal continuity. This is very advantageous, because an algorithm which exploits temporal continuity in a more aggressive manner, such as smoothing across time using a temporal Gaussian, may require the input of several additional images before an output is produced. For real-time robotics, this is generally unacceptable ([FL93] recognizes this by using recursive filters instead of a temporal Gaussian). Conversely, a single frame of lookahead is a relatively modest requirement.

Figure 4.27 shows some grey-level coded maps of the actions taken as a result of temporal aliasing (as detected by the algorithm). White areas correspond to pixels whose best-matching motion have the same direction as the fastest possible motion detected for that pixel. Light grey pixels move in directions that disagree with the fastest zero-or-one pixel detected motion for that pixel, but continue for an additional frame in their current direction and so are not altered. If neither of these conditions hold then we invoke our temporal antialiasing heuristic. Among all remaining motions faster than the current best-match for the pixel in question, we pick the motion with the best matching value such that either 1) it has the same direction as the fastest zero-or-one pixel motion detected for that pixel, or 2) does continue in its current direction for an additional frame. One of the latter conditions must hold for some motion faster than the current best matching motion (exclusive) and

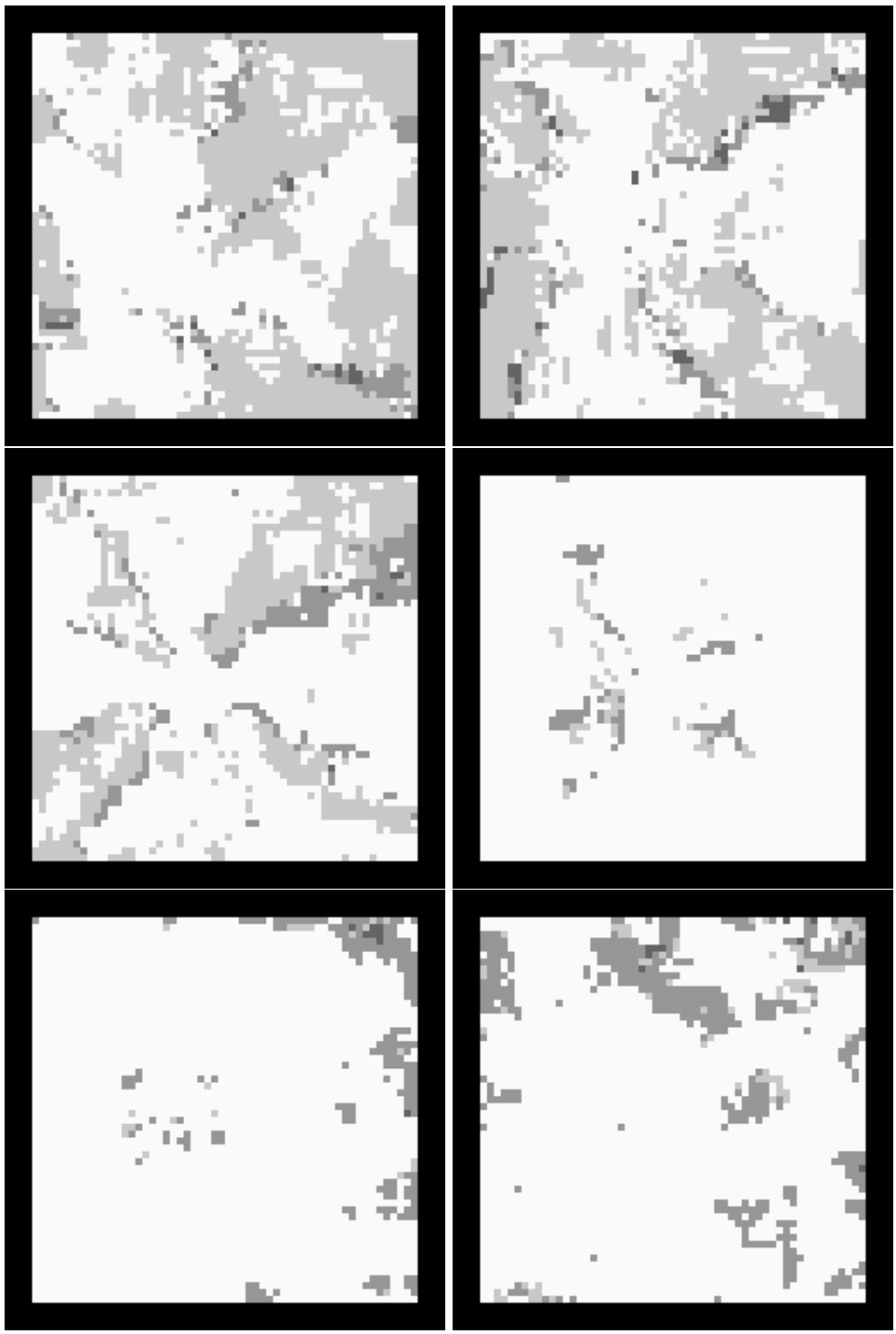


Figure 4.27: Grey-coded map of temporal aliasing (in row-major order) for frames 12, 41, 96, 126, 134, and 139. White: Best-match direction agrees with fastest motion. Light grey: Best-match direction continues one frame. Medium grey: Accept a faster motion that matches direction of fastest. Dark grey: Accept a faster motion whose direction continues one frame.

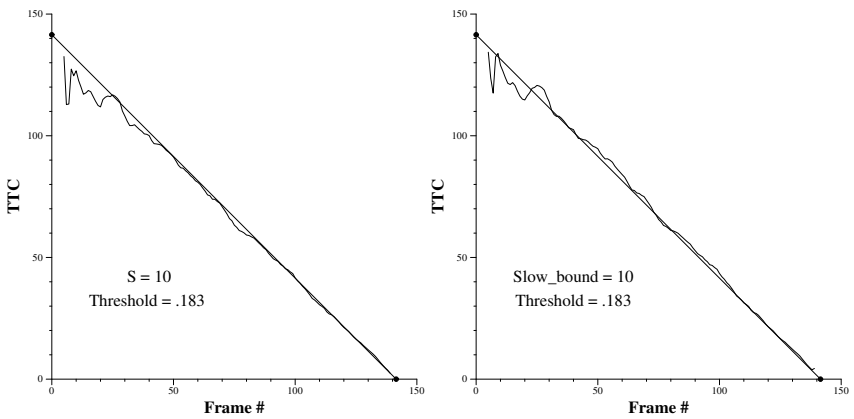
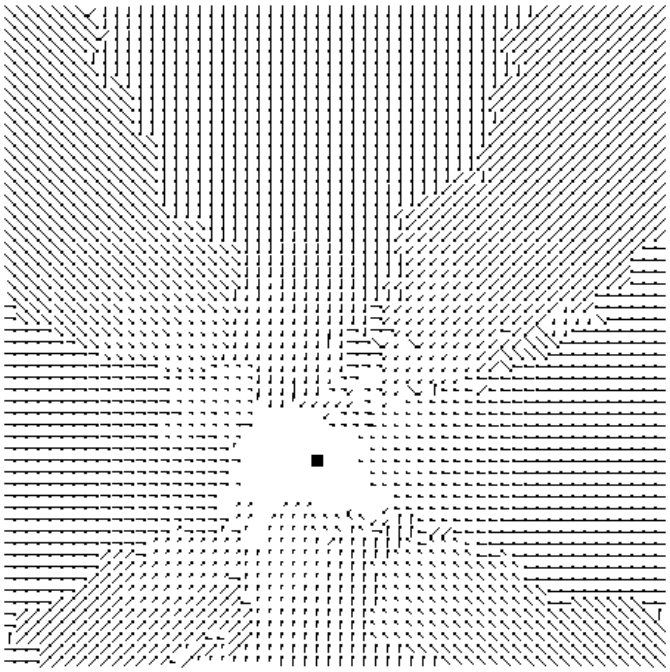


Figure 4.28: Left is normal time-to-contact experiment with heuristic antialiasing. Right is experiment without antialiasing.

the fastest zero-or-one pixel motion because the latter is itself a valid candidate. Medium grey areas correspond to the first condition, and dark grey areas correspond to the latter.

The effect of these corrections can be seen in Figure 4.28; the left image is the normal result from the right of Figure 4.12, and the right is the same algorithm run without the temporal antialiasing. Without the temporal antialiasing the time-to-contact measurements are slightly higher than they should be, resulting from optical flow measurements that are slightly slower than they should be. Figure 4.29 shows the change in optical flow directions for frame 41. Since the differences in magnitudes cannot be seen in these plots, Figure 4.30 shows a log plot of the magnitudes of the optical flow vectors (the differences would not be visible in a straight plot).

As noted in Section 4.1, motions along the diagonal represent $\sqrt{2}$ times as much motion as those along NEWS directions. The question arises, what happens for motions that are $1/X$ (for integral X) pixels/frame along a diagonal, or motions that are $\sqrt{2}/X$ pixels/frame along one of the NEWS direction? These motions do not have an ideal match, but will instead pick the "closest" match to its actual motion. Without the temporal antialiasing, the best match in these cases tend to be along the diagonals. With temporal antialiasing, some of these motions are corrected to lie along the NEWS directions. Interestingly, these



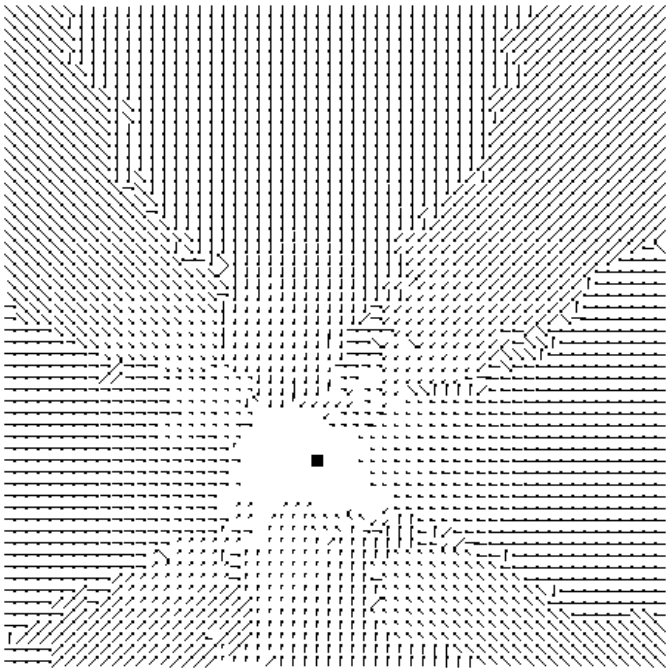


Figure 4.29: Optical flow at frame 41 with and without temporal antialiasing.

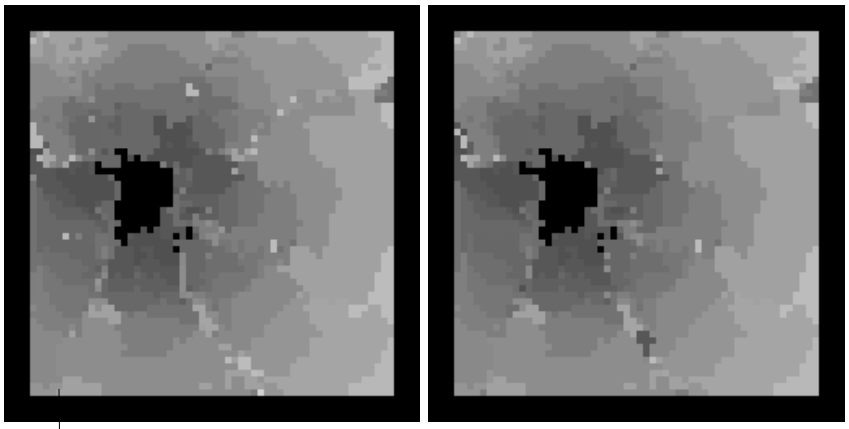


Figure 4.30: Greyscale log plots of the magnitude of the motion vectors. Left is with heuristic antialiasing, right is without.

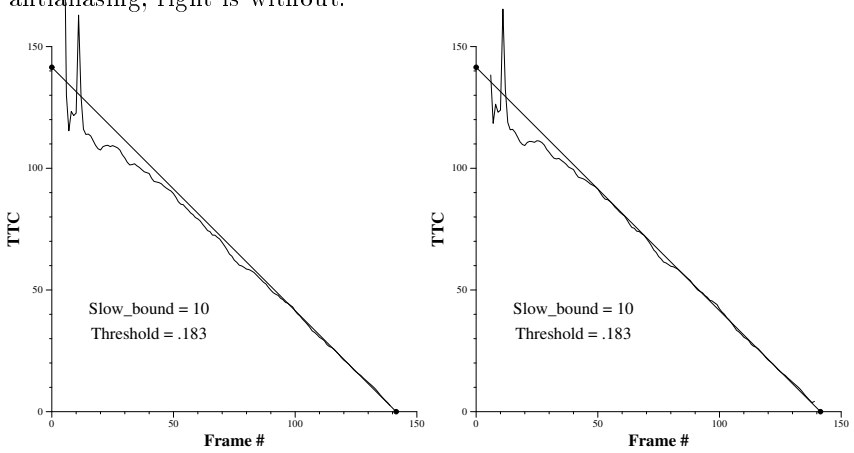


Figure 4.31: Left is time-to-contact experiment with only a fastest-first-agreement rule. Right is experiment with only a temporal continuity rule. See text for details.

"corrections" can be made due to both conditions 1) (the fastest-motion constraint) and 2) (the temporal continuity constraint) as described above.

Figure 4.31 shows the result of the algorithm when using only one of the two possible antialiasing conditions, the fastest-motion constraint or the temporal continuity constraint, but not both. I.e., if the direction of motion of the best-match motion disagrees with that of the fastest motion, the corrected, faster best-match motion must match the direction of the fastest motion (Figure 4.31 left) or it must continue for one frame (Figure 4.31 right). For this example, the temporal continuity constraint performs nearly as well as the full heuristic. This is not surprising for the time-to-contact application where motion is indeed

continuous for many frames. In general however, motion may not continue into the future for several frames, so we do not extend the temporal continuity further than one frame.

4.5 Rotation examples

Algorithms and techniques that work for crash detection are often easily modifiable to apply for rotation computation as well (for example see [PVT91]). Recall equation 4.1 :

$$\frac{y}{\dot{y}} = -\frac{Z}{\dot{Z}} = \tau \tag{4.4}$$

For a given approach to an object, at any instant there will be a corresponding time-to-contact τ, and a point a fixed distance y from the FOE will have velocity y/τ. Thus the instantaneous velocity of a point will be linearly proportional to its distance from the FOE. Similarly, by simple geometry for a rotating circle, the instantaneous rotational velocity of a point is proportional to its distance from the Focus of Rotation (FOR). This can be seen in Figure 4.32. This suggests that it might be possible to use a time-to-contact detector as a rotation detector simply by calculating motion in the normal way, rotating the optical flow vectors by 90 degrees so that they point away from the focus of rotation (now the focus of expansion), calculating time-to-contact normally, and finally performing a simple units conversion.

As noted in Figure 4.33, normally instantaneous rotational velocity can be computed by the $\arctan(\dot{y}/y)$, where $\dot{y}$ is the instantaneous tangential motion at that point on a circle of radius y. By rotating the motion vector by 90 degrees, $\dot{y}$ becomes the instantaneous divergence of that point. Computing τ on this new optical flow field gives us a robust estimate of $y/\dot{y}$, which can be then inverted and used in calculating rotational velocity. Of course, one would normally construct a rotational motion detector directly without translating to and from diverging motion, however it is interesting to note that much of the same machinery can be reused since it is trivial to convert a time-to-contact detector into a rotational

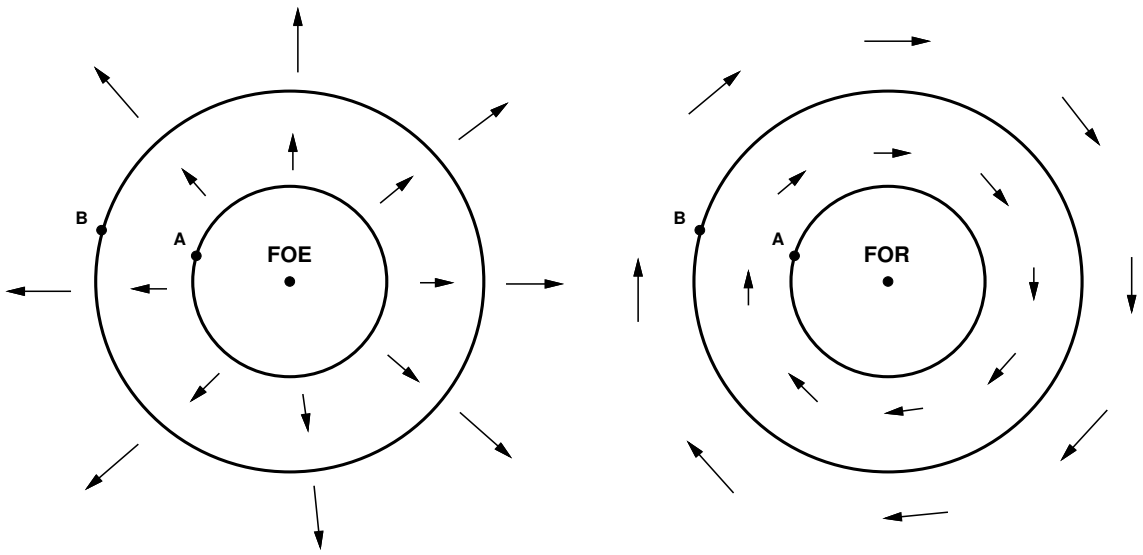


Figure 4.32: Relation of expansion and rotation. If point B is twice the distance as point A from the foci of expansion (FOE) and rotation (FOR) respectively, its instantaneous velocity will be double that of A's for both cases.

velocity detector.

This algorithm was tested on a single image from the SRI translating tree sequence (Figure 4.34). Many tools were tested to rotate the images, but unfortunately many produced unsatisfactory results, such as shifting the image slightly in addition to rotating the image. Others clearly used naive algorithms as well in rotating an image. A requirement in performing such a rotation is to perform an inverse transformation (rotation in this case) from the new image to the old image, otherwise there may be unmapped pixels in the new image. A naive algorithm (which some public domain packages seem to use) is to simply use the old pixel value, but this produced unsatisfactory results such as "block-moves" of rectangular areas of pixels. The method chosen for rotating the images was to use a weighted average of the nearest four pixels for each point in the reverse transformation, as was used in Figure 4.7. This produced high quality rotated images, far more than required considering that the original 256x256 images are also subsampled to 64x64 before processing. Background points were left black.

A sample optical flow field superimposed on the rotated image appears in Figure 4.35

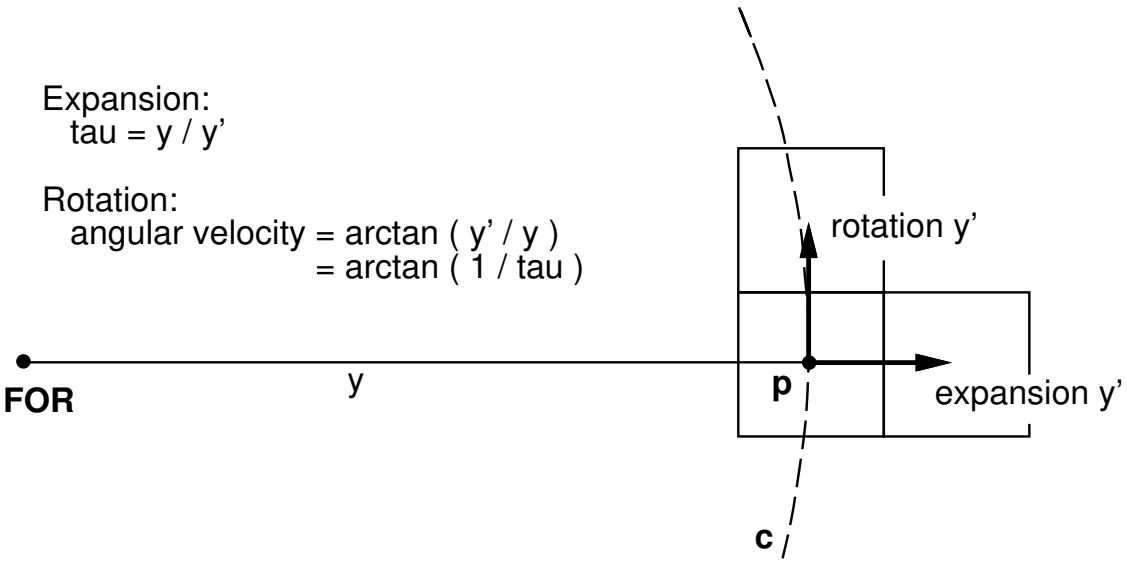


Figure 4.33: Conversion of expansion motion vector to rotation. Point p undergoes an instantaneous rotation along circle c, measured as a tangential motion. By flipping this motion vector by 90 degrees, the machinery for time-to-contact calculations can be easily reused for the calculation of rotational velocity.

(note that the full resolution image was used to enhance visibility). (Points just off the actual image are calculated as having moved because the background is perfectly textureless and thus the matching window "captures" [RA86] some off-image points). For this sequence, the images were rotated clockwise by .5 degrees per frame. Using the same parameters as for the fast collision sequence ($S = 10$, lower-bound threshold of 1.83, and upper-bound threshold of 1.0, and average of last 8 frames for final estimate), a result of .535 degrees per frame is calculated, or an error of 7%.

This particular algorithm does not attempt to identify rotational motion per se, although it can calculate the amount of rotation given that rotation is known to exist. It is possible that an algorithm could hypothesize rotational motion and then calculate the deviations from this hypothesis throughout the image, with a low degree of deviations indicating rotational motion. For example, the focus of rotation could be calculated for both a hypothesized clockwise and a counter-clockwise motion; in general only one of the two will lie within the

Figure 4.34: Rotating the SRI tree sequence. The left image is rotated CW by 7.5 degrees to produce the right image.

central area of the image. Since rotation detection however is not considered as important as collision detection (since rotation about the visual axis is normally impossible for a mobile robot fixed on the ground plane), this particular application was not optimized further.

4.6 Summary

Our three criteria for practical robotic vision were: robustness, real-time performance, and finally accuracy. Although the linear-time optical flow algorithm has been shown to be very fast and robust, one limitation is that it does not give truly real-valued image velocity measurements. Therefore, it is not obvious that it can be used for a wide range of robotics vision tasks. We have shown that in the context of calculating time-to-collision, remarkably accurate measurements may be made by integrating the resulting optical flow measurements across space and time. Furthermore, this quantity is calculated independently at several points in the optical flow field and experiments have shown that these time-to-contact measurements are relatively constant regardless of where in the optical flow field they are measured. This means that there are dozens of measurements all of which confirm a single hypothesis of the time-to-collision, resulting in the desirable property of extreme redundancy (and thus robustness) in the measurements. Furthermore, the calculations for

time-to-collision are extremely fast, even when compared to the real-time linear-time optical flow algorithm. Several additional experiments have demonstrated that the time-to-collision calculations are extremely robust against noise, to the point where even bitmaps may be used as input to the algorithm. This particular application demonstrates that, at least for this application, the potential limitations of non-real-valued optical flow measurements do not manifest themselves.

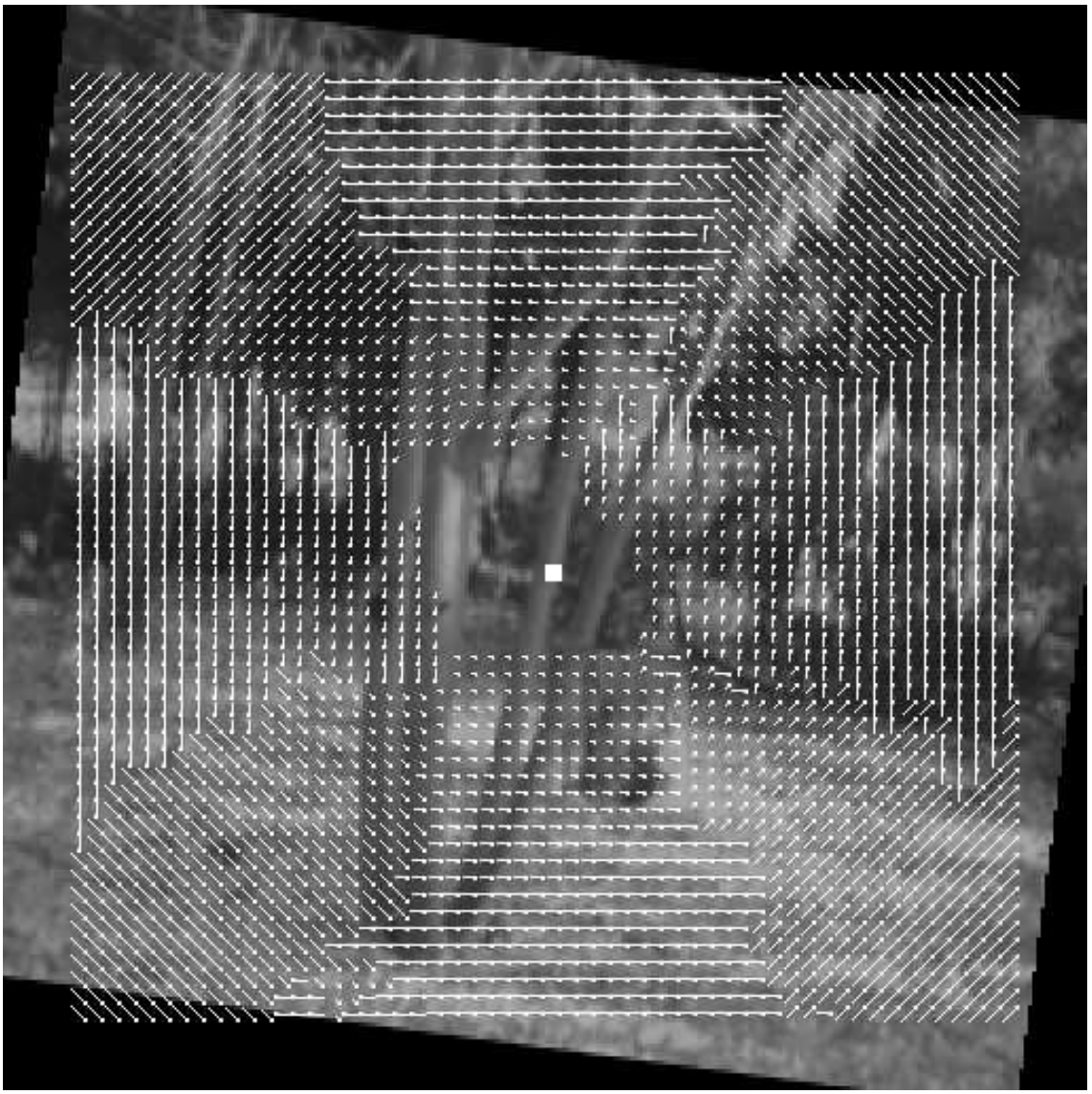


Figure 4.35: Optical flow superimposed on the right of Figure 4.34.

Chapter 5

Real-Time Performance and Hardware Considerations

[M82] defines three levels at which an information processing device can be understood :

1) Computational theory

2) Representation and algorithm

3) Hardware implementation

The first level describes the goal of the computation and the logic by which it is carried out. The second level describes the representations of the input and output and the algorithm for the transformation between the two. Finally the third level describes the physical realization of the algorithm. Research concerned with *what* and *how* information may be computed from images often focuses only on just the first and second levels. Computer vision is no exception to this rule. Given how different neuronal hardware is from CCD sensors and computer chips, it is almost mandatory.

However, for computer vision to ever become practical, some consideration must be given to performance issues. The vast computational requirements of real-time computer vision make implementational issues more important than they are for many other fields of computer science. From [O85] :

> "It is only by using machines with [levels of performance in the Giga operations per second] that significant real-time image processing becomes a realistic

proposition."

Entire workshops [CAMP93] and specialized texts [Uhr87], [O85] are devoted to custom hardware or massively parallel implementations of computer vision algorithms. For practical mobile robot vision, considerations must be given to efficiency. One factor related to performance is an efficient representation and algorithm used to analyze the image(s). In the case of the linear-time optical flow algorithm, this has been addressed by the space-time tradeoff which converted a quadratic search in space into a linear search in time. The other important factor is the hardware implementation. Often, computer vision literature is theoretically oriented in nature and gives little consideration to these practical implementation issues, which are essential to real-time performance. A major advantage of the linear-time optical flow algorithm is that it can be implemented using only simple integer arithmetic such as shifts, adds, and integer compares, which can execute very quickly compared to floating point calculations. This means that the algorithm is suitable for computation on 1) desktop personal computers, and 2) dedicated image processors. Although this algorithm is suitable for the latter, the controversial view will be taken that vision algorithms (particularly the one described in this thesis) are perhaps better suited for standard desktop computers than specialized processors; since this view is likely to be in the minority, it requires some justification. The first section describes the most computationally intensive portion of the patch-matching algorithm, and the sections following discuss specific software and hardware implementations.

5.1 Box Filter Implementation

About two-thirds of the computational time of this algorithm is consumed during the patch-matching (equation 2.3) stage during which the best matching pixel shift is found. A key part of this stage is known as a *box filter*. This single operation is so important to real-time operation that it will be described in some detail here.

The first step of the patch-shift-match operation is the shifting stage ([BLP89a], [MBLB91]). In a massively parallel machine such as the Connection Machine 2, it might be desirable to implement this as an explicit image shift so that corresponding pixels are located in the same processing node. In the case of a serial processor however, this can be implemented at negligible cost by using normal addressing arithmetic on certain microprocessors. Once the base addresses of corresponding shifted rows are calculated, a single index register can be updated with each new pixel in architectures such as SPARC [Sun87], PA-RISC [hp94], and PowerPC [Mot94a]. Architectures with only base + displacement addressing, such as Alpha [SW94] and MIPS [PH94], require an explicit address add. The former two architectures have existing superscalar implementations, however such implementations with multiple integer or load/store units may require special coding to take full advantage of their abilities (for example, to process two rows in parallel). The latter two architectures generally have superpipelined implementations for a very high clock rate, and do not need special coding, although they are hurt somewhat by their lack of index register addressing.

After the shift stage (whether explicit or implicit) each pair of corresponding pixels are compared. This comparison could be either the sum of absolute differences (SAD) of each corresponding pair of pixels' intensity values, or the sum of their squared differences (SSD). Both sum-of-absolute-differences and sum-of-squared-differences could be implemented with no difference in computational time by using lookup tables for the absolute-value and squared-value functions (although the latter would require somewhat more memory to hold the larger aggregate patch match values). [A87a] chooses to use the SSD for easier mathematical analysis, since the derivatives of the SAD function are discontinuous at zero; [S91] follows suit. However, for this algorithm the sum-of-absolute-differences has been used since it appears less sensitive to large deviations such as edges produced by shadows, and high-frequency noise; the discussion of robust measures in Appendix A reinforces this view as well. If a lookup table is not convenient (as it might not be in certain custom hardware), ab-

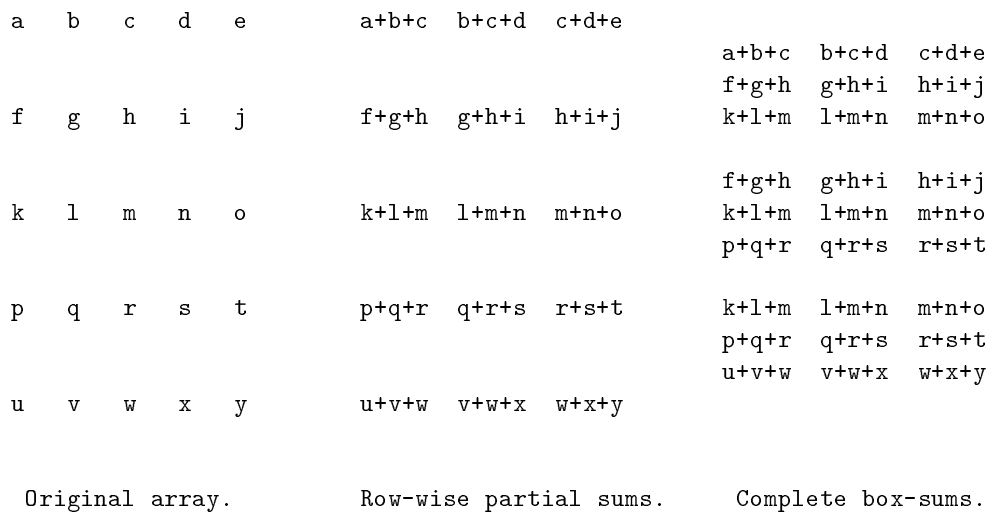


Figure 5.1: Demonstration of a 3x3 box filter.

solute value can be easily calculated by the expression : `abs(x) = (x^(x>>31))-(x>>31)`, where `^` is the standard C operator for bitwise XOR and `>>` represents an arithmetic shift. Other schemes such as in [BLP89a] suggest the ANDing of certain features such as edges. Edges are used in some methods of motion detection (as in [Hil84], [P90], [WWB88], [ZL93]) but are not used here because the resulting optical flow density would then depend on the density of the features in the image sequence, which might be too low for subsequent processing, or in any event would vary from one image sequence to another. It would also have required some high-level feature detection, which would have added further to the computational time used. However, note that the performance of the algorithm when run on images reduced to 2 bits precision, resulting in effectively a 1-bit feature map, is still very good, at least when the camera is close to the target; see Section 4.3.

The next step is referred to as the "Excitation" stage in [BLP89a], or the local neighborhood summation stage, as demonstrated in Figure 5.1. Here each element of the absolute difference (or squared difference) array is replaced with the sum of all elements in a local neighborhood $\nu * \nu$ in size. In Figure 5.1, the neighborhood is 3x3 in size. Note that the

final box-sums array is not defined on the very edges of the image, since that would require pixels not available in the image itself. A naive implementation would perform an explicit $\nu * \nu$ summation for each pixel location; such an algorithm would be quadratic in ν. A much better way is to perform a box filter, whose running time is independent of the neighborhood size ν. A box filter is performed in two passes, one along the rows of the absolute difference array and another along the columns. The first pass creates the partial row-wise summations, which are then added along the columns to yield the full neighborhood summations. Each pass is performed by taking the value for the previous location, adding one new absolute difference (corresponding to the new scope of the $\nu * \nu$ neighborhood), and subtracting an old absolute difference (corresponding to the neighborhood of the previous location). This takes advantage of the fact that the neighborhoods of adjacent locations overlap completely except for the extreme most elements. Therefore, a complete box summation consists of only a constant four computations, two along the element's row and two along the element's column (except for the very first element of each row or column which initializes the running summations), and this is independent of the size of ν.

The final "winner-take-all" stage is then done based on the match values. In practice on a serial machine it is convenient to compare the current best match value with each new one as it is calculated, using a traditional `IF-THEN` construct. In a massively parallel processor it may be undesirable to require a data-dependent conditional construct in the algorithm (e.g. [LPK93]), since this means that some processors will be executing different instructions depending on the data being processed; this may be difficult or impossible with SIMD (single-instruction-multiple-data) hardware. In this case the updates can easily be performed directly using shift and mask operations, as shown in Figure 5.1.

On some parallel machines such as on the Connection Machine the match values could be calculated independently and combined using built-in MINIMUM calculating hardware ([D86], [LBC87], [L88]). This operation could also be performed very efficiently without

```
/*  Select motion with the minimum match value.

    best_motion :  motion with the best matching value thus far
    best_match  :  the match value for best_motion
    cand_motion :  a candidate motion
    cand_match  :  the match value for cand_motion
    mask        :  will be assigned a mask of all 0's or 1's

    This code assumes a 2's complement integer representation,
    and that there will not be an arithmetic overflow.
*/

mask = (best_match - cand_match) >> 31;    /* arithmetic shift */

best_match  = ( best_match & mask) | ( cand_match &~ mask);
best_motion = (best_motion & mask) | (cand_motion &~ mask);
```

Figure 5.2: Direct updating of best motion without conditionals.

branches by using a "conditional move" operation, such as found on the Alpha [McL93] and SPARC V9 [Cas93] architectures.

Often, some tight loops are memory limited (i.e. byte loads and stores dominate). This can be overcome by loading and storing 4 or 8 bytes at a time (depending on whether or not the architecture is 32-bit or 64-bit), if the appropriate byte or bitfield extracting instructions exist in the architecture.

5.2 Personal Computers and Workstations

There is a tremendous advantage in implementing a computer vision algorithm on a general-purpose computer rather than special purpose hardware. A software program is obviously much more flexible than a hardware implementation; a change in algorithm requires nothing more than a recompile, which allows for extensive experimentation. A dedicated image processor can be much more difficult to program than a general purpose computer ([OLT93]).

When designing a custom hardware implementation, there may be some operations that are not possible or practical to implement, and even a simple operation may be impossible to add after the implementation has actually been built. General purpose computers are already commonly available, and can be used for other computationally expensive tasks such as planning, when it is not doing vision. This multiplexing of a computer both benefits all software algorithms that use the computer as well as helps justifies its initial purchase (or upgrade) in the first place. In addition, as noted in the introduction, a software algorithm can conceptually be distributed to everyone with a computer at effectively no cost, in contrast to the purchase or building of a special purpose processor. This effectively means that thousands of researchers can make use of any techniques learned. In particular, the algorithms discussed in this thesis are now usable by anyone with a computer (and in real-time by anyone with a modern, low-end workstation). Finally, general-purpose computers have experienced an exponential growth for the last several years which will likely continue for several more years. [PH94] gives a 54% yearly increase for the years 1987-1992. A software algorithm can trivially make use of a new computer generation with only a recompile (see also Section 6.1). Conversely, moving from one brand of dedicated image processor to another may require a complete redesign of the algorithm to match the new processor's capabilities.

Scientific workstations are generally optimized for intensive floating-point computations, requiring advanced floating-point hardware as well as a high memory bandwidth; as a result, they can be quite expensive, around $20-30,000 for a mid-level machine. Dedicated image processing boards are available, but can add considerable expense to a vision system. Conversely, personal computers generally make use of mostly integer arithmetic. Although multimedia applications will certainly increase the use of floating-point, personal computers will most likely emphasize their integer performance for the foreseeable future. Examples are Intel's Pentium (tm) [AA93] and IBM/Motorola's PowerPC (tm) 604 processor [Mot94b],

both of which contain multiple integer units but only a single floating-point unit. In addition, pixel data is generally only a single byte long, thus with proper coding several image bytes can be loaded in parallel, compared to a double-precision floating point datum which spans eight bytes, thus less memory bandwidth would be required per datum. Such memory accesses are generally sequential and highly predictable, and thus can take advantage of instructions that provide memory-access "hints" as are found on the Alpha [DEC92], PA-RISC [hp94], PowerPC [Mot94a], and SPARC V9 [Cas93] architectures. Thus this optical flow algorithm is well-suited for operation on common personal computers, since it does not require the floating point capabilities of the much more expensive scientific workstations, and can achieve a more balanced memory-bandwidth to computation ratio.

Digital signal processors (DSP's) have some of the advantages of general-purpose computers, in that they have the flexibility and reprogrammability of software. However, they often must be programmed in assembly language to fully exploit their capabilities, due to the rigid program structure imposed by their instruction sets. In addition, DSP instruction sets generally vary from one generation to another, so they must be reprogrammed, usually in assembly language, with each new generation [SPL92]. One attractive feature of DSP's is low overhead looping, which is ideal for the box-filter stage of the basic algorithm described in Section 5.1. However, implementations of zero-cycle branching and branch prediction are commonly found in modern CPU architectures, negating this advantage, since such loops are highly predictable. Another feature of DSP's are their ability to perform multiple operations in parallel, however this too is offset by superscalar implementations of modern architectures. When combined with faster cycle times, a modern general-purpose processor is usually as fast or faster than a DSP, while remaining much easier to program and debug [SPL92]. Thus, while a DSP would be appropriate for a low-cost (or low-power) implementation of a fully developed system, it is not necessary to resort to them to achieve real-time performance.

A common measure of performance for a computer vision algorithm is its frame rate, in frames per second. In the case of this algorithm, it is dependent on the number of speeds detected. Since the algorithm is linearly dependent on the number of speeds detected per frame, the frame rate times the number of speeds detected per frame is constant for a given machine; this constant is called the "magic number". Of course on most modern machines, memory access time is non-linear due to caches and paging effects, so there will inevitably be a non-linear factor increase with problem size, an effect which will only increase as microprocessors get faster and memory gets larger (but not much faster, as is the current trend). For this problem on a machine with 1 MB of secondary cache, this effect is non-existent to at least 20 speeds. Since the algorithm exhibits excellent locality of reference, performance can be expected to degrade gracefully with even larger problem sizes. For a 40 Mhz Sun Sparcstation 10/41, the "magic number" is approximately 45 for images 64 pixels by 64 pixels in size.

The following chart gives estimated values of the "magic numbers" for various personal-computer class machines currently available as well as those expected in the next year, based on comparisons with the Sparcstation 10/41 using industry standard benchmarks. All machines are assumed to have an external cache ranging in size from 256KB to 1 MB. For example, the algorithm would be expected to run at a peak rate of 8 frames per second, calculating 8 speeds per frame, on an 66 MhZ PowerPC machine. Alternatively, it could run at 4 frames per second calculating 16 speeds per frame, or possibly 32 frames per second calculating only 2 speeds per frame.

(Trademarks are the property of their respective owners.) For computer architectures such as the PowerPC, which is designed for fast looping across arrays [WD93], an optimized program could yield even higher "magic numbers". In addition, as discussed in Section 4.2.3 , for a given maximum image delay S it is not necessary to calculate all motions $\{1/1, 1/2, ..., 1/S\}$ pixels per frame; any subset of velocities may be chosen, allowing

Est. "Magic Numbers" for 64x64 images		
machine/CPU	CPU Mhz	magic number
Sun Sparcstation 10/41	40	45
Intel 486 DX2	66	27
Intel Pentium	66	55
Intel Pentium	100	85
PowerPC 601	66	64
PowerPC 601	80	73
PowerPC 604	100	140

Figure 5.3: Magic numbers for various computers/CPU's

considerable "range vs. resolution" performance tradeoffs.

In the upcoming years, ever increasing performance can be expected from personal computer makers that will allow the algorithm to be run at even higher frame rates, image velocity ranges, and/or image resolutions. In particular, CPU's intended for desktop personal computers such as Intel's x86 and the PowerPC series will be built with multiple integer functional units (but not necessarily multiple floating point units), the Pentium [AA93] and PowerPC 604 [Mot94b] being the first such examples. Whereas most general-purpose software can only take partial advantage of the superscalar capabilities of these CPU's due to data dependencies, and image processing algorithms requiring floating point are limited by a single FPU on these chips, algorithms such as this one can take full advantage of the available integer instruction-level parallelism (with proper coding) by processing two or more rows of the image in parallel, for example. HP's PA7100LC chip, and the recently announced UltraSPARC chip employ special instructions used to handle MPEG compressed video streams, such as multiple 16-bit additions performed within a single register. The correlation-based algorithm could potentially make use of these features to dramatically increase the performance of a patch-matching optical flow algorithm. Chips exploiting Very Long Instruction Word technology (such as announced by Hewlett-Packard and Intel) would be similarly suitable for this form of image processing due to the inherent parallelism in the algorithm. Thus, this algorithm is well positioned to take full advantage of all of these

developments.

5.3 Parallel Implementations

This type of algorithm is attractive in that there are many places where parallelism can be exploited, both in hardware and software, and at both fine and coarse levels of parallelism. Of course, a major strength of the algorithm is that is does not need massive parallelism to run in real-time, however, such a processor could be fully utilized if available.

Parallelism exploited in software (i.e., explicitly running multiple parts of a program on multiple processors simultaneously) can include running the algorithm on multithreaded machines with a small number of processors, typically two or four. Since such machines are now being sold for the desktop by several popular manufacturers, it is entirely appropriate to make such considerations. At the coarsest level, each zero-or-one pixel shift-and-match stage, one for each time delay, could be processed independently. Once the pair of images are transferred to each processor, each processor could then process its own image pair independently. The only potential bottleneck here would be to transfer the incoming image to several processors; however, the rows of the image could be transferred to each processor in a pipelined fashion. Alternatively, each processor could begin accessing the image at a different row per processor, so that memory "collisions" are reduced. At a slightly finer level, the correlation step for each time delay could be divided into its component multiple directions and computed individually. For the current implementation, this is eight directions of motion, plus zero motion. Future implementations could have greatly increased angular accuracy by interpolating grey level values and considering inter-pixel motions, i.e. motion to an interpolated "virtual" pixel lying in between to existing pixels, or by using pixels created by subsampling a shifted input image (in cases where a subsampled image is used).

Hardware implementations could take parallelism to an even finer scale, and includes

implementation on special-purpose image processors (such as the PIPE or Datacube) and in VLSI [O85]. Although this algorithm would be much more difficult to implement in VLSI in its entirety than say the one-dimensional technique of [AP93], the simple nature of the box filter makes it suitable for implementation in VLSI, Field-Programmable Gate Arrays (FPGA's), systolic arrays, or digital signal processors. At the finest scale, each row of an image can processed individually by a pipelined VLSI chip and fed into further processing modules, as has been shown by a partial implementation in CMOS VLSI [C91]. At a maximum, in the current implementation, all 64 rows/columns in all 9 directions (8 directions plus zero motion) for each of 20 speeds could be usefully parallelized, exploiting up to 11520 separate processing nodes. Running the algorithm by processing individual *pixels* on separate processors (as in [BLP89a]) is not particularly recommended on a typical modern parallel machine which uses tens or hundreds of much more powerful microprocessors instead of the 1-bit processors of a Connection Machine-2 [TM88], since I/O and communication latencies are then likely to become the bottlenecks.

Using the technique of [BLP89b] (the same as described in Section 2.3) on a Datacube MaxVideo 200, [LK93] calculates optical flow within a limited radius of +/- 3 pixels horizontally and +/- 2 pixels vertically using a 7x7 correlation window at 10 Hz on 128x120 images. Since this implementation uses the same basic correlation algorithm as that described in this thesis, it is instructive to compare the performance of the two. We will use single-pixel-shift units as a basic measure, i.e. the comparison of a single given pixel with a single hypothesized pixel shift. Recall that in the traditional algorithm ([BLP89a], [BLP89b]) only two frames are used, and a pixel is constrained to lie within a certain neighborhood, in their case 7x5. Motion is calculated everywhere except for a border region where motion calculation would require pixels not available in the image itself. This border region is the sum of the maximum pixel displacement in any direction plus half the correlation window; thus the "active" image area is (128-2(3+3))*(120-2(2+3)) = 116*110. Given a frame rate of 10

Hz, this is 7*5*116*110*10 = 4466000 single-pixel-shift units per second. (The correlation window size, which is 7x7 for both algorithms, does not affect computational complexity; see Section 5.1.) In the linear-time algorithm, the local neighborhood is +/- 1 pixel for a search neighborhood of 3x3. In addition, there are multiple time delays, which adds another factor. The active image area is (64-2(1+3))*(64-2(1+3)) = 56*56. When calculating 10 speeds the frame rate is 4.5 frames per second on a Sun Sparcstation 10/41. This results in 3*3*10*56*56*4.5 = 1270080 single-pixel-shift units per second. Thus the workstation implementation is only 3.5 times slower than the implementation on the dedicated image processor.[1] By early 1995, this gap is expected to be closed; see Section 6.1.

[Uhr87] argues strongly for massively parallel processing systems for computer vision :

> "Perception is a massively parallel process... Only massively parallel networks are fast enough to process the large resulting image arrays, and the successive resulting structures of information."

For example, monkeys can recognize complex objects such as faces in only 70 to 200 milliseconds, which only allows for at most a few hundred serial steps in neural hardware; clearly massive parallelism is needed in this case. However, this argument does not consider the possibility that *moderate* parallelism, combined with extremely high-speed digital computers (each of which can operate at a moderate level of parallelism) can approach the performance of massively parallel biological hardware. Companies such as Thinking Machines has recognized the advantages of using moderate numbers of commercial microprocessors in their latest offering (the CM-5) in favor of previous models which are composed of up to tens of thousands of 1-bit custom processors ([D86], [LBC87], [L88]). [Uhr87] notes that the overhead for communication in a MIMD (multiple-instruction, multiple data) computer may

[1] Note that this only considers the raw amount of work done, and not whether or not the work done is particularly efficient or useful. As noted in Section 3.1, the linear-time algorithm only grows linearly with the number of velocities detected rather than quadratically as with the traditional approach. In addition, Section 3.4 argues for the harmonic series of velocity measurements versus the linear set of measurements of the traditional algorithm. These are algorithmic issues however, and orthogonal to this discussion.

be much more than the time for the computation itself. In a moderately parallel machine however, tasks are generally divided up in a coarser fashion, such that a variable amount of communication versus computation can take place depending on what is appropriate for a given machine. For example as noted above, the various shift-and-match stages of the correlation algorithm can be divided up into individual frame pairs (one per time delay), pairs of shifted images (eight or more pixel shift directions plus zero motion per frame pair), or individual rows and columns (64 or more per shifted image pair), depending on the communication to computation ratio for a particular machine. Flexible network topologies such as described in [BA93] would likely be required for a massively parallel system to be able to operate on a wide variety of computer vision problems.

Amdahl's Law (attributed to Gene Amdahl) [HP90] states that as sections of a program that are parallelizable are increasingly parallelized, the remaining serial sections of the program will become the bottleneck. In the case of this algorithm, if the shift and match stage were completely parallelized, the temporal antialiasing stage would then become the bottleneck. In the case of very high parallelism each pixel could then be processed independently, however this form of parallelism could be limited by I/O to the processing nodes, unlike for example processing entire rows in parallel (as assumed above), which has a better I/O to computation ratio. At high levels of parallelism however, it may be the case that the temporal antialiasing stage does not produce a significant benefit, since there would not be the very coarse quantizations which seem to cause many of the problems in the first place.

Although this algorithm was designed to operate on ordinary scientific workstations, and would also run well on a modern personal computer, it would run very quickly on a massively parallel processor (MPP), up to the level of parallelism noted above (i.e. parallelized at the level of individual rows of an image). Such processing is facilitated by the fact that the algorithm only requires simple integer arithmetic which is easily implemented by basic hardware; floating-point hardware would be much more expensive to implement

on a massively parallel scale. Modern highly (if not massively) parallel processors from Cray, Thinking Machines, Convex, and IBM are generally composed of tens or hundreds of "off-the-shelf" microprocessors, and thus the algorithm would benefit from all the reasons mentioned for personal computers, as well as from massive parallelism itself.

It should also be noted that even hardware implementations such as described in [DW93] could benefit from the linear-time space-time tradeoff described in this thesis since only a linear number of pixel shifts would be calculated instead of a quadratic number. In addition, the detection of sub-pixel motions would be detectable, as well as the harmonic sequence of motions.

5.4 Pyramid Architectures

One popular hardware image processor is the pyramid processor. A *pyramid* structure attempts to divide an image into its component spatial scales, and by doing so allows the efficient computation of certain properties that are appropriate at that given scale; an example pyramid representation of an image is shown in Figure 5.4. Pyramids can be excellent tools for hierarchal binary morphology (such as the ANDing of various features and region growing), segmentation, and the calculation of various geometrical concepts such as extrema [TLL87], [SHBV87], [St87].

The strength of a pyramid structure with regard to optical flow is that it can efficiently capture large motions of large contiguous areas of an image ([A87a], [A87b], [C90], [G87]), for example see Figure 5.5. However, the general requirement that the area of the moving surface be roughly proportional to the magnitude of motion does not always hold for interesting types of motion ([C93]). For example, in the top-left of Figure 5.6 there appears to be some angular downward motion of a straight object. The finest level pyramid representation of course gives good motion, and the next higher level is more or less representative, however the upper two levels are not very representative at all. One basic problem is that each

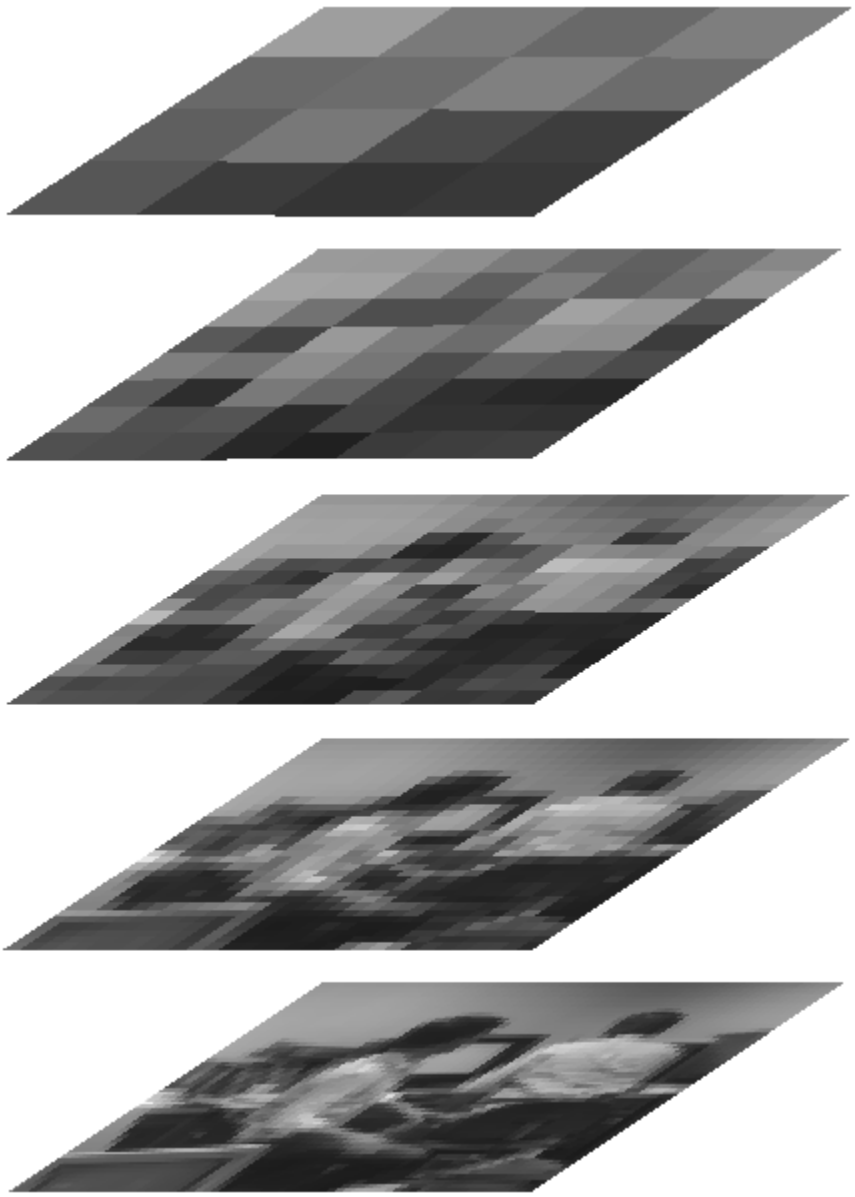


Figure 5.4: A pyramid representation of an image, at scales (bottom to top): 64x64, 32x32, 16x16, 8x8, and 4x4.

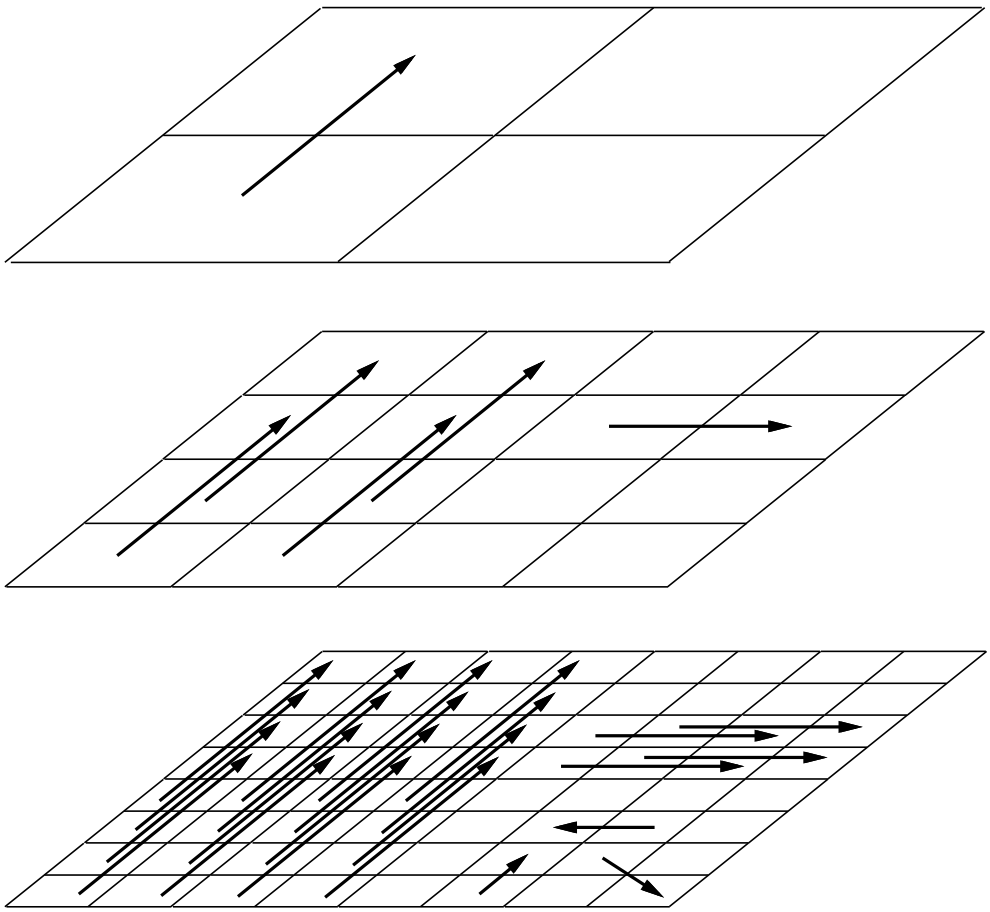


Figure 5.5: Scales of a pyramid representation. Some forms of motion are best represented at different spatial scales. This is a somewhat idealized example.

level in the pyramid is coarser by a factor of 4, thus resolution decreases exponentially with each layer. [PFH87], [Uhr87] suggest the construction of "gentler" pyramids, with ratios somewhat less than a power-of-2 on each side. [A87a] suggests using overlapping pyramids to avoid missing motion lying on a pixel boundary. Resampling techniques used in computer graphics can help here ([F86], [Burt81], [Burt83]), however, such complexity greatly reduces the practicality of a *hardware* implementation of the traditional 3-dimensional pyramid.

Another problem with a physical implementation of a pyramid is that the axis connecting the separate levels (beginning with the base going up to the apex) is not in the same plane as the 2-dimensional images at each level. I.e., there usually needs to be a backplane of

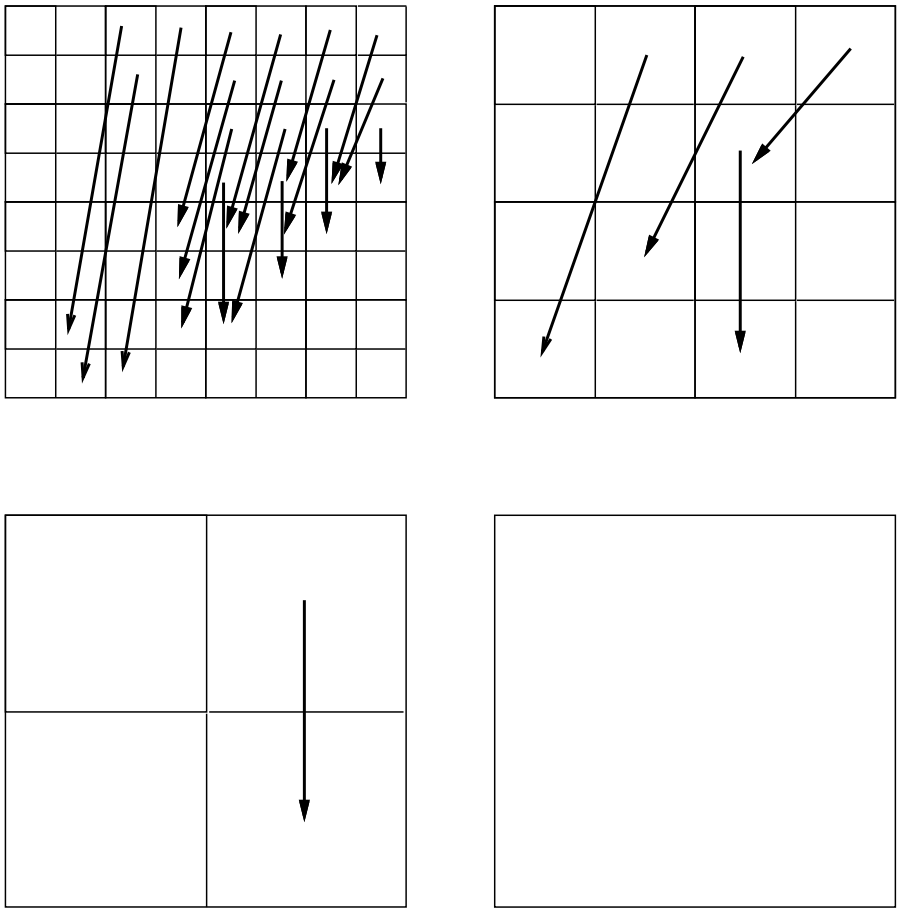


Figure 5.6: An example demonstrating the limits of pyramids. Top left shows very fast motion at a very fine scale. Although two levels in the pyramid give reasonable approximations, the higher levels are not appropriate representations. Solutions to this problem may be performed in software, but would be more difficult in hardware; see text for details.

some sort connecting these levels, if they are physically implemented similar to their logical layout (e.g. [SHBV87]). Conversely, current VLSI (Very Large Scale Integration) chips are largely flat devices. A typical chip may include several metal or polysilicate "layers", but they are basically 2-dimensional devices; there is a strong practical incentive in keeping such devices two-dimensional. This is both the pyramid's greatest strength and weakness. The topology of the pyramid allows for $O(\log n)$ communication between processors, even distant processors, but this added complexity reduces the feasibility of widely available, low-cost systems. Parallelism, both instruction-level and task-level (multiprocessing and

multithreading) will certainly see increased use as a means of increasing speed, and image processing/computer vision will be able to make full use of these advances, regardless of whether pyramid structures are used.

Although pyramid *structures* can be very useful for understanding an image at multiple scales (and are not inconsistent with the linear-time optical flow algorithm), pyramid-based special-purpose *hardware* is less attractive for real-time operation, except in expensive special-purpose systems. Given the current state of exponential growth in computing power, there is also the risk that any such system, once finally built, will be obsolete. The concept of characterizing a scene in increasingly abstract descriptions is of course attractive, however it would be unwise to fix any particular description in hardware until visual processing is much better understood. For example, [PFH87] describes pyramids using "gentler" scale ratios than the usual power-of-two pyramid structures. This reduces the normally very large detail differences found between pyramid layers. Building such a custom pyramid in hardware however would be much more difficult than the usual 4-to-1 child-to-parent ratio found in most pyramids. [Uhr87]'s suggestion of a massively parallel lowest level augmented with a number of much more powerful processors appears to be a practical alternative to the typical hardware pyramid. [W93] describes UMASS's second-generation Image Understanding Architecture, consisting of a low-level reconfigurable array of bit-serial processors for sensory data, an intermediate level of digital signal processors to execute grouping of tokens and graph matching, and a high level of RISC processors for knowledge-based processing. The term *pseudomorphism* is defined to describe an automatic management of multiple implementations of possibly widely varying types and topologies of parallel computing resources. This type of flexibility surely represents a difficult software challenge in the years to come, but will be necessary for portable and parallel computer vision programs.

5.5 Summary

Issues of computational efficiency often focus on algorithmic complexity. The tremendous computational demands of computer vision demand computational efficiency at all levels, including the hardware implementation. This chapter has described in detail the characteristics of the linear-time optical flow correlation-based algorithm which make it especially suitable for commonly available computing hardware, and several reasons why it will be able to take full advantage of virtually any advance in computer technology expected in the near future. As a result, the linear-time algorithm is likely to remain at the forefront of real-time optical flow for many years to come.

Chapter 6

Conclusions and Future Work

We have defined three criteria for practical robotic vision: robustness, real-time performance, and accuracy. Robustness has been addressed by the patch-matching correlation-based optical flow algorithm, and at the application level by the redundancy in the time-to-contact measurements. Real-time performance has been addressed by the linear-time optical flow algorithm and the suitability of the necessary calculations for modern computing hardware. Finally, remarkable accuracy may be achieved, at least for some applications such as time-to-contact, by integrating optical flow measurements across time and space. By pushing forward with the coarsely quantized optical flow measurements, we have learned that it can be possible to achieve quantitatively accurate computer vision, even when the low-level measurements are not precise. Thus we have learned that complicated models or very accurate low-level measurements are not always necessary for quantitatively accurate computer vision. It is worthwhile to approach other computer vision problems with the same philosophy.

The current implementation of the time-to-contact algorithm assumes that the mobile robot is moving such that the focus-of-expansion is always within the field of view (but not necessarily straight forward). Ideally of course, we would not want to have this limitation; we would want the robot to be able to move in any direction it can, particularly arbitrary rotational motion (for the purposes of flexibility only; it is only the translational compo-

nent of motion, and not the rotational component, which provided information about the environment and the observer's relation to it [T90]. In addition, rotational movement of an active camera head on a mobile robot will generally be known, since this is generally programmed explicitly). At this particular moment however, this functionality is not available. One way to deal with this situation is to insist on full general motion or nothing at all, and not make use of what means of perception are available. A better option is to limit the robot's motion to those behaviors which are essential for completing a task. A robot which can move forwards and backwards, and turn clockwise and counter-clockwise, may be able to achieve the same set of behaviors as one that can combine rotation with translation (although maybe not quite as efficiently). In this case fully general motion may not give our robot any new fundamental behaviors; fully general motion would just be a luxury. In some tasks, fully general motion may be necessary, and these behaviors will have to wait to be implemented until the necessary modes of perception are available. However, many other behaviors need not wait. In the meantime of course, research into more general forms of motion vision will continue.

Relating this argument to the top-down versus bottom-up debate, a top-down approach might analyze a situation and then determine what means of perception are necessary to implement certain behaviors. For example, structure-from-motion might require very precise optical flow calculations. If the state-of-the-art in optical flow cannot yield these measurements quickly, reliably and accurately, the top-down approach would likely insist on finding better optical flow algorithms. Unfortunately this may just prove to be too difficult. Conversely, a bottom-up approach would declare that the general problem is too difficult, and impose constraints on the problem at hand (*Active perception* can be seen as an implementation of this philosophy; see [AWB87], [Bal90], [CF88]). Indeed, the approach taken by this author is to determine what information is available from the optical flow (in particular, quantized measurements) and verify what can be calculated from them. This yields

a certain functionality (time-to-contact given straight, linear motion) which can be used to implement a wide range of behaviors, at least until the time when more information can be calculated.

The current implementation of the algorithm still suffers from many limitations. One example is the limited angular accuracy due to the standard rectangular tessellation of images; this can be clearly seen in Figure 2.1. This can be overcome in principle by shifting the input image by a half-pixel in either the X or the Y direction, and performing the search over these new pixels as well. For applications that demand increased angular accuracy, it may be worth the additional computational cost. This approximation can be performed by either interpolation, or exactly in cases when subsampling is done by shifting the input array appropriately. A similar form of interpolation via pixel-shifting could be used to increase the accuracy of the magnitudes of the motions detected.

Figure 6.1 shows a grey-level plot of the match values for the best pixel-shift (i.e. the selected velocity) for two typical frames of the "fast" collision sequence. (For this particular plot, the sum-of-squared differences measure was used instead of the usual sum-of-absolute differences). The areas where matches are poor generally correspond to motions that cannot be matched exactly by the algorithm, such as motions that are in between two of the usual 8 directions or in between velocities of $1/x$ and $1/(x+1)$ for some time delay, or both. These values could be used for interpolation of velocities.

Precise boundary detection from motion remains as future work [SU87]. One option is to run the algorithm using two patch sizes; one would expect that the "quality" of the match of the larger patch would be much worse than that of the smaller patch, when both are placed at the motion boundary [LBP88]; this could run completely in parallel as well. [AD91] notes that often precise boundary detection is not necessary for accurate motion perception; for example, Johansson [J73] demonstrated that people can easily identify a walking person while only viewing lights attached to the moving person's joints (see also [J75]). These

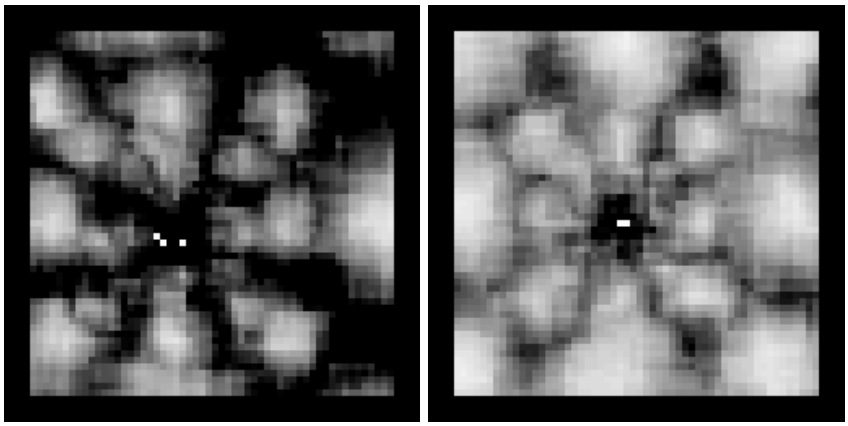


Figure 6.1: Grey-level plot of best-match sum-of-squared-differences for two typical frames; a lighter shade means a better match.

"Moving Light Displays" do however sweep out a spatio-temporal path whose temporal coherence can be exploited. The current optical flow algorithm makes only limited use of temporal coherence; although it is undesirable to place too strong a prior importance on temporal continuity in *future* frames (i.e. for frames at time $t + k$ when considering frame t), temporal continuity in *past* frames in the current algorithm is limited to the implicit assumption that motion must be equal a full pixel before being detected. Further exploiting temporal coherence may be required for cases when there are multiple moving objects, for example. Given that the algorithm runs best with a temporally dense set of images (i.e. a fast frame rate), 3-dimensional spatio-temporal coherence is an excellent candidate for furthering this algorithm. Occlusion and disocclusion detection can also be handled by spatiotemporal coherence [AD91].

Detecting multiple moving objects requires an initial segmentation step. [SRT92] considers a *FOE triangle* created by the intersection of any three optical flow vectors on an object in segmenting out objects. One possibility with this algorithm is to calculate an initial FOE, then calculate the deviation from the hypothesized FOE at all points in the image. [GG84] uses a Markov Random Field model incorporating "line processes" to separate motions across occluding boundaries. These techniques can be slow to converge and

difficult to apply to practical problems however.

[BBHP90] describes six types of basic motion: 1) single object motion; 2) two moving surfaces separated by a boundary; 3) two transparent surfaces with different motions, including moving shadows, reflections, and transparent surfaces; 4) interleaved components; 5) small moving object in front of moving background; and 6) two motions each of which suffer from the aperture problem.

Type 1) motion is of course straightforward. The detection of type 2) motion implies boundary detection, which the current algorithm can detect roughly, but not exactly. In Figure 3.9, the motion of the branches blurs over some of the background behind them. This is because the neighborhood patches sometimes "capture" pixels that are at the edge of occluding boundaries, and assign to them the motion of nearby objects. Reducing the size of the patches would decrease this effect, however this would result in less resistance to noise. This tradeoff between uncertainty in localization in space vs. localization in frequency is quantitatively described by the *uncertainty relation* ([Br78]) : $\Delta x \Delta s \geq \frac{\pi}{4}$, where Δx is spatial variance and Δs is variance in frequency. The only distribution which optimizes this relation is the Gaussian ([MH80]). Many computer vision optical flow algorithms find it necessary to preprocess the image by smoothing with a Gaussian before calculating optical flow, due to the sensitivity of mathematical differentiation to noise. Even though this operation can be separated into two one-dimensional convolutions it still often requires special-purpose hardware to be performed in real-time, thus it was not used in this algorithm, nor were Gaussian "patches" used in the matching stage. Preliminary tests do not seem to indicate any performance improvement using Gaussian patches, probably due to the already large size of the patches and subsampling of the images.

The capturing of off-object pixels is not an issue in many types of qualitative computer vision where precise boundary detection is not needed, nor is it an issue in the case of time-to-contact with a single surface. In cases were more precise boundary detection is

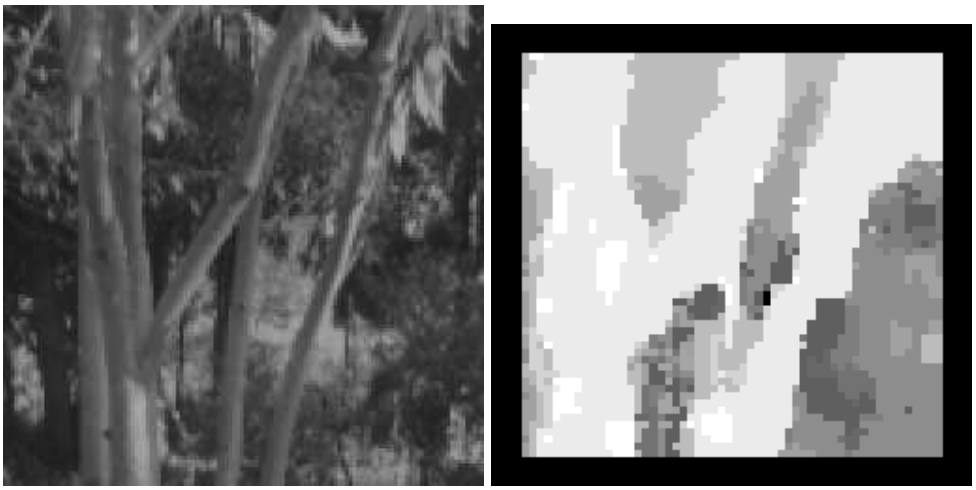


Figure 6.2: Depth from motion parallax, using image zoomed by a factor of 2. Camera is translating left, with the resulting optical flow and grey-level shaded relative depth map.

desired however, one option is to zoom in on boundaries using a pyramid technique such as in [BBH+89]. Figure 6.2 shows the optical flow algorithm run on the same SRI tree sequence as in Figure 3.9, except that the image is zoomed by a factor of 2, and shows the outline of the branches much more clearly. One issue with this type of processing is that the image motion for such a zoomed image is double that of the normal sequence; a method of dynamically adjusting the range of velocities detected would be useful in this case.

Cases 3) and 4), transparent surfaces and interleaved components are more difficult; since the algorithm has thus far only relied on the single best match value for all possible motions, it is unknown how well the algorithm will do when faced with more than one motion at a single location (in the case of transparency) or more than one motion very close to each other (in the case of interleaved objects). The "winner-take-all" nature of the algorithm can support the notion of a bimodal response, however some mechanism would be needed to differentiate such a bimodal response from a normal, single response (unless it was known *a priori* that there were two motions present in the image).

One possibility to handle transparency is to use an iterative procedure as in [BBHP90]. Here two motions are hypothesized, $\mathbf{p}$ and $\mathbf{q}$. They define a sequence of *difference images* :

$$D_i \equiv I(x, y, i+1) - I^{\mathbf{p}}(x, y, i)$$

where $I^{\mathbf{p}}(x, y, 1)$ is image 1 transformed by the motion $\mathbf{p}$. Thus D_1 consists of the second frame minus the first frame transformed by motion $\mathbf{p}$; this yields an estimate for motion $\mathbf{q}$. A similar definition holds when D_1 is defined to be the second frame minus the first frame transformed by motion $\mathbf{q}$. Given an initial estimate for $\mathbf{p}$, iterate :

1. Form the difference images D_1 and D_2.

2. Estimate the motion from D_1 to D_2 to estimate $\mathbf{q}_{n+1}$.

3. Form the new difference images D_1 and D_2 using the estimate $\mathbf{q}_{n+1}$.

4. Estimate the motion from the new D_1 to the new D_2 to obtain $\mathbf{p}_{n+2}$.

This is repeated, and they report convergence within a few iterations in most cases, regardless of the initial estimate for $\mathbf{p}_0$.

Case 5), a small moving object in front of moving background, could make use of some form of temporal integration such as in [IRP92]. Here they define a temporally integrated image $Av(t)$:

$$Av(t) \equiv I(0)Av(t+1) \equiv w \cdot I(t+1) + (1-w) \cdot register(Av(t), I(t+1))$$

where w is a weighting factor (they use $w = 0.3$), and $register(P, Q)$ denotes the registration of images P and Q by warping P to Q by the motion from P to Q. Iterate (where M denotes a segmentation mask of a tracked object of interest) :

1. Compute the dominant motion parameters between $Av(t)$ and the new frame $I(t+1)$ in the region of $M(t)$.

2. Warp $Av(t)$ and $M(t)$ towards the new frame $I(t+1)$ by the calculated motion parameters.

3. Identify the stationary regions in the registered images, using $M(t)$ as an initial guess, and set $M(t+1)$ to the new tracked region.

4. Update the temporally integrated image $Av(t+1)$, and repeat.

[BFBB93], [NG92], [VG92], [SAH91] make use of confidence measures in calculating optical flow. Conversely, the "winner-take-all" nature of this algorithm makes no use of confidence measures whatsoever, instead relying on the algorithm's inherent smoothness in producing a 100% dense output. If dense measurements are required, this fact may be viewed as a strength; if only sparse measurements are acceptable, it may be viewed as a weakness. In fact, there would be little computational savings in calculating optical flow only at areas of high contrast such as edges [Hil84] or corners ([Nag87]) since the pipelined nature of the box-filter generally assumes that optical flow is calculated at all points in the image to be efficient (see Section 5.1 for details). Although it may be possible to use the match strengths as a measure of confidence, this author is not optimistic of such an approach. Although Figure 6.1 does imply that the match values could be used for this purpose, it seems more beneficial to use the match values for interpolation of velocities. In this way all velocities become more accurate, rather than throwing out "bad" matches.

The time-to-contact and FOE calculations made several assumptions, such as constant unidirectional motion, and a flat object that completely fills the field of view. (This last condition is actually not satisfied until about frame 42. This is significant since the current implementation does not attempt to segment the relevant flat surface and omit the background. Performance of the algorithm does improve after this point.) In addition, the sweatshirt used as a target, although solid-colored, is not as textureless as many objects found in a typical office, such as a black filing cabinet. Subsequent work will examine these issues in turn.

Non-rigid motion (for example [KG92]) would seem to require more precise measurements than is possible from this algorithm without some form of interpolation. Since the basic assumption of the patch-matching approach is local rigidity, it is likely that major modifications would be needed to make the algorithm suitable for this application. Applica-

tions that calculate the 3-dimensional structure of the environment will likely be the main focus of this algorithm for some time.

Although the field of computer graphics is mostly concerned with the opposite of computer vision, that is the process of converting 3-dimensional scenes into two-dimensional images, concepts from one field are very often applicable to the other. For example, spatial resampling techniques can be used in the construction of pyramids [F86]. [Burt81] presents a Hierarchal Discrete Correlation technique which can very efficiently approximate a Gaussian kernel. It remains to be seen how useful antialiasing techniques such as in [SS93] are applicable to the problems described in this thesis. In addition, there would be a tremendous potential for using commercial graphics hardware to assist computer vision algorithms should a common ground be found. Special-purpose MPEG encoders/decoders, or special-purpose instructions found on general-purpose CPU's (as noted in Section 5.2) are the first examples of this.

The relationship between the time-space tradeoff and spatio-temporal models of motion ([AB85], [SS85], [WA85]) is an area of future research.

Issues such as lighting and image stabilization (particularly when the camera is mounted on a small, unstable mobile robot) are common problems with machine vision, and it is desirable to develop techniques to deal with these issues.

Ultimately, it is likely that accurate and robust structure-from-motion will make use of both depth from stereo as well as depth from motion, as in [TGS91]. The process of correspondence in optical flow can be similar to that of binocular stereo, except that in the former case the matching is across time, and in the latter the matching is across space [R75]. Both applications can make use of various methods for correspondence (issues of correspondence and image matching are discussed in [HS93]), however tracking across space is not equivalent to tracking across time. In particular, the former can make use of the *epipolar constraint* in matching across space. The latter's matching across space and time

can make use of the *velocity equation* in matching across both space and time. Of course, stereo can make use of spatiotemporal coherence as well.

The major barrier to applying this algorithm as a generic optical flow "black box" for robotic vision is that it must be carefully adapted to the specific task in order to produce quantitatively accurate results, as in the case of time-to-contact. It would be pointless to attempt to measure the performance of this particular algorithm *per se* using the methods used in [BFBB92] and [WM93]; since the individual optical flow vectors are quantized in magnitude and direction, the average error for a single pixel would be very poor. But note that optical flow is only valuable when used in the context of a specific task, such as obstacle avoidance [MBLB91] or time-to-contact, and the performance of this algorithm when applied to the latter has been shown to be exceptionally good. However, it remains to be seen if such performance can be equaled for other robotic vision tasks. Given the algorithm's inherent robustness, computational efficiency, and demonstrated *potential* accuracy, there is a good chance for success. Ideally, a more general procedure for adapting this real-time optical flow algorithm to specific robotic vision tasks will someday emerge.

6.1 Timeline

All versions of this algorithm since its inception in 1990 run in real-time. Continuous improvements in computing power have allowed and resulted in corresponding increases in functionality while still maintaining real-time performance. This section presents a timeline of both past, present and predicted future results, listing the approximate date, the computer or CPU type used or expected to be used at about that time, the actual or expected "magic number" for such a machine, a defining characteristic of the optical flow practical given that level of computing power, and a sample application possible or predicted by that date.

(1) Using pyramid techniques it was possible to segment a moving object in about a half second.

Timeline				
Date	machine/CPU	Magic #	optical flow	application
3/90	Sun 4/110	-	pyramid; 2Hz	(1) segmentation
6/90	Sun 4/110	6	1 speed	(2) tracking
8/91	Sparc 2	16	2 speeds	(3) heading determination
6/93	Sparc 10/41	32	6 speeds	(4) qualitative depth map
3/94	Sparc 10/41	45	20 speeds	(5) time-to-contact
11/94	150MHz R4400	70	>1 pixels	(6) planar tilt
1995-96	100MHz PPC604	140	16 angles	(7) obstacle divergence
1997-99	VLIW, multi-CPU	200-500	128x128 images	(8) active motion vision
2000+	?	?	256x256 images	(9) 3D stereo and motion

Figure 6.3: Timeline of linear-time optical flow.

(2) An early version of the algorithm calculating a single speed (1 pixel/frame) was able to continuously center a slowly moving hand in a rotating camera's field of view.

(3) Calculating 2 speeds of optical flow was sufficient to roughly determine a mobile robot's heading on test images.

(4) Calculating up to six speeds per frame was sufficient to give a relative depth map using motion parallax on test images.

(5) Code improvements allowed the calculation of up to 10 or even 20 speeds per frame in real-time and was sufficient to give remarkably accurate time-to-contact calculations.

(6) Time-to-contact calculations no longer assume an approach perpendicular to an flat object's surface; planar tilt is calculated along with time-to-contact.

(7) The optical flow algorithm will be used for obstacle avoidance based on the divergence of the optical flow field (a generalization of time-to-contact).

(8) Active motion vision will be exploited to determine the 3-dimensional structure of the environment.

(9) Stereo will be combined with active motion vision to determine 3-D structure.

Appendix A

Robust M-estimation

Least-squares techniques are desirable when the measurement noise is normally (Gaussian) distributed, however they are undesirable when there are *outliers*, measurements that do not conform to a Gaussian distribution, since error in the final result is linear in the error of a single measurement. This has led to the study of robust estimators, which have the desirable property of insensitivity to small measurement errors, emphasizing that "small measurement errors" also include gross errors in a small fraction of the data. A primary goal of robust procedures is resistance against these gross errors.

In particular, a robust estimator should have the following three properties ([H81]) :

1) It should have reasonably good efficiency at the assumed noise model,

2) It should be robust in the sense that small deviations from the model assumptions should impair performance only slightly, and

3) somewhat larger deviations from the model should not cause a catastrophe.

The necessity of robust estimators can be demonstrated by comparing two methods of measuring error, the average squared deviation d_n and the average absolute deviation s_n defined by :

$$d_n = \frac{1}{n} \sum |x_i - \overline{x}|$$

$$s_n = \left[\frac{1}{n} \sum (x_i - \overline{x})^2 \right]^{1/2}$$

ε	ARE(ε)
0	0.876
0.001	0.948
0.002	1.016
0.005	1.198
0.01	1.439
0.02	1.752
0.05	2.035
0.10	1.903
0.15	1.689
0.25	1.371
0.5	1.017
1.0	0.876

Figure A.1: Asymptotic Relative Efficiency for given error probabilities.

As an example, consider a batch of n observations of a quantity μ, with probability $1-\varepsilon$ of being a good observation with distribution $N(\mu, \sigma^2)$ and with probability ε of being a bad observation with distribution $N(\mu, 9\sigma^2)$. For normal observations, s_n is 12% more efficient than d_n. However, in this example all errors are not normally distributed, some observations with probability ε will have their errors multiplied by a factor of 3. To analyze this case, [H81] defines a ratio *asymptotic relative efficiency* (ARE) :

$$\text{ARE}(\varepsilon) = lim_{n\to\infty} \frac{\text{var}(s_n)/(Es_n)^2}{\text{var}(d_n)/(Ed_n)^2}$$

Results from [H81] summarized in the following table show that for this example only 2 bad observations in 1000 are enough to offset the 12% advantage of s_n in favor of d_n. Therefore, the standard classic least-squares measure is clearly inappropriate when there are outliers present.

A procedure for robust maximum-likelihood estimation (M-estimation) is derived in [H81] and repeated here using the notation of [HS93]. The goal is to minimize a function of the form:

$$\min_{T_k} \sum_{i=1}^{n} \rho(x_i - T_k) \tag{A.1}$$

where T_k is the parameter to be estimated. In this form, ρ is called the object function,

and T_k is called an M-estimate. This can be defined equivalently by differentiating with respect to T_k :

$$\sum_{i=1}^{n} \psi(x_i - T_k) = 0 \tag{A.2}$$

Huber proposed as an object function the following :

$$\rho(x) = \begin{cases} 0.5x^2, & |x| \leq a; \\ a|x| - 0.5a^2, & \text{o.w.} \end{cases}$$

which naturally results in :

$$\psi(x) = \begin{cases} -a, & if x < -a; \\ x, & if |x| \leq a; \\ a, & if x > a \end{cases}$$

[H81] recommends setting the tuning constant a to some value between 1 and 2; the time-to-contact algorithm seems extremely insensitive to the exact value of tuning constant a. In general, the procedure can be used to estimate a m-dimensional vector θ using n functions f_i which map m-dimensional space into an observation y_i :

$$f_i(\theta) = y_i, \qquad i = 1, ..., n.$$

In our case we are only interested in a single parameter, time-to-contact, so $m = 1$. We have $n = 31$ observations y_i of time-to-contact, one for each radius from the FOE. By equation 4.1 the function that transforms the parameter θ into observations y_i is given by :

$$f_i(\theta) = \frac{i}{\theta},$$

where i is equal to the radius. θ should minimize :

$$\sum_{i=1}^{n} \rho\left(\frac{y_i - f_i(\theta)}{S}\right)$$

where S is a robust estimate of scale (for a scale-invariant M-estimate) defined by:

$$S = \frac{\text{median}_i |y_i - f_i(\theta)|}{0.6745}$$

where 0.6745 is half the interquantile range of the normal distribution $N(0,1)$. The time-to-contact algorithm appears very insensitive to this constant. Here the median of only the non-zero deviations are used, so as to not unduly bias the estimate.

There is no closed form for the M-estimate, so it is solved with an iterative procedure. As an initial estimate for time-to-contact, the geometric average of the individual measurements (one per radius) is used, considering only values that lie between the lower and upper-bound thresholds.

For each iteration k, the modified residuals are computed:

$$r_i^* = \psi\left(\frac{y_i - f_i(\theta^k)}{S^k}\right) S^k$$

where S^k is the estimate of scale at iteration k as defined above. Finally the least-squares solution $\hat{\delta}$ to $X\delta = r^*$, where $X = [x_{ij}]$ is the Jacobian matrix:

$$x_{ij} = \frac{\partial}{\partial\theta_j} f_i(\theta^k) = -\frac{i}{\theta^2}$$

Since we are only solving for one parameter (time-to-contact), $j = m = 1$ and the m by n matrix X^T is only a 1x31 vector, the solution of

$$X^T X\hat{\delta} = X^T r^*$$

reduces to :

$$\hat{\delta} = \frac{X \cdot r^*}{X \cdot X}$$

where '$\cdot$' represents the inner product of two vectors. Finally, set $\theta^{k+1} = \theta^k + \hat{\delta}$ and iterate until $\hat{\delta} \leq \epsilon$ for some small ϵ. For the time-to-contact algorithm with $\epsilon = .001$, the procedure usually converges after only 4-6 iterations.

Bibliography

[A87a] P. Anandan, Measuring Visual Motion from Image Sequences, PhD Dissertation, COINS TR 87-21, University of Massachusetts at Amherst, 1987

[A87b] P. Anandan, A Unified Perspective on Computational Techniques for the Measurement of Visual Motion, Proceedings of the IEEE ICCV, p.219-230, 1987

[AA93] D. Alpert, D. Avnon, Architecture of the Pentium Microprocessor, p.11-21, June 1993 IEEE Micro

[AB85] E. Adelson, J. Bergen, Spatiotemporal Energy Models for the Perception of Motion, Journal of the Optical Society of America, 2(2):284-299, 1985

[AD91] M. Allmen, C. Dyer, Long-Range Spatiotemporal Motion Understanding Using Spatiotemporal Flow Curves, Proceedings of the IEEE CVPR, p.303-309, 1991

[AP93] N.Ancona, T.Poggio, Optical Flow from 1D Correlation: Application to a Time-To-Crash Detector,Fourth International Conference on Computer Vision,p.209-214,1993

[ATYM] P.K.Allen,A.Timcenko,B.Yoshimi,P.Michelman,Automated Tracking and Grasping of a Moving Object with a Robotic Hand-Eye System, Columbia University, accepted for publication IEEE Transactions on Robotics and Automation

[AWB87] J. Aloimonos, I. Weiss, A. Bandyopadhyay, Active Vision, Proceedings of the IEEE ICCV, 1987

[BA93] S. Bhandarkar, H. Arabnia, The Multi-Ring Reconfigurable Network for Computer Vision, p.180-190, Workshop on Computer Architectures for Machine Perception, New Orleans Louisiana, IEEE Computer Society Press, Los Alamitos CA, 1993

[Bal90] D. Ballard, Animate Vision, University of Rochester Technical Report 329, Feb. 1990

[Br78] R. Bracewell, The Fourier Transform and Its Applications, McGraw-Hill, New York, 1978

[B81] H. Buelthoff, Figure-Ground Discrimination in the Visual System of Drosophila melanogaster, Biological Cybernetics 41 p.139-145, 1981

[BB88] A. Borst, S. Bahde, Spatio-Temporal Integration of Motion, Naturwissenschaften 75, p.265-267, 1988

[BBH+89] P. Burt, J. Bergen, R. Hingorani, R. Kolezynski, W. Lee, A. Leung, J. Lubin, H. Shvaytser, Object Tracking With a Moving Camera, IEEE 1989 Workshop on Visual Motion, p.2-12, March 1989

[BBHP90] J. Bergen, P. Burt, R. Hingorani, S. Peleg, Computing Two

Motions from Three frames, Proceedings of the IEEE ICCV, p.27-32, August 1990

[BFBB92] J. Barron, D. Fleet, S.S. Beauchemin, T. Burkitt, Performance of optical Flow Techniques, Proceedings of the IEEE CVPR, p.236-242, 1992.

[BFBB93] J. Barron, D. Fleet, S.S. Beauchemin, T. Burkitt, Performance of Optical Flow Techniques, Revised Tech Report RPL-TR-9107, Queen's University, July 1993.

[BJ94] M. Black, A. Jepson, Estimating Multiple Independent Motions in Segmented Images using Parametric Models with Local Deformations, IEEE Workshop on Motion of Non-Rigid and Articulated Objects, Austin, Texas, Nov. 1994

[BLP89a] H. Buelthoff, J. Little, T. Poggio, A Parallel Algorithm for Real-time Computation of Optical Flow, Nature 337(6207):549-553, 9 Feb 1989

[BLP89b] H. Buelthoff, J. Little, T. Poggio, A Parallel Motion Algorithm Consistent with Psychophysics and Physiology, IEEE 1989 Workshop on Visual Motion, 165-172, March 1989

[Bor90] A. Borst, How Do Flies Land ?, BioScience 40(4): 292-299, April 1990

[Burt81] P. Burt, Fast Filter Transforms for Image Processing, Computer Graphics and Image Processing 16:20-51, 1981

[Burt83] P. Burt, Fast Algorithms for Estimating Local Image Properties, Computer Vision, Graphics, and Image Processing 21:368-382, 1983

[C90] T. Camus, Applications of Pyramid Structures to Multiscale Optical Flow, Brown University Technical Report CS-90-09, August 1990

[C91] T. Camus, Accelerating Optical Flow Computations in VLSI, unpublished report, Brown University January 1991

[C93] T. Camus, Thesis Proposal, Brown AI Memo 93-1005, October 1993

[C94a] T. Camus, Real-Time Optical Flow, SME Technical Paper MS94-176, MVA/SME Applied Machine Vision 1994, Minneapolis Minnesota, June 1994

[C94b] T. Camus, Calculating Time-to-Collision with Real-time Optical Flow, SPIE Visual Communications and Image Processing, Chicago IL, Sept. 1994

[CAMP93] Workshop on Computer Architectures for Machine Perception, New Orleans Louisiana, IEEE Computer Society Press, Los Alamitos CA, 1993

[Cas93] B. Case, SPARC V9 Adds a Wealth of New Features, Microprocessor Report, Feb. 15 1993

[CB91] T. Camus, H. Buelthoff, Space-Time Tradeoffs for Adaptive Real-Time Tracking, Mobile Robots VI, William J. Wolfe, Wendall H. Chun ed., Proc. SPIE 1613, p.268-276, Nov. 1991

[CF88] J. Clark, N. Ferrier, Modal Control of an Attentive Vision System, Proceedings of the IEEE ICCV, p.514-523, 1988

[D86] M. Drumheller, Connection Machine Stereomatching, Proceedings of the 1986 AAAI, Philadelphia PA, p.748-753, August 1986

[DEC92] Alpha Architecture Handbook, Preliminary Edition, Digital Equipment Corp., Maynard, MA, 1992

[DN93] A. Del Bimbo, P. Nesi, Optical Flow Estimation on the Connection

Machine 2, p.267-274, Workshop on Computer Architectures for Machine Perception, New Orleans Louisiana, IEEE Computer Society Press, Los Alamitos CA, 1993

[DNS92] A. Del Bimbo, P. Nesi, J. Sanz, Optical Flow Computation Using Extended Constraints, Department of Systems and Informatics Tech Report DSI-RT 19/92, Faculty of Engineering, University of Florence, Italy, 1992

[Du94a] A. Duchon, Robot Navigation from a Gibsonian Viewpoint. 1994 IEEE International Conference on Systems, Man and Cybernetics, San Antonio, Texas. October 2-5, 1994. IEEE, Piscataway, NJ, pp 2272-2277.

[Du94b] A. Duchon, Ecological Robotics. in preparation, 1994.

[DW93] R. Dutta. C. Weems, Parallel Dense Depth from Motion on the Image Understanding Architecture, Proceedings of the IEEE CVPR, p.154-159, 1993

[EB93] M. Egelhaaf, A. Borst, Motion Computation and Visual Orientation in Flies, Comp. Biochem. Physiol., 104A(4):659-673, 1993

[F86] K. Fant, A Nonaliasing, Real-Time Spatial Transform Technique, IEEE Computer Graphics and Applications 6:71-80, 1986

[Fl92] D. Fleet, Measurement of Image Velocity, Kluwer Academic Publ., Norwell MA, 1992

[FL93] D. Fleet, K. Langley, Recursive Filters for Optical Flow, RPL-TR-9308, Robotics and Perception Lab, Queen's University, Ontario May 1993

[FvD+90] J. Foley, A. vanDam, S. Feiner, J. Hughes, Computer Graphics Principles and Practice, Addison-Wesley, Reading MA, 1990

[G87] F. Glazer, Hierarchal Motion Detection, PhD Dissertation, COINS TR 87-02 University of Massachusetts at Amherst, 1987

[GG84] S. Geman, D. Geman, Stochastic Relaxation, Gibbs distributions, and the Bayesian Restoration of Images, IEEE Transactions on Pattern Analysis and Machine Intelligence, 6:721-741, Nov. 1984

[Hil84] E. Hildreth, The Measurement of Visual Motion, MIT Press, Cambridge MA

[Horn86] B. Horn, Robotic Vision, MIT press, Cambridge MA, 1986

[H81] P. Huber, Robust Statistics, Wiley, New York, 1981

[HB88] I. Horswill, R.Brooks, Situated Vision in a Dynamic World: Chasing Objects, Seventh National Conference on Artificial Intelligence, p.796-800,1988

[Heg87] D. Heger, Optical Flow from Spatiotemporal Filters, Proceedings of the IEEE ICCV, p.181-190, 1987

[HG91] J. Huber, V. Graefe, Quantitative Interpretation of Image Velocities in Real-Time, IEEE 1991 Workshop on Visual Motion, p.211-216, Oct. 1991

[Hil91] E. Hildreth, Recovering Heading for Visually-Guided Navigation, Vision Research 32(6):1177-1192, 1992

[HP90] J. Hennessy, D. Patterson, Computer Architecture: A Quantitative Approach, Morgan Kaufman, San Mateo, CA, 1990

[hp94] PA-RISC 1.1 Architecture and Instruction Set Reference Manual, third edition, Hewlett Packard, Feb. 1994

[HS81] B. Horn, P. Schunck, Determining Optical Flow, Artificial Intelligence 17:185-203, August 1981

[HS93] R. Haralick, L. Shapiro, Computer and Robot Vision II, Addison-Wesley, Reading, MA, 1993

[Hub94a] S. Huber, video presentation, From Animals to Animats: Third International Conference on Simulation of Adaptive Behavior, Brighton, G.B., 8-12 August 94

[Hub94b] S. Huber, personal communication.

[HW91] N. Hatsopoulos, W. Warren Jr., Visual Navigation with a Neural Network, Neural Networks, 4:303-317, 1991

[IRP92] M. Irani, B. Rousso, S. Peleg, Detecting and Tracking Multiple Moving Objects Using Temporal Integration, Proceedings of the ECCV, p.282-287, 1992

[J73] G. Johansson, Visual Perception of Biological Motion and a Model for its Analysis, Perception and Psychophysics, 14:201-211, 1973

[J75] G. Johansson, Visual Motion Perception, Scientific American 232:6 p.76-88, June 1975

[K81] D. Knuth, The Art of Computer Programming, Volume 2: Seminumerical Algorithms, Addison-Wesley, Reading, MA, 1981

[K86] J.J. Koenderink, Optic Flow, Vision Research 26(1):161-180, 1986

[KG92] C. Kambhamettu, D. Goldgof, Point Correspondence Recovery in Non-rigid Motion, Proceedings of the IEEE CVPR, p.222-227, 1992

[KMB+91] C. Koch, A. Moore, W. Bair, T. Horiuchi, B. Bishofberger, J. Lazzaro, Computing Motion Using Analog VLSI Vision Chips, An Experimental Comparison Among Four Approaches, IEEE Workshop on Visual Motion, p.312-324, 1991

[KSL85] E. Kent, M. Shneier, R. Lumia, PIPE: Pipelined Image Processing Engine, Journal Parallel and Distributed Computing 2, P. 50-78, 1985

[KvD78] J.J. Koenderink, A.J. van Doorn, How an Ambulant Observer can Construct a Model of the Environment from the Geometrical Structure of the Visual Flow, Kybernetic 1978, Oldenbourg, Muenchen

[KWM89] C. Koch, H.T. Wang, B. Mathur, Computing Motion in the Primate's Visual System, Journal of Experimental Biology, 146:115-139,1989

[Lee76] D. Lee, A Theory of Visual Control of Braking Based on Information about Time-to-Collision, Perception 5:437-459, 1976

[L88] J.Little, Integrating Vision Modules on a Fine-Grained Parallel Machine, in Machine Vision, Academic Press, p.57-96, 1988

[LBC87] J. Little, G. Blelloch, T. Cass, Parallel Algorithms for Computer Vision on the Connection Machine, Proceedings of the IEEE ICCV, p.587-591, 1987

[LBP88] J. Little, H. Buelthoff, T. Poggio, Parallel Optical Flow Using Local Voting, Second International Conference on Computer Vision, p.454-9, 1988

[LK93] J. Little, J. Kahn, A Smart Buffer for Tracking Using Motion Data, p.257-266, Workshop on Computer Architectures for Machine Perception, New Orleans Louisiana, IEEE Computer Society Press, Los Alamitos CA, 1993

[LP80] H. Longuet-Higgins, K. Prazdny, The Interpretation of a Moving Retinal Image, Proc. of the Royal Society of London, B 208:385-397, 1980

[LPK93] C-C Lin, V. Prasanna, A. Khokhar, Scalable Parallel Extraction of Linear Features on MP-2, p.352-361, Workshop on Computer Architectures for Machine Perception, New Orleans Louisiana, IEEE Computer Society Press, Los Alamitos CA, 1993

[LV89] J. Little, A. Verri, Analysis of Differential and Matching Methods for Optical Flow, IEEE 1989 Workshop on Visual Motion, 173-180, March 1989

[Mor77] H. Moravec, Towards Automatic Visual Obstacle Avoidance, 5th IJCAI, p.584, 1977

[MBLB91] H. Mallot, H. Buelthoff, J. Little, S. Bohrer, Inverse Perspective Mapping Simplifies Optical Flow Computation and Obstacle Detection, Biological Cybernetics, 64:177-185,1991

[M82] D. Marr, Vision, W. H. Freeman and Company, New York, 1982

[MB90] D. Murray, B. Buxton, Experiments in the Machine Interpretation of Visual Motion, MIT Press, Cambridge MA, 1990

[McL93] E. McLellan, The Alpha AXP Architecture and 21064 Processor, p.36-47, June 1993 IEEE Micro

[MH80] D. Marr, E. Hildreth, Theory of Edge Detection, Proceedings of the Royal Society of London, B:207 p.187-217, 1980

[Mot94a] PowerPC 603 RISC Microprocessor User's Manual, part# MPC603UM/AD, Motorola, 1994

[Mot94b] PowerPC 604 RISC Microprocessor Technical Summary, part# MPC604/D, Motorola, 1994

[MU81] D. Marr, S. Ullman, Directional Sensitivity and Its Use in Early Visual Processing, Proc.R.Soc.Lond. B 211, p.151-180, 1981

[Nag87] H-H Nagel, On the Estimation of Optical Flow: Relations between Different Approaches and Some New Results, Artificial Intelligence 33:299-324, 1987

[N91] R.Nelson,Qualitative Detection of Motion by a Moving Observer, Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, p.173-178,1991

[NA89] R. Nelson, J. Aloimonos, Obstacle Avoidance Using Flow Field Divergence, IEEE PAMI-11 no. 10, p.1102-1106, October 1987

[Nal93] V. Nalwa, A Guided Tour of Computer Vision, Addison-Wesley, Reading, Massachusetts, 1993

[NG92] S. Negahdaripour, V. Ganesan, Simple Direct Computation of the FOE, Proceedings of the IEEE CVPR, p.228-235, 1992

[NH87] S. Negahdaripour, B. Horn, Direct Passive Navigation, IEEE PAMI, 9:1, Jan. 1991

[NS88] K. Nakayama, G. Silverman, The Aperture Problem-I. Vision Research 28(6):739-746, 1988

[NSY91] S. Negahdaripour, A. Shokrollahi, C.H. Yu, Optical Sensing for Undersea Robotic Vehicles, Robotics and Autonomous Systems 7:151-163, 1991

[O85] VLSI Image Processing, R. Offen ed., McGraw-Hill, New York, 1985

[OLT93] T. Olsen, R. Lockwood, J. Taylor, Programming a Pipelined Image Processor, p.93-100, Workshop on Computer Architectures for Machine Perception, New Orleans Louisiana, IEEE Computer Society Press, Los Alamitos CA, 1993

[P90] J. Perrone, Simple Technique for Optical Flow Estimation, Journal of the Optical Society of America, 7(2):264-278, 1990

[P92] J. Perrone, Model for the Computation of Self-Motion in Biological Systems, Journal of the Optical Society of America, 9(2):177-194

[PBF89] J-M Pichon, C. Blanes, N. Franceschini, Visual Guidance of a Mobile Robot Equipped with a Network of Self-Motion Sensors, SPIE Vol. 1195 Mobile Robots IV, p.44-53, 1989

[PFH87] S. Peleg, O. Federbush, R. Hummel, Custom-made Pyramids, p. 125-146, in Parallel Computer Vision, L. Uhr ed., Academic Press, Orlando Florida, 1987

[PH94] D. Patterson, J. Hennessy, Computer Organization and Design: the Hardware/Software Interface, Morgan Kaufmann, San Mateo CA, 1994

[PRH81] T. Poggio, W. Reichardt, K. Hausen, A Neuronal Circuitry for Relative Movement Discrimination by the Visual System of the Fly, Naturwissenschaften 68, p.443-446, 1981

[PVT91] T. Poggio, A. Verri, V. Torre, Green Theorems and Qualitative Properties of the Optical Flow, AI Memo 1289, Art. Int. Lab, MIT 1991

[R75] I. Rock, An Introduction to Perception, Macmillan Publishing Co., New York 1975

[RA86] V. Ramachandran, S. Anstis, The Perception of Apparent Motion, Scientific American, June 1986; also in compilation The Perceptual World, p.139-151,

[Reg86] D. Regan, Visual Processing of Four Kinds of Relative Motion, Vision Research (26)1:127-145, 1986

[RL85] J. Rieger, D. Lawton, Processing Differential Image Motion, Journal of the Optical Society of America, 2(2):354-359

[S91] A. Singh, Optic Flow Computation A Unified Perspective, IEEE Computer Society Press, Los Alamitos CA, 1991

[SAH91] E. P. Simoncelli, E. H. Adelson, D. J. Heeger, Probability Distributions of Optical flow, Proceedings of the IEEE CVPR, p.310-315, May 1991

[SHBV87] D. Schaefer, P. Ho, J. Boyd, C. Vallejos, The GAM Pyramid, p. 15-42, in Parallel Computer Vision, L. Uhr ed., Academic Press, Orlando Florida, 1987

[SPL92] L. Stewart, A. Payne, T. Levergood, Are DSP Chips Obsolete?, DEC Cambridge Research Lab, technical report CRL 92/10, Nov. 1992

[SRT92] M. Shah, K. Rangarajan, P-S Tsai, Motion Trajectories, in Proceedings of the IEEE CVPR, p.839-841, 1992

[SS85] J. van Santen, G. Sperling, Elaborated Reichardt Detectors, Journal of the Optical Society of America, 2(2):300-321, 1985

[SS93] A. Schilling, W. Strasser, EXACT: Algorithm and Hardware Architecture for an Improved A-Buffer, ACM SIGGRAPH, p.85-91, 1993

[St87] Q. Stout, Pyramid Algorithms Optimal for the Worst Case, p.147-168

[SU87] A.Spoerri,S.Ullman,The Early Detection of Motion Boundaries, International Conference on Computer Vision, p.209-218, 1987

[Sun87] The SPARC Architecture Manual, part# 800-1399-07, Sun Microsystems, Mountain View CA, 1987

[SW94] J. Smith, S. Weiss, PowerPC 601 and Alpha 21064: A Tale of Two RISC's, p.46-58, June 1994 IEEE Computer

[T90] J. Tresilian, Perceptual Information for the Timing of Interceptive Action, Perception 19:223-239, 1990

[T91] J. Tresilian, Empirical and Theoretical Issues in the Perception of Time to Contact, Journal of Experimental Psychology: Human Perception and Performance, 17:3 p.865-876, 1991

[TGS91] M. Tistarelli, E. Grosso, G. Sandini, Dynamic Stereo in Visual Navigation, Proceedings of the IEEE CVPR, p.186-193, 1991

[TLL87] S. Tanimoto, T. Ligocki, R. Ling, A Prototype Pyramid Machine for Hierarchal Cellular Logic, p.43-83, in Parallel Computer Vision, L. Uhr ed., Orlando Florida, Academic Press, 1987

[TM88] Thinking Machine technical report HA87-4, April 1987

[To92a] S. Toelg, Gaze Control for an Active Camera System by Modeling Human Pursuit Eye Movements, SPIE Intelligent Robots and Computer Vision XI, p.585-598, Nov. 1992

[To92b] S. Toelg, personal communication.

[TR93] S. Tubaro, F. Rocca, Motion Field Estimators and Their Application to Image Interplation, in Motion Analysis and Image Sequence Processing, 1993, Kluwer Academic Publishers, Norwell MA

[TS91] M. Tistarelli, G. Sandini, Direct Estimation of Time-to-impact from Optical Flow, IEEE 1991 Workshop on Visual Motion, p.226-233, Oct. 1991

[TWR94] D. Touretzky, H. Wan, A. Redish, Neural Representation of Space in Rats and Robots, in J.M. Zurada and R.J. Marks, eds., Computational Intelligence: Imitating Life. Proceedings of the symposium at the 1994 IEEE World Congress on Computational Intelligence, IEEE Press, 1994

[U79] S. Ullman, The Interpretation of Visual Motion, The MIT Press, Cambridge MA, 1979

[UGVT88] S. Uras, F. Girosi, A. Verri, V. Torre, A Computational Approach to Motion Perception,Biological Cybernetics, 60:79-87, 1988

[Uhr87] L. Uhr, Highly Parallel, Hierarchal, Recognition Cone Perceptual Structures, p.249-292, in Parallel Computer Vision, L. Uhr ed., Academic Press, Orlando Florida, 1987

[VG92] J.A. Vlontzos, D. Geiger, A MRF Approach to Optical Flow Estimation, Proceedings of the IEEE CVPR, p.853-856, 1992

[VP87] A. Verri, T. Poggio, Against Quantitative Optical Flow, Proceedings of the IEEE ICCV, p.171-180, 1987

[W93] C. Weems, Jr., The Second Generation Image Understanding Architecture and Beyond, p.276-285, Workshop on Computer Architectures for Machine Perception, New Orleans Louisiana, IEEE Computer Society Press, Los Alamitos CA, 1993

[WA85] A. Watson, A. Ahumada Jr, Model of Human Motion Sensing, Journal of the Optical Society of America, 2(2):322-342, 1985

[War88] W. Warren Jr., Action Modes and Laws of Control for the Visual Guidance of Action, in Complex Movement Behavior: The motor-action controversy, p.339-380, O. Meijer and K. Roth eds., Elsevier Science Pubs., B.V. (North-Holland), 1988

[WD93] S. White, S. Dhawan, POWER2: Next Generation of the RISC System/6000 Family, in RISC System/6000 Technology: Volume II, p.2-12, IBM, 1993

[WH89] W. Warren Jr., D. Hannon, Eye Movements and Optical Flow, Journal of the Optical Society of America, 7(1):160-169, 1989

[WM93] J. Weber, J. Malik, Robust Computation of Optical Flow in a Multi-Scale Differential Framework, Fourth International Conference on Computer Vision, p.12-20, 1993

[WWB88] A. Waxman, J. Wu, F. Bergholm, Convected Activation Profiles: Receptive fields for Real-Time Measurement of Short-Range Visual Motion, Proceedings of the IEEE CVPR, p.717-723, 1988

[WZ91] J. Woodfill, R. Zabih, An Algorithm for Real-Time Tracking of Non-Rigid Objects,Ninth National Conference on Artificial Intelligence, p.718-723,1991

[ZL93] A. Zakhor, F. Lari, Edge Based 3-D Camera Motion Estimation with Application to Video Coding, in Motion Analysis and Image Sequence Processing, 1993, Kluwer Academic Publishers, Norwell MA

Real-Time Optical Flow

Ted Camus

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-94-36
May 1995