Time-to-Collision Algorithm and Real-Time Implementation

by

Haiqian Cheng

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1999

Author ....
Department of Electrical Engineering and Computer Science
May 18, 1999

Certified by ....[illegible]
Berthold K.P. Horn
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by ....[illegible]
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Time-to-Collision Algorithm and Real-Time Implementation

by

Haiqian Cheng

Submitted to the Department of Electrical Engineering and Computer Science
on May 18, 1999, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

The time-to-collision task is that of estimating the ratio of the distance of an object to the relative velocity in the direction of that object, based on time-varying images resulting from the 3-D motion of a camera with respect to the object. Traditional methods require large amounts of memory and computation time, and are not suited for real time hardware implementations. We used a direct method to provide a non-linear closed form solution for the case of translational motion, because it is more robust to noise, and provides a straightforward solution for time-to-collision. Three approaches – gradient descent, a grid search method and an improved grid search were used for solving the non-linear equations. Two alternatives – analog VLSI chip implementation and digital signal processing microprocessor based system design were investigated. Finally, the time-to-collision algorithms were simulated on a Texas Instruments digital signal processor(DSP) TMS320C67 based system. From the results of the simulation on the TMS320C67 Code Generator and Simulator, it is shown that the improved grid search method provides reliable results and high computational efficiency with real time performance when implemented with the DSP TMS320C67.

Thesis Supervisor: Berthold K.P. Horn
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

I am indebted to many professors, students, and friends for their assistance, support, and suggestions.

I must thank my thesis supervisor, Professor Berthold K.P. Horn for his support and guidance. Professor Horn's advice and insight into machine vision is invaluable. It has been a great learning experience to work with him. I also extend my deep gratitude to Dr. Ichiro Masaki for his support and numerous helpful suggestions on the system level issues for this project. Professor Hae-Seung Lee also gave me some advice on analog circuit design.

The students at AI lab and MTL provide a lot of technical advice. I wish to extend special thanks to Andrew Ng for his immense help and numerous suggestions made regarding my research. Many thanks also to my friends Paul Fiore, Nicole Love, Kinh Tieu, Erik Rauch, Zubair Talib, Kush Gulati, Bo Feng, Yong Chen, Yangjun Fang, Bikui Chen, An Gao, Jing Yu, Zhitao Cao and Minghao Qi in MIT. They provided huge amount of both friendship and technical advice. I also thank my friends Chengyang Li and Xia Bo from the University of Arizona for their valuable suggestions regarding digital signal processing hardware.

Above all, I thank my family for their love and support. Without that, this work would not have been be possible. This thesis is dedicated to my parents.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The motion vision processing problem involves using a sequence of camera images to estimate motion in the world. This has been well-studied over the past few decades, and numerous algorithms having been proposed, ranging from feature based methods to motion field methods [12]. In this thesis, we focus on the time-to-collision problem and its real-time implementations, which in particular has immediate and practical application to automatic navigation.

1.1 Time-to-collision for Automatic Navigation

At a high level, the goal of automatic navigation is to move a mobile robot or vehicle from some starting position to a goal location. The task of having a program automatically drive a car from Boston to New York City is an example of this, and so is the task of having a mail-delivery robot wander from the mail-room to a particular office in a building. In both of these tasks, it is generally crucial for the car or robot to avoid running into other objects.

Focusing on the driving example, by exploring emerging intelligent transportation system(ITS) technologies for automatic navigation, road-vehicle systems can be safer, more efficient, and more environment-friendly [16]. ITS systems can broadly be classified into two classes: On-road systems and in-vehicle systems [1]. As its name suggest, the on-road approach involves building sensors into the environment (the "road") to monitor and analyze vehicular motion, and uses the results of that analysis to make appropriate decisions to control the traffic. In contrast, the in-vehicular approach involves building a system into the mobile object itself (the "car") to monitor local conditions and to make appropriate navigational movements so as to avoid collisions. In this thesis, we focus on implementations that generally fall into this latter category of in-vehicular approaches.

Given images from a camera mounted on the mobile object (the car or robot), it is the task of the time-to-collision algorithm to give a warning whenever a collision with another object is imminent, and in fact to further estimate the time remaining to the collision. This facilitates the building of an early-warning collision system, which can often be crucial to avoiding such collisions.

A curious but very useful property of the time-to-collision problem is that, despite operating in a three-dimensional world where stereoscopic vision is needed for most vision processing tasks (such as depth estimation), the time-to-collision problem requires only *one* camera. It is certainly true that, without additional information, it is impossible to estimate the absolute value of depth or absolute velocity when we are given a sequence of images taken by a single moving camera. Thus, it would seem impossible, using just a single camera, either to estimate the distances of objects we might collide into, or to estimate how quickly we are approaching them, and therefore that time-to-collision cannot be solved if we had only a single camera.

Happily, this is not the case. While there is an ambiguity in the scaling of distances and in the scaling of velocities, there is no ambiguity in the *ratio* of distance to velocity, and this ratio gives exactly the time-to-collision. This fact will be explained in considerable detail later in this thesis, and is the key fact that will enable us to design sound time-to-collision algorithms that require only one camera mounted on the moving object.

1.2 Real-time Implementations

Most interesting vision problems are difficult, perhaps even "AI-complete," and so the algorithms designed for them are often only attempts at approximating truly "optimal" solutions. And because many vision algorithms were designed to try to achieve the best possible performance perhaps even under adverse conditions, it is often the case that they will engage in a large amount of costly statistical estimation and processing to give answers that are as accurate as possible. Taking this approach, the literature reports numerous successes of these algorithms, which occasionally even beat human performance on restricted tasks. [5]

Yet because of a growing desire to apply vision algorithms to real systems that often have tight time constraints, much recent attention has also been focused on the particular case of *real-time* implementations of such algorithms. Indeed, this is particularly important when applying time-to-collision algorithms to obstacle avoidance—if we are about to collide into another object in 1s, then any algorithm that takes longer than 1s to warn us about it would not be of much use!

This thesis addresses the problem of real-time implementations of solutions to the time-to-collision problem. To successfully build such a system, our requirements are two-fold:

- (Algorithm) We need to design an algorithm that is sound and robust, yet fast enough for real-time applications.
- (Implementation) We need to make careful choices in the implementation of the algorithms, both to reduce cost and to ensure sufficiently fast processing for a quick time-to-collision estimate.

This work addresses the real-time implementation of the time-to-collision problem. The system does not perform high level navigation such as following certain lane markings. Instead, it is designed to detect and avoid obstacles, in order for the vehicle to travel safely in the unknown environment. Collision avoidance is a key consideration for people working with vehicles. Time-to-collision is an effective tool in collision avoidance, because it provides the time it takes for the camera to hit the object without physical contact in the measurement. Once implemented in real time, a time-to-collision system can be useful in guiding a moving vehicle, a robot or perhaps a blind person.

There are two possible approaches for real-time implementation of time-to-collision. The first is based on the system level integration of an imager (usually a CMOS or CCD) camera and a digital processor to run the time-to-collision algorithm [18]. The other solution is a specialized chip which includes both image sensors and image processing units [22]. In either case, the designed hardware has to have both reliable results and real-time performance.

The time-to-collision problem is concerned with time dependent features of the scenes, which requires inputs from two time-frames. Due to the fast motion of vehicle, the camera inside the vehicle must take images at real-time video rate, which is 30 frames per second for a real-time implementation. The computation of the time-to-collision is required to be completed in less than 33ms per frame. Because of the large number of pixels in each image, the number of arithmetic operations involved will be huge. Most existing motion vision processing problems are computationally expensive, and are not suited for real-time implementation. In this research, we have explored new algorithms with less computation and reliable results for time-to-collision and its possible real-time implementation.

1.3 Thesis Organization

The remainder of this thesis is structured as follows: Chapter 2 surveys related work in time-to-collision processing, and then introduces the theoretical framework and assumptions that are used throughout this thesis. Based on this framework, Chapter 3 then describes and gives the mathematical derivations and characteristic features of different time-to-collision algorithms—gradient descent algorithm, grid search algorithm, and improved grid search algorithm. Chapter 4 then investigates various issues in the hardware-implementations of these algorithms, and shows the tradeoff between two main alternatives—Analog VLSI with a parallel algorithm (the gradient descent algorithm), and Digital signal processing (DSP) with a serial algorithm (the improved grid search algorithm). Eventually choosing the latter, we also show how software that was specialized for the particular chosen chip architecture (a DSP TMS320C67 chip) was able to significantly improve computational and memory efficiency. Chapter 5 then reports on experiments using a DSP microprocessor simulator, that compares the efficiency of different time-to-collision algorithm implementations. Finally, Chapter 6 closes with conclusions and future work.

Chapter 2

Theoretical Foundation

This chapter presents a brief discussion of some background and theoretical foundation that is directly related to this work.

2.1 Background

The standard approach for motion vision processing is to first compute correspondences. Generally, the correspondences can be either characteristic features or motion field estimates[24]. From the computed correspondences, camera motion or scene structure can be obtained. In the characteristic feature based methods, an estimate of camera motion and scene structure is found by identifying characteristic features of objects, such as edges or corners, and tracking them through the image sequence. The other method is based on motion field estimation. In the motion field methods, optical flow, the apparent velocity of each location in the image, is first calculated through the image brightness at every location to approximate the motion field. The calculated optical flow is then used to estimate the camera motion and scene structure.

In the standard approaches mentioned above, finding correspondence is a hard problem. Features have to be recovered accurately in order to correctly recover camera motion. For characteristic feature based methods, it is still an open research subject to find out what set of features such as edges, corners is best for different images. In addition, a window search, which is generally used in characteristic feature based method, is a computationally expensive process. For motion field methods, iterative approaches are very often used in obtaining optical flow. Accurate optical flow usually requires a large number of iterations. Meanwhile, optical flow suffers from the "aperture problem", which is that local flow estimates are only in the direction of the image brightness gradient. Thus, recovery of optical flow is computationally expensive and ill-conditioned [5]. Errors calculated in the correspondences will be carried to the next stage of camera motion estimation, therefore causing the final result to be not very robust in the presence of noise. Because of the reasons above, the standard methods are not suitable for camera motion estimation in a real time system. We are now exploring some new approaches that promise to be more robust and computationally less expensive than the traditional methods.

Time-to-collision is solved with a direct method, which was developed by Horn and Negahdaripour [21]. It forms the theoretical foundation for the time-to-collision problem. This method is an improved motion field method, and uses the direct relationship between camera geometry and image brightness. Therefore it solves the time-to-collision problem, while skipping the optical flow computation as an intermediate quantity, causing less computation for obtaining the camera motion. At the same time, by making use of image intensity for every pixel on a two-image sequence and minimizing error using a least squares approach, our approach avoids the aperture problem, and is relatively robust to quantization error, noise, illumination gradients, and other effects [6].

2.2 Previous Work

Given the direct method as the theoretical foundation, we should also mention a few projects related to the present work. McQuirk implemented analog circuitry to solve a related problem, finding the focus of expansion (FOE) of time-varying image sequences [17]. Focus of expansion is the intersection of the translation vector of the camera with the image plane. As one moves through a world of static objects, for a given direction of translational motion and direction of gaze, the world seems to be flowing out of one particular retinal point. That point is the focus of expansion. The focus-of-expansion problem finds the direction of the motion, while velocity and distance are not of interest. Conversely, our time-to-collision problem obtains the ratio of distance and velocity in the direction of the motion, while direction of motion itself is not of interest. Both problems can be solved using the direct method and least squares error as the theoretical basis.

Frumkin investigated the problem of time-to-collision in a digital chip design [3]. Because of computation complexity, only a 1-D special case was implemented. As a generalization of that work, in this project we will explore algorithms for the 2-D case and their possible real time implementations.

2.3 Direct Method

The direct method builds the theoretical foundation for time-to-collision problem. It is derived from the following two constraints [6]:

- Perspective Projection Constraint
- Constant Brightness Constraint

To understand how the direct method works, we first review these two constraints individually, then combine these two constraints to form the method.

2.3.1 Perspective Projection Constraint

In our application, a camera based coordinate system is used to express the correspondence between objects in the 3-D world and images in the image plane. Figure 2-1 shows this camera based coordinate system.

In this figure, the origin is located at the center of projection, which is taken as the camera location. The image plane is positioned at the principal distance f, parallel to the xy-plane. The z axis represents the direction that is perpendicular to the image plane. Position $R = (X, Y, Z)^T$ in the real world system is mapped into the

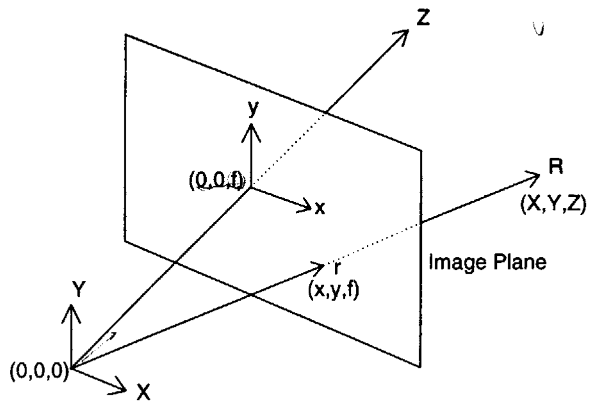


Figure 2-1: Camera-centered geometry and perspective projection

image plane at $r = (x, y, f)^T$ according to the *perspective projection constraint.* This constraint is modeled as a pinhole camera which provides a relation between object coordinate R and image coordinate r as follows:

$$r = \frac{fR}{R \cdot \hat{z}} \tag{2.1}$$

where $\hat{z}$ is the unit vector in z direction. Equation (2.1) is called the *perspective projection equation.*

As we know, the motion of the object relative to the camera is the opposite of the motion of the camera relative to the object. If the camera moves with instantaneous translational velocity $t = (t_x, t_y, t_z)^T$ and instantaneous rotational velocity $\omega = (\omega_x, \omega_y, \omega_z)^T$, the motion of a world point in the camera coordinate system will be:

$$R_t = -t - (\omega \times R) = -t - (\omega \times r)(\frac{R \cdot \hat{z}}{f}) \tag{2.2}$$

The motion R_t of the object in the world system generates a motion on the image plane, which can be represented as $r_t = (u, v, 0)$. The relation between these two motions can be determined by taking the derivative of equation 2.1 with respect to time:

$$r_t = \frac{f}{(R \cdot \hat{z})^2}\Big((R \cdot \hat{z})R_t - (R_t \cdot \hat{z})R\Big) \tag{2.3}$$

From the motion R_t obtained in equation 2.2, we can rewrite equation 2.3 to represent the motion field r_t on the image plane as:

$$r_t = -\hat{z} \times \left(r \times (\frac{r \times \omega}{f} - \frac{t}{R \cdot \hat{z}})\right) \tag{2.4}$$

2.3.2 Constant Brightness Constraint

Using the perspective projection, we have built the relation between camera motion and the motion field. To relate the motion field to the image brightness, we will introduce the concept of optical flow.

Optical flow is a two-dimensional field of vectors (u,v) corresponding to the apparent motion of these patterns over a sequence of frames. It can be used to approximate the motion field [6]. Optical flow can be recovered using temporal variations in image intensity patterns by computing the partial derivatives using discrete approximation to derivatives.

The *constant brightness constraint* assumes that the brightness of a particular point in the image pattern remains steady with the change of view. This is saying that the perceived motion of brightness patterns within the image is only due to actual motion of the scene and not due to varying illumination conditions. Under this constraint, the optical flow for the 2-D case $(u,\ v)$ is given by:

$$\frac{dE}{dt} = E_t + E_r \cdot r_t = E_t + E_x u + E_y v = \frac{\partial E}{\partial t} + \frac{\partial E}{\partial x} u + \frac{\partial E}{\partial y} v = 0 \tag{2.5}$$

where $E(x, y, t)$ is the image brightness at the point (x, y) in the image plane at time t. E_x, E_y, E_t are the brightness gradients with respect to x, y coordinates and time, respectively. The vector (u, v) is the optical flow field. Equation 2.5 is called the *constant brightness constraint.*

It should be noted that there is a limit beyond which the first order *constant brightness constraint* will not apply: when motion between images is more than 2 pixel/frame. Above that we cannot neglect higher order terms in the Taylor series expansion of $\frac{dE}{dt}$.In addition, from equation 2.5, we can see that in the location where the brightness gradient is zero, the optical flow field is locally unrecoverable. So, usually the input image is required to have rich visual texture in order to use the equation above.

2.3.3 Recovering General Motion of the Camera

In most cases, the optical flow will correspond to the motion field. This will allow us to estimate relative motion with time-varying images. In the following, we will explain how the *constant brightness constraint* and the *perspective projection constraint* are combined in one direct equation to recover the general motion of the camera. The *constant brightness constraint* builds the relationship between image brightness and optical flow field, while the *perspective projection constraint* connects camera motion to the motion field. The direct method combines those two separate stages into one step. Putting together the *constant brightness equation* and the *perspective projection equation*, we can get:

$$E_t + \frac{v \cdot \omega}{f} + \frac{s \cdot t}{R \cdot \hat{z}} = 0 \tag{2.6}$$

where the new variables s and v are 3×1 vectors defined as:

$$s = (E_r \times \hat{z}) \times r = \begin{bmatrix} E_x \\ E_y \\ -(\frac{x}{f}E_x + \frac{y}{f}E_y) \end{bmatrix}$$

and

$$v = r \times s = \begin{bmatrix} E_y + y(xE_x + yE_y) \\ -E_x - x(xE_x + yE_y) \\ yE_x - xE_y \end{bmatrix}$$

2.4 Solving for Time-to-Collision

2.4.1 Assumptions

We now perform the mathematical derivation of time-to-collision. Three assumptions were made to simplify this problem. Firstly, it is assumed that we have either a fixed camera in a changing environment or a moving camera in a static environment. If there is more than one object with independent motion, then we shall assume that the image has been segmented and that we can concentrate on one region corresponding to a single object. Secondly, in this project, we consider only the case where the relative motion is purely translational, because motion of vehicles on the road is generally pure translational and the theoretical derivation will provide a nice closed form solution for translational motion. Thirdly, a further assumption is made that the local shape of the surface of the object is planar. These assumptions are approximately valid for many cases in a real world problem when the camera has a narrow view angle, and simplify the time-to-collision problem to reduce a large amount of computation.

2.4.2 Time-to-Collision for Translational Case on Planar Surface

Assuming the object has only translational motion, time-to-collision is defined as the ratio of the distance of an object to the relative translational velocity in the direction of that object [3]. With the camera based coordinate system defined in section 2.3.1, the time-to-collision is expressed as:

$$\tau = \frac{R \cdot \hat{z}}{t \cdot \hat{z}} \tag{2.7}$$

where $R \cdot \hat{z}$ is the distance of object and the camera in the direction of the optical axis, and $t \cdot \hat{z}$ is the translational velocity in the direction of the optical axis.

As mentioned above, the shape of the object is assumed to be a planar surface, and have only translational motion. The moving object can be described by a normal vector $n = (n_x, n_y, n_z)$ of the planar surface and the translational velocity vector $t = (t_x, t_y, t_z)^T$. Vector n and t are shown in figure 2-2.

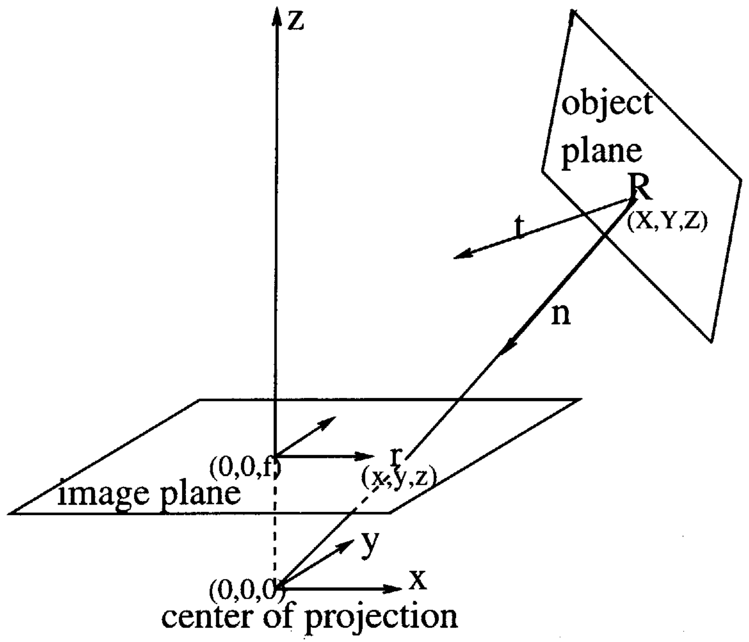


Figure 2-2: The time-to-collision is defined as $\frac{1}{n \cdot t}$

We choose the scale of n such that the value of $1/\|n\|$ is the distance from the center of projection to the object plane. With this scale, time-to-collision can also be expressed as the reciprocal of the velocity along the normal vector n:

$$\tau = \frac{R \cdot n}{t \cdot n} = \frac{1}{t \cdot n} \tag{2.8}$$

Equation 2.6 is used to solve for t and n in equation 2.8. In the case of pure translation, $\omega = 0$ and equation (2.6) simplifies to become:

$$E_t + \frac{s \cdot t}{R \cdot \hat{z}} = 0 \tag{2.9}$$

Equation 2.9 provides the relationship between the image brightness and the motion of the object.

The planar object constraint mentioned above gives the equation below:

$$R \cdot n = 1 \tag{2.10}$$

Combining with the *perspective projection equation* (2.1), equation 2.10 can be used to relate depth (z component of R expressed as $R \cdot \hat{z}$) to the point in the image plane as:

$$R \cdot \hat{z} = \frac{f}{r \cdot n} \tag{2.11}$$

Combining equation 2.11 with equation 2.9, we have:

$$E_t + (r \cdot n)(t \cdot s) = 0 \tag{2.12}$$

Equation 2.12 is valid at any instant in time for every pixel in the image. In this nonlinear equation, there are six unknowns, a 3-D translational motion $t = (t_x, t_y, t_y)$ and a 3-D normal vector $n = (n_x, n_y, n_z)$. Because of the scaling factor ambiguity mentioned in section1.1, this equation has five degrees of freedom. In general, five points in the image plane uniquely determine the purely translational motion of a camera. However, there is a drawback to utilizing so little of the available information. The brightness derivatives are hard to estimate accurately with only five points especially if that the image is noisy, and the optical flow we measure is usually corrupted by noise. Reliable results can be obtained if the brightnesses values from the whole image plane are used. We therefore use a least squares error method, which is more robust against noise. When we are given N points, vectors t and n obtained from the least squares error method are those best fitted for all the given points. The error function for this case is defined as:

$$\Lambda(n, t) = \sum_{x,y} \Big(E_t + (r \cdot n)(t \cdot s) \Big)^2 \tag{2.13}$$

Equation 2.8 and equation 2.13 are the heart of the time-to-collision problem. Equation 2.13 is used to find vector n and t when minimizing the error function. Then time-to-collision is solved with equation 2.8 from vector n and t.

Chapter 3

Time-to-Collision Algorithm

The mathematical derivation outlined in the previous chapter provides nonlinear equations for the time-to-collision problem. Now we turn to algorithms to solve the nonlinear equation with a goal of constructing a dedicated chip or system to estimate time-to-collision. Recall that the defining equations are given by:

$$\Lambda(n,t) = \sum_{x,y}\Big(E_t + (r \cdot n)(t \cdot s)\Big)^2 \tag{3.1}$$

$$\tau = \frac{1}{t \cdot n} \tag{3.2}$$

The nonlinear equation is a minimization problem with unknown vectors t and n. A simple approach to the minimization problem is the method of gradient descent. Besides the traditional method, we also introduce a new approach, a grid search method and its improved version.

3.1 Gradient Descent Method

3.1.1 Mathematical Derivation

Gradient descent is a traditional method to solve non-linear minimization problems. The gradient descent method is based on the well-known fact that a function decreases locally fastest in the direction of the negative gradient [14]. For the problem of minimizing $f(x)$ with respect to x, the gradient descent method will provide an iterative solution as:

$$x^{i+1} = x^i - \alpha \Delta f(x^i) \tag{3.3}$$

where α is a positive scale factor that can be used to adjust the step size. It is also called the "learning rate" for the function $f(x)$. And x^i and x^{i+1} denote the estimates of x after the ith iteration and after the $i+1$th iteration. Vector x is an $N \times 1$ column vector consisting of N parameters:

$$x = [x_1, x_2, ..., x_N]^T$$

$$\Delta f(x^i) = [\frac{\partial f(x)}{\partial x_1}, \frac{\partial f(x)}{\partial x_2}, ..., \frac{\partial f(x)}{\partial x_N}]^T_{x=x^i}$$

Figure 3-1 illustrates gradient descent for special case when x consists of only one element ($x = a$). In figure 3-1, $f(a)$ has a positive gradient. The estimate of a after ith iteration will be in the negative direction of a.

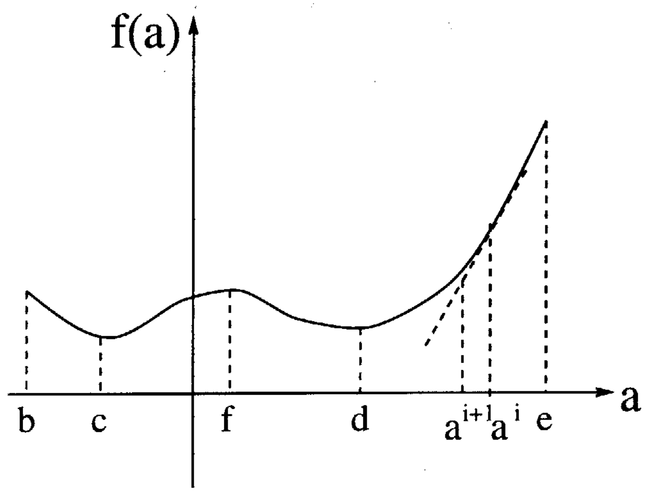


Figure 3-1: Gradient descent: a^i and a^{i+1} denoting the estimates of a after ith iteration and $i+1$th iteration

In our application, the gradient descent method is used to minimize the error function $\Lambda(x)$ in the nonlinear equation 3.1 with respect to x, where x is a column vector:

$$x = [n_x, n_y, n_z, t_x, t_y, t_z]^T$$

The vector $n = (n_x, n_y, x_z)$ and $t = (t_x, t_y, t_z)$ can thus be obtained iteratively as:

$$n^{i+1} = n^i - \alpha \frac{\partial \Lambda}{\partial n^i} = n^i - \alpha \sum_{x,y} 2\Big(E_t + (r \cdot n^i)(s \cdot t^i)\Big)(s \cdot t^i) r \tag{3.4}$$

$$t^{i+1} = t^i - \alpha \frac{\partial \Lambda}{\partial t^i} = t^i - \alpha \sum_{x,y} 2\Big(E_t + (r \cdot n^i)(s \cdot t^i)\Big)(r \cdot n^i) s \tag{3.5}$$

Notice that in equation 3.4, solving for n, we need to use the value of t, while in the equation 3.5 solving for t, the value of n is needed too. The vector n is estimated with equation 3.4, by assuming t known; the vector t is estimated with equation 3.5, by assuming n known. In the next iteration, the vectors n and t will be re-estimated using the same equations with the new value of n and t, and the iterative procedure continues until both converge to a minimum error. In general, the gradient descent method uses an iterative computation to solve equation 3.1.

3.1.2 Characteristic Features

The gradient descent method is very effective in decreasing the function in the initial stage of the algorithm, when the iterative solution is far from the true solution. However, we need to carefully choose the learning rate α. If the learning rate α is too large, the a^{i+1} may overshoot to get into an oscillation around the solution and not converge. On the other hand, if α is too small, this algorithm will converge very slowly. Generally, because different image sequences may have a different type of image data, the learning rate α is usually adjusted based on the experimental results.

In this iterative computation, an initial value of n and t should be provided. Because the gradient descent method always searches in the direction of negative gradient, it may converge to a local minimum of $f(x)$. For example, in figure 3-1, when the initial value of a is located in the region between f and e, the final value will be converged to local minimum d. When the initial value of a is located in the region between b and f, gradient descent method will converge to the global minimum c. So when a function has more than one minimum, it is hard to tell if it will converge to a local minimum or the global minimum from the gradient descent method. From experiments, we know that the error function in equation 3.1 has many local minima, whereas the actual solution is the global minimum. Because the value where the iterative computation converges depends on the initial value at the start of the algorithm, it is important to choose good initial values for n and t. This can be achieved from doing some experiments with different initial values.

3.1.3 Algorithm Description

The computational structure of the gradient descent method is shown below.

```
...; load image 1 and image 2
...; smooth image 1 and image 2
...; compute derivatives $(E_x, E_y, E_t)$ from image 1 and image 2
...; initialize vector $t$ and $n$

for (i=0; i<iterations; i++)
   for (j=0; j<rows; j++)
      for (k=0; k<cols; k++)
         ...; compute the gradient of error function $\Lambda$
            ; respect to vector $n$ and sum up for each pixel
         ...; compute the gradient of error function $\Lambda$
            ; respect to vector $t$ and sum up for each pixel
      end;
   end;
   ...; update $t$ with equation 3.5
   ...; update $n$ with equation 3.4
end;
...; compute time-to-collision with calculated $t$ and $n$
```

3.2 Grid search Method

Another method we use to solve the nonlinear equation is a grid search method. In this method, we make use of the scale-factor ambiguity to change the nonlinear equation to a number of linear equations. It provides a global minimum solution by explicitly searching over a fine grid for a solution with small error.

3.2.1 Mathematical Derivation

In motion vision problems, there exists a scale-factor ambiguity in recovering motion and the distance of the planar object from the motion field [17]. If we scale distance in the scene by some constant factor λ and the velocity by the same factor, the image sequences obtained from the camera remains the same. Because of this scale factor ambiguity, we cannot recover the magnitudes of the vectors n and t individually. However, since time-to-collision is the ratio of the distance and speed, (calculated as the reciprocal of the dot product of translation velocity t and the normal vector n of the image), it is not ambiguous. Equation 3.1 remains the same and the absolute value of time-to-collision does not change, if we multiply $1/n$ by a constant factor λ and t with the same factor λ. We also should point out that time-to-collision is not defined when either n or t is zero. Because of all the reasons above, we can treat t as a unit vector to simplify the problem. Vector t can be any points on a unit sphere, which can be denoted in a polar coordinate system as below:

$$t_x = \sin\theta\cos\phi \tag{3.6}$$

$$t_y = \sin\theta\sin\phi \tag{3.7}$$

$$t_z = \cos\theta \tag{3.8}$$

where $\theta \in [0, \pi]$ and $\phi \in [0, 2\pi]$.

When vector t is known, the error function becomes a quadratic function of vector n. More importantly, it provides a closed form solution for vector n. Because a quadratic function has only one minimum, it is very straightforward to minimize the error Λ with respect to vector n. Vector t can be any point on a unit sphere, so there are an infinite number of possible vectors t. A simple but effective technique to approximate all the points on the unit sphere is to construct an artificial grid so that the divided small areas can be approximated by the closest grid intersection point. When this artificial grid is fine enough, it is good enough to approximate the whole sphere with grid points. Also because we create the artificial grid, we limit the computation on a finite number of vectors t. This artificial grid is shown in figure 3-2:

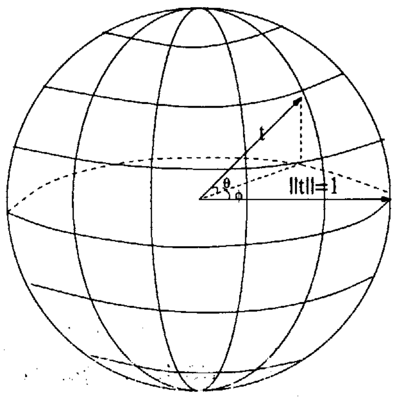


Figure 3-2: Grid search method

The artificial grid is created by dividing θ into N_1 parts, and dividing ϕ into N_2 parts. This unit sphere now is represented by $N_1 \times N_2$ grid intersection points on the sphere. If N_1 and N_2 are large enough, actually we can get a very fine grid on the sphere to represent the whole sphere. Rewrite t_x, t_y, and t_z as:

$$t_x = \sin\frac{\pi k_1}{N_1}\cos\frac{2\pi k_2}{N_2} \tag{3.9}$$

$$t_y = \sin\frac{\pi k_1}{N_1}\sin\frac{2\pi k_2}{N_2} \tag{3.10}$$

$$t_z = \cos\frac{\pi k_1}{N_1} \tag{3.11}$$

where $k_1 = (0, 1, 2, ..., N_1)$ and $k_2 = (0, 1, 2, ..., N_2)$. Every grid intersection point represents a value for vector t. For each value of t, we minimize the error function with respect to the unknown vector n. The error function is a quadratic function which is concave up with respect to n, so the minimum error occurs where the partial derivative of the error function Λ with respect to n is zero:

$$\frac{d\Lambda}{dn} = \sum_{x,y}\Big(E_t + (r \cdot n)(s \cdot t)\Big)(s \cdot t)r = 0 \tag{3.12}$$

The above equation is a linear equation. We can solve that equation for n shown as below:

$$n = -\Big(\sum_{x,y}(s \cdot t)^2(rr^T)\Big)^{-1}\sum_{x,y}E_t(s \cdot t)r = -M^{-1}(t)L(t) \tag{3.13}$$

where

$$M(t) = \begin{bmatrix} \sum_{x,y}(s \cdot t)^2x^2 & \sum_{x,y}(s \cdot t)^2xy & \sum_{x,y}(s \cdot t)^2xf \\ \sum_{x,y}(s \cdot t)^2xy & \sum_{x,y}(s \cdot t)^2y^2 & \sum_{x,y}(s \cdot t)^2yf \\ \sum_{x,y}(s \cdot t)^2xf & \sum_{x,y}(s \cdot t)^2yf & \sum_{x,y}(s \cdot t)^2f^2 \end{bmatrix}$$

and

$$L(t) = \begin{bmatrix} \sum_{x,y}E_t(s \cdot t)x \\ \sum_{x,y}E_t(s \cdot t)y \\ \sum_{x,y}E_t(s \cdot t)f \end{bmatrix}$$

There are $N_1 \times N_2$ grid intersection points on the unit sphere, which correspond to $N_1 \times N_2$ different values of t. At each value of t, the minimum error and its corresponding n are computed. The computation above is repeated for $N_1 \times N_2$ different values of t. Afterwards, the minimized error obtained for each different vector t is compared to obtain the minimal one among all possible values of t. The time-to-collision can be calculated from equation 2.8, with the corresponding n and t. This provides a global-minimum solution for the error function by explicitly searching over a fine grid. By creating an artificial grid, our search for minimum error will only be limited to a finite number of grid intersection points, and obtain a solution with small error.

3.2.2 Characteristic Features

The main advantage of this method is that the grid search approach provides a global minimum solution. In this method, we can always guarantee that the solution obtained from this algorithm corresponds the global minimum of the error function, provided the artificial grid is fine enough.

Another advantage of the grid search is to convert the nonlinear equation 3.1 into a set of closed form linear equations. The computation speed is significantly better than the gradient descent method. Solving those linear equations is much faster than solving the previous nonlinear equation with iterative computation. The linear operation is shown in the algorithm structure below:

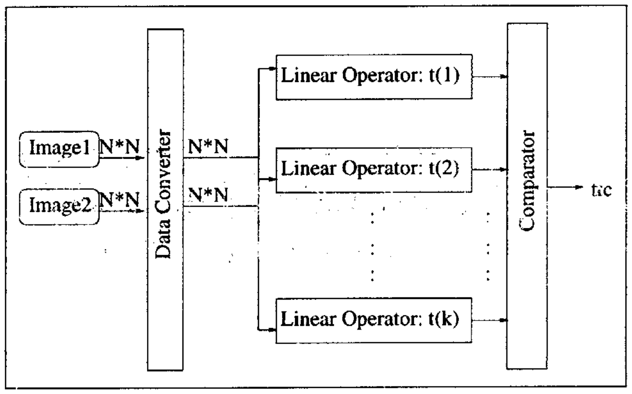


Figure 3-3: Block diagram of grid search method

it is too coarse, we might miss the global minimum; if it is too fine, there will be more computation than necessary. We strike the balance between these two extremes by considering the results of experiments on different grids, which will be shown in Chapter 5.

3.2.3 Algorithm Description

Below is a straightforward implementation of the grid search method.

```
...; load image 1 and image 2
...; smooth image 1 and image 2
...; compute derivatives $(E_x, E_y, E_t)$ from image 1 and image 2
```

```
...; initialize vector $t$ and $n$

for (i=0; i<gridpoints; i++)
  ...;assign $t$ to each grid intersection point on the unit sphere
  for (j=0; j<rows; j++)
    for (k=0; k<cols; k++)
      ...; compute matrix M (sum up over all the pixels) in equation 3.13
      ...; compute Matrix L (sum up over all the pixels) in equation 3.13
    end;
  end;
  ...; from matrix $M$ and $N$ calculate $n$ with equation 3.13
  ...; initialize error to zero;
  for (j=0; j<rows; j++)
    for (k=0; k<cols; k++)
      ...; compute error $\Lambda$ with the assigned $t$ and calculated $n$
    end;
  end;
  ...; compare error to obtain minimal error
  ...; update minimal error $\Lambda$, $n$, $t$
end;
...; using equation 3.2 to compute time-to-collision with $t$ and $n$
  ;that corresponding to the global minimal error
```

3.3 Improved Grid Search Method

There are some changes we made in the grid search method to improve the performance. These changes make the computation more efficient, while still keeping the final solution as accurate as the original version. The most significant changes we make are to use a hierarchical search and sufficient statistic.

3.3.1 Hierarchical Search

We can reduce the computational load in the grid search method by using a more efficient search procedure, rather than a brute force approach. In the grid search method, the error expression is evaluated at the grid intersection points on the t unit sphere. Then obviously, the finer the grid is, the more accurate the final result will be. It will cost quite a lot of computation, if we start with a very fine grid. Instead, a hierarchical search is applied to define the region for grid search. The grid search will initialized with a relatively coarse grid. After obtaining the minimum error, the local area where the minimum error occurs will be divided into finer grid. A finer search will be utilized in that local area. It is a coarse-to-fine search [14]. A relatively coarse search is made by making comparisons between the grid points on the whole sphere. Once we get the area where the minimum error occurs, a finer grid will be used on the local area to get better accuracy for our final result.

This operation is performed recursively until the error in the error function does not change significantly. In general, two or three steps of coarse-to-fine processes will be enough to obtain a good result.

3.3.2 Sufficient Statistics

In this project, our algorithms will have an image sequence as input and generate time-to-collision as the output. The information of time-to-collision that the input image sequence brings, can also be brought by sufficient statistics, which are a set of functions of the input image sequence. Once we extract those functions from the input data to form sufficient statistics, the rest of the information is irrelevant and can be ignored. The solution obtained from the sufficient statistics is exactly the same as the one obtained from all the input data, but with a lot less computation.

So far we have treated each pixel as an individual input datum. Notice that the derivatives of pixel intensity and pixel location appear in each iteration, when selecting a new value of vector t. For an image size of 128×128, the number of pixels is on the order of 10^4. If we choose 50 different values of t, we must process on the order of 50×10^4 pixels. That amount of computation is significant. Instead, from the idea of sufficient statistics, if we can loop through each image pixel only once to extract a small number of significant quantities and process only these quantities, we will be able to save a lot of computation time and memory.

To obtain sufficient statistics, the error function must first be expanded. To make it easy to explain, we will represent the error function (3.14) in matrix notation for detailed computation. Here is the error function again:

$$\Lambda(n,t) = \sum_{x,y} \Big(E_t + (r \cdot n)(t \cdot s)\Big)^2 \tag{3.14}$$

In the equation above, index x and y run along the width and height of input images. In the following derivation, instead of representing an image as a two-dimensional array, represent it as a one-dimensional array with an index i. The size of this one dimensional array is m, where $m = width \times height$. As we can see, E_t, r, and s are different for each pixel. With this new notation, we represent the temporal gradient E_t as a new vector G with index i:

$$G_i = [G_1, G_2, G_3, ..., G_m]^T$$

G_1 refers to E_t for 1th pixel, G_2 refers to E_t for 2th pixel, ... , G_m refers to E_t for mth pixel. For each pixel, r and s are 3-D vectors. In equation 3.14, r and s are $m \times 3$ matrices represented as $r_{i,l}$ and $s_{i,k}$. The translational velocity t and the normal vector n are 3-D vectors. In matrix notation, matrix-vector multiplication $z = Ax$ is usually represented as:

$$z_i = \sum_{j=1}^{n} A_{i,j} x_j$$

With the notation above, equation 3.14 expands:

$$\begin{aligned}
\Lambda &= \sum_{i=1}^{m}\Big(G_i + \sum_{k=1}^{3}(s_{i,k}t_k)\sum_{l=1}^{3}(r_{i,l}n_l)\Big)^2 \\
&= G_{total} + \sum_{k,l} C_{k,l}t_kn_l + \sum_{k,k'}\sum_{l,l'} D_{kk'll'}t_kt_{k'}n_ln_{l'} \\
&= G_{total} + \sum_{l}(\sum_{k} C_{k,l}t_k)n_l + \sum_{l,l'}(\sum_{k,k'} D_{k,k',l,l'}t_kt_{k'})n_ln_{l'} \\
&= G_{total} + \sum_{l} F_ln_l + \sum_{l,l'} H_{l,l'}n_ln_{l'}
\end{aligned} \tag{3.15}$$

where G_{total} is a scalar, C is a 3×3 matrix, D is a $3 \times 3 \times 3 \times 3$ tensor, F is a 3×1 matrix, and H is a 3×3 matrix:

$$G_{total} = \sum_{i=1}^{m}(G_i)^2$$

$$C_{k,l} = \sum_{i=1}^{m}(2G_is_{i,k}r_{i,l})$$

$$D_{k,k',l,l'} = \sum_{i=1}^{m} s_{i,k}s_{i,k'}r_{i,l}r_{i,l'}$$

$$F_l = \sum_{k=1}^{3} C_{k,l}t_k$$

$$H_{l,l'} = \sum_{k,k'} D_{k,k',l,l'}t_kt_{k'}$$

By representing our error function Λ as equation 3.15 above, vector n is obtained when the partial derivative of the error function Λ with respect to n is zero:

$$\frac{d\Lambda}{dn} = F + 2(H)n \tag{3.16}$$

From this, we easily get

$$n = -\frac{1}{2}H^{-1}F \tag{3.17}$$

As shown in equation 3.15, matrix C and D are not dependent on the unknown vectors t and n. They are sufficient statistics, which can be computed directly from input data. Once C and D are computed, the rest of the input data can be ignored. From characteristics of matrix multiplication, $D(l, l', :, :) = D(l', l, :, :)$ and $D(:, :, k, k') = D(:, :, k', k)$. Because of this, there are 36 independent variables for tensor D, 9 independent variables for matrix C. Compared to an order of 10^4 of image data, using 45 independent variables saves quite a significant amount of computational time for further computation. Thus, sufficient statistics make the grid search method possible for real time performance with digital signal processor design. The overall structure of this algorithm is shown in figure 3-4.

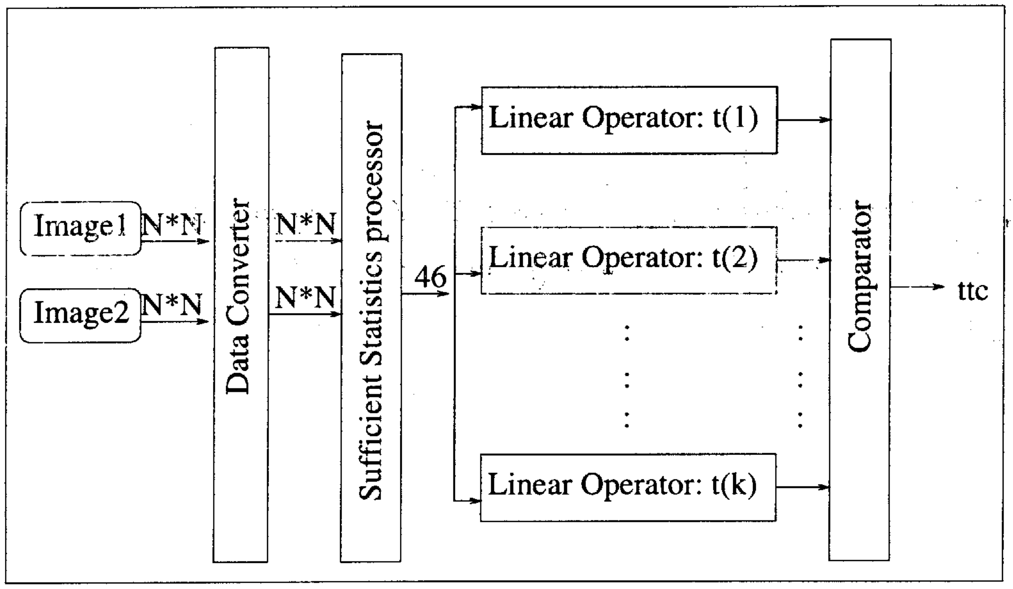


Figure 3-4: Block diagram of improved grid search method

Algorithm Description

Below is the algorithm description of grid search with sufficient statistics:

```
...; load image 1 and image 2
...; smooth image 1 and image 2
...; compute derivatives ($E_x, E_y, E_t$) from image 1 and image 2
for (i=0; i<rows; i++)
    for (j=0; j<cols; j++)
```

```
            ...; compute Matrix C and D (sum up over all the pixels)
        end;
    end;

    for (i=0; i<grid; i++)
        ...; assign vector $t$ to the grid intersection points on the grid
        ...; compute matrix $F$ and $H$ from $C$, $D$ and $t$
        ...; compute vector $n$ from matrix $F$ and $H$
        ...; compute error $\Lambda$ from matrix $F$, $H$ and computed $n$
            end;
        end;
        ...; calculate $n$ with equation 3.13
        ...; compare errors to obtain minimal error
        ...; update minimal error $\Lambda$, $n$, $t$
    end;
    ...; using equation 3.2 to compute ttc with calculated $t$ and $n$
```

Chapter 4

Hardware Implementation

4.1 Possible Solutions

When designing a vision system, we will have to face different design challenges, such as computational speed, reliability of the processing, size of the system, power dissipation, and flexibility. When making design decisions, we have to determine the priorities of these design challenges, and make tradeoffs among them. In our application, computational speed is our major concern due to the requirement for real-time performance.

In many applications, calculations corresponding to the early vision tasks are often the main computational bottleneck [12]. Towards speeding these up, hardware implementations are often used, and the main two common architectures used are application-specific analog chip design, and a programmable digital signal processor system design.[1] Both of these approaches have already found a number of success stories in the vision literation: For example, analog chip design has successfully been applied to motion detection [2] and to calculating focus of expansion [17], and a robust tracking system was also built with a digital signal processor system design [22]. In this Chapter, we will discuss these two architectural choices, as well as their associated

[1]Other alternatives that some authors have tried include application-specific digital chip design and system-level integration of camera and programmable analog processor. (The interested reader is referred to the excellent papers [23, 15]). But since these approaches often have prohibitively high complexity and do not really address our goals in this thesis, we will not consider them here.

algorithms, in considerable detail.

4.1.1 Analog VLSI Chip Design

Analog processing provides a possible solution for time-to-collision, because analog circuits can do certain computations that are time consuming when implemented in the conventional digital paradigm with much less power [17]. However, analog vision chip designs face limitations, such as low processing precision, and low resolution. In addition, designing single analog vision chips is time consuming and error-prone, because each single chip is fully custom designed.

To solve the nonlinear equation, the gradient descent method provides an iterative solution, which can be easily represented as a feedback scheme in an analog approach. A negative feedback loop can be used to correct an estimated value until it fits into the non-linear equation constraint. The iteration property makes the gradient descent method a very promising algorithm for analog VLSI chip design. On the other hand, the grid search method provides the solution by explicitly searching and comparing over a fine grid, which is a more digital approach, and less suitable for the nature of analog chip design. Therefore, below we only consider the gradient descent algorithm for an analog VLSI chip design.

Applications for vision usually have pixel based processing, which leads to a highly parallel overall architecture [4]. The gradient descent method is a highly parallel algorithm; all the pixels in the image have the same processing. This process will be performed by identical processing elements (PEs) in the chip. Each PE corresponds to a pixel in the image, and executes the same function. These PEs are interconnected in a regular fashion, and data streams can flow in single or multiple directions as shown in the figure 4-1:

In our application, there are certain operations required for each PE unit:

- **Multiplication**; this operation requires the use of a four-quadrant multiplier, where each quadrant corresponds to a particular combination of signs of input signals.

PE Array

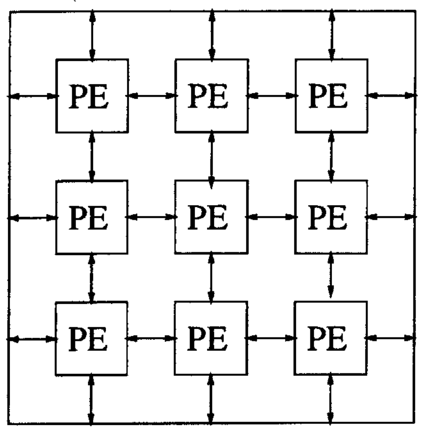


Figure 4-1: Array processor with PE

- **Aggregation**, where a very large number of inputs are brought together. This operation is done by an adder in current mode.
- **Scaling**, where a quantity of interest is multiplied by a scaling factor.
- **Normalization**, the purpose of which is to reduce the dynamic range of input signals to levels compatible with the needs of subsequent processing stages.
- **Differentiation**; this operation will be done with a hysteretic differentiator.

Circuits that implement these functions are described in Mead (1989, Analog and Neural System) [19]. Every pixel on the chip contains: an adaptive photocircuit which transduces light into a voltage signal that responds to contrast, independently of the background illumination; a hysteretic differentiator, which calculates the temporal gradient, whereas the spatial gradient is obtained simply by taking the difference between the output of the photoreceptors on either side; and a processing unit including adders, multipliers, and scaling and normalization units ??.

For the gradient descent algorithm, the processing unit will implement the equations below:

$$n^{i+1} = n^i - \alpha\frac{\delta\Lambda}{\delta n^i} = n^i - \alpha\sum_{x,y} 2\Big(E_t + (r\cdot n^i)(s\cdot t^i)\Big)(s\cdot t^i)r \tag{4.1}$$

$$t^{i+1} = t^i - \alpha\frac{\delta\Lambda}{\delta t^i} = t^i - \alpha\sum_{x,y} 2\Big(E_t + (r\cdot n^i)(s\cdot t^i)\Big)(r\cdot n^i)s \tag{4.2}$$

As with any other vision chip, the major drawback of this pixel level analog vision chip is low resolution. Each pixel includes a photocircuit and a processing unit that occupies a large pixel area. The area required for implementing the circuits and routing the information across the chip puts an upper bound on the number of pixels. Therefore, vision chips in general have a very low resolution. The average size of a vision chip reported has only 64×64 pixels, even for each process element implementing much simpler functions than the functions described above [20]. As rough estimate with $0.25\mu m$ process technology shows that we can have only around 10×10 pixels on a large chip ($9.40mm \times 9.70mm$). Experiments on test images with such a low resolution showed that the results obtained have a very low precision. So unless we find a better way to make the photocircuit or processing unit part smaller, we will not be able to obtain reliable results with pixel level analog vision chips for this application.

4.1.2 Digital Signal Processor (DSP) System Design

The other possible approach is a system level integration of an imaging camera and a digital signal processor. The challenge for digital signal processor system designs is still speed. Typically, motion vision algorithms are parallel algorithms. Usually they require thousands of operations per pixel for each input image [12]. The gradient descent algorithm and grid search algorithm are good examples of parallel algorithm. Due to use of sufficient statistics, our improved grid search method is a serial algorithm, which is more suited for digital signal processing system design. Meanwhile, the improved grid search algorithm successfully decreases the most amount of computation and makes such a task possible in real time with a fast digital signal processor unit. The detailed performance of gradient descent and the improved grid search in Chapter 5 will prove that the improved grid search method is well suited for a digital signal processor system with real-time performance.

Figure 4-2 shows the digital signal processor system architecture [22]. Included is a video camera (imager) for image acquisition, an A/D circuit block to convert the image format (data converter), and a digital signal processing unit for time-to-collision calculation.

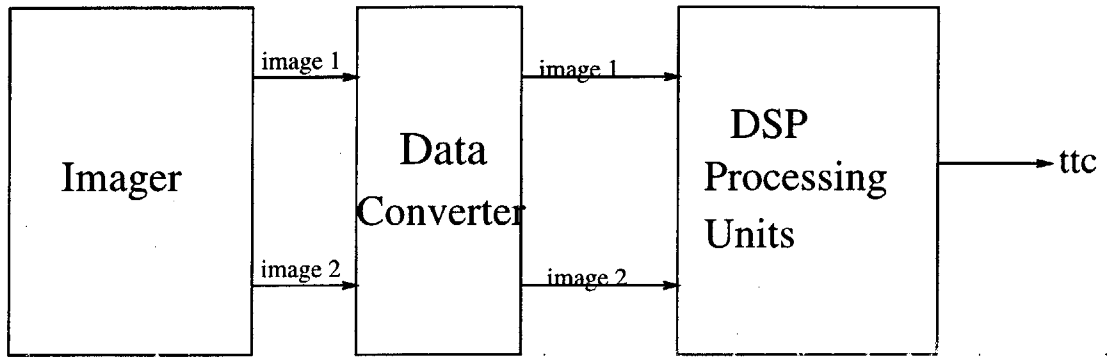


Figure 4-2: DSP system design

Digital signal processors are optimized for three types of multimedia applications: data compression, graphics, and image processing. A digital signal processor performs image processing tasks much more efficiently than a conventional microprocessor because of the following reasons [7]:

- Pipelined architecture
- Multiple operations of multiplier and ALU per cycle
- Single cycle multiplier and arithmetic logic unit (ALU)
- Zero overhead looping in hardware
- Fast on-chip memory
- Efficient program sequencing

Time-to-collision algorithms contain quite a lot of looped computation. Loop computations have potentially more parallelism than non-looped code because there are multiple iterations in the same code executing with limited dependencies between each iteration. Because of the pipelined architecture and zero overhead looping in DSP hardware, our algorithms are well suited for a digital signal processor system implementation.

4.2 Final Solution: DSP Design

As previously mentioned, the facts of low resolution and low precision make analog VLSI design unsuitable for the time-to-collision task. A digital signal processor system design is chosen for the real-time implementation. Popular high speed digital signal processors includes:

- the TMS320 series of chips from Texas Instruments
- the DSP56000, DSP56100, DSP56600 from Motorola
- DSP1600 and DSP3200 series from Lucent Technologies
- ADSP2100 and ADSP21000 series from Analog Devices

After comparisons among those different digital signal processors, we chose TMS320C67 as the core processing unit. The TMS320C67xx is the latest family of floating-point DSP processor from Texas Instruments. We decided to use floating-point DSP as our processing unit due to the large dynamic range of the data during the computation. This also makes software implementation easier, and giving better accuracy for the final results.

Because of time limitations, this DSP microprocessor based system will be a "proof-of-concept", and simulations were done on the DSP microprocessor TMS320C67 Code Generator and C Source Debugger/Simulator. The simulator lets us run the time-to-collision code in a simulation environment to test the software without using an actual hardware system. The results of simulation provide memory requirements and the maximum speed that a time-to-collision could provide on a TMS320C67 DSP microprocessor system.

4.2.1 TMS320C67x Architecture and Tool Overview

Before we discuss how to efficiently implement the real-valued time-to-collision algorithms on the 'C67x, it is helpful to take a brief look at the 'C67x architecture and code development tools.

The 'C67x is a floating-point DSP with the VelociTI architecture. It is an Advanced Very Long Instruction Word (VLIW) CPU architecture and designed to achieve high performance through increased instruction-level parallelism. This is done by avoiding the need for complex instruction scheduling and dispatch hardware in the processor. The device's core CPU consists of 32 general-purpose registers of 32-bit word length and eight function units, including two multipliers and six arithmetic units. These function units operate in parallel and can perform up to eight 32-bit instructions during a single cycle. It operates at 167-MHz clock rate [7].

Table 4.1: Characteristic Features for TMS320C67

Parameter Name	Value
Cycle Time (ns)	6
Data/Program Memory (bits)	512k/512k
DMA	4
Synchronous Memory Interface	(1) 32-bit
Nominal Voltage (V)	1.8/3.3
Host Port	(1) 16-bit
McBSP	2

The TMS320C67xx is supported by a set of software development tools, which includes an optimizing C compiler, an assembler, a linker, a C source Debugger and assorted utilities. Its orthogonal RISC-like CPU architecture makes the 'C67x CPU a good C compiler target.

4.2.2 Implementation and Optimization of TTC for DSP 'C67x

Our algorithms are implemented in C to provide an easy way to verify the functionality of these algorithms and obtain results of an optimized version.

Some optimizations were made on the software codes in accordance with the TMS320C67 architecture, for better efficiency. We focus our optimization on the the procedures which are most important in terms of MIPS requirements. One of the easiest methods used to optimize the C code is to evoke the C compiler's optimizer, using compiler options, such as -o3, -pm, -mt. Besides that, look-up tables, loop unrolling, and dynamical memory access strategies were used [9]. These optimizations successfully improved the code performance by 10% in total.

Look-up Table

A common strategy used in digital signal processing code implementation is to replace some complex arithmetic computation with a fast direct look-up table [11]. In the grid search method, the vector t is assigned to the grid intersection points as:

$$t_x = \sin\frac{\pi k_1}{N_1}\cos\frac{2\pi k_2}{N_2} \tag{4.3}$$

$$t_y = \sin\frac{\pi k_1}{N_1}\sin\frac{2\pi k_2}{N_2} \tag{4.4}$$

$$t_z = \cos\frac{\pi k_1}{N_1} \tag{4.5}$$

where $k_1 = (0, 1, 2, ..., N_1)$ and $k_2 = (0, 1, 2, ..., N_2)$. The quickest way to implement this is to create a table with all different values of t_x, t_y, and t_z. When we run the application on the digital signal processor, this table is directly loaded into the memory and read one by one. Obviously, it will be much faster than computing trigonometric functions from the equations above.

Loop Unrolling

The TMS320C67 has a pipelined architecture, which can dispatch up to eight parallel instructions every cycle. These parallel instructions proceed simultaneously through the same pipeline phases, which greatly improves the performance of our code. To schedule instructions in parallel, the compiler must determine the relationships, or dependencies, between instructions. If the compiler can determine that two instructions are independent of one another, it can schedule them in parallel. Otherwise, it assumes a dependency and schedules the two instructions sequentially. Loops inherently have more parallelism than non-looping code because there are multiple iterations of the same code executing with limited dependencies between each iteration. To maximize the number of instructions available to execute in parallel, in parts of the code, we use a loop unrolling strategy to expand small loops when the operations in a single iteration do not use all the resources of the DSP architecture [8]. Use of this strategy is limited by the resources of the DSP architecture, which contains eight functional units, including two multipliers and six arithmetic units. Through increasing the pipeline, the 'C6x code generation tools use the multiple resources of the VelociTi architecture efficiently and obtain very high performance.

Dynamical Memory Access for C Programs

There are two memory models for the DSP 'C67 to access data when programming in C [10]. In the small memory model, the external bus is used to access data from memory. Direct addressing limits the number of words that the small model can access to 32K bytes. However, it produces fast and compact code. The big model is not limited to 32K bytes, but it uses an indirect addressing mode. So it has a cost of two instructions per data access. Because our application requires a large image data reach and fast execution, dynamically allocated memory is used. The MALLOC function from the runtime support library is called at run time to reserve a block of memory in the .SYSMEM section. Code referring to the dynamically allocated array is one instruction per data access. Meanwhile it has a large word address reach. The price is a one-time call to MALLOC for each dynamically allocated array. In our application, dynamically allocated memory is efficient, because the overhead associated with the MALLOC call is insignificant when compared to the large number of data accesses.

Chapter 5

Testing and Experiments

5.1 Software Implementation Details

5.1.1 Input Image Smoothing

Input images for the DSP processing unit are discrete in time and space. Gradients of the image intensity are needed for time-to-collision algorithms, which requires that the image intensity be differentiable. Therefore, a smoothing process was applied to the input images to avoid aliasing, and improve the subsequence derivative estimates. Because there is a built-in lowpass filter in time — the video camera smears the input image sequence and decreases alias, — we are more interested in smoothing images in space (in x and y directions). The input images can be pre-smoothed with a lowpass filter — a Gaussian filter or a binomial filter[1]. In this project, a 2-D Gaussian lowpass filter [14] is used:

$$f(x) = \frac{1}{\sqrt{2\pi}\,\sigma} exp\Big(-\frac{(x-\mu_x)^2}{2\sigma^2} - \frac{(y-\mu_y)^2}{2\sigma^2}\Big) \tag{5.1}$$

where the mean in x direction and y direction (μ_x, μ_y) are set to zero, and the standard deviation (σ) of the Gaussian function is an user input. The user can change the value of σ depending on the need for the smoothness of the input images. The shape of the

[1]For the discrete images, a binomial lowpass filter usually works better than a Gaussian truncated lowpass filter

2-D Gaussian filters with different σ values are shown in figure 5-1.

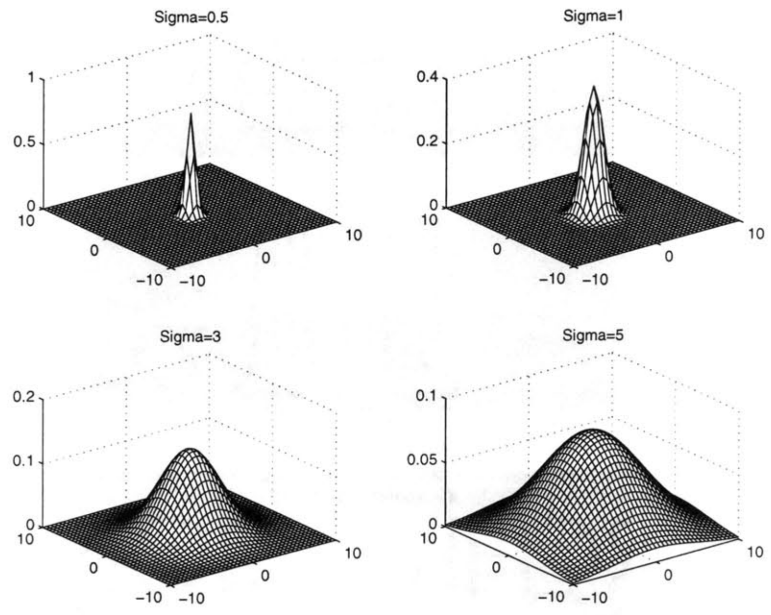


Figure 5-1: Shape of 2-D Gaussian filter with different sigma values

When using a Gaussian filter to smooth images, the larger the value for σ is, the larger the smoothing window will be. Figure 5-2 illustrates the performance of a Gaussian filter with different σ values. Clearly, the Gaussian filter blurs the image by reducing high frequency components while preserving low-frequency components.

Another place we need a low-pass filter to smooth images is when there is a large displacement between each input image frame. As mentioned before, the direct method only applies for small motions between image frames. To meet this constraint, we usually smooth the input image sequences with a Gaussian filter or a binomial lowpass filter first, then the images are sub-sampled into a low-resolution image with less motion. This process is usually repeated to obtain a hierarchical, multi-resolution image representation, which allows us to use the direct method.

5.1.2 Gradient Derivation

From the derivation in previous chapters, the time-to-collision algorithms involves computation of the partial spatial and temporal derivatives (E_x, E_y, E_t) of each pixel

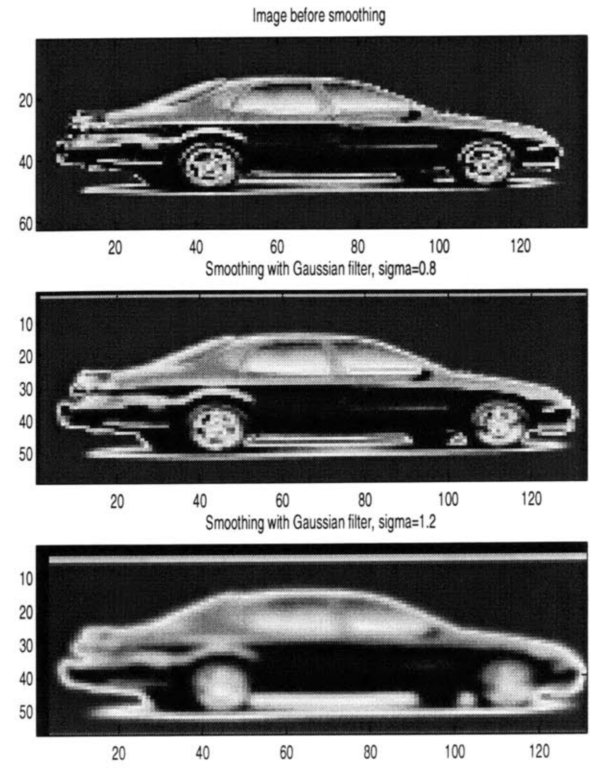


Figure 5-2: Smoothing with Gaussian filter

in the image. To get time derivatives E_t, an image-pair taken sequentially in time is required. In our approach, the partial derivatives of image intensity were computed with a first order derivative. The first order difference approximations are:

$$E_x|_{i,j,k} = \frac{E(j+1) - E(j)}{\Delta x} \tag{5.2}$$

$$E_y|_{i,j,k} = \frac{E(i+1) - E(i)}{\Delta y} \tag{5.3}$$

$$E_t|_{i,j,k} = \frac{E(k+1) - E(k)}{\Delta t} \tag{5.4}$$

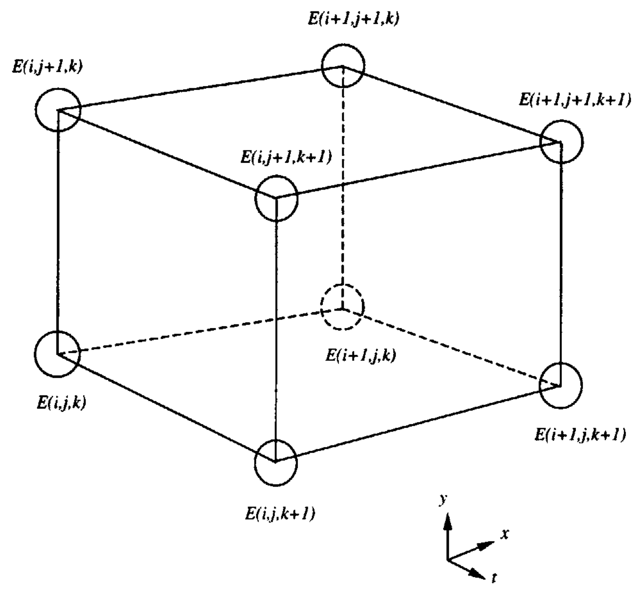


Figure 5-3: Pixel cube used in estimating the three partial derivatives of image brightness from an image pair

where

$$E(i+1) = \frac{1}{4}\Big(E(i+1,j,k), E(i+1,j+1,k), E(i+1,j,k+1), E(i+1,j+1,k+1)\Big)$$
$$E(i) = \frac{1}{4}\Big(E(i,j,k), E(i,j+1,k), E(i,j,k+1), E(i,j+1,k+1)\Big)$$
$$E(j+1) = \frac{1}{4}\Big(E(i,j+1,k), E(i+1,j+1,k), E(i,j+1,k+1), E(i+1,j+1,k+1)\Big)$$
$$E(j) = \frac{1}{4}\Big(E(i,j,k), E(i+1,j,k), E(i,j,k+1), E(i+1,j,k+1)\Big)$$
$$E(k+1) = \frac{1}{4}\Big(E(i,j,k+1), E(i+1,j,k+1), E(i,j+1,k+1), E(i+1,j+1,k+1)\Big)$$
$$E(k) = \frac{1}{4}\Big(E(i,j,k), E(i+1,j,k), E(i,j+1,k), E(i+1,j+1,k)\Big)$$

$E(i,j,k)$ here corresponds to the image intensity for pixel (i,j,k). Here i is in the x direction, j is in the y direction and k is in the time direction. The three partial derivatives of image brightness at the center of the cube are estimated from the average of the first four differences along the four parallel edges. This is illustrated in figure 5-3.

5.2 Synthetic Image Sequences for Planar Scenes

The reason we test the program on the synthetic image sequences is that it is easy to control the motion field, i.e. we know the "ground truth". Given a planar scene brightness function $E(x, y, 1)$, we would like to be able to render this function onto the camera image plane as the plane containing the scene moves through space. With this ability, a synthetic image pair $E(x, y, 1)$ and $E(x, y, 2)$ that corresponds to real world motion can be generated.

The following mathematical derivation is used to create image $E(x, y, 2)$ from image $E(x, y, 1)$ given the motion vector t and normal vector n. As shown in previous chapters, the relation of the translational motion of the camera and the motion of the image points is given as:

$$r_t = \hat{z} \times r \times \frac{t}{R \cdot \hat{z}} \tag{5.5}$$

The instantaneous translational velocity is presented as $t = (t_x, t_y, t_z)^T$. Optical flow, the apparent motion on the image plane, is represented as $r_t = (u, v)^T$. Equation 5.5 can be rewritten as:

$$u = (r \cdot n)(-t_x + \frac{xt_z}{f}) = (xn_x + yn_y + fn_z)(-t_x + \frac{xt_z}{f}) \tag{5.6}$$

$$v = (r \cdot n)(-t_y + \frac{yt_z}{f}) = (xn_x + yn_y + fn_z)(-t_y + \frac{yt_z}{f}) \tag{5.7}$$

From the *constant brightness equation* we know that:

$$E_t + E_x u + E_y v = 0 \tag{5.8}$$

Therefore, the second image frame can be created as:

$$E(x, y, 2) = \Delta T[E_x(x, y, 1)(-u) + E_y(x, y, 1)(-v)] + E(x, y, 1) \tag{5.9}$$

where $E(x, y, 1)$ is the image brightness for the first image in the image pair, while

$E(x, y, 2)$ is for the second image. In this way, the second image in the image pair is generated given motion vector t, normal vector n, and first image $E(x, y, 1)$ in the image pair.

Terrain image[2] pairs and car image pairs were created for testing the time-to-collision algorithm. Figure 5-4, 5-5, and 5-6 present terrain image pairs and car image pair and their motion fields, given the first image in the image pair $(E(x, y, 1))$, a motion vector t and a normal vector n. .

The reason we create terrain image pairs is that they provide smooth images, which were created from some smooth functions. The original functions project to the image plane, creating a continuous function on image plane. On the other hand, when car images are projected to the image plane, because of the fact that car images are discrete in space, we have to use sub-pixel interpolation estimation, which causing some distortion and aliasing. In addition, terrain image pairs have rich visual texture, which is generally required for motion field methods.

[2]The terrain images were created with several random points in the image plane. The image intensities are set to be $exp(-25 * dist)$, where $dist$ is the distance from the closest random point

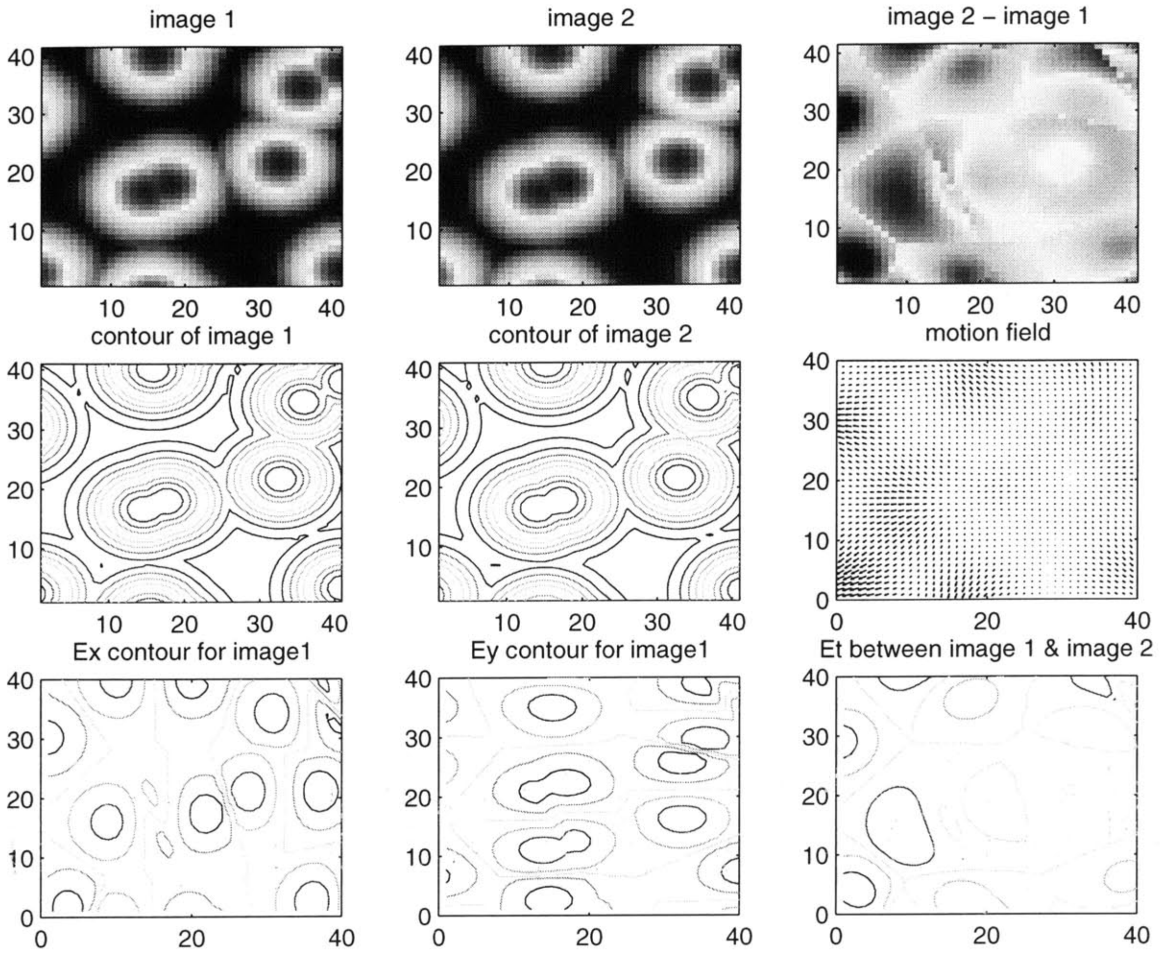


Figure 5-4: A terrain image pair created with $t = (0.0011, 0.4247, -0.9053)$ and $n = (0, -0.0009, -0.0379)$: high resolution

5.3 Performance Evaluation

After software implementation, we need to verify the performance of the algorithms. Items of interest include:

- Computational efficiency, including hardware resources and computational speed
- Verification of mathematical and numerical consistency and accuracy

5.3.1 Computational Efficiency

As we mentioned in chapter 3, because of the hierarchical search and sufficient statistics, the improved grid search will have faster computational speed compared to the

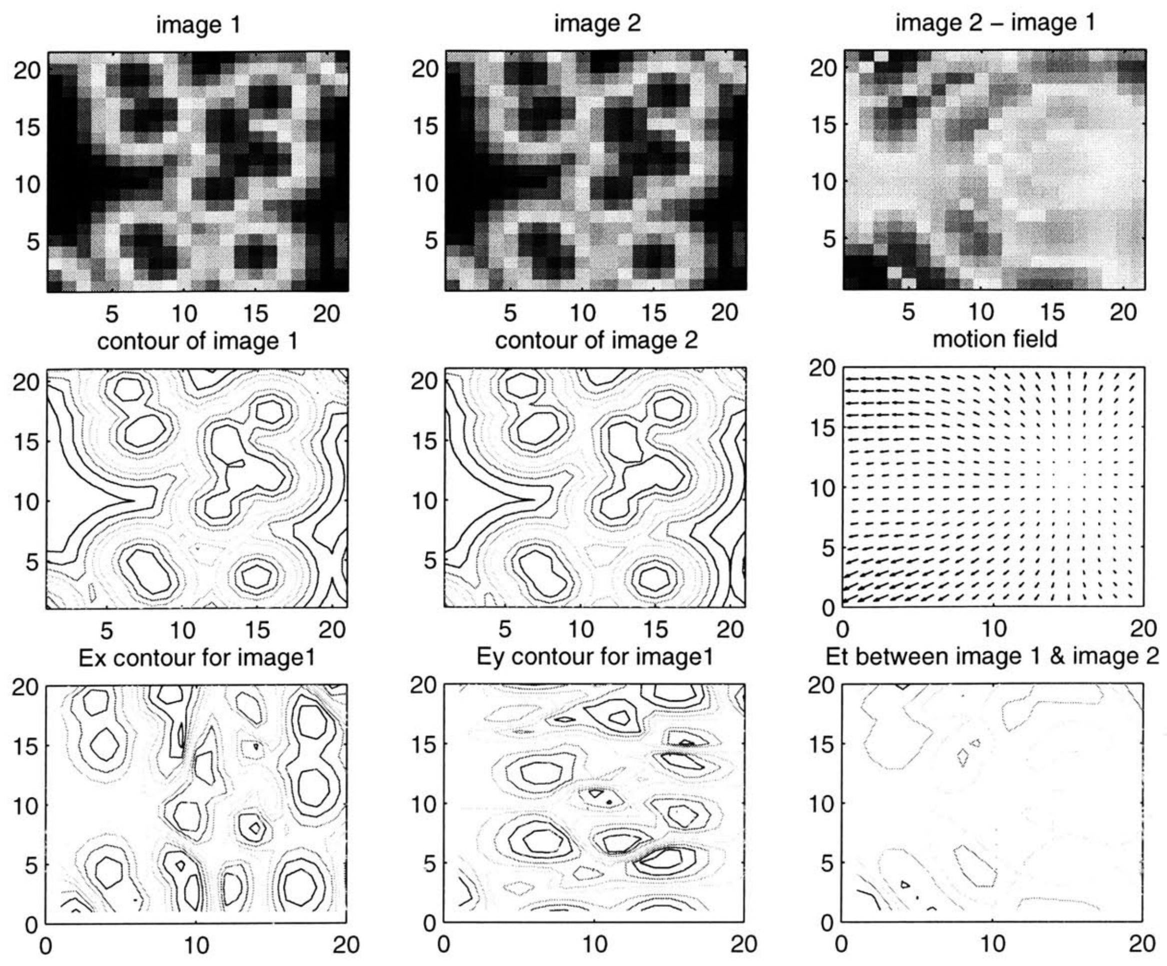


Figure 5-5: A terrain image pair created with $t = (0.0011, 0.4247, -0.9053)$ and $n = (0, -0.0009, -0.0579)$: low resolution

simple grid search. Here we compare the computational speed between the gradient descent and the improved grid search on TMS320C67 compiler and simulator. Experiments were performed on the image pairs of different image size with the gradient descent method; the number of iterations for convergence in gradient descent was set to 50. The gradient descent algorithm includes three functions – *imageAcquisition*, *getDeriv*, and *solveTtc*. The computational cycles for each function are shown in table 5.1. Experiments were repeated with the improved grid search method, when grid division N_1 and N_2 for angle θ and ϕ of the unit vector sphere were both set to 15. The improved grid search algorithm includes four functions – *imageAcquisition*, *getDeriv*, *obtainMatrix*, and *solveTtc*. The simulation results are shown in table 5.2.

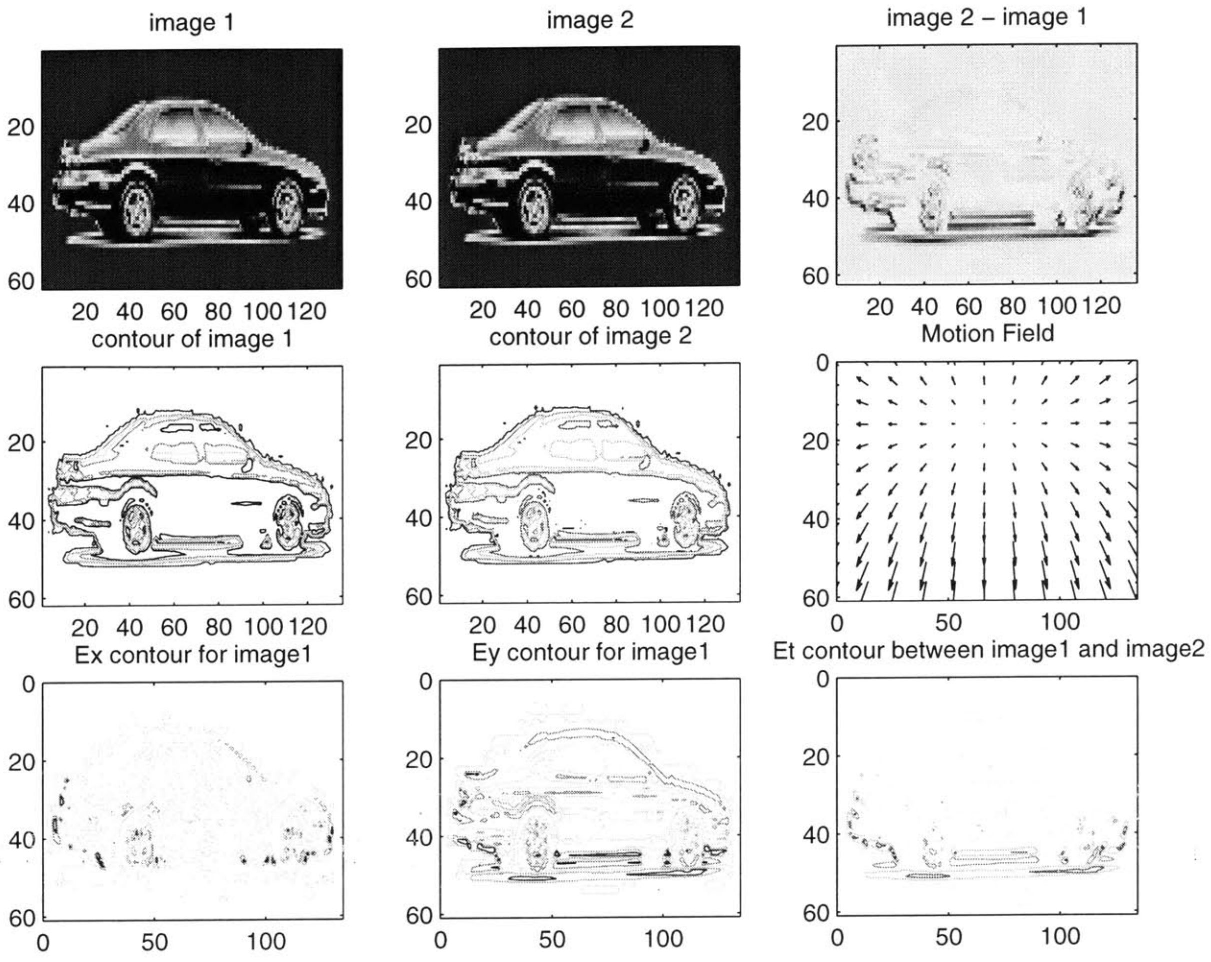


Figure 5-6: A car image pair created with $t = (0.0011, 0.4247, -0.9053)$; $n = (0, -0.0009, -0.0379)$

From table 5.1 and table 5.2, it is obvious computation cycles for gradient descent method is about 6 times bigger than the results from improved grid search method for three different image sizes. Computational time for functions *getDeriv* and *solveTtc* in gradient descent method are both proportional to the size of the image pair. Therefore, computation time for overall gradient descent algorithm is proportional to the size of image. In the improved grid search search method, computational time for functions *getDeriv* and *obtainMatrix* are proportional to the size of image, while the time for function *solveTtc* is independent of image size. The reason for this is that after obtaining sufficient statistics from the input image pare in function *obtainMatrix*, the rest of the program, including function *solveTtc*, will not depend on the input

Table 5.1: Performance of gradient descent method on TMS320C67

Function	Cycles (32 × 64)	Cycles (64 × 128)	Cycles (128 × 256)
imageAcquisition	24933	51237	103845
getDeriv	576662	2061288	7087263
solveTtc	15843973	64445573	251433785
Total	16403520	65538613	264173681

Table 5.2: Performance of improved grid search method on TMS320C67

Function	Cycles (32 × 64)	Cycles (64 × 128)	Cycles (128 × 256)
imageAcquisition	24933	51237	103845
getDeriv	576662	2061288	7087263
obtainMatrix	1945039	7949135	32147023
solveTtc	380841	380841	380841
Total	2928518	10417686	39642058

image pair at all. Compared to the computational time for function *obtainMatrix*, time for function *solveTtc* is insignificant. Therefore, we will say that the computational time for the overall improved grid search method is proportional to the input image size. Given cycle time at $6ns$ for the digital signal processor TMS320C67, the computational time for an image pair with the size of 32×64 is 17ms, when using the improved grid search method. Because the computational time for the overall improved grid search method is approximately proportional to the input image size, the computation of the time-to-collision can be completed in 34ms each frame for an image size of (64×64) with the TMS320C67 to achieve real-time performance.

For a larger image size, it is hard to obtain such high computational throughput with available DSP chips; multiple DSPs must be used. Generally, we will break down, or partition, the computation into smaller units, and distribute them among several processors. Theoretically, computational time will be reduced by a maximum factor of n (the number of parallel processors) [13]. In the parallel processor systems, our task – time-to-collision, can be partitioned so that each processor operates on different sub-blocks of image data.

Table 5.3: Effects of N_1 and N_2 on improved grid search method on TMS320C67

Function	$N_1 = N_2 = 5$	$N_1 = N_2 = 10$	$N_1 = N_2 = 15$	$N_1 = N_2 = 20$
imageAcquisition	24933	24933	24933	24933
getDeriv	576662	57662	57662	57662
obtainMatrix	1945039	1945039	1945039	1945039
solveTtc	34947	159890	380841	698469
Total	2582243	2707483	2928518	3246206

Effects of grid division N_1 and N_2 for angle θ and ϕ of the unit vector sphere on computational speed were investigated. From table 5.3, it is obvious that only the function *solveTtc* is related to N_1 and N_2. The computational cycle of function *solveTtc* is proportional to $N_1 \times N_2$ the number of grid points on the sphere. But as we mentioned above, compared to the computational time for the overall improved grid method, the computational cycle for function *solveTtc* is insignificant. By using a large value of N_1 and N_2, the DSP system obtains a more accurate result for the time-to-collision without losing much in computational speed.

Now that we have studied computational speed, let us examine hardware resources. As we know, our algorithms for time-to-collision need to use derivatives for every pixel in the image. For both gradient descent method and grid search method, the processing units will need a storage buffer to hold the derivatives for every pixel since all of them are needed in every iteration of the algorithm. On the other hand, because of the sufficient statistics, for the improved grid search method, the derivatives are not needed for every iteration of the algorithm. Storage buffers are only needed to hold derivatives for the current pixel, because they are used only once in obtaining sufficient statistics. To obtain spatial and temporal derivatives, $3 \times N$ memory units are needed for a row by row computation, when N is the column size in the input images.

A brief summary of hardware resources and computational bandwidth for the three different algorithms is as follows. Let M and N be the width and height in pixels of the image, respectively. Let T be the number of iterations for convergence in

Table 5.4: Performance summary of different algorithms

	Gradient Descent	Grid Search	Improved Grid Search
Cycles	$O(P \times T \times M \times N)$	$O(K \times M \times N)$	$O(M \times N) + O(K)$
Memory	$M \times N$	$M \times N$	$3 \times M + 45$

gradient descent method, and K be the number of grid intersection points representing different vectors t. Whether the gradient descent computation converges to a global minimum or local minimum depends on the initial values for vector n and t. Therefore, the process is usually repeated for several different initial values to ensure about convergence to the global minimal solution. Here P represents the number of initial points for the gradient descent method. As shown in table 5.4, the computational cycles for gradient descent are $O(P \times T \times M \times N)$, which can be explained as the running time is proportional to $P \times T \times M \times N$.

5.3.2 Accuracy

Now, let us determine how accurate the mathematical and numerical calculations for the gradient descent method and the improved grid search method are. Sequences of car images were created with a different translational velocity t and normal vector n. Experiments were performed on synthetic image pairs in figure 5-4. Effects of grid division N_1 and N_2 for angle θ and ϕ of the unit vector sphere on the experimental results of the time-to-collision were investigated. Figure 5-7 shows the experimental results using the grid search method. In figure 5-7, variable N is set as $N = N_1 = N_2$. From the result of this figure, we can see that when N takes value bigger than 15, the improved grid search method obtains a solution with very small error.

To compare the gradient descent method with the improved grid search method, experiments were performed on the same image pair in figure 5-4. Figure 5-8 shows the experimental results for the gradient descent method with two different sets of initial values of vector n and t. From figure 5-8, the solution starts to converge after about fifty iterations. However, the value where the iterative computation converges

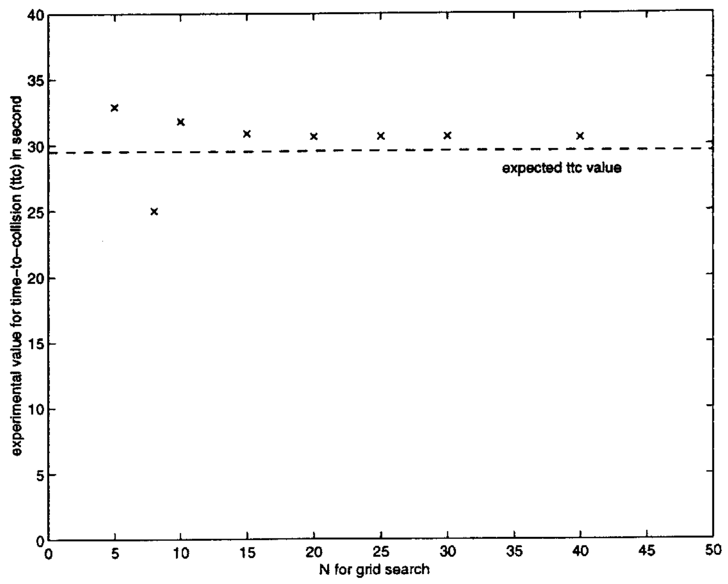
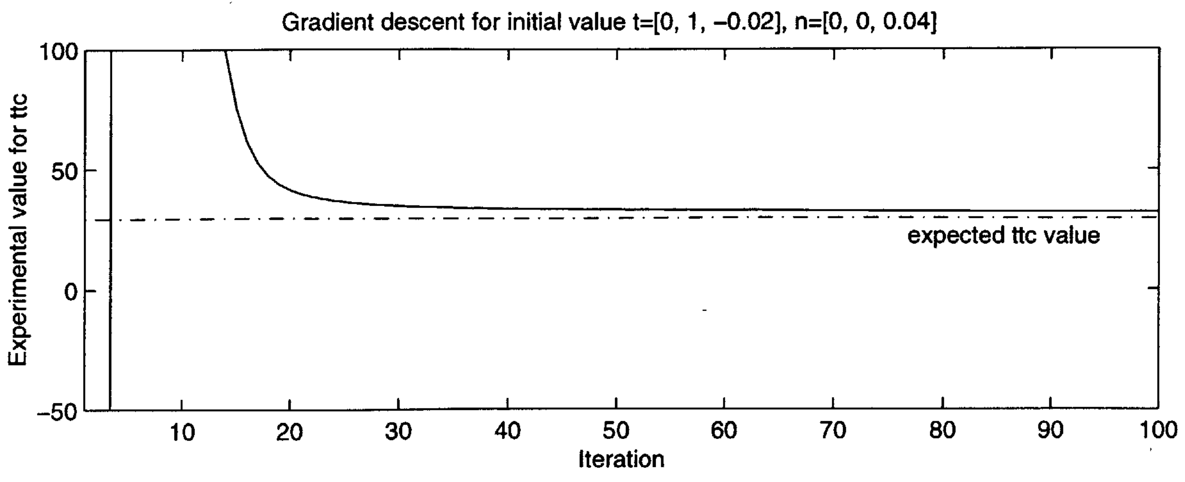


Figure 5-7: Solving TTC with improved grid search method

depends on the initial values of vector n and t. Therefore, when the error function have more than one local minima, the gradient descent method provides inconsistent results depending on the initial values.

To check if the error function have more than one local minima, error distributions were obtained from the improved grid search method. In figure 5-9, the value for error is shown in z axis, and the location on the intersection of the grid on the sphere is projected to the 2-D $x-y$ plane. Figure 5-10 shows corresponding ttc distribution on the different grid points. From figure 5-9, we can see that there are two local minima in the error function, and our actual solution is the global minimum marked as 'x' in figure 5-9 and figure 5-10. The improved grid method provides the global-minimum solution once N is greater than 15. Comparing the performance of the improved grid search method and the gradient descent method, it is obvious that the grid search method provides a more consistent and reliable result for the time-to-collision.

The same experiments were repeated for the terrain image pair with lower resolution in figure 5-5 and the car image pair in figure 5-6. Similar behaviors were observed from the simulation. The results show that terrain image pairs provide more accurate results than car image pairs, and apparently, high resolution image pairs have better performance than low resolution image pairs.



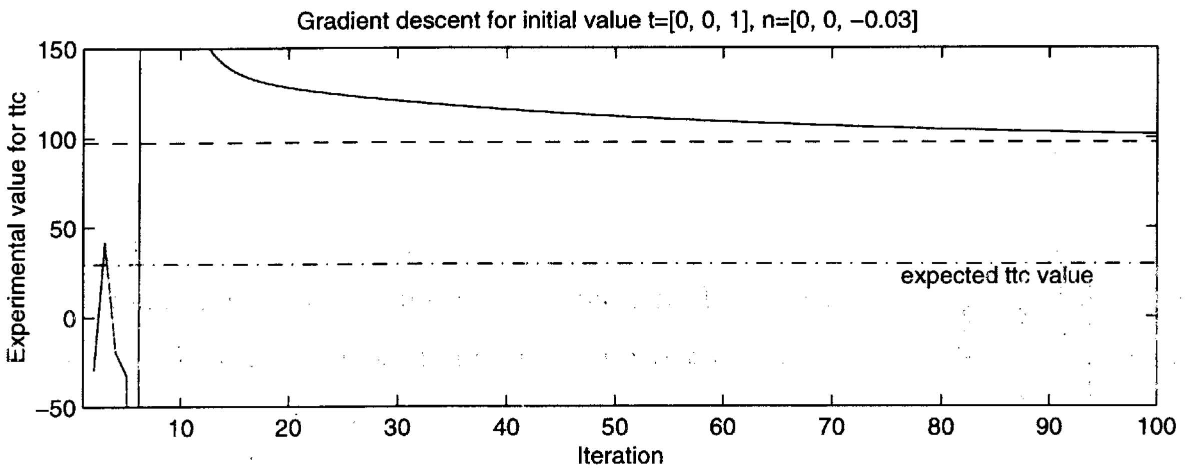


Figure 5-8: Solving ttc with gradient descent method

More experiments were performed to test the performance of the improved grid search algorithm for the image pairs with different translational motion. Each experiment involved setting the translational motion vector t, and the normal vector n appropriately. Keeping the normal vector n constant, while increasing the value of translational velocity t in the direction of normal vector of planar object, the experimental results of ttc are shown in figure 5-11, in comparison with the expected value of ttc. Our experiments exhibit the same behavior as we expected. When motion between two sequential image frames is very small, noise in the input image pairs will have more effect on our final solution, causing a less accurate result. But when the motion is not too small, our algorithm provides a very accurate result. The typical error in our algorithm is less than 4%.

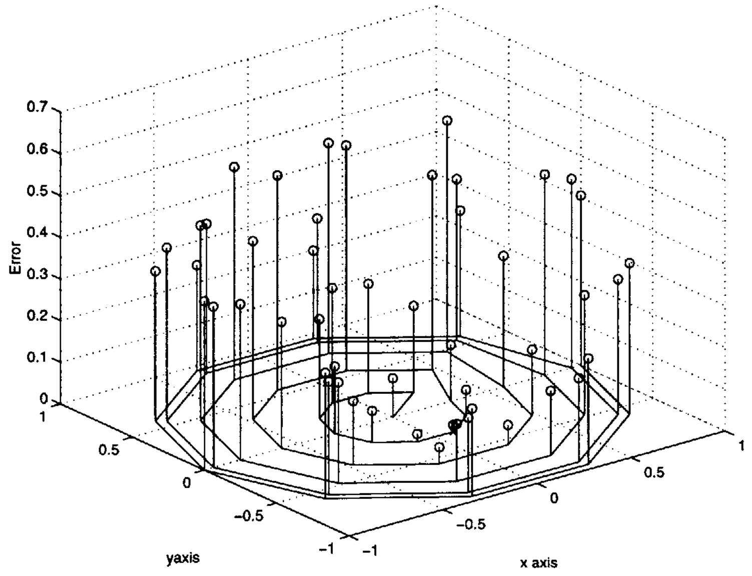


Figure 5-9: Error distribution on the sphere

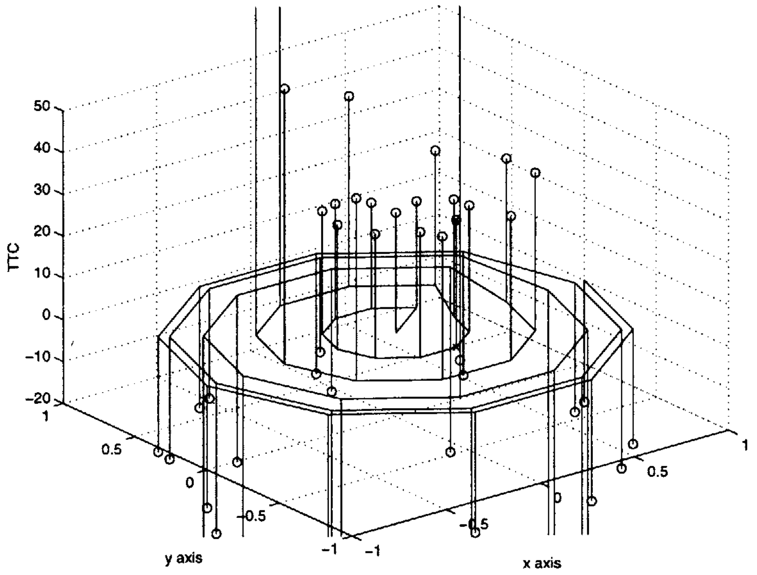


Figure 5-10: Time-to-collision distribution on the sphere

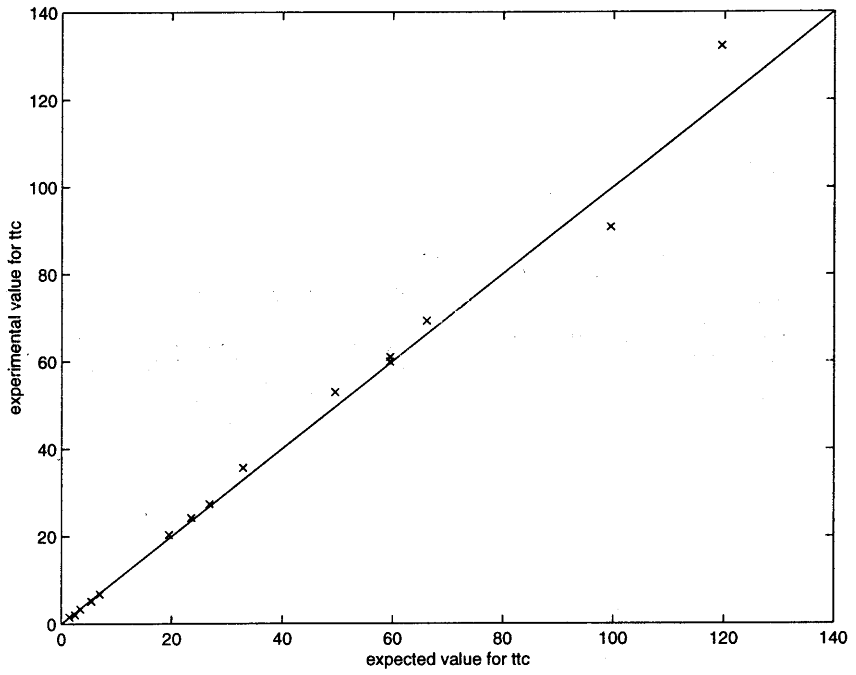


Figure 5-11: Estimate time-to-collision

Chapter 6

Conclusion

6.1 Summary

This thesis investigated the potential of using a digital signal processor to realize a real-time system for the estimation of time-to-collision for camera motion. The direct method builds the theoretical basis for time-to-collision, because it is more robust to noise, and provides a straightforward solution for time-to-collision. The direct method and least squares approach provide a non-linear closed form solution for determining translational camera motion with respect to a planar surface [6]. To solve these non-linear equations, we explored gradient descent, grid search, and improved grid search. The performance of these algorithms were compared in terms of reliability, amount of arithmetic operations required, and memory use.

The gradient descent approach provides an iterative solution. The iteration property of this approach makes it well suited for analog implementation, because the iterative solution can be easily obtained by closing a feedback loop in analog circuit design. A major limitation of gradient descent is the need to choose a suitable value for the learning rate parameter and the issue of convergence to local minima. The gradient descent algorithm has pixel based processing, which leads to a highly parallel overall architecture, if implemented with analog VLSI circuitry.

The grid search approach provides a global-minimum solution by explicitly searching, over a fine grid, for a solution with small error. The improved grid search method uses hierarchical search and sufficient statistics strategies, basing on the original grid search method. It successfully reduces a large amount of arithmetic operations. With a Digital Signal Processing (DSP) processor having properties such as fast arithmetic processing capability, fast on-chip memory, zero overhead looping in hardware, and extended precision and dynamic range in the computation units, the improved grid search approach seems well matched to a real time implementation with a digital signal processor.

Due to computational complexity for the gradient descent method, the pixel level analog VLSI chip has large pixel size, causing very low resolution. Therefore it is hard to get reliable results for analog VLSI design for this application. A digital signal processor TMS320C67 was chosen for our current real time implementation. Because of time limitations, this DSP microprocessor based system will be a 'proof-of-concept,' and simulations were on the DSP microprocessor TMS320C67 Code Generator and C Source Debugger/Simulator. From the results of simulation, we conclude that the improved grid search is well suited for DSP system design. It has provided a reliable solution and high computational efficiency when implemented with the Texas Instruments digital signal processor TMS320C67.

6.2 Recommendation

There are a number of improvements which can be made for future real-time implementation of time-to-collision.

In this project, there is a trade-off between speed and precision. When implemented time-to-collision algorithms with a digital signal processor (DSP) system design, a floating point DSP processor was used due to the large dynamic range of the data in the implementation of the time-to-collision problem. If we are willing to accept lower accuracy for this problem, fixed point DSP combined with FPGA (Field Programmable Gate Array) can be used to achieve even higher speeds.

The time-to-collision algorithm using the improved grid search method was simulated on a DSP microprocessor based system to provide real time performance. In the future, we hope to combine silicon VLSI technology and analog circuits for the implementation of time-to-collision. Although analog circuits do indeed suffer from low precision, this shortcoming is compensated for by the efficiency of computations and less power consumption [12]. The gradient descent approach has a strong potential for robustness and a real time analog VLSI chip design. By investigating how to decrease arithmetic operations on each pixel and the analog circuit size of each pixel, the shortcoming of low resolution can be overcome, making a vision chip for time-to-collision task possible.

Bibliography

[1] J. R. Bergendahl. A computationally efficient stereo vision algorithm for adaptive cruise control. Master's thesis, MIT, June 1997.

[2] Chu Phoon Chong. Image-motion detection using analog vlsi. *IEEE Journal of Solid-State Circuits*, 27(1):93–96, 1992.

[3] S. E. Frumkin. Time to collision algorithm: Properties and real time implementation. Master's thesis, MIT, May 1997.

[4] Jeffref Carl Gealow. *An integrated Computing Structure for pixel-parallel Image Processing.* PhD thesis, MIT, June 1997.

[5] Ellen C. Hildreth. *Recovering Heading for Visually-Guided Navigation.* 1984.

[6] Berthold K. P. Horn. *Robot Vision.* The MIT Press, 1986.

[7] Texas Instruments. *TMS320C6000 CPU and Instruction Set Reference Guide.* 1996.

[8] Texas Instruments. *TMS320C6000 Optimizing C Compiler User's Guide.* 1996.

[9] Texas Instruments. *TMS320C6000 Programmer's Guide.* 1996.

[10] Texas Instruments. *TMS320C6x C Source Debugger User's Guide.* 1996.

[11] Texas Instruments. Table loop-up and interpolation on the tms320c2xx. Technical Report BPRA046, January 1997.

[12] Christof Koch and Hua Li. *Vision Chips: Implementing Vision Algorithms with Analog VLSI Circuit.* IEEE COmputer Society Press, 1997.

[13] Brewster Lamacchia. Improve real-time product development using parallel dsp. *DSP and Multimedia Technology*, March 1998.

[14] Jae S. Lim. *Two-dimensional Signal and Image Processing.* Prentice-Hall, 1990.

[15] David Andrew Martin. *ADAP: A Mixed-Signal Array Processor with Early Vision Applications.* PhD thesis, MIT, June 1996.

[16] Ichiro Masaki. Machine-vision systems for intelligent transportation systems. *IEEE Intelligent Systems and their applications*, 13(6):24–29, Nov 1998.

[17] Ignacio Mcquirk. An analog VLSI chip for estimating the focus of expansion. Technical Report AIM-1577, MIT Artificial Intelligence Laboratory, June 1996.

[18] Ignacio Mcquirk, Berthold K. P. Horn, Hae-Seung Lee, and John L. Wyatt Jr. Estimating the focus of expansion in analog VLSI. *International Journal of Computer Vision*, 1997.

[19] Carver Mead. *Analog and Neural Systems.* 1989.

[20] Alireza Moini. *Vision Chips or Seeing Silicon.* WWW page at URL: http://www.eleceng.adelaide.edu.au/Groups/GAAS/Bugeye/visionchips/, March 1997.

[21] Shahriar Negahdaripour and Berthold K. P. Horn. Direct passive navigation. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 9(1), January 1987.

[22] Ryuzo Okada, Shinya Yamamoto, and Yasushi Mae. Developing a real-time person tracking system using the tms320c40 dsp. Technical Report SPRA189, January 1996.

[23] D. Mark Royals. On the design and implementation of a lossless data compression and decompression chip. *IEEE Journal of Solid-State Circuits*, 28(9):948–953, 1993.

[24] Gideon P. Stein. *Geometric and Photometric Constraints: Motion and Structure from Three Views.* PhD thesis, MIT, June 1998.