A High-Speed Iterative Closest Point Tracker on an FPGA Platform

by

Michael Sweeney Belshaw

A thesis submitted to the
Department of Electrical & Computer Engineering
in conformity with the requirements for
the degree of Master of Science (Engineering)

Queen's University
Kingston, Ontario, Canada
July 2008

Abstract

The Iterative Closest Point (ICP) algorithm is one of the most commonly used range image processing methods. However, slow operational speeds and high input bandwidths limit the use of ICP in high-speed real-time applications.

This thesis presents and examines a novel hardware implementation of a high-speed ICP object tracking system that uses stereo vision disparities as input. Although software ICP trackers already exist, this innovative hardware tracker utilizes the efficiencies of custom hardware processing, thus enabling faster high-speed real-time tracking. A custom hardware design has been implemented in an FPGA to handle the inherent bottlenecks that result from the large input and processing bandwidths of the range data. The hardware ICP design consists of four stages: **Pre-filter**, **Transform**, **Nearest Neighbor**, and **Transform Recovery**.

This custom hardware has been implemented and tested on various objects, using both software simulation and hardware tests. Results indicate that the tracker is able to successfully track free-form objects at over 200 frames-per-second along arbitrary paths. Tracking errors are low, in spite of substantial noisy stereo input. The tracker is able to track stationary paths within 0.42 mm and 1.42 degrees, linear paths within 1.57 mm and 2.80 degrees, and rotational paths within 0.39 degrees axis error. With further degraded data by occlusion, the tracker is able to handle 60% occlusion before

a slow decline in performance. The high-speed hardware implementation (that uses 16 parallel nearest neighbor circuits), is more then **five times** faster than the software K-D tree implementation.

This tracker has been designed as the hardware component of 'FastTrack', a high frame rate, stereo vision tracking system, that will provide a known object's pose in real-time at 200 frames per second. This hardware ICP tracker is compact, lightweight, has low power requirements, and is integratable with the stereo sensor and stereo extraction components of the 'FastTrack' system on a single FPGA platform.

High-speed object tracking is useful for many innovative applications, including advanced spaced-based robotics. Because of this project's success, the 'FastTrack' system will be able to aid in performing in-orbit, automated, remote satellite recovery for maintenance.

Acknowledgments

I would like to express my gratitude to my supervisor, Dr. Michael A. Greenspan (P.Eng.), for his continual support during my graduate studies at Queen's University. An expert in Computer Vision, he has guided me with his extensive knowledge, endless patience and calming reassurance, enabling me to explore this exciting field.

I would also like to thank MDA Space Missions (*Dr. Piotr Jasiobedzki*), Ontario Centres of Excellence CITO, the University of Toronto Computer Vision Laboratory (*Dr. James MacLean*), and the University of Ryerson Embedded and Re-configurable Systems Laboratory (*Dr. Lev Kirischian*) for their support throughout my work. Many thanks to my colleagues in the Robotics and Computer Vision laboratory at Queen's University for welcoming me into this innovative research group.

My deepest thanks to my family and friends for their constant help, support and understanding - especially for tolerating my endless interpretations of computer vision.

Thanks sincerely to all of you :)

Contents

List of Tables

List of Figures

Glossary

ϵ_d the number of non-hidden data read cycles required, page 60

ϵ_m the number of non-hidden model read cycles required, page 62

ϵ_p the pipeline priming delay of the circuit, page 58

$\hat{\vec{p}}$ a data point translated by a centroid, page 73

$\hat{\vec{q}}$ a model point translated by a centroid, page 74

$\vec{c}$ a centroid, page 73

$\vec{p}$ a data point, page 73

$\vec{q}$ a model point, page 74

D the distance between two points, page 70

D_i the i^{th} data point in scene data $\mathbf{D}$, page 21

H the number of high-resolution points in a cluster, page 69

L the number of low-resolution model points, page 69

M_i the i^{th} data point in model data $\mathbf{M}$, page 21

N the number of points compared, page 26

N_{cyc} the number of cycles for operation, page 58

P a probability, page 49

P_i point pair that will be composed of D_i and its nearest neighbor M_j points, page 56

$\mathbf{D}$ a scene data set, page 21

M a model data set, page 21

R a rotation matrix with dimensions 3 by 3, page 21

T a transform matrix with dimensions 3 by 1, page 21

ASIC Application-Specific Integrated Circuit, page 8

baseline the distance between cameras in a stereo sensor, page 44

BRAM block random access memory, page 40

CLB configurable logic block, page 6

FIFO First-in First-out, page 75

FPGA Field-Programmable Gate Array, page 2

FSL Fast Simplex Link, page 75

GCLK global clock, page 103

HDL hardware description language, page 40

ICP Iterative Closest Point, page 2

IO input/output, page 40

LUT look-up table, page 38

PDF probability distribution function, page 19

RAM random access memory, page 40

RMS root mean square, page 83

ROI region of interest, page 47

SAD the sum of absolute difference, page 18

Self-Occlusion an object that partially covers itself when viewed from certain vantage points, page 80

Specularities direct reflections of light sources off objects, page 81

SRAM static random access memory, page 6

SRAM static random access memory, page 38

SSD the sum of squared difference, page 18

SVD singular value decomposition, page 21

VHDL very-high-speed integrated circuits hardware description language, page 40

Chapter 1

Introduction

1.1 Background

Recent developments in reconfigurable hardware have resulted in significant increases in logic densities. As a result, many realtime computer vision tasks that are too computationally or band-width intensive for general processors can now be performed effectively on reconfigurable hardware. This thesis examines a high-speed object tracking system that is designed to utilize stereo vision disparities for the purpose of remote satellite recovery and is implemented in these advanced reconfigurable devices.

The tracking system in this thesis is one component of a three-part team project that has been designed concurrently to utilize one reconfigurable device, field-programmable gate array (FPGA).

1. **Stereo Sensor** platform is capable of imaging and rectifying a stereo image pair with 640×480 pixel resolution at 200 frames-per-second (fps). This platform is being developed by Ryerson University.

2. **Stereo Extraction** platform is capable of analyzing a stereo pair to provide 640×480 pixels by 128 disparities at 200 fps using the maximum-likelihood stereo correspondence algorithm [1]. This platform is being developed by University of Toronto.

3. **Object Tracker** platform that is capable of processing the disparity images to track an object's pose at 200 fps, utilizing a technique known as the Iterative Closest Point (ICP) algorithm. This platform is the focus of this thesis.

These three components have been designed to utilize a single FPGA to address the large bandwidths associated with the communication between components. The focus of this thesis is the third component.

1.1.1 Computational Visual Tracking

Although visual tracking is a seemingly simple task for humans to perform, it is difficult to imitate properly with computers. As a result, many tracking algorithms have been conceived, each one having its own particular features [2]. Figure 1.1 provides a 2D face tracking example. Figure 1.2 provides a 3D object tracking example.

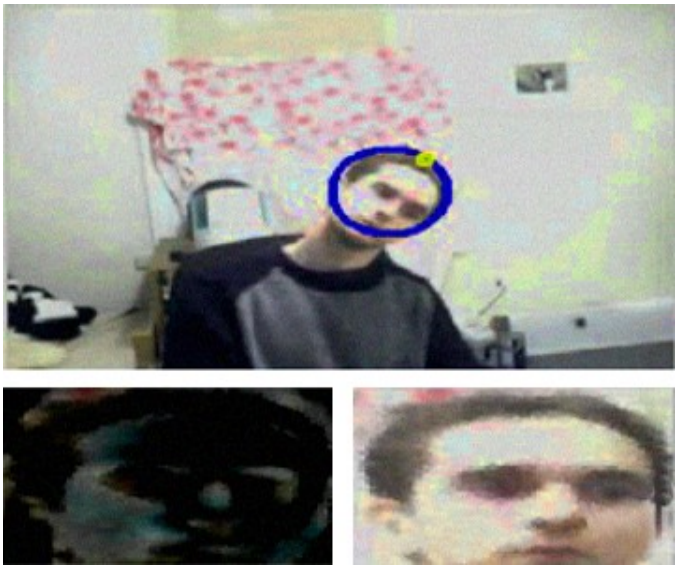


Figure 1.1: Face tracking using a 2D variant of the Bounded Hough Transform. The top image shows the scene with a face being tracked. The blue circle is the recovered pose of the face with yellow indicating angle of face. The bottom left image is the error between the best matched face model image alongside the reoriented region of interest of the current frame.

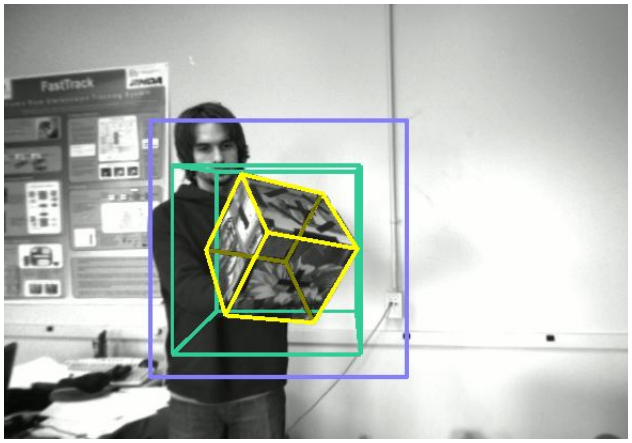


Figure 1.2: An example of a 3D object tracker. Shown here is a frame from the FastTrack hardware ICP tracker that is presented in this thesis. Overlays have been added to the input left stereo image so that the recovered pose can be observed. The region of interest overlays are the blue box and the green cube. The estimated pose overlay is the yellow cube.

In general, visual tracking is the process of continuously following a desired feature in an observed scene over a period of time. The observations are usually passive and usually only take into account the human visual spectrum of light. Although humans can easily recognize and track objects in complex scenes, tracking is a demanding task for computers. Even though computational visual tracking has been studied for over 30 years, difficulties in handling the dynamic nature of scenes still pose challenges. These challenges include: variable lighting, background clutter, occlusions and complex objects. Despite many successes in this field, research efforts are continuing in order to achieve more reliable computational tracking for these challenging situations.

Most humans rely on visual tracking for almost any activity. For example, visual tracking is extremely important while driving a car. Visual tracking allows for complex decision-making processes based on the tracking of numerous different road features such as signs, cars, pedestrians. Computational visual tracking can enable computers to perform similarly complex activities. This technology enables innovative developments, including autonomous robotic technology and human-computer interaction.

Stereo Vision Tracking

One particular class of computational trackers utilizes stereo vision sensors that mimic human stereoscopic vision. Because of the limited processing capabilities of current computers, stereo vision systems have a performance bottleneck. Either they lack the detailed visual and 3D scene information or they fail to track at the high frame rates required by advanced tasks. Concurrent research has circumvented the inherent bottleneck by implementing a high-speed stereo vision extraction algorithm using

FPGA hardware [1]. Eventually, this related stereo vision research will be joined to custom stereo sensors and this object tracker to form the complete 'Fast Track' stereo vision system. The goal of the Fast Track project is to design a system capable of sensing a scene in stereo, performing stereo extraction and tracking an object at over 200 fps in order to perform autonomous remote satellite recovery.

When stereo vision is utilized, a left-right image pair is required for each frame. This image pair is rectified to facilitate correspondence matching, and is then analyzed with stereo extraction techniques utilizing feature matching. This approach results in the formation of a disparity image that indicates the offset of features between image pairs. Finally, the disparity image can be converted into a 3D point cloud using triangulation with the stereo sensor's baseline and focal point.

There are many approaches to visual tracking. Although monocular vision is widely used, stereo vision also allows depth information to be considered when tracking [2]. This depth information, which describes the surface of shapes in the scene, can be used independently for the tracking of an object that has a known 3D surface model. One such method, which can operate on shape alone, is the ICP algorithm that represents an object as a collection of points that define its shape [3].

ICP Tracking

When tracking with ICP, an attempt is made to sample only scene points that fall on the object of interest. Because objects often have speed limitations, the temporal coherence between frames can be used to reduce the selection of scene points to the area around the previous pose estimate. These scene points are matched to the object model and used to determine the movements that have taken place since the

last frame. For every sampled scene point in this region, its nearest neighbor model point is determined. Because this approach requires an expensive search, the nearest neighbor step is the bottleneck of the ICP algorithm. This inherent problem has been addressed by many techniques [4]. For this thesis, data structures and parallel hardware processing are used to partition the nearest neighbor calculation to increase processing speeds.

1.1.2 Reprogrammable Hardware - FPGA

Description

Reprogrammable hardware, such as the FPGA, has been available for the past 30 years, and the amount of configurable logic that they contain has drastically increased over that time period. FPGAs enable a user to design and reconfigure the integrated circuit's overall function by loading hardware configurations into dedicated on-chip static random access memory (SRAM). Because of the ability to reconfigure these devices quickly, they are beneficial for hardware development. The hardware design can be quickly implemented, tested, and updated when modifications are necessary.

Current FPGA architecture consists of an array of configurable logic blocks (CLB) and routing channels. These CLBs are composed of units called 'slices' that contain look-up tables (LUTs) with which basic logic components are configured. For example, the simple logic of 'And','Or' and 'Not' gates are created by filling the look-up table with the corresponding truth table. If the logic function is more complex than what can be represented within a single LUT, multiple LUTs are combined together with outputs of certain LUTs feeding the input of other LUTs. In addition, each LUT is associated with a single flip-flop, enabling the creation of registers by combining

multiple LUT/flip-flop elements together.

The routing channels linking the CLBs together are configured using switch blocks that enable signals to pass between components. These switch blocks and LUTs are all configured using an uploaded configuration file that is stored in SRAM.

Xilinx, one of the largest FPGA vendors, produces a series of FPGA devices with various features. Xilinx's Vertex II Pro FPGA device utilizes CLBs that contain four slices. Each of these slices contain two LUTs and other related elements. Because of the multiple slices per CLBs, many optimizations can be performed when implementing designs on the FPGA.

In addition to standard LUTs, the Vertex II Pro FPGA also contains specialty blocks for integer multiplication, memories, clocking and two PowerPC processing cores. These blocks can be integrated into circuits allowing for more efficient designs. There are no arithmetic blocks within the FPGA supporting division or floating-point operations. If such operations are desired, they can be approximated, fully configured within the FPGA logic, or implemented in on-chip memory as a look-up table.

Features

There are several advantages to using FPGAs for processing.

- **Parallel Calculations:** Large increases in performance can be obtained despite the relatively slow clock speeds of FPGAs (∼150MHz for the Vertex II Pro) if an algorithm can be divided into a sufficiently large number of computations that are performed by dedicated parallel hardware elements.

- **Configurable Data Bit Widths:** The ability to control data bit widths enables more efficient resource utilization during processing and provides greater

bandwidths, compared to general processors (which have fixed bit widths).

- **Compact Size and Weight:** All related circuit logic can be implemented into one FPGA that requires no external components (other than IO and power), resulting in a compact design.
- **Durability:** FPGAs are compact integrated circuits, which can be easily encased to provide a durable unit with no mechanical components.
- **Low Power Requirements:** FPGAs have low power requirements because they implement custom logic that runs at relatively slow clock speeds and operates at low voltages (less than 3 volts). Therefore, FPGAs dissipate less heat. As a result, cooling is less of a problem for these devices compared to general processors.

Although, application-specific integrated circuits (ASIC) can provide even greater clock speeds and have lower power requirements, the high cost of designing and fabricating these components in small quantities make FPGAs a more economical option in many cases.

Drawbacks

Even though there are many advantages in using FPGAs rather than a general processor, the cost of increased development time for converting software algorithms into hardware implementations must be considered. Unlike sequential software development, hardware development requires consideration of circuit timing, multiplexing of buses, designing of state machines, managing of on-chip memory, debugging using logic analysers, and many other factors. While software debugging allows the halting

of program execution and quick viewing of variables, hardware debugging must be built into a design. It is also necessary to capture hardware signal behavior instead of examining variable values.

1.2 Application - Visual Tracking in Space

Many aging satellites that are in orbit around the Earth are reaching the end of their lifespans and therefore require maintenance in order to continue operating [5]. Due to limitations in space shuttle orbit altitudes, current space shuttle satellite maintenance is only performed on low orbit satellites. Because of the enormous expense, space shuttle satellite recovery is limited to a few extremely expensive satellites, such as the Hubble Telescope. In 1993, for example, the Hubble Telescope was repaired by a manned shuttle mission to overcome numerous manufacturing defects that hindered its optical system. Figure 1.3 illustrates the Hubble Space Telescope during these repair missions.

These manned missions pose serious risks to the astronauts performing the extravehicular activities required for docking procedures and maintenance. Although these operations could be performed via telerobotics, inherent difficulties with communications and teleoperations could occur. For these reasons, an automated vision guided servicer module (utilizing visual tracking during the satellite capturing phase) could replace these manned recovery and repair missions. An autonomous robotic device with a telerobotics option would be more practical [6].

Visual tracking for satellite recovery can be implemented with a realtime high-speed (200 fps) ICP-based FPGA platform. The low power requirements of FPGAs make them ideal for space applications where energy is extremely limited. Compact

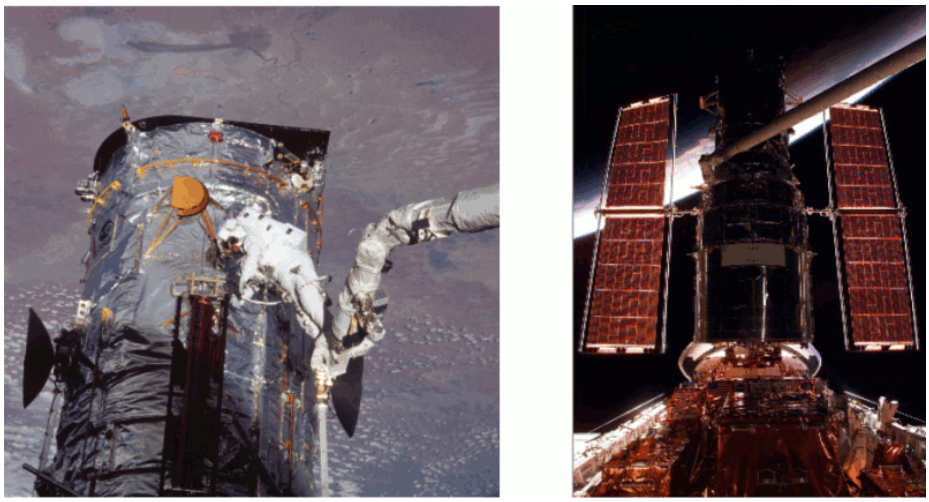


Figure 1.3: Photos of the Hubble Space Telescope during manned repair missions (permission to reproduce from NASA).

size and minimal package weight make the FPGA an ideal choice because the payload size and weight significantly affect launch costs. Also, the solid state of FPGA hardware makes them resistant to the harsh forces of launching systems.

1.3 Contributions

The contributions of this thesis include:

- **Design of an FPGA ICP based tracker.** This design was investigated because the ICP is the most widely used range image processing method and speed is important to process the large amounts of data. Despite its widespread utility, there is no documented research on a hardware implementation of ICP. Therefore, research reported in this thesis may be of broad interest to the community. In order to effectively utilize FPGAs, the different ICP stages were analyzed for appropriate conversion to hardware components.

- **Design of an FPGA nearest neighbor method.** This design was investigated because the nearest neighbor is the bottleneck of the ICP algorithm. Analysis of the nearest neighbor problem for an FPGA implementation was performed. Several novel FPGA nearest neighbor variations were considered. This variations involve a trade off hardware complexity, FPGA utilization, and nearest neighbor correctness.

- **Implementation.** The ICP based tracker was implemented into an FPGA so that system performance could be evaluated. To enable this evaluation, debugging and IO interfaces were designed so that a personal computer could be directly interfaced with the FPGA system. Furthermore, the tracker was designed to be integrated within a larger stereo vision system on a single FPGA platform.

- **Experiments.** The tracker was analysed in order to characterize the system's performance. Tests were designed to evaluate particular aspects of the tracker while working around physical limits of the FPGA, and the testing apparatus.

1.4 Organization of Thesis

The remainder of this thesis is organized as follows. Chapter 2 presents past research in computer vision, particularly ICP based tracking and ICP optimizations. Chapter 3 provides an explanation of the methods explored to solve the tracking problem and includes the final solution that is physically implemented in hardware. Chapter 4 describes the testing methods explored to characterize the tracker's performance and includes a discussion of results. Chapter 5 concludes the thesis by summarizing the outcomes of the research and indicates directions for future work. po

Chapter 2

Literature Review

2.1 Visual Tracking

Computational visual tracking is the technology that imitates the human visual system when it follows the path of a moving object. There are numerous applications for visual tracking including: video surveillance, innovative medical procedures and advanced robotics. At present, there are many diverse tracking techniques - each with its own advantages and disadvantages. Many factors must be considered when designing or selecting a technique that will track successfully.

2.1.1 Classes of Trackers

There are three general classes of trackers: 2D-2D, 2D-3D, and 3D-3D. These classes are named according to the type of data that the trackers use as input, and the type of pose information that they return as output.

- 2D-2D trackers accept a stream of 2D intensity images as input and return the

2D translational position of objects being tracked as output.

- 2D-3D trackers accept intensity data as input and return the 3D six-degree-of-freedom rigid transformation of tracked objects as output.
- 3D-3D trackers accept 3D range data as input and return 3D pose information.

2D-2D Trackers

2D-2D trackers use 2D intensity image input. A 2D intensity image is used to determine an object's pose only within the 2D translational domain defined by the $x-$axis and $y-$axis that corresponds to the row and column space of the image. An object's translation or rotation along the $z-$axis cannot be determined. These types of trackers can be used in surveillance situations where the object's position, relative to the camera, is used to realign the camera position. In Li et al. [7] a 2D-2D multi-scale gesture recognition system is designed to track sequences of human gestures. This technique (and many other 2D-2D tracking techniques) generally fail when any moderate 3D transformations are applied changing the object's vantage.

2D-3D Trackers

Because 2D-2D trackers fail during 3D transformations, many of the 2D-2D approaches have been extended to recover the extra dimension [8]. Trackers using this technique can be described as 2D-3D. These trackers have the advantage of using simple intensity image sensors such as webcams. Although it may seem that 2D-3D trackers are the optimal choice for tracking 3D, this tracking technique usually suffers from the inherent complexity of inferring the 3D transformations.

3D-3D Trackers

The 3D-3D trackers utilize 3D data, typically in the form of a stream of point cloud data [9] [10]. Because data is already describing the scene in 3D, there is no need to process it to extract 3D meaning. This use of 3D in itself can drastically simplify and improve tracking in 3D. The focus of this thesis is the 3D-3D tracker, specifically model-based tracking of 3D data, utilizing data from a stereo vision sensor.

2.1.2 Pose Determination vs. Tracking

Pose determination is the process of finding the pose of an object without any prior knowledge of the object's position in the scene. Generally, pose determination is a relatively slow task that cannot be performed in realtime. Exceptions to this problem do exist. For example, target-based pose determination systems that use well known, easy to segment markers that allow for fast object recognition or pose determination are available [11]. Realtime tracking provides the object's pose with no significant time delay. The pose recovered is therefore still current.

Using 3D data, the pose determination task becomes slower but more accurate. This performance difference is due to the larger amount of scene information from the increase in dimensionality. For example, [12] [13] describes a system capable of performing pose determination using 3D data by extracting object features at approximately 4 fps. Because these pose determination systems are unable to function in realtime, trackers are often utilized so that pose determination is only required at the beginning of a sequence of frames.

Typically, an object will maintain similar poses between adjacent frames. This similarity between adjacent frames is called temporal coherence. Because of this

temporal coherence, tracking systems usually only search a limited region around the previous pose. This results in a less costly, more time-efficient technique that can be performed in realtime.

2.1.3 Tracking by Fast Pose Determination

As stated previously, one method of tracking an object through a sequence of frames is to perform pose determination quickly. Because of the physical limitation on an object's movement and the limitations of sensor acquisition rate, fast pose determination is not an ideal tracking approach. The object's movement limitations impose that the object will not move far from its last known pose. Stated another way temporal coherence exists between consecutive frames. There are numerous ways to perform pose determination quickly. One such method that has been used to track in 2D video streams utilizes SIFT features [14], repetitively re-acquiring the object's pose.

2.1.4 Target Trackers

Pose-determination-based tracking has also been implemented using special tracking targets to act as key features [11] [15]. Target trackers seek to simplify the search space by affixing easy-to-detect markers on the object. The markers are designed specifically so that either the 2D or 3D pose can be extracted robustly under varying lighting conditions and varying sensor marker vantages. An example of a tracking target is illustrated in Figure 2.1.

Because the markers are designed to be robust and easily detectable, the systems are usually reliable with high recognition and tracking speeds.

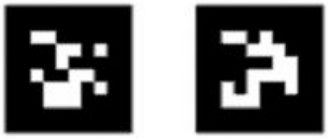


Figure 2.1: Example targets used by ARTag. The targets are chosen to be easily distinguishable from one another and can be robustly segmented from images.

One such marker system, ARTag [11], which is based on ARToolkit, provides fast and robust pose results using inexpensive webcams and markers printed on paper. Commercial implementations of target trackers also exist, such as the OptiTrack [15] system made by NaturalPoint Incorporated. This system relies on target-based techniques, using active light-emitting diodes as targets. The tracker is able to acquire motion at high frame rates due to its ease of segmenting the markers from the background clutter.

These pose recognition and tracking methods are limited to situations where markers can be applied in advance. This limitation prevents the tracking of objects that cannot be readily marked (such as satellites requiring maintenance while orbiting in space).

2.1.5 Tracking by Template Matching

Template matching is a well-developed area of computer vision. The main objective is to determine if and where something lies in a data stream. For 2D-2D vision systems, this could be simply a digital image of a view of an object of interest. This image is called the template. It is used, along with a distance metric, to determine where this object might be in another image. Various techniques exist for determining the proper

location of the object, most notably the sum of squared differences (SSD) metric, or the sum of absolute differences (SAD). Although template matching has been used mainly for 2D-2D systems, such techniques are also used for object recognition and tracking in 3D. In [8], template matching with SSD is performed to reconstruct the transformations occurring on planar objects. Many small templates are generated on a planar surface so that its transformation can be recovered during tracking. Some 3D-3D approaches use template matching.

The Bounded Hough Transform [10] utilizes template matching in a bounded voxel space to allow for efficient tracking of 3D range data. The Bounded Hough Transform tracking system utilizes physical characteristics of the scene. For example, the object's speed is usually bounded and the camera frame rate is usually fixed. Because of these characteristics, the search area between frames is bounded. Voxel template matching is then used to search discrete small regions around the last observed position of the object.

2.1.6 Predictive Filters

Kalman Filters

In 1960, Rudolf Kalman developed the Kalman Filter as a method for tracking linear systems that have both process and measurement noise [16]. This noise is considered to be independent with Gaussian probability distributions. The Kalman Filter cycle involves 3 steps as listed below.

1. **Predict** the next state.

2. **Update** the estimate using observations.

3. Proceed to the next time instance and **repeat** the cycle.

In order to determine the current position of the object being tracked, the recursive Kalman Filter utilizes only the previous state and the current measurement. The Kalman Filter uses only minimal memory and requires minimal processing. These features made it ideal for spacecraft navigation. As a result, the Kalman Filter was developed for use during the Apollo space program [17]. Because the tracked data was noisy and the objects being tracked could be modeled as a linear system, the Kalman Filter was a suitable choice for tracking trajectories. In order to overcome measurement noise, this filter considers both the prediction from a model and the current observations, thereby improving accuracy. During each time instance, weights are calculated to determine whether the model or observation is more reliable. Because of the filter's low processing overhead, robustness to noise, and simple operation, the filter is now being used for visual object tracking [18].

Particle Filter

Particle Filters are similar to the Kalman Filter. Both filters use a filtering cycle with a prediction phase followed by an update phase. Rather than tracking one estimate through filter cycles, the Particle Filter utilizes many different predictions based on a probability distribution function (PDF). These predictions are called 'particles' and are used together to form one estimate of the object's location. The particles are updated every cycle based on the current modeled PDF. These particles evolve over time. If the validity of one particle drops below some chosen threshold it is re-sampled by sampling near other particles with a higher probability of being valid.

This approach has been used for visual tracking in a number of applications.

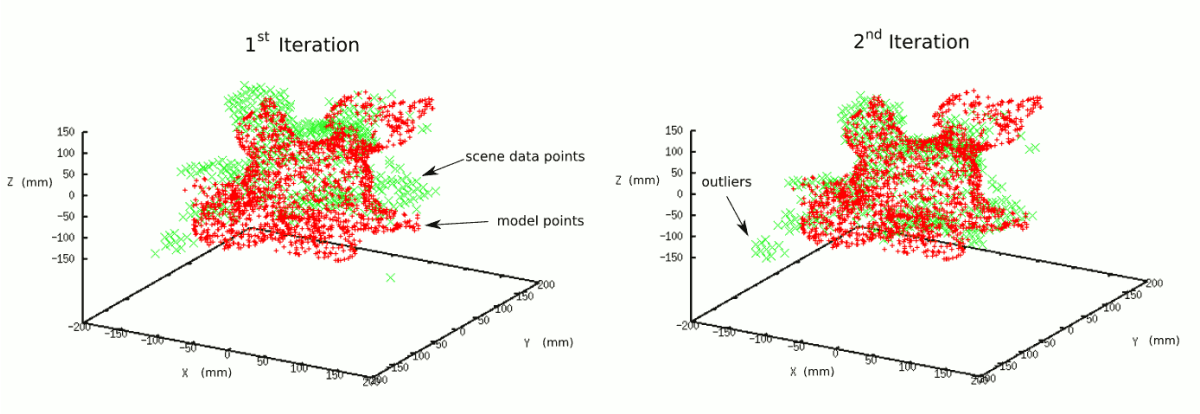


Figure 2.2: The first and second iterations of ICP registering scene data (in green) to an object model of the 'Big Bird' (in red).

For example, Nummiaro et al.[19] use a Particle Filter with color histograms to track people in 2D video. Histograms are invariant to scaling and rotation, and the Particle Filter is robust to short-term occlusion. Therefore, the combined histogram and Particle Filter results were utilized for tracking people involved in various sporting activities. Unlike Kalman Filters, the Particle Filter is not limited to linear models with Gaussian noise distributions. They do, however, require more computational resources to follow each particle.

2.2 Iterative Closest Point Algorithm

In 3D imaging, the Iterative Closest Point algorithm (ICP) is a common registration tool that was first described by Besl et al.[3]. This algorithm allows two 3D point data sets to be registered using a least-squares minimization. Figure 2.2 provides an example of ICP iterations in which a data is registered to a model. The ICP does not perform registration using any features other than a global feature—this is the

entire object's surface model. This feature enables the creation of object models from computer aided design drawings for objects that are not readily available—such as orbiting satellites [5]. Because of the simple and robust nature of the ICP, space research—specifically satellite recovery research, has taken an interest in using the ICP for autonomous control of space robotics. This thesis is specifically focused on the use of the ICP for autonomous satellite recovery.

2.2.1 Base ICP Algorithm

The ICP minimization is performed iteratively. The resulting registration transforms are accumulated and applied to either the model data **M** or scene data **D**, which are used for the next iteration. Once the least-squares error or some similar metric between the two point sets falls below a threshold, or the number of iterations rises above a threshold, the accumulated transform is then considered to be the estimated registration transform.

The problem is to find a transform between the two data sets in order to minimize the distance measure

$$\sum_i ||M_i - (\mathbf{R}D_i + \mathbf{T})||^2 \tag{2.1}$$

where M_i is the i^{th} model point, D_i is the i^{th} data point, and **R** and **T** are the rotation [3x3] and translation [3x1] matrices, respectively.

The base ICP algorithm consists of three steps, as listed below.

1. Correspondence - Nearest Neighbor Search
2. Recovery of Transform - Singular Value Decomposition (SVD)
3. Accumulate Transform

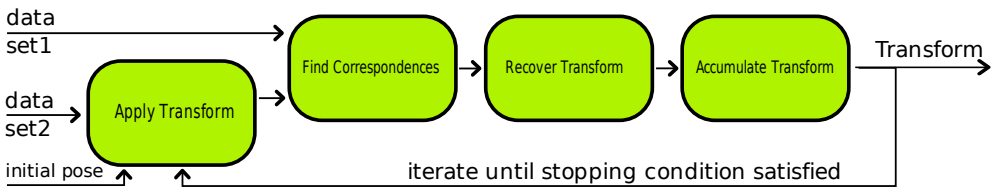


Figure 2.3: The flow of data through the main modules composing the ICP algorithm. After each iteration of the ICP, its output is used to re-transform the incoming data.

A data flow diagram of ICP is illustrated in Figure 2.3.

2.2.2 ICP Variations

Even before ICP tracking was available, research continued improving the speed and efficiency of ICP. Many diverse approaches have been utilized to achieve different effects. Some approaches focus on increasing the ICP iteration speed, the correspondent search correctness, and the correspondent search speed [4]. The correspondents search will be explained below.

Nearest Neighbor Techniques

The ICP requires the constant updating of point correspondences between two data sets. This operation is performed for each iteration. The corresponding point pairs are found using a nearest neighbor search. The objective is to find, for each point in one data set, the closest matching point in the other data set, as illustrated in Figure 2.4.

For ICP tracking, the distance metric used is generally the Euclidean distance.

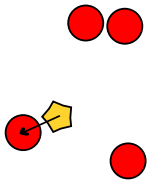


Figure 2.4: An example of the nearest neighbor search to find the closest point in a given set. Here the point represented by the hexagon is matched with its closest neighbor, determined by comparing the Euclidean distance with respect to all points of set circles. The result of this processing is the nearest neighbor pair, represented here by an arrow.

However, other metrics can also be applied [3] [4] [20]. The quality of the correspondences will control the ICP convergence rate as well as its robustness to noise and occlusion. The method for calculating the correspondences will also affect the speed of the system, depending on the complexity of the method chosen.

1. **Pre-Filtering Points**: Pre-filtering in ICP tracking is usually performed using a region of interest (ROI) surrounding the last pose estimate. This ROI is generally a cube region that quickly filters any points from the data that lie outside the desired volume. Because this ROI is applied before determining point correspondences, the number of points that are sampled from the incoming data can still be guaranteed. This filtering ensures a consistent number of data points for later processing.

2. **Selecting Points**: The various methods for selecting points have been studied thoroughly, and there are many sampling techniques available [3] [21] [22] [23]. The sampling method should be appropriate for the given conditions, such as the type of object, presence of clutter, and noise conditions.

   (a) ***Using All Points***: This method was originally proposed by Besl et

al.[3]. It provides a robust method for selecting points in all conditions. The search for the nearest neighbor can become quite large and expensive. Many of the points may be redundant and uninformative, such as points on a plane. However, in situations where complex surface variations exist, it is useful to fully sample the area.

(b) ***Using Subsampling***: Subsampling is a common practice for reducing the bandwidth requirement of a system.

i. *Uniform Subsampling*: This process is simple to implement and can drastically reduce the number of points required for correspondence [21]. Because the points are uniformly chosen, the only noticeable change is the increased number of ICP iterations that are required before convergence, as noted in [22]. This property is important when deciding on an ICP implementation.

ii. *Random Subsampling*: This method can also reduce the number of points required for correspondences [23]. Because this technique is performed randomly, it can be used to reduce the effects of noisy data over a sequence of frames.

Performance improvements for the nearest neighbor algorithm (that are gained by reducing the number of points sent to the nearest neighbor) directly increase the number of ICP iterations required before convergence. With the reduced number of point pairs, there are less correspondences that are used to register the point clouds. This reduction will cause the ICP to require more iterations, effectively reducing the speed of the system back to its original state.

This compromise between subsampling vs. number of iterations is important when deciding on a system architecture. When working with FPGAs that have limited memories and other resources, this trade-off can help improve results.

(c) ***Using Color and Intensity Information***: Color and intensity information can improve the selection of the nearest neighbor. However, it is necessary to ensure that the color and intensity information is invariant to view changes because these changes can drastically effect the gains of this information. Weik [20] used view invariant features based on intensity with color and demonstrated that these features speed up the ICP convergence.

(d) ***Extending Selection To Both Point Clouds***: The process of selecting points can be performed on either point cloud or even both point clouds as described by Godin et al.[24].

(e) ***Selecting Normals as Diverse as Possible***: Rusinkiewicz et al.[4] used a simple criterion to select points in areas of details. This process is called normal space sampling. Areas of 3D imagery that have greater details also contain diverse point normals; therefore, these normals can be used as a guide to control the number of samples for a particular region. Results using this method have the ability to converge on the solution in situations where few object details are present. Objects with minimal descriptive elements can still be registered if these details are not overlooked during the subsampling phase. Because most objects are usually not as featureless as a primitive plane, this approach should not always be necessary in order to achieve a proper pose estimate.

3. **Weighing Points**: In order to efficiently utilize the available scanned points on an object, point weights can be assigned so that informative regions are taken into greater consideration.

   (a) ***Constant Weight***: This technique is the basic ICP method for weighing points which considers all points with the same priority. The technique has no extra overhead and is ideal for systems with fewer hardware resources.

   (b) ***Similarity Between Normals***: This method compares weights by considering the solid angle between normals [24]. This method enables the matching of points that have a higher probability of being in the correct location (when dealing with non-rotated transformations). Unfortunately, there is the increased overhead associated with this calculation. In addition, not all sensors provide point normals. The extra calculations required to derive normals will further increase computational overhead.

4. **Point Correspondence Search**: Many techniques have been developed in an attempt to speed up the matching of points, which is the greatest bottleneck of the ICP algorithm [3] [25].

   (a) ***Closest Point***: Besl et al.[3] use the most basic brute-force approach for selecting points. The complexity of this search method is $O(N^2)$, where N is the number of points to be compared. This method always guarantees finding the best match (closest point) from the data.

   (b) ***Data Structure - K-D Trees***: The K-D Tree structure has a complexity of $O(N \log N)$ and has been effectively used for correspondence search [26].

This approach allows for a fast search but requires the initial construction of the tree structure.

(c) ***Multi-Resolution Models***: A type of model with various levels of resolution was proposed by Jost et al.[27]. For initial coarse registration, a low-resolution model was used. After a few iterations, higher resolutions were then used until the desired registration was achieved. This correspondence search allows for faster point matching in early iterations but still provides rapid convergence at a cost of storing many more models in memory.

(d) ***Intersection of Point Normal to Surface***: Chen et al.[28] use point normals projected through a surface model of an object. This method extrapolates the model. The goal is to find the nearest point on a plane rather than just the typical point-to-point correspondence. This technique is ideal for smooth surfaces because extrapolation works best in these cases.

(e) ***Reverse Calibration***: This method finds the matching point by projecting the point using the direction of the scanner's observation towards the object [29].

5. **Point Rejection**: Rejecting points is an appropriate method to remove outliers and other incorrectly matched pairs. This point rejection is performed after the pairing of points in the nearest neighbor phase, so that it reduces the number of considered point correspondences.

(a) ***No Rejection***: This method assumes that there are no outliers, and allows all point pairs to be processed to recover the transform. This method

works adequately for situations where a large portion of the data are inliers.

(b) ***Rejection Based on Distance Threshold***: For any point pair beyond some distance threshold, the points beyond this threshold will then be rejected from future consideration. This method is effective for filtering outliers, which may be caused by noise or clutter, because of its simple nature.

(c) ***Rejection Based on the Lowest Quality Pairs*** (distance metric...etc): Only the top percentage of the highest quality point pairs are considered. Any lower quality point pairs are rejected, such as those that fall at a greater distance, have poor quality matches [23], or that have some other poorly matched criterion, such as color and intensity.

(d) ***Rejection Based on Pairs that are not Similar in Nature***: This method selects points only if they have surrounding points that agree with the chosen criteria. This method prevents sensor errors from being used as inliers, but may also remove small details from the object, thereby preventing them from being observed.

Singular Value Decomposition

Singular Value Decomposition (*SVD*) is a useful factorizing technique that can be applied to solve the least-squares minimization. SVD is used to recover the unknown registration transform between the two data sets. The corresponding points are first generated using nearest neighbor search. Then, these correspondences are used to compose the covariance matrix that is to be decomposed by SVD in order to generate the estimated transform.

Both software and hardware approaches have been developed to perform SVD. A software implementation of SVD is available in Numerical Recipes [30]. There has also been research into SVD on distributed re-configurable systems [31] [32]. Some commercial available HDL packages also exist to implement SVD in hardware, such as Accelachip and System Generator. However, the required hardware resource overhead is typically high.

ICP Optimization

Optimizations to ICP are methods designed to improve ICP performance. For example, reducing the required number of ICP iterations can increase frame rates.

1. ***Pre-alignment of Object Before ICP***: The Bounded Hough Transform has been utilized to improve ICP for high-frame-rate object tracking [10]. This tracker, which was explained in Section 2.1.5, uses a voting scheme approach to track at over 100 fps on a standard PC. Integrating ICP with the Bounded Hough Transform enhances ICP convergence speed by providing a much better initial pose estimate. This closer initial estimate requires fewer ICP iterations for convergence to the solution.

2. ***Parametric Acceleration***: Acceleration is another method to decrease the number of iterations required for the ICP to converge. The main idea is to extrapolate the overall direction of each ICP iteration. This extrapolation is then used to update the transformation, thereby bypassing some iterations [3]. Because this extrapolation process can be a lightweight task when compared to the processing required within a standard ICP iteration, efficiency improvements can be achieved.

In general, it is preferable to register D scene data points to a static M model points, and not the other way around. This registering direction is chosen because sensors are normally only able to acquire a partial 3D view of the object and scene. This partial view is known as 2.5D, as the back surfaces are occluded from the sensor view.

If the M model points were registered to the D scene data points, any of the occluded object points (which may be present in the model) would not have any true corresponding points in the scene data, which could cause registration errors. These registration errors can be avoided by either rejecting points based on a threshold, or simply by performing the matching from D to M.

2.2.3 ICP-Based Tracking

Although ICP was developed originally for data registration, it was later extended to 3D tracking. The ICP allows for tracking with various types of 3D sensor data and is robust to sensor noise, sparse data, and occlusion. In addition, ICP can track erratic paths.

Simon et al.[9] were the first researchers to use ICP for the realtime tracking of rigid objects. Their ICP tracker used a 32×32 pixel stereo vision sensor and was able to track an arbitrary object at a frame rate of up to 10 fps. They used efficient K-D Trees as well as point caching to further increase operating speeds.

This approach uses temporal coherence for efficient tracking. The temporal coherence between frames can provide closer initial pose estimates to enable the ICP to function at increased speeds. In ICP tracking, the previous frame's pose estimate is used as the initial pose estimate for the current frame, as shown in Figure 2.5.

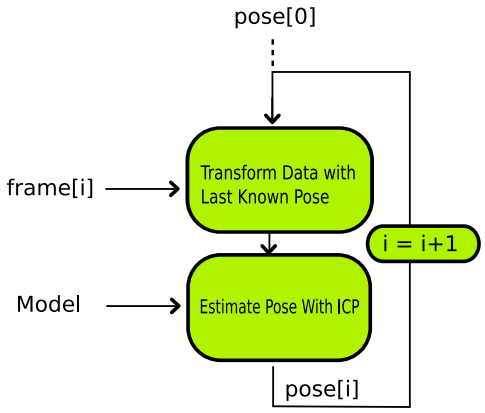


Figure 2.5: The simple loop used for tracking a sequence of frames i with ICP. At the start of tracking, the initial pose is input into the system. For all later frames, the system relies on feedback from the previous frame.

This method works because objects generally follow certain bounded paths, with only small relative changes in their pose values between frames. Typically, objects have a maximum speed and sensors have a set frame rate. Because of the object and sensor speed limitations, the previous frame's pose estimate provides an approximate position of the pose of the object in the current frame. In turn, the previous frame pose estimate allows the system to be more robust to outliers that could cause the system to settle at a local minimum.

The steps involved in ICP tracking are:

1. Load pre-computed object model.
2. Acquire sensor data.
3. Update pose with given initial estimate.
4. Acquire Sensor Data.
5. Calculate incremental pose (estimate via ICP).

6. Update pose and go to step 4 and continue.

2.2.4 Parallel ICP

Because there are numerous repeated tasks performed throughout the ICP algorithm, research has been performed to exploit this repetition through the use of parallel processing computers [22]. Parallel ICP methods attempt to segment the ICP problem into many sub-problems in order to separately compute partial solutions. Once a partial solution has been performed, the partial results are sent back to the parent processor so that the full solution can be integrated.

Quaternion Approach

The main goal of the quaternion approach is to spread the correspondence search among the processors. After the correspondences are found, they are returned to the main processor and the transform is then reconstructed from the different correspondence pools. This method creates a communication bottleneck for the system as each processor is required to send their correspondences back to the main processor.

Rather than returning the correspondences, the communication bottleneck can be relieved by having the child-nodes compute the partial sums. These partial sums are then combined to form the cross-covariance matrix that is used to reconstruct the registration transform. This use of partial sums not only reduces the amount of communication between processing nodes, it also reduces the amount of processing required by the parent-node to calculate the covariance. Thus, the overall system is able to compute the ICP with near linear speed-up, dependent only on the number of child-nodes.

Shuffling Approach

Rather than focusing on the correspondence search, shuffling breaks the ICP problem into many smaller ICP problems that are then solved in parallel. The shuffling technique uses a tree search structure. A common property of most tree structures is the tendency to execute with increasing efficiency as the actual distance to the nearest neighbor decreases. Because of this inherent property, shuffling is performed by uniformly subsampling the data in order to obtain an average nearest neighbor distance for all processing nodes. This uniform subsampling results in a more balanced load among the processing resources and a more consistent runtime for each iteration on every processing node. In effect, this balance reduces the chance of any one node from being a bottleneck of the system. An improvement of approximately 30 percent was noted by using this Shuffling technique [22].

2.3 Hardware Trackers

Hardware acceleration is a common tool for computing. Most desktop computers utilize hardware-accelerated 3D graphic cards that allow for considerable speed improvements—compared to the general processor functioning on its own. These speed gains are due to the application-specific hardware that has been parallelized for increased throughput.

Research has been attempting to utilize the massive parallel capabilities of hardware for tracking solutions. For example, Mühlbauer et al. [33] have worked on a hardware-software feature tracker that focuses on hardware/software partitioning. This tracker was able to detect specific features at 162 fps for a 640×480 pixel video

source with hardware acceleration. However, object tracking was later performed only at 4 fps by tracking these features. Speeds were limited due to sequential processing in software, which caused a bottleneck in the system. These result demonstrate that care must be taken when designing such hardware-software systems.

Chapter 3

Hardware Based ICP Tracker

3.1 Problem Definition

The objective of this work is to construct a hardware component capable of tracking objects at a high frame rate of over 200 fps, utilizing custom hardware. This tracker is required to track objects of various sizes and shapes at a distance of approximately 2 - 10 meters from a stereo vision sensor. The stereo vision sensor provides data at a resolution of 640×480 pixels, with 128 disparities. The pose of the object is reported by the tracker as x,y,z translation and rotation along the x,y,z axes of the camera coordinate reference frame. The tracker must be compact, lightweight, have low power requirements, and be capable of being integrated with both a stereo sensor and stereo extraction module, all using a single FPGA.

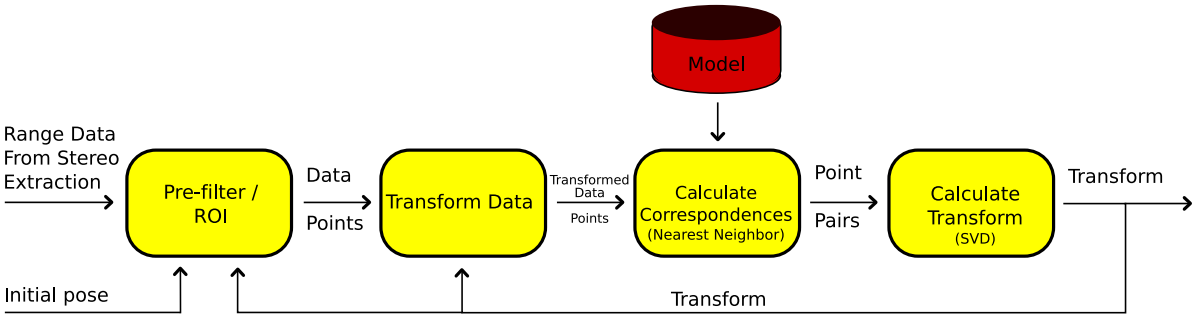


Figure 3.1: The data flow through components of the ICP hardware.

3.1.1 ICP Algorithm

In order to track objects in 3D, the ICP algorithm is applied across a series of frames by exploiting inter-frame object pose coherence. The ICP is a simple and robust approach for tracking. However, processing the nearest neighboring pairs can easily become the bottleneck of the system when working with large numbers of model and scene points. Indeed, the nearest neighbor calculation can consume approximately 95 present of the processing time when using a sequential brute-force approach [3] [25]. This work focuses on implementing ICP for tracking on an FPGA platform utilizing custom logic components in order to reduce this bottleneck.

3.2 System Components

The tracking system is decomposed into four main components, as seen in Figure 3.1. This system is closed loop requiring feedback from only the previous frame's processing.

Pre-Filter - Filter Range Data

Before data is processed by the ICP algorithm, the surrounding clutter, any outliers, and any redundant points should be removed for optimal performance. If pre-filtering is not performed, the ICP may settle into a local minimum. This initial filtering stage attempts to remove most of the unwanted points so that the algorithm can operate quickly. Otherwise, the ICP speed may decrease due to the excessive number of points being considered.

Transform Data

During execution of ICP, the incoming scene data points must be transformed every iteration and cycle of the algorithm. This module performs the task using the transform recovered from the previous frame's processing. At the initial frame, the system has no previous state information, and therefore it must be provided with the initial pose of the object. This initial pose is chosen to be within the minimum well space of the actual object pose to prevent it from being caught in a local minimum. The initial pose is used to bootstrap the entire tracking process.

Nearest Neighbor Search - Calculate Correspondences

Once the scene data points have been transformed, the nearest neighbor search then determines the closest matching pairs of points from the scene and model data. That is, every point from the scene is matched with a corresponding point of the model. This search is a one-to-many mapping that allows for three possibilities for the model points: unselected, selected, or selected multiple times.

Transform Recovery - Calculate Transform

Once the point correspondences have been made, transform recovery is performed using Singular Value Decomposition (SVD). The resulting partial transform, which corrects for the alignment error between the scene and model points, is added with the previous accumulated transform. Then, the whole iterative process is repeated beginning with transforming the scene points with the accumulated transform. It repeats as long as the number of iterations is less than a given threshold.

Once a new frame is received, the process is restarted by the Filter Range Data step, using the newly determined accumulated transform from the previous frame.

3.2.1 High-Level Hardware Design

The tracker's overall circuit design is shown in Figure 3.2. Buffers between individual components are used for this prototype implementation and may be removed for resource optimization. The system has been developed modularly in order to facilitate integration and debugging.

3.3 Apparatus

3.3.1 Field-Programmable Gate Array

For the design and implementation of the hardware ICP tracker, the Amirix AP1100 Development Board was selected due to its availability, connectivity to off-chip memory, and overall FPGA logic capacity. The AP1100 is comprised of a single Xilinx Vertex II Pro (XC2VP100) SRAM FPGA that has over 99,000 logic cells, each having one 4-input look-up table (LUT). Approximately 1000KB of block random access

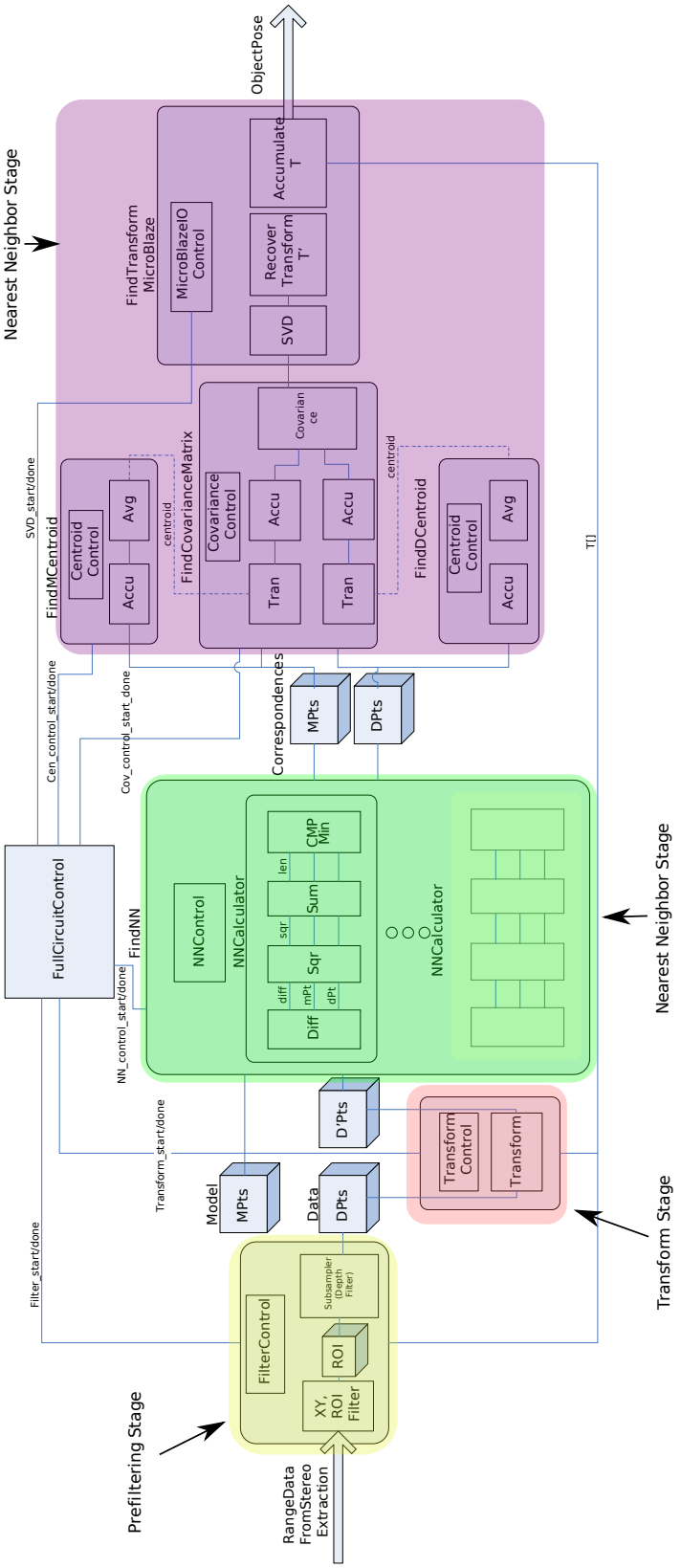


Figure 3.2: A system block diagram.

memory (BRAM) is available along with 1164 IO pins, two PowerPC cores, and 444 18×18 bit multipliers.

Specifying Hardware for FPGA

In order to design hardware systems, special programming languages called hardware description languages (HDL) are used. For this work, VHDL has been chosen as the HDL to implement the design as it is a leading standard. These HDL languages describe the desired circuits in a human-readable form that is interpreted by a logic synthesizer, and then compiled for placement and routing of components on the physical FPGA. The final output file, referred to as a 'bit-stream', then describes the state of the entire FPGA. For example, a bit-stream will indicate the contents of LUTs that are used to specify logic, and also the contents of embedded memory blocks. In addition, the bit-stream will also establish the connections among all utilized elements in the FPGA. The wiring on an FPGA is actually fixed; however, paths can be modified using switch boxes. The bits in the bit-stream indicate which wires are connected by the switch box so that electrical signals can pass. All LUT and switch box configurations are stored in dedicated SRAM within an FPGA, as outlined previously in Section 1.1.2. Once the SRAM in an FPGA is programmed, the FPGA retains its configuration until power is removed from the device.

There are specific blocks present in some FPGAs, such as BRAM and multipliers. These blocks require special HDL descriptions to enable the synthesizer to recognize and implement them correctly. Specifying RAM is a common example. One option is to specify it as distributed RAM that the individual flip-flops and LUTs as elements. Another option is to use the available BRAM elements present in the FPGA fabric.

Specifying hardware is drastically different than specifying software. Each action in hardware is timed to a system clock so that all operations can be synchronized. With hardware, precise timing of actions must be specifically stated. With sequential software, precise timing is not required—only an order of execution for the operations to be performed. For example, if the operation $X = Y$, followed by $Z = X$, is converted from software to hardware, then two registers can be used to capture the values of X and Z. If proper care is not taken to ensure that the $X = Y$ assignment is performed before the $Z = X$ assignment, then we may actually obtain the incorrect result of $X_i = Y$ (the initial value of X) and $Z = X_{i-1}$, rather than the intended result of $Z = Y$. This problem is due to the inherent parallel nature of the logic operating in FPGAs. If used correctly, this parallel flow can allow for substantial increases in throughput.

Unlike general-purpose processors that usually have one arithmetic logic unit (ALU), an FPGA can have many independent arithmetic units that can function simultaneously in different parts of a large logic circuit. Because of the distributed arithmetic nature of FPGAs, floating-point operations are usually avoided so that the complicated logic that they require does not consume logic elements and produce slow system performance. Instead, fixed-point representation is utilized to approximate these numbers, which simplifies arithmetic calculations. Therefore, for every fixed-point signal, register, multiplier, RAM, or other FPGA element, the user must decide on a fixed bit width.

Another technique for enhancing performance is pipelining. Pipelining introduces logic and timing overhead to enable different stages in a pipeline to maintain the necessary per-stage information for performing the relevant operations in an overlapped

manner. Each stage can then process a portion of the problem and pass the result to the next buffered stage. Because of this overlapping of operations, pipelining enables higher clock speeds as well as enhanced resource usage (with a penalty of pipelining logic and timing overhead).

3.3.2 FPGA Design Cycle

In the FPGA design cycle, the first task is to determine the system architecture, after which the system components are specified in HDL. Because of the complexity of 'place and route', the step in which components are positioned onto an FPGA, and the difficulty of debugging in hardware, simulation is then performed to ensure proper functionality. Once components have been finalized, they are then tested in hardware and system integration is performed. Integration tests are next performed in hardware. The complete design cycle is illustrated in Figure 3.3.

Test Tools

Simulators are a common hardware testing tool. Because of the difficulty of debugging digital circuitry, and the long compile times for place and route, simulations are utilized to verify functionality before proceeding to the time-consuming steps of the overall design flow in Figure 3.3. For this work, both Xilinx Simulator and Mentor Graphics ModelSim have been used.

If the use of simulation yields satisfactory results, the design can then be implemented in hardware and tested using a logic analyzer. In recent years, FPGAs have become large enough to implement IP core logic analyzers into designs. These embedded logic analyzers are implemented with the design being tested utilizing memory

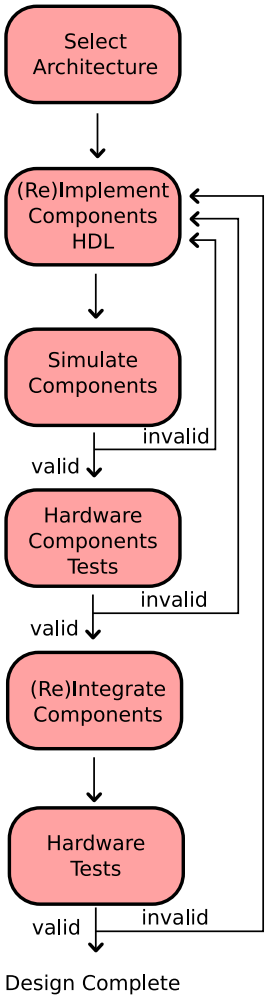


Figure 3.3: The design cycle used for implementing circuity hardware in a FPGA. Simulation of components is performed initially to quickly remove circuit errors before integration and testing in hardware.

and logic within the FPGA. They can be quickly integrated into an FPGA design, and internal signals can be selected to capture required data. This data is then uploaded with a common IO interface (JTAG) for debugging on a PC. All logic analyzer debugging for work in this has been performed using the Xilinx ChipScope IP core.

3.3.3 External Components

The tracking system that is described in this thesis is designed to be integrated with a custom stereo vision sensor (with rectification) and a stereo extraction module that are being co-developed by groups at the University of Toronto and Ryerson University. Because these components are being implemented externally to the work described in this thesis, they will be described briefly in this section. An initial prototype of the FPGA development board and stereo vision sensors can be seen in Figure 3.4. The external components are listed below.

- Sensors: The cameras are state-of-the-art custom digital video sensors able to capture 200 fps at 640×480 pixels with the distance between sensors (baseline) of 160 mm.

- Rectification: Rectification is performed, using camera calibration parameters, to correct lens and sensor misalignments. The goal of this rectification is to align the image pair so that features appearing in the left image's scan line also occur in the same right image scan line. This enables a simpler 1D search to be performed along a single scan line during stereo extraction. The rectification circuit is provided with camera calibration parameters that are calculated off-line.

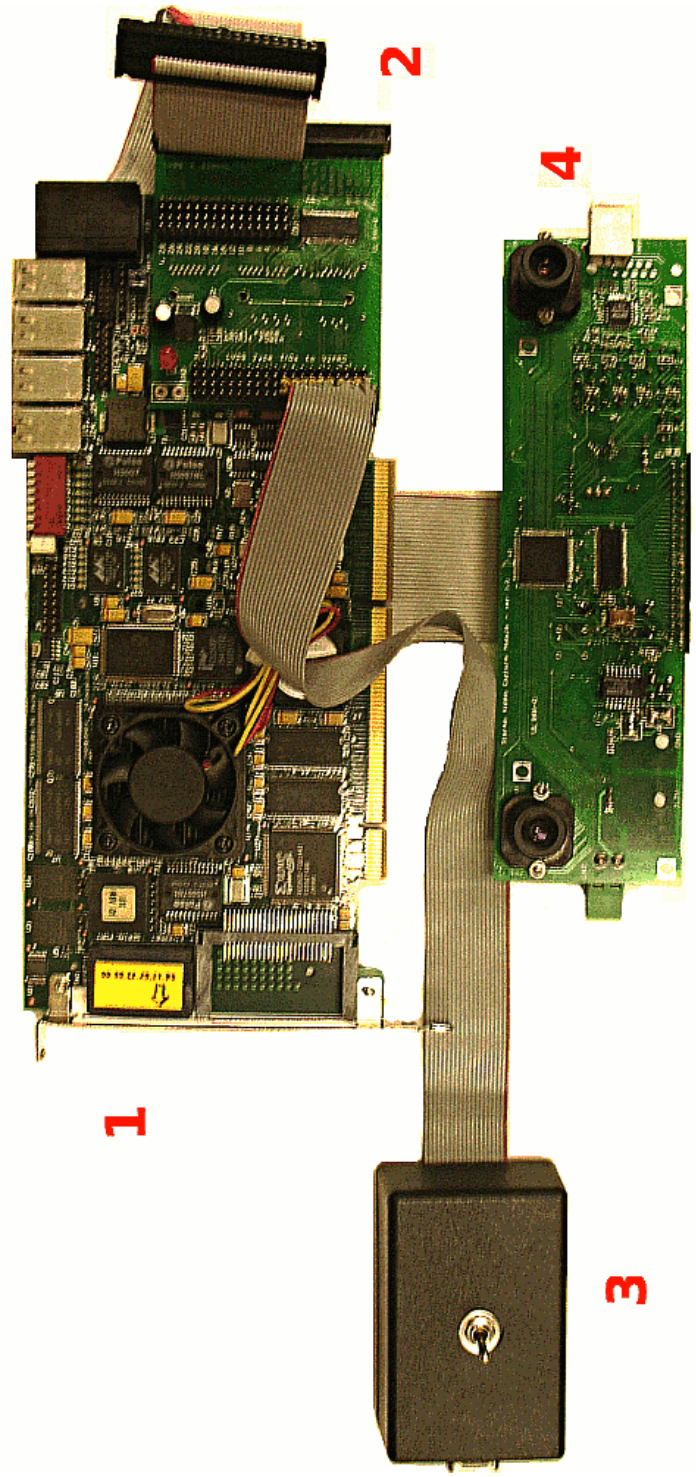


Figure 3.4: The 30 fps hardware prototype used for initial integration tests - 4 components are indicated: 1) Amerix AP1100 prototyping board 2) Custom camera and VGA interface 3) VGA debugging output/controller 4) Custom 30 fps stereo sensors

The off-line calibration processes generates intrinsic and extrinsic camera parameters. The intrinsic parameters are: the focal length f, the pixel size, and the principal point. These intrinsic parameters describe the configuration of the internal geometry of a camera. There are two extrinsic matrices, translation $\mathbf{T}$ and rotation $\mathbf{R}$, that align the image to a known frame. Because the stereo sensor utilizes two cameras, the sensor's focal length is an approximate average of the two camera focal lengths. In addition, the distance between the two cameras (the baseline $\mathbf{B}$), which is required for converting disparities to depth estimates, can be determined as the difference of translations $\mathbf{T}$ of the cameras.

- Stereo Extraction: The stereo extraction module has been designed to use dynamic programming in order to provide the desired 200 fps data frame rate at 128 disparities at 640×480 pixels [1]. The interface between the stereo extraction module and the tracker module is timed by a valid data signal from the stereo extraction module.

3.4 Data Filtering

The data filtering task can be separated into two parts:

1. Segment the object from the surrounding clutter.

2. Subsample the remaining data rejecting obvious outliers.

The goal of this data filtering is to remove most of the unnecessary data from consideration during the execution of ICP. This extra data would slow down the ICP process and could even drive the solution to an incorrect result.

Because the filtering is performed on the compact disparity data, it is beneficial to preserve this disparity data as far as possible through the filtering stage. The conversion to point data is performed at the final step in the filtering stage, just before writing to memory. Data filtering functions in a series of 3 processes: disparity region of interest (ROI), cube ROI, and subsampling, as described below and illustrated in Figure 3.5.

3.4.1 ROI Filter

Disparity Region of Interest

The filtering stage begins with a processing step that removes all data not within the desired ROI. Because of limited FPGA hardware resources, the filtering has been performed starting with the raw disparity data of 7 bits (128 disparity) per pixel rather than the 3×16 bit x, y, z point data. This technique saves storage space as well as the number of logic blocks required.

The ROI is applied to the disparity image where a square is used to form the bounding constraints. The resulting filtered region, if converted from disparity into 3D point data, will fill a frustum that will be filtered further in the following stages.

Because of the necessary hardware memory resources, which are used for buffering the maximum sized ROI, only one disparity ROI size is used. This fixed sized ROI also reduces the control overhead associated with a variable-size ROI. As a result, for objects that are further away from the sensor, more surrounding clutter will be passed though the ROI. This excess clutter will then be filtered out during the next stage of filtering.

Figure 3.5: Depicted here is the filtration process on a sample scene. 1) The rectified left stereo intensity image of the scene before stereo extraction. 2) The disparity image of the scene provided by stereo extraction. The dashed box represents the disparity region of interest. 3) Disparity region of interest segmented from the disparity image, showing points sampled. Here, orange points are considered valid and blue points are considered invalid. 4) A view of the scene data, looking down the Y axis, showing a frustum with a cube region of interest further segmenting points. Orange points inside the cube are considered valid, while all other blue points are invalid. Here, cube size is kept constant because there is no perspective scaling in 3D. 5) Valid subsampled points are stored in a list for later nearest neighbor processing.

3D Cube ROI

Once the disparity ROI is obtained, the data is then filtered further using another ROI, but this time it is performed in 3D. This cube ROI is again constant in size, but is in 3D. Because it is not affected by perspective scaling (as in the disparity ROI case), the cube ROI should segment the object of interest in all locations.

The size of this 3D ROI is chosen to enable the object of interest to move between frames and not move outside the cube ROI. The resulting 3D point data, which passes this filter, is not immediately stored but is streamed into the next subsampling filter stage.

3.4.2 Subsampling Valid Points

When processing with bandwidths that exceed the system limits, subsampling is necessary to reduce the total amount of data. Because a high percentage of the data provided to the tracking system may be redundant, subsampling can effectively decrease the required bandwidth. Two methods were considered for subsampling the incoming stereo extracted disparity data for this project: statistical subsampling and uniform subsampling.

Statistical Subsampling

One method to solve the filtering task of subsampling valid points involves starting with all of the data and selecting the first 512 valid points. Then, with probability P, points in the incoming disparity stream are checked for validity and overwrite existing 512 values. The probability P is derived from the previous frame's validity statistics and is chosen so that the data is sampled as evenly as possible. When a valid point

is found, one of the initial 512 points is selected for replacement by the new point.

Statistical filtering enables streaming of data so that no frame buffering is required. Because the subsampling filter always selects the first 512 valid points, it guarantees the selection of at least 512 valid points—if they exist. When there are fewer than 512 valid points in the region, the filter cannot guarantee the return of 512 valid points, unless a mechanism for repeating points is introduced, which requires in turn memory. In order to randomly overwrite the initial 512 points, a simple random number generator can be used for deciding which point to reject.

A weakness of the statistical subsampling method is its inability to guarantee 512 points. In addition, this method requires the valid point statistics from the previous frame, which may not always be a good estimate for the current frame. Therefore, a simpler and more robust sampling method was used.

Uniform Subsampling

Uniform subsampling is performed using an iterator to cycle completely through all points in the disparity ROI before a new cycle begins. For every line in the ROI, the iterator (which selects the disparity to potentially sample) increments through the line by a set constant. Once the line is complete, a new line is chosen by incrementing by the same constant. This technique may result in rolling over the last line and starting from the beginning of the disparity image on a new line, as shown in Figure 3.6.

Because there is a buffer for the disparity ROI (which allows this process to be repeated), this method is always able to guarantee the selection of 512 points, as long as at least one valid point exists in the ROI. It is important to maintain 512 (a power of 2) point samples so subsequent centroid calculations can be performed using simple

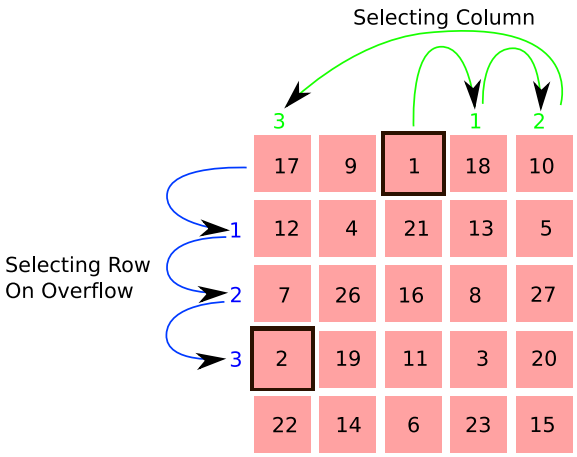


Figure 3.6: This depicts the subsampling method used by the tracker. In this example, the disparity region of interest is 5 by 5 pixels (shown as pink squares). An iterator with constant incrementing value of 3 is used. The black numbers indicate the sampling order. The first two selected samples are bordered in black. The first sample #1 has been picked arbitrarily. Then the iterator proceeds in the row from left to right with the constant increment of 3, until it overflows the number of pixels in the line. Once this overflow occurs, the iterator moves down 3 rows (the same constant increment of 3). The overflow amount is then used to index in the new row. The method for calculating the second pixel #2 is indicated with blue and green numbering.

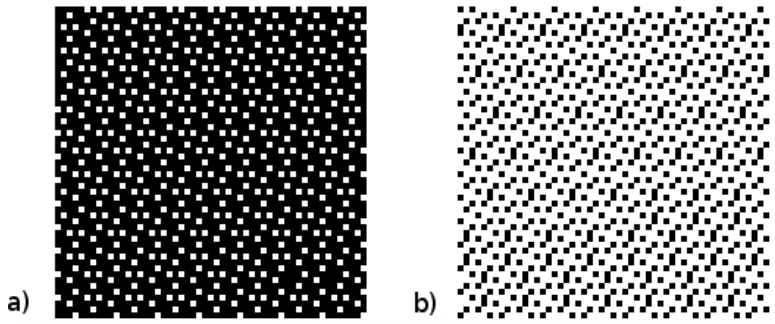


Figure 3.7: Example software-generated output of uniform subsampling circuit. These patterns show the selected points using uniform subsampling with various incrementing constants. Black signifies a sampled pixel. a) Using an indexing constant of 7 while sampling 512 points with a ROI of 53 by 53 pixels. b) Using an indexing constant of 7 while sampling 2048 points with a ROI of 53 by 53 pixels.

bit truncation. Uniform subsampling provides a good overall sampling of the object and is simple to implement in hardware using counters.

The results of software and hardware simulations are shown in Figure 3.7. As expected, the output displays an even subsampling, which is consistent regardless of the number of points sampled. This consistency is useful for situations in which many invalid points exist.

Disparity data provided for initial tests is an ideal case with no outliers. The model is composed of the object depicted in Figure 3.8. In Figure 3.9, a hardware simulation has been performed and shows output with point verification. This test displays the correct output, with points sampled only in the expected regions on the object. It also indicates the correct depths expected.

In order to generate the disparity ROI bounds, a conversion from world coordinates to disparity image coordinates is necessary, as shown in Figure 3.10.

The possible location of the ROI is limited to the disparity image bounds. Figure 3.11

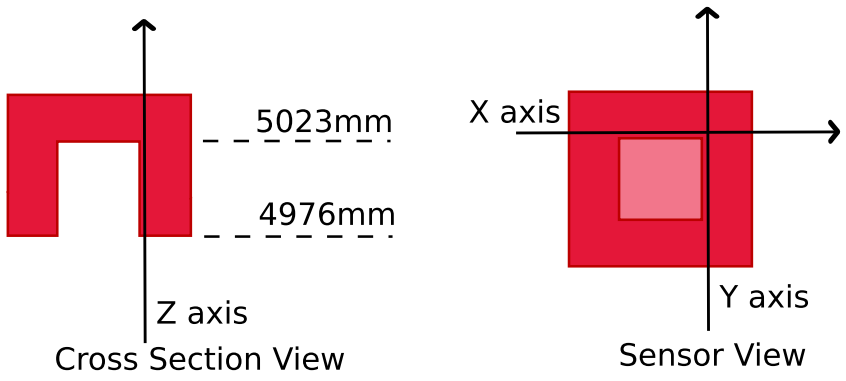


Figure 3.8: A sample object used for pre-filter simulation. On the left (view looking down y-axis) is a cross section of the object. On the right is the object viewed from the sensor vantage .

depicts the valid region that the disparity ROI may be in.

3.5 Data Transformation

During an ICP iteration, data is transformed using a fixed-point matrix multiplier module. This multiplier module has been designed for transforming X, Y, Z to X', Y', Z' in 3 cycles using a parallel circuit (see Figure 3.12).

During the calculation, all bits are preserved. In order to provide the most accurate results, they are only truncated at the final output stage. The total circuit consisting of three layers will consume nine block multipliers.

The accumulation of this transform is offloaded onto the Microblaze SVD calculation (which is described below). This offloading is done in order to perform the accumulation utilizing floating-point arithmetic to preserve accuracy. The number is then truncated and returned to fixed-point form to perform this data transformation.

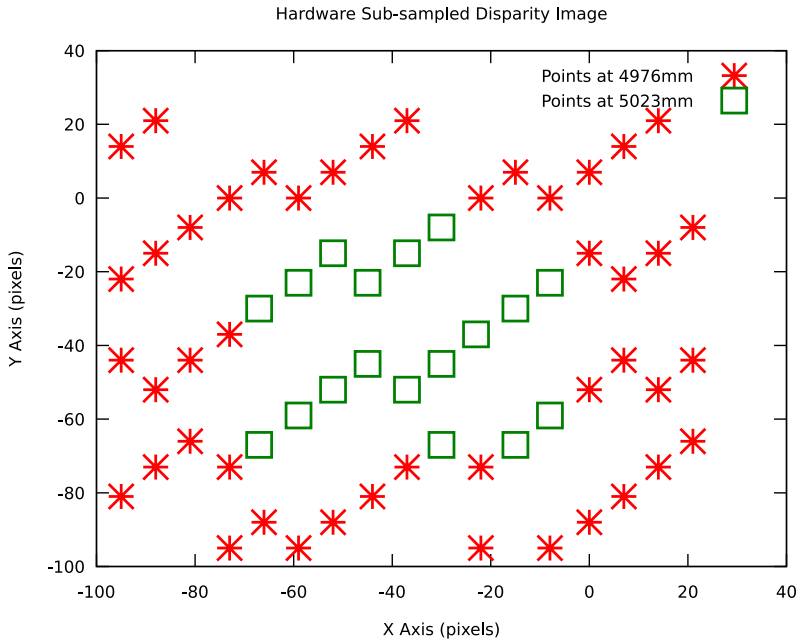


Figure 3.9: A simulated hardware implementation of the pre-filter stage with an sample object (which is shown in Figure 3.8). The two distinct levels are indicated by the two types of points, while the surrounding background has been filtered out. Squares represent object points subsampled at distance of 5023mm and x's represent object points subsampled at a distance of 4976mm.

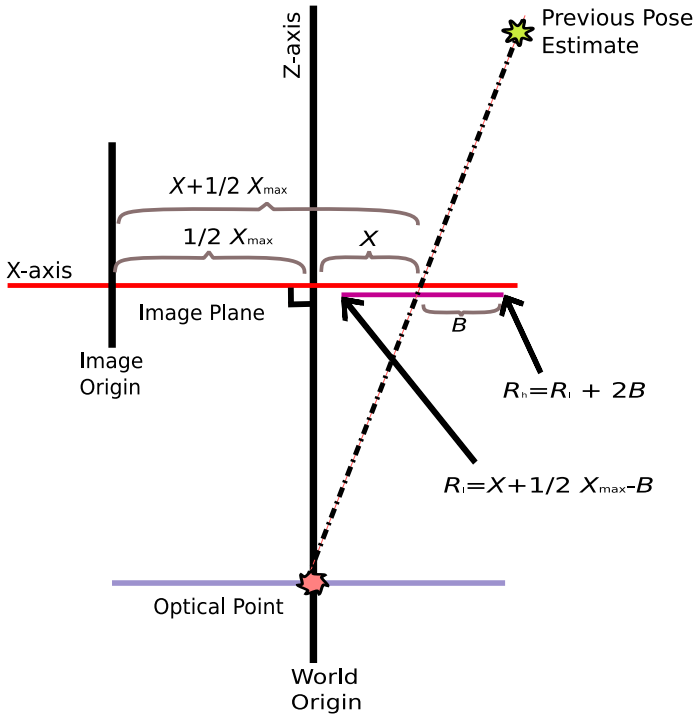


Figure 3.10: Converting the previous estimated object pose (only x-axis is considered in this example) to form the disparity ROI origin. The figure is a view looking into the y-axis. The ROI is used to filter out outlying points. X is the X-axis position of the object in world coordinates, projected onto the image plane. X_{max} is 640 pixels wide, B is half of the ROI size, R_l is the ROI's low boundary (the ROI's origin), R_h is the ROI's high boundary.

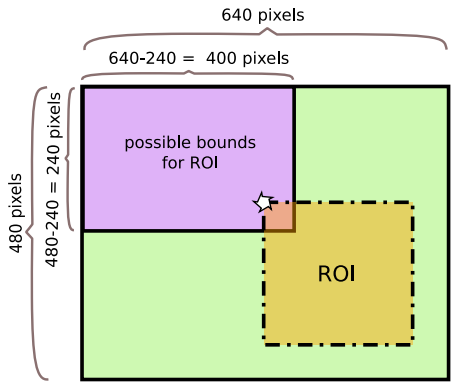


Figure 3.11: The possible disparity ROI bounds for a ROI size of 240 by 240 pixels with the top left corner as origin. This ROI is used to limit the amount of data required to be buffered before subsampling and point rejection.

3.6 Nearest Neighbor Search

Due to the high computational cost of the nearest neighbor search, considerable time was spent on designing this module. The design trade-off tended to be one of processing speed versus FPGA utilization.

The two tasks required for the nearest neighbor search were:

1. Compute the point to point distance: This distance metric, which is usually the Euclidean distance, can be computed in a pipelined fashion so that one point pair distance is computed every cycle.

2. Search for the closest point: This search uses the distance calculated to determine the minimum distance point pair.

These two tasks can be expressed in the following Equation with D_i the data point whose nearest neighbor is being sought, $\mathbf{M}$ be the set of all model points being searched, and P_i be the resulting point pair which will be composed of D_i and its

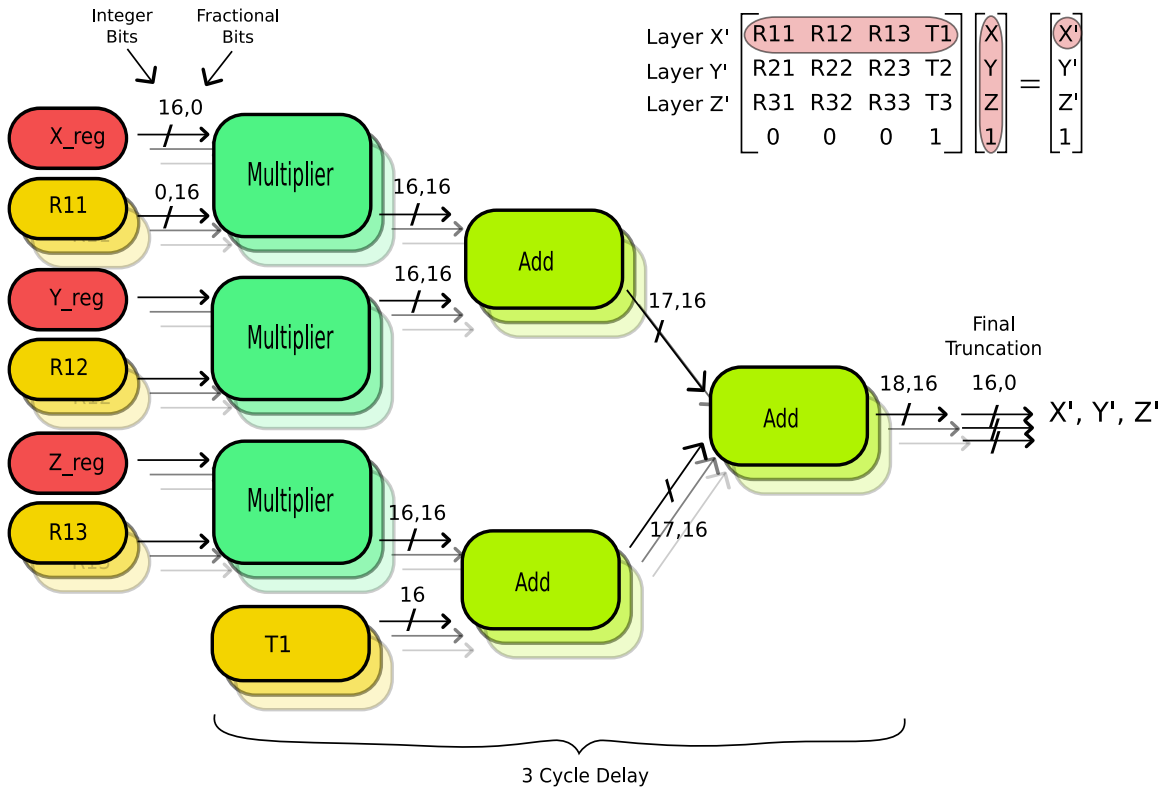


Figure 3.12: A depiction of the transform circuit. Transform circuit is composed of three identical layers which perform the matrix multiplication using fixed point math. Initial point input X, Y, Z is stored in X_reg, Y_reg, Z_reg in order to decouple the multiplier from the external circuits. Bit widths for both integer and fractional components are chosen so that no data loss occurs until the resulting truncation is performed.

nearest neighbor M_j

$$P_i = (D_i, M_j),\ j = \text{argmin } dist(D_i, M_k)\ \forall\ M_k \in \mathbf{M} \tag{3.1}$$

All of the nearest neighbor searches were performed using fixed-point representation in order to efficiently utilize the FPGA logic. Because of the difficulty (in terms of space and speed) to implement a divider in hardware, in most circumstances divisions were enforced as a power of two, and therefore could be performed by simple bit truncation. If this bit truncation was not possible, LUTs were used to implement the division.

3.6.1 Brute-Force Search

The most straightforward method to perform the nearest neighbor search is to tackle it in a sequential manner, usually referred to as brute-force. This brute-force approach, depicted in Figure 3.13, is the slowest processing method but it is simple to implement, requiring very low control overhead.

For every (scene) data point D_i, all model points $M_k \in \mathbf{M}$ are to be processed. Because point D_i is static for the search for the model point M_j, D_i is only required to be buffered and not stored in the brute-force search circuit.

For $N_d = 512$ data points, $N_m = 2048$ model points (this number of data and model points will be maintained throughout this chapter for comparison) and pipelining priming delay of $\epsilon_p = 4$ (see Figure 3.13 for an illustration of pipelining delay), it will take N_{cyc} cycles to complete the search:

$$N_{cyc} = \epsilon_p + N_d N_m \tag{3.2}$$

A circuit running at 125 MHz would be able to perform 14.9 fps nearest neighbor

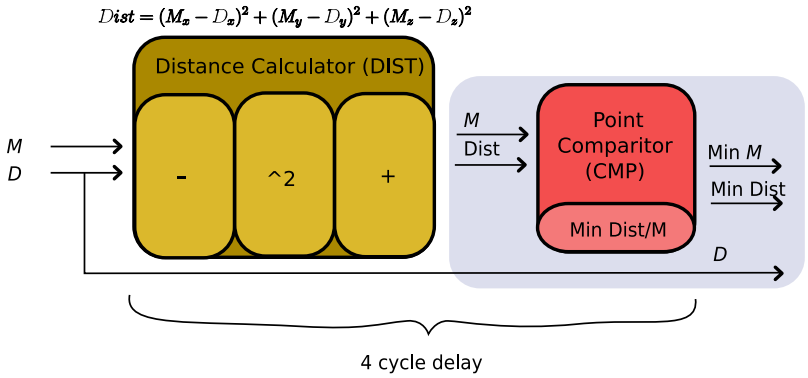


Figure 3.13: A depiction of the brute-force search. In order to find the nearest neighbor to data point D_i the circuit iterates through all model points $\mathbf{M}$. The distance between points is first calculated for every point to point pairing and is then followed by a comparison to the minimal distance found thus far. After the model $\mathbf{M}$ has been compared, the circuit will report the nearest neighbor point pair at its output. Because D_i never changes throughout operation, it is not passed through the circuit but is only buffered to be read at the output.

calculations using this approach. The benefit of the brute-force approach is that it is simple to implement and is guaranteed to find the closest point. It also uses minimal hardware resources. Therefore, it is ideal for situations in which hardware space is limited and high-speed is not a requirement.

Parallel Brute-Force

The parallel brute-force approach takes the brute-force circuit (previously described) and parallelizes it using multiple copies of hardware that search for multiple corresponding points simultaneously. By utilizing the fact that all data points are searched with the same model points, the different data points can be preloaded into registers. Then, the data points can iterate through all model points just as it would be done in the sequential brute-force search, but in parallel. Because it is still performing the

brute-force search on individual data points, finding the closest point is guaranteed.

A minimal amount of overhead is required for this circuit. This overhead is for preloading the data points and writing the resulting point-pair correspondences sequentially into memory. The number of cycles needed for one nearest neighbor search is:

$$N_{cyc} = 2\epsilon_d + \epsilon_p \frac{N_d}{\alpha} + N_m \frac{N_d}{\alpha} \tag{3.3}$$

$$= (2 + \frac{\epsilon_p}{\alpha} + \frac{N_m}{\alpha})N_d \tag{3.4}$$

where $\epsilon_d = N_d$ is the number data memory reads required by the system and $2\epsilon_d$ is used to account for data reads and pair writes (if data read write IO is performed before and after each parallel search). Also, $\epsilon_p \frac{N_d}{\alpha}$ represents the priming delay required for every group of nearest neighbor calculations when it is split into groups of size α. Here, α is an integer representing the number of parallel circuits and ϵ_p is the pipeline priming delay for the distance and comparison operation.

The parallel brute-force method results in a slight delay to read and write data every invocation of a brute-force search. This delay is due to the fact that all of the parallel units require their particular D_i to be loaded from memory at the same time. To prevent this contention, the parallel brute-force initially loads all parallel units in sequence. However, this creates some IO overhead of $2\epsilon_d$ for these reads and writes. Given a clock speed of 125 MHz, 8 nearest neighbor iterations per frame, $N_d = 512$, and $N_m = 2048$, the resulting execution rate will be 234.3 fps when $2^4 = 16$ parallel circuits are used.

Staggered Start

The staggered start aims to pipeline the IO operations in order to reduce the IO overhead. Each brute-force unit can load its data point D_i and start searching with a one-cycle delay. This delayed start enables the initial data to be read without the loading overhead. It also enables each point pair to be directly written once the nearest neighbor search is complete and the correspondences are found. Therefore, Equation 3.4 is reduced to

$$N_{cyc} = (\frac{\epsilon_p}{\alpha} + \frac{N_m}{\alpha})N_d + \alpha \tag{3.5}$$

where the number of parallel circuits α is added to account for staggered start delay.

For data sets with a relatively larger number of model points (compared to the number of data points), this method will not be worthwhile, as the IO delays $2 + \frac{\epsilon_p}{\alpha}$ will become insignificant compared to the processing time of the nearest neighbor, see Equation 3.4. Given a clock speed of 125 MHz, 8 nearest neighbor iterations per frame, $N_d = 512$ and $N_m = 2048$, the resulting execution rate will be 237.9 fps when $2^4 = 16$ parallel circuits are used.

3.6.2 Binary Tree Search

A binary tree is a data structure that can speed up the nearest neighbor search, but its performance is IO-bounded. The binary tree is common in hardware such as the fanout tree [34] (although a fanout tree is not necessary binary). The fanout tree is used for propagating clocking and other signals evenly throughout a digital design. This type of structure is quite simple to implement in hardware using recursive coding styles.

The binary search tree is the complimentary structure to the fanout tree. The binary search tree is used to compare S signals at the input leaf nodes of the tree. On every layer, the number of results is divided by two, until a single solution is obtained (minimum or maximum). For a binary tree with S inputs, we need a total of $S-1$ comparators. To fully utilize this type of structure through pipelining, every layer of the tree must be stored in registers. Figure 3.14 provides a binary tree example.

The number of cycles required for a binary tree to complete is given by

$$N_{cyc} = \epsilon_m + \epsilon_p + \log_2 N_m + (N_d - 1) \tag{3.6}$$

where $\epsilon_m = N_m$ is the number of model memory reads required by the system, $\epsilon_p = 3$ is the priming delay for the distance calculation, and $\log_2 N_m$ is the priming delay for the binary tree. For the same number of model and data points and nearest neighbor iterations as in the previous searches, this circuit will operate at 6070.3 fps. Although the speed of the binary tree search is much higher than the speed of the brute-force search, it requires an enormous amount of hardware resources (for the binary tree and pipelining buffers) and a greater IO bandwidth (for initial loading of model points). In fact, this design will not fit into the selected FPGA, the Xilinx XC2VP100.

Initially, in order to run the binary tree, the point-pair distances must be calculated so that these distances can be compared using the binary tree. These Euclidean distances require the use of multipliers, which are a limited resource on an FPGA. A circuit to find the nearest neighbor for N_m points requires $3N_m$ multipliers. For example, if $N_m = 2048$ model points must be searched, $3 \times 2048 = 6144$ multipliers are required, which is far more than the 444 multipliers that are available. Therefore, the largest number of points that can be processed at one time is 148 model points,

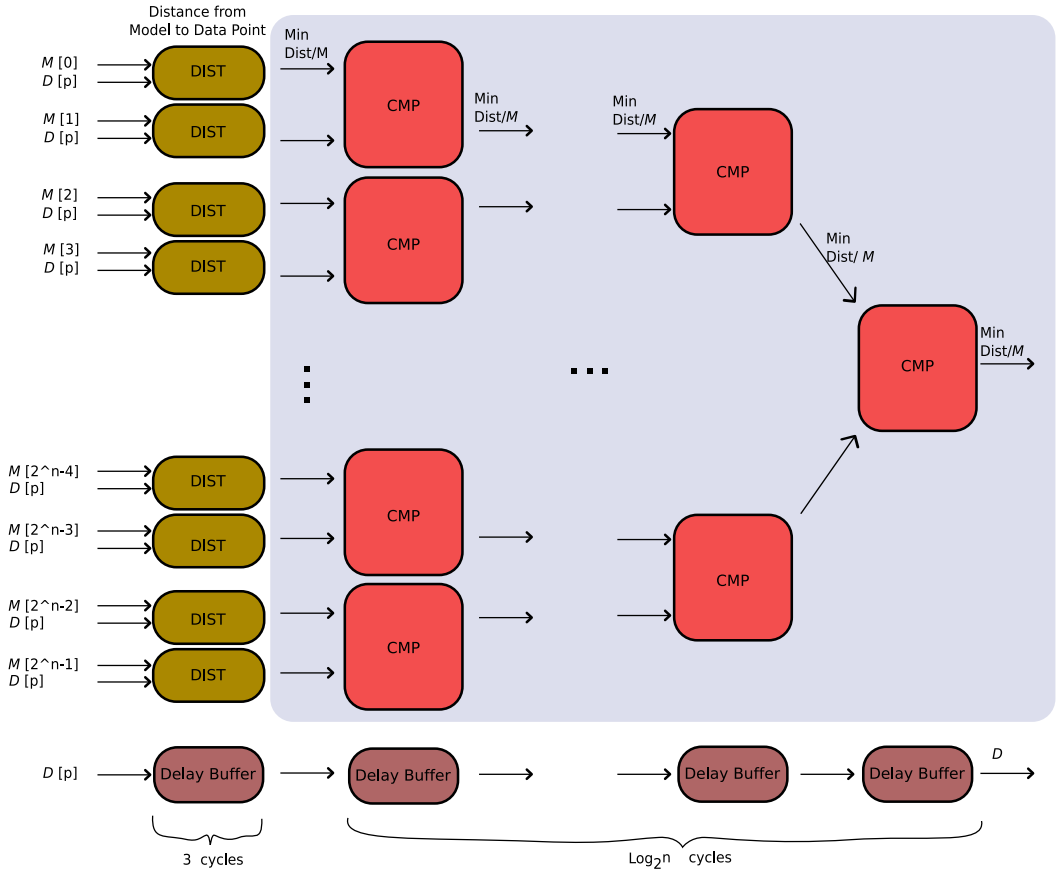


Figure 3.14: This binary tree structure enables rapid determination of the minimal point pair by halving the possibilities at each depth. All inputs are required before the tree processes the data. Initially, the inputs to the tree are pre-processed to calculate each point pair distance. These distances are propagated through the tree and used by each node to decide which point pair and corresponding distance should be forwarded. After N cycles, all N layers have completed their processing and the minimum distance point pair is presented at the output.

assuming that no other logic is present on the FPGA. In this application, some multipliers are also required for other circuits, hence such an approach is not desirable. Even if Euclidean distances are approximated without multipliers, there will still be a considerable amount of logic consumed.

In addition to the considerable amount of hardware, there is a great deal of preparation required for reading of the model points because of the memory IO limitations. Every model point must first be loaded into memory before running the search tree. This loading step is performed sequentially.

Flat Binary Tree

The flat binary tree is similar to the binary tree, but there are no layers (depths) of nodes. The data is propagated along one layer until only one minimum is left (see Figure 3.15).

For S inputs, $S/2 - 1$ comparators are required. This architecture requires fewer hardware resources than the binary tree because the flat binary tree reuses the first layer of nodes. As a result, the flat binary tree uses approximately 50 percent fewer comparators then the binary tree. Again, IO will limit the effectiveness of the flat binary tree's operation, as all of the input data must be loaded before the tree can start processing.

In terms of pipelining, the flat binary tree would require more complex control. This control is required to initially fill the structure and then resupply the lower half of the tree with the remainder of the data (see arrows in Figure 3.15). The number

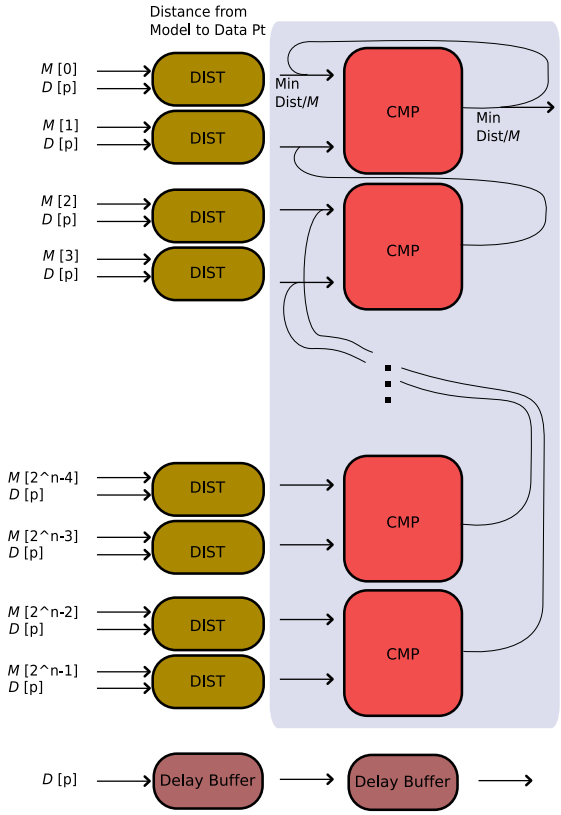


Figure 3.15: The flat binary tree to determine the nearest neighbor. During every cycle the tree shifts its contents and again determines the minimal point pairs. The shifting of the minimal point pairs continues until only one pair remains. This structure decreases the number of comparators by $\sim$50%.

of cycles required for this search (no pipelining) is given by the following equation

$$N_{cyc} = \epsilon_m + \epsilon_p N_d + N_d \log_2 N_m \tag{3.7}$$

$$= \epsilon_m + (\epsilon_p + \log_2 N_m) N_d \tag{3.8}$$

Therefore, the flat binary tree could provide 508.6 fps; however, the flat binary tree will still require considerably more logic than is available in the FPGA.

3.6.3 Low-to-High Search

Because of the simplicity of the coarse to fine search method by Jost et al., the application of this search was investigated for solving the hardware nearest neighbor problem. From these investigations a novel hardware based search technique 'low-to-high' search was developed. The search utilizes a clustered pre-processed model to speed up the nearest neighbor.

The main idea of this approach is to pre-segment the model around clustering points before runtime. These clustering points will contain neighboring points to be further searched to obtain the nearest neighbor. This method differs from the coarse to fine method of Jost et al. [27]. This low-to-high search only searches partial higher-resolution models. During runtime, a point is compared to the small number (low-resolution model) of clustering points to find its nearest neighbor. The nearest neighbor clustering point is then used to select the surrounding neighborhood for a higher-resolution search (see Figure 3.16).

For example, if the search involves 2048 points, the model could be preprocessed into 32 clustering points and then 64 neighborhood points ($32 \times 64 = 2048$). This case only requires a search of 96 points, which is only $2048/96 = 21.3$ percent of the original brute-force search expense. This result shows a substantial reduction in the

Figure 3.16: An example of the low-to-high search. Yellow points are the clustering points for the low-resolution model. Blue points are the points to be used as the high-resolution model; they are associated with the nearest clustering point. a) shows points which are to be used to build the low-to-high-resolution model. b) shows evenly spread clustering points selected as the low-resolution model. Also shown here in red is one arbitrarily selected data point being compared to the low-resolution model. The arrow indicates the nearest cluster neighbor. c) shows the nearest cluster neighbor and its surrounding model points. It also shows the nearest neighbor of these model points.

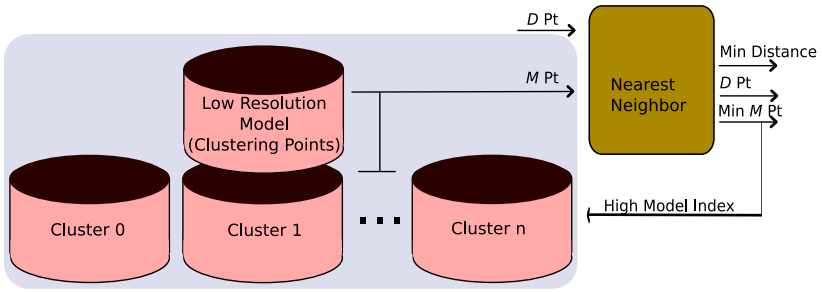


Figure 3.17: A high level view of the low-to-high nearest neighbor circuit. This figure depicts two levels of resolution. The first level will always consist of one model memory (which holds the low-resolution clustering points). This memory is searched and then provides its result to index through the second level of model memory (which holds the clustered high-resolution points). The second model memory is then search for the nearest neighbor. This does not guarantee the true nearest neighbor—only an approximation.

number of searched elements.

Due to the simplistic nature of the low-to-high search, the brute-force hardware can be utilized to perform its operations. The initial low-resolution search functions by searching the data with a low-resolution model using brute-force. Once this low-resolution search is complete, the low-resolution solution can be used to index the selection of the high-resolution model cluster, as explained in Figures 3.16 and 3.17.

Although this search enables a drastic increase in performance (hardware size and speed), it may not always provide the true nearest neighbor. Due to the robustness of the ICP algorithm to noise, this approach does not cause much of a problem in practise. As previous work has established, some percentage of incorrect correspondences do not significantly affect the overall convergence of the ICP algorithm. Indeed, in some cases, small incorrect correspondences may actually accelerate convergence [25].

This low-to-high technique permits the utilization of larger models while using fewer processing resources. The number of cycles required for a low-to-high search with σ model levels is given by:

$$N_{cyc} = \sigma\epsilon_p + \sigma\lceil \sqrt[\sigma]{N_m} \rceil N_d \tag{3.9}$$

$$= \sigma \left(\epsilon_p + \lceil \sqrt[\sigma]{N_m} \rceil N_d \right). \tag{3.10}$$

This equation gives the optimal cycle count when the number of points per level is equal.

A proof that $\sqrt{N_m}$ points is the optimal number per level for a two-level model is as follows (where L is the number of low-resolution model points and H is the number of high-resolution points in a cluster):

Proof.

$$N_{cyc} = \epsilon_p + LN_d + \epsilon_p + HN_d \tag{3.11}$$

$$= 2\epsilon_p + N_d(L + H) \tag{3.12}$$

We are trying to minimize N_{cyc} so that $LH = N_m$ or $H = \frac{N_m}{L}$. Substituting H gives:

$$N_{cyc} = 2\epsilon_p + N_d(L + \frac{N_m}{L}) \tag{3.13}$$

Because we wish to minimize N_{cyc} where $2\epsilon_p$ and N_d are constants we have:

$$\frac{dN_{cyc}}{dL} = 1 - \frac{N_m}{L^2} \tag{3.14}$$

and

$$\frac{ddN_{cyc}}{ddL} = 2\frac{N_m}{L^3} \tag{3.15}$$

which indicates that we have found the minimum.

To minimize we have:

$$\frac{dN_{cyc}}{dL} = \quad 0 = 1 - \frac{N_m}{L^2} \tag{3.16}$$

$$L = \sqrt{N_m} = H \tag{3.17}$$

□

For a two-level model with 32 points (for the low-resolution) and 64 points (for the high-resolution clusters), the circuit can operate at 317.8 fps, using only one brute-force nearest neighbor calculator with a slightly more complex model (consisting of a low-resolution clustering points and higher-resolution clusters). This uneven level size is not the optimal setting of $\sqrt{N_m}$ points per level; but, is the nearest integer alternative that result in $LH = N_m$. If $\sigma = 1$, the equation reverts back to the standard brute-force calculation, as in Equation 3.2.

3.6.4 Distance Metrics Optimization

Different distance metric optimizations have been considered. The true Euclidean distance D can be calculated as:

$$D = \sqrt{x^2 + y^2 + z^2}. \tag{3.18}$$

Because only the relative magnitude of the distances is important, this calculation can be simplified by removing the square root operation:

$$D = x^2 + y^2 + z^2 \tag{3.19}$$

This simpler equation can be readily implemented on the current FPGA utilizing on-chip multiplier blocks.

An efficient approximation to the Euclidean distance described by [35] is as follows:

$$D = Max + \frac{1}{4}Med + \frac{1}{4} \tag{3.20}$$

Where Max, Med and Min are defined by the maximum, median, and minimum ranked x, y, z components of the difference vector. This equation enables a hardware solution that does not require any multipliers. Only bit truncation, addition logic and comparators are required, which makes it quite efficient to execute. This approximation is accurate to within ± 13 percent of the true value [35].

3.6.5 Rejecting Points: Post Nearest Neighbor Processing

Point rejection can be performed after the nearest neighbor calculation. This point rejection can be used to remove outliers using the distance calculated from the nearest neighbor stage.

For this implementation, point rejection is not performed after the nearest neighbor calculation because it is beneficial to maintain the number of point pairs as a power of 2 (as seen in Section 3.4.2). If the point pairs are not left as a power of 2, then bit truncation will be unable to perform division, and LUTs or dividers would be required to form the $\mathcal{M}$ matrix that is discussed in the following section on singular value decomposition (SVD).

3.7 Transform Recovery

The transform recovery component has been implemented utilizing a combined hardware-software technique.

Because the nearest neighbor search generates a considerable number of point

pairs, it is beneficial to perform data reduction on hardware before utilizing the SVD function in software. These point pairs must be processed to form the $\mathcal{M}$ and $\mathcal{N}$ matrix, as described in [36]. The $\mathcal{M}$ matrix is as follows:

$$\mathcal{M} = \begin{bmatrix} S_{xx} & S_{xy} & S_{xz} \\ S_{yx} & S_{yy} & S_{yz} \\ S_{zx} & S_{zy} & S_{zz} \end{bmatrix} \tag{3.21}$$

with

$$S_{xx} = \sum_{i=1}^{n} \hat{\vec{p}}_{x,i} \hat{\vec{q}}_{x,i} \tag{3.22}$$

(where $\hat{\vec{p}}$ and $\hat{\vec{q}}$ are defined later in Equation 3.26)

$$S_{xy} = \sum_{i=1}^{n} \hat{\vec{p}}_{x,i} \hat{\vec{q}}_{y,i} \tag{3.23}$$

and so on.

The $\mathcal{N}$ matrix is as follows:

$$\mathcal{N} = \begin{bmatrix} S_{xx}+S_{yy}+S_{zz} & S_{yz}-S_{zy} & S_{zx}-S_{xz} & S_{xy}-S_{yx} \\ S_{yz}-S_{zy} & S_{xx}-S_{yy}-S_{zz} & S_{xy}+S_{yx} & S_{zx}+S_{xz} \\ S_{zx}-S_{xz} & S_{xy}+S_{yx} & -S_{xx}+S_{yy}-S_{zz} & S_{yz}+S_{zy} \\ S_{xy}-S_{yx} & S_{zx}+S_{xz} & S_{yz}+S_{zy} & -S_{xx}-S_{yy}-S_{zz} \end{bmatrix} \tag{3.24}$$

The generation of the $\mathcal{M}$ matrix is a repetitive task that generates all of the required data to be processed by the SVD. Because of the repetitive nature, hardware was used to form this matrix. The calculation of this matrix relies on first calculating the centroids of both the model and data points in the point pairs. This calculation requires a division by the number of point pairs and is simplified by enforcing the number of point pairs to be exactly a power of 2. As a result, simple bit truncation

is now able to determine this division.

Once this matrix is derived, it is then sent for processing on a Microblaze microprocessor. The Microblaze decomposes the provided matrix, using SVD to determine the registration transform between the two point sets. All calculations inside the microcontroller are performed with floating-point arithmetic to achieve accurate results. This registration transform is then accumulated within the microcontroller so that floating-point arithmetic can be used. Once the registration is complete, the estimated transform is transferred back to the hardware for the Filtering and Transform modules.

3.8 Transform Recovery - Streaming Optimized

Another transform recovery approach to generate the $\mathcal{M}$ matrix and centroids was considered. This approach could be performed in one step, so that data streaming could be utilized.

For a set of n points $\{\vec{p}_i\}_{i=1}^{n}$, the centroid is calculated as:

$$\vec{c} = \frac{1}{n}\sum_{i=1}^{n}\vec{p}_i \tag{3.25}$$

The usual approach to form the $\mathcal{M}$ matrix is to initially center the data and model points around their centroids before further calculations:

$$\hat{\vec{p}} = \vec{p} - \vec{c} \tag{3.26}$$

where $\hat{\vec{p}}$ represents a data point translated by a centroid.

After the point sets are centered, the $\mathcal{M}$ matrix elements are calculated as follows:

$$S_{ab} = \sum_{i=1}^{n}\hat{\vec{p}}_{a,i}\hat{\vec{q}}_{b,i} \tag{3.27}$$

where $\vec{q}$ represents a model point and $a, b \in x, y, z$ specifies the components being multiplied; for example S_{xx}, S_{xy},...,S_{zz}.

Rather than centering the points with the centroid prior to the computation of the $\mathcal{M}$ matrix, the centering can be performed afterwards, as follows:

$$S_{ab} = \sum_{i=1}^{n} \hat{\vec{p}}_{a,i} \hat{\vec{q}}_{b,i} \tag{3.28}$$

$$= \sum_{i=1}^{n} (\vec{p}_{a,i} - \vec{d}_a)(\vec{q}_{b,i} - \vec{m}_b) \tag{3.29}$$

$$= \sum_{i=1}^{n} \vec{p}_{a,i}\vec{q}_{b,i} \; + \; n\vec{d}_a\vec{m}_b \; - \; \vec{d}_a \sum_{i=1}^{n} \vec{q}_{b,i} \; - \; \vec{m}_b \sum_{i=1}^{n} \vec{p}_{a,i} \tag{3.30}$$

where $\vec{d}$ and $\vec{m}$ are the centroids for the data and model point sets, respectively.

This calculation results in the need to have 9 multipliers and accumulators to calculate:

$$\sum_{i=1}^{n} \vec{p}_{a,i}\vec{q}_{b,i} \tag{3.31}$$

for the 9 elements of the $\mathcal{M}$ matrix, and six accumulators for

$$\sum_{i=1}^{n} \vec{p}_{a,i} \tag{3.32}$$

and

$$\sum_{i=1}^{n} \vec{q}_{b,i} \tag{3.33}$$

for $a, b \in x, y, z$.

All these accumulations can be performed in hardware. In total, this circuit requires 15 accumulators. Once these hardware accumulations are finished, they can then be sent to a microcontroller to be combined to produce Equation 3.30. The centroids $\vec{d}$ and $\vec{m}$ can then be computed in software using Equations 3.32 to 3.34

along with the number of point pairs. For example:

$$\vec{d} = \frac{1}{n}\sum_{i=1}^{n}\vec{p}_i \tag{3.34}$$

The benefits to this approach are explained as follows. The centroid calculations are now performed in parallel with all other hardware accumulations; therefore, only one cycle through the point pairs is required. This enables the circuit to stream the point pairs and does not require the point pair data to be buffered. Buffering would require extra BRAM units. Furthermore, because the centroid divisions are now in software, these divisions can be performed using the Microblaze's ALU. This software division approach enables the processing of point pairs, without additional division units in hardware. Therefore, a none power of two point pairs can be processed with the software ALU divider.

3.8.1 Interface to Microblaze

The interface with the Microblaze micro-controller (for SVD calculation and debugging) makes use of the Fast Simplex Link (FSL) for data transfers. The FSL is a bidirectional FIFO buffer used to provide fast one-cycle asynchronous communication. Because the Micoblaze IO bandwidth is limited to 32 bits per transfer, data reduction is performed by reducing the point pair data into the symmetric $\mathcal{N}$ matrix. As a result, only 8 elements (one half of the symmetric $\mathcal{N}$ matrix) must be transferred. The resulting accumulated transform is then transferred back through the FSL so that it is available to the Pre-filter and Transform components.

Chapter 4

Experimental Results

In this chapter, the performance of the hardware tracker is characterized using a variety of testing techniques. Not only is the implementation of the tracker's algorithm tested, but the external components relating to the data capture and stereo extraction, which are used during experimentation, are also tested. External components significantly affect the performance of the system; therefore, it is useful to understand their influences.

4.1 Experimental Setup and Overview

As indicated in the introduction, this tracker was developed concurrently within a larger project called the 'FastTrack' stereo vision tracking system. The external custom components are still in development by the partner organizations at the time of this writing and therefore were unavailable for testing. As a result, during testing of this work, these components were replaced with available commodity components. The replacements were chosen to meet the specifications of the components under

development as closely as possible.

4.1.1 Capturing and Processing of Tracking Data

Because of the limited bandwidth between the substituted external components and the FPGA, the data cannot be supplied to the tracker in realtime. Also, the limited memory of the FPGA that is used for the tracking cannot store the entire sequence of frames required for testing. To overcome this, each frame is sent to the FPGA and is then processed sequentially.

At the beginning of the tracking process, the user must first reset and then transmit the following parameters to the FPGA:

- Number of ICP iterations.
- Cube ROI size.
- Initial pose of the object (consisting of the x, y, z position and the x-, y-, and z-axis rotations).

Once these parameters have been set, the model and disparity frames are transmitted, one at a time, over the serial port for processing. After each frame has been sent to the hardware (which operates at 125.0 MHz), the hardware begins processing to recover the pose of the object. The hardware immediately returns the object's pose through the serial port to the PC.

4.1.2 Stereo Sensors

The stereo vision hardware used for testing was the Point Grey Research Bumblebee stereo vision sensor. The maximum acquisition rate of this sensor was 30 fps

in comparision to the FastTrack sensor at 200 fps (under development by Ryerson University). It was further limited by a serial IO channel connected to the FPGA. Even though the sensor was limited by this low bandwidth, the processing of data frames was still performed at full speed, once a frame was received by the FPGA hardware.

The Bumblebee sensor has a 0.12 m baseline, whereas the FastTrack sensor has a 0.16 m baseline. Due to this difference, the Bumblebee sensor is somewhat less accurate at estimating depths further away from the sensor. However, it performs better at determining closer depths.

The stereo extraction was performed with the default Bumblebee software settings, with the exception of the number of calculated disparities. The designed number of disparity levels for the FastTrack sensor is 128. As a result, the Bumblebee's stereo extraction parameters were adjusted to provide this number of disparities.

4.1.3 Path Generation

For path testing, four types of paths were used: Stationary, Linear, Rotational, and Free-Form.

Stationary Path

In the stationary path, the objects were held stationary to determine the error associated with tracking a static object. This is useful because it identifies the characteristics of the system for the simplest case.

Linear Path

To create a linear path that can be easily registered to a line (to determine the path's quality), a robotic arm was used to manipulate the object. The object was placed in consecutive positions along a linear path and an image frame was acquired at each distinct position. The robotic arm and object were completely stationary while each frame was being acquired by the sensor.

Rotational Path

The rotational path test was similar to the linear path test. The object was fixed to a robotic arm. The arm was then instructed to rotate at its hip joint, in small increments, for stereo imaging. The generated movement of the object was rotational, but also contained a translational component. This translational component was caused by the robot's hip axis not coinciding with the object's centroid. The resulting rotation was extracted by determining the axis of rotation. The translational component was ignored.

Free-Form Path

Finally, hand held movements of the objects were performed to ensure proper tracking of free-form paths.

4.1.4 Objects of Interest

Three objects of interest were chosen for the tracker experiments, as illustrated in Figure 4.1. All three objects had their own unique characteristics that were used to test various aspects of the tracker.

Figure 4.1: The three objects used for tracking tests are shown. On the left is the Angel model, in the middle the Big Bird model, and on the right the Cube model.

1. **Angel:** The angel model was selected because of its free-form shape, including self-occlusion. Because of the model's small size (190 mm $\times$ 160 mm $\times$ 250 mm), it could not produce as many disparity values during stereo extraction. To account for this, the object was moved closer to the sensor in order to fill a greater field of view.

2. **Big Bird:** The Big Bird model was selected because of its relatively large size (410 mm $\times$ 310 mm $\times$ 230 mm) and descriptive shape. An interesting element of this model was that it was not spherical in nature. Its elongated shape made it difficult to segment from clutter using either a sphere or cube of interest. In addition, this model also had self-occlusion.

3. **Cube:** The cube model (180 mm × 180 mm × 180 mm) was chosen for initial testing because of its simple geometry that allowed for simple construction of both the physical and computer models. This model had six different textures applied to the six sides to make its planar sides easier to sense with stereo vision. Unfortunately, because of the reflective nature of the photographic paper used for the textures, the object was quite susceptible to specularities. The cube model also had very few descriptive 3D features. Therefore, it was best tracked when three faces were visible.

4.1.5 Experimental Errors

During testing, there were three types of errors associated with the tracked path:

- The robotic positioning error.
- The sensor and stereo extraction error.
- The tracker's algorithmic error.

The robotic positioning errors are considered to be much smaller than the combined sensor and stereo extraction errors. Because the robotic positioning error is considered minimal, it has been ignored.

4.2 External Components - Testing of Camera and Stereo Extraction

In order to evaluate the tracker's behaviour, both the camera and the stereo extraction had to be tested and characterized.

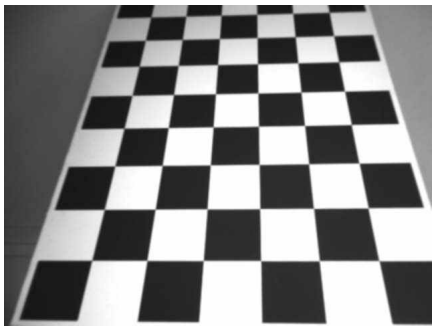


Figure 4.2: The rectified right image from the Bumblebee sensor shows the calibration pattern setup for the sensor and stereo extraction test. The pattern was placed at approximately a 45 degree angle to the camera so that various depths could be extracted. As seen in this image, the Bumblebee camera only provides black and white imagery.

4.2.1 Calibration Pattern - Selecting All Points

One calibration pattern test was utilized to provide an indication of the quality of the stereo extraction method in a controlled situation. A calibration pattern with a 7×9 grid of calibration squares, with each square measuring 0.15 m by 0.15 m, was placed in front of the stereo vision sensor. The calibration pattern was positioned at an approximately 45 degrees so that the different disparities that were to be used during the tracking could be measured.

Figure 4.2 shows the right image used for the stereo extraction process, and Figure 4.3 shows the resulting disparity image. In the disparity image, there are many false disparities that can potentially disrupt tracking. These false disparities are caused by the untextured regions of the calibration pattern, being matched incorrectly with other untextured regions. The calibration pattern has many of these regions, which are a potential source of error.

The stereo extraction algorithm utilizes similarity measurements between the

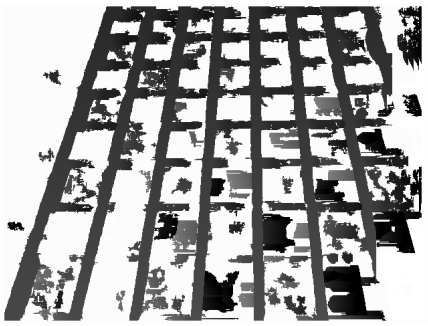


Figure 4.3: This shows the resulting disparity image of the calibration test pattern positioned on a 45 degree angle. In this image, white indicates no left/right image correspondences. For the remaining pixels, the darker the pixel, the farther the object is away from the camera.

stereo images along similar scanlines. If the scanline has no discriminating information, such as a constant horizontal line, then the disparity estimates will be inaccurate for that line.

Although there is a considerable amount of noise in the disparity image, the plane can still be clearly seen. This disparity image of the calibration pattern was then converted to x, y, z point data and registered to an infinite plane. The RMS error and standard deviation were calculated to see how well the sensor and stereo extraction algorithms operate. Results of this test gave a 58.05 mm average error and a 79.07 mm standard deviation, indicating that the data is noisy and unsettled. This provides an indication of how inaccurate this commercial sensor and stereo extraction can be.

4.2.2 Calibration Pattern - Selecting Ideal Points

The second calibration pattern test utilized the same disparity images from the previous test, but with one major difference. In this test, the data was manually filtered to

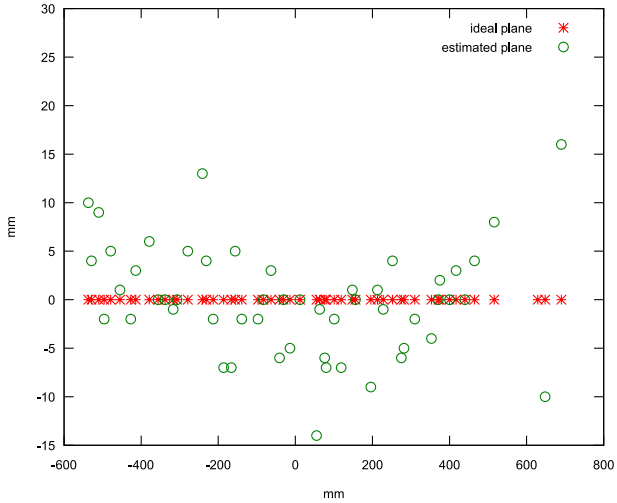


Figure 4.4: The results of registering 54 manually sampled points from a disparity image of a calibration pattern to a plane. The view is looking along the edge of the ideal plane indicated in red stars.

extract those pixels that were most likely valid disparities. These pixels were sampled from the intersection in the grid of the calibration pattern because they contained the greatest number of high frequency details that were most descriptive (see Figure 4.2).

After extracting 54 of the intersection disparities, these disparities were converted to x, y, z point data. The point data was then registered to an infinite plane so that the RMS error and standard deviation could once again be calculated. Results for the average displacement error for all points to the plane were 4.50 mm, with a standard deviation of 4.84 mm.

Figure 4.4 shows a view looking along one side of the plane. From this view, a trend is evident where the sensor and stereo extraction phase cause the depth to decrease closer to the edges of the stereo image. This is most likely caused by radial distortion that has not been adequately corrected in the Bumblebee camera rectification phase.

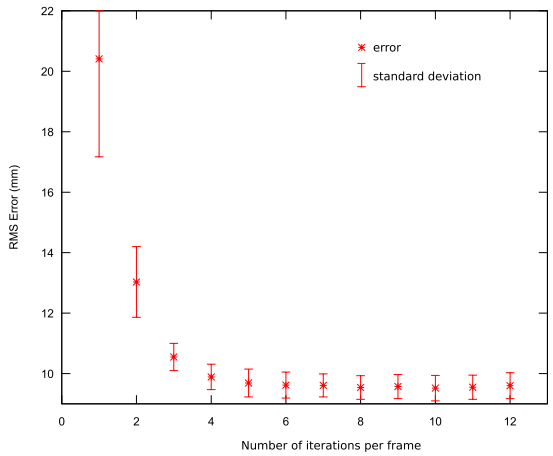


Figure 4.5: A graph that shows the effect of increasing the number of iterations on a sequence of 100 frames. The sequence used here was the cube model moving on a linear path controlled by a robotic arm. The graph indicates both the average RMS error in mm as well as the standard deviation of the error for each run.

These distortion effects will be evident later with the linearly tracked paths in Section 4.4.2

4.3 ICP Iterations

To determine the number of iterations required for the tracking algorithm to settle to a local minimum, an ICP iteration test was performed. A sequence of 100 consecutive frames of an object moving linearly was tracked using a software implementation of ICP. The number of iterations was varied from 1 to 12 iterations per frame. The results are shown in Figure 4.5.

As expected, a decreasing curve was obtained for the iteration test. However, as the number of iterations increased, the error did not always decrease monotonically.

These variations can be expected due to the varying stream of frames where slight disparity differences can change the RMS.

For convergence to a good path estimate, with an RMS error of under 10 mm, the tracker only required approximately four iterations. Because other paths may have a longer convergence time (due to further displacements in rotation and translation from frame to frame), eight iterations were selected for the remainder of the tests in this section. In addition, to emphasize that a good registration is achieved in four iterations, the standard deviation can be seen to remain consistent after four iterations. In fact, the standard deviation settles to under 0.45 mm within three iterations per frame.

4.4 Tracker Testing - Quantitative

The recovered paths of an object's movement were analyzed quantitatively in these experiments. The ground truth paths (paths with known reference frames) of the object's movement were unavailable, so only relative paths (paths having no known reference frames) were taken into consideration. Paths with simple geometry were chosen so that they could be easily aligned and compared to the extracted paths, using least-mean-squares fitting.

The three quantitative tests are stationary path, linear path, and rotational path. They were performed using both the hardware and software versions of the ICP algorithm. The results show that the tracked paths in both hardware and software have similar errors for each test. This indicates that the hardware implementation has correctly replicated the software implementation.

4.4.1 Stationary Path

The three objects of interest were processed in hardware to determine how well a stationary object could be tracked. Because the object and robot were stationary during the 100-frame sequence, this test guaranteed that no robot positioning error could occur.

The following two measurements were made from the recovered path:

1. **Distance Off Stationary Position:** Both the average distance off the path and the standard deviation were calculated.

2. **Angle Of Object:** The average error in estimating object angle and the standard deviation were calculated.

Table 4.1 shows the results of the stationary tests. The percents in this Table indicate the percent translation error based on each objects' minimum dimension. The relatively small average pose displacement of 0.42 mm was excellent when compared to the sensor and stereo extraction calibration pattern tests, in which the position of individual points varied on average by 4.5 mm. These results demonstrate the tracker's ability to compensate for noisy data inputs. Also observed is that the hardware and software provide consistent results with only slight variations.

In addition to the presented statistics, Figure 4.6 shows various views of the Angel's recovered positions during tracking of a 100-frame sequence with the object held stationary. The views show that all 100 positions of the object that have been recovered describe a small, concise region in which all recovered positions coincide across the frame sequence. Consequently, the tracking has been performed correctly.

Stationary Test						
		Angel	**Big Bird**	**Cube**	**Average**	
Hardware	**Translational**				(mm)	(%)
	Average Error (mm)	0.46	0.42	0.38	0.42	0.23
	Standard Dev (mm)	0.57	0.65	0.53	0.58	0.31
	Rotational				(deg.)	
	Average Error (deg.)	1.34	1.29	1.64	1.42	
	Standard Dev (deg.)	0.81	1.02	1.02	1.15	
Software	**Translational**				(mm)	(%)
	Average Error (mm)	0.42	0.55	0.09	0.35	0.18
	Standard Dev (mm)	0.50	0.86	0.30	0.56	0.28
	Rotational				(deg.)	
	Average Error (deg.)	0.40	0.86	2.58	1.28	
	Standard Dev (deg.)	0.57	0.36	3.18	1.37	

Table 4.1: The statistics of both rotational and translational components of the Angel, the Big Bird and the Cube, following a stationary path using both hardware and software.

Because fixed-point numbers have been used for representing the points' x, y, z components, a regular spacing can also be seen.

4.4.2 Linear Path

The three objects of interest were processed in hardware to determine how well the tracker could track a 100 frame linear-path 632 mm long with the object moving at effective rate of 316 mm per second (tracker operating at 200 fps). Figure 4.7 shows an image of the linear path experimental setup with the Cube object attached on a CRS robotic arm.

After the tracker was used to track the linear path, the estimated path was fitted

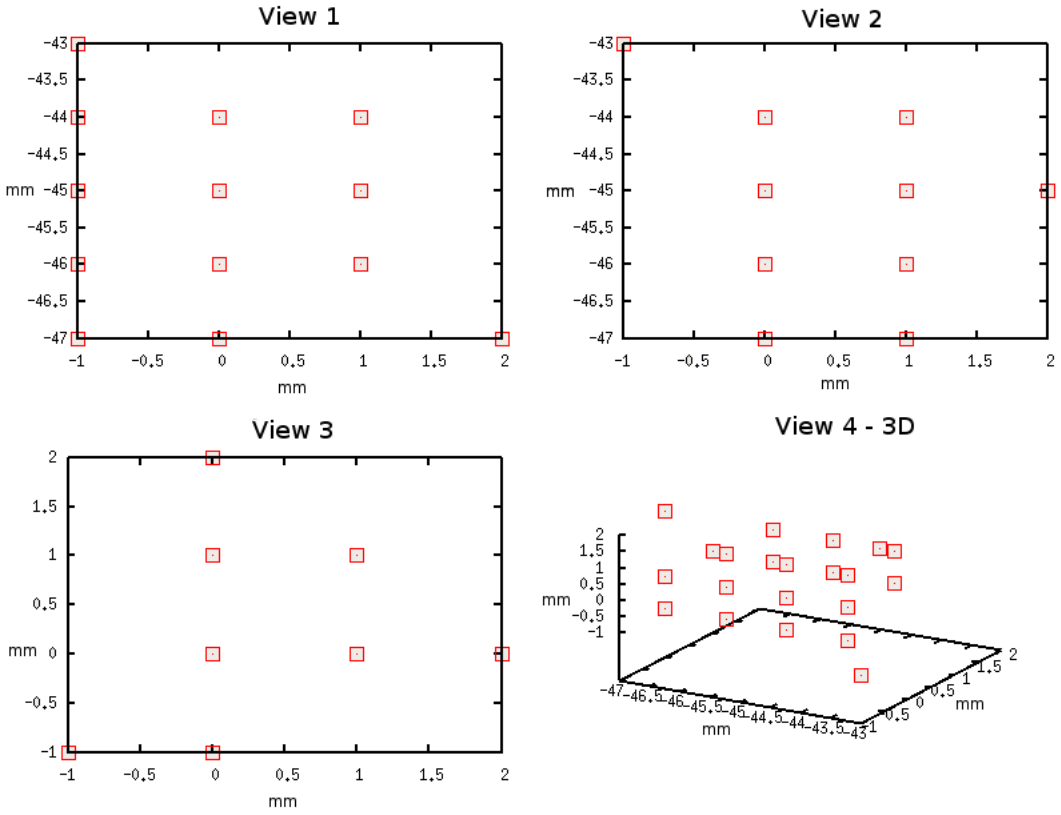


Figure 4.6: Four views of the tracked position of the stationary Angel during a 100-frame sequence. The small red boxes indicate the recovered positions of the object during the 100-frame sequence. View 1, 2 and 3 are along orthogonal axes. View 4 is a complete 3D view, displaying the compact, contained shape of the data.

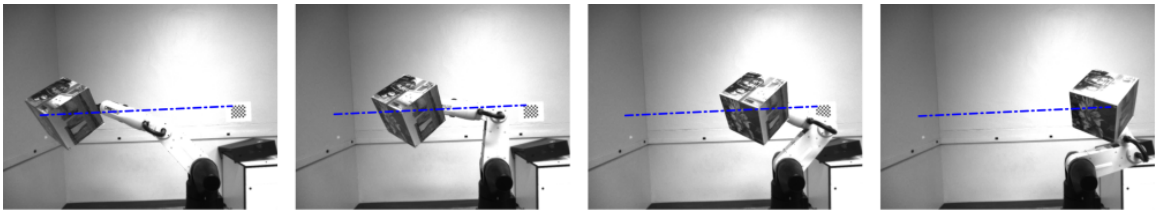


Figure 4.7: Images of the robotic arm with the cube object attached to the end effector. The left stereo images of the cube at frame 0, 33, 66, 99 (left to right) of the 100 frame linear path sequence. The dotted line indicates the expected path of the object.

to a line to determine how well the object was tracked during its linear movement. Two measurements were then made with the recovered path positions:

1. **Distance Off Path:** Both the average distance away from the ideal path and the standard deviation were calculated using Euclidean distance.

2. **Angle of Object:** The average angle error and standard deviation were calculated.

Linear Test						
		Angel	**Big Bird**	**Cube**	**Average**	
Hardware	**Translational**				(mm)	(%)
	Average Error(mm)	1.26	1.31	2.14	1.57	0.85
	Standard Dev (mm)	0.94	1.00	1.76	1.23	0.59
	Rotational				(degs.)	
	Average Error(deg.)	3.08	2.88	2.01	2.66	
	Standard Dev (deg.)	1.19	1.30	1.26	1.25	
Software	**Translational**				(mm)	(%)
	Average Error(mm)	1.30	1.28	2.20	1.59	0.81
	Standard Dev (mm)	1.24	0.94	1.55	1.24	0.68
	Rotational				(degs.)	
	Average Error(deg.)	2.82	2.69	2.89	2.80	
	Standard Dev (deg.)	1.22	1.46	1.91	1.53	

Table 4.2: The statistics of both rotational and translational components of the Angel, Big Bird, and the Cube following a linear path using both hardware and software.

Similar to the stationary tests statistics, these statistics indicated good tracking (see Table 4.2). The percents in this Table indicate the percent translation error based on each objects' minimum dimension. The average translational error in the

linear tests was 1.57 mm with a rotational error of only 2.66 degrees. These values are higher than the values obtained in the stationary tracking experiment. These results were expected because the linear tracking experiment had a moving object. Because the object moves after each frame, ICP only performs eight iterations per view. In comparison, a stationary object would have 100 aligned views each having eight ICP iterations. This is similar to having 800 ICP iterations for one view, which is more than enough time for the ICP to settle into the correct pose (local minimum).

Sensor and Stereo Extraction Influences

The paths that the tracker estimated for the Angel, the Big Bird, and the Cube are shown in Figures 4.8, 4.9, and 4.10 respectively. In these figures, a common curve is visible. The curve is most likely due to sensor and stereo extraction error. It is solely along the z-axis and sensor distortion increases towards the edge of lens. In fact, the previous calibration pattern test also indicated this trend despite the input images being rectified, as shown in Figure 4.4. Because of this sensor and stereo extraction influence, the average translation error and angle error would be expected to increase towards the edges.

Model Influences

During tracking, the three objects displayed different behaviour. The Angel and the Big Bird model displayed an even variation of their general curves (see Figures 4.8 and 4.9). This even variation was because the objects possessed equally descriptive shapes from all vantage points, thereby allowing the ICP algorithm to operate well from any view.

In contrast, the Cube model path, shown in Figure 4.10, appeared to thin out in one section, indicating the presence of a better shape description for that period of time. The cube model only had a maximum of three distinct surfaces to track. These thin regions coincided with the period when the sensor had a very good view of all three surfaces.

4.4.3 Rotational Path

The rotational test attempted to determine how well the tracker could follow a rotating object during a sequence of frames, while ignoring the translational components. The physical setup for this test was identical to the linear test, except that the robot was rotated through 80 degrees along one joint over a sequence of 100 frames. The rotation was limited to 80 degrees to prevent robot object occlusion. The effective rotatioal rate was 160 degrees per second with tracker operating at 200 fps. Because the object could not be rotated exactly around its centroid, the resulting translation of the object was ignored.

After the tracker had recovered the path of the object, the path was then processed to determine both the average change in the rotational axis and its standard deviation. These metrics provided an indication of how well the tracker was able to follow the rotation.

The results of the rotational test can be seen in Table 4.3. The average axis error of 0.39 degrees indicates a proper track of the single axis rotations being performed.

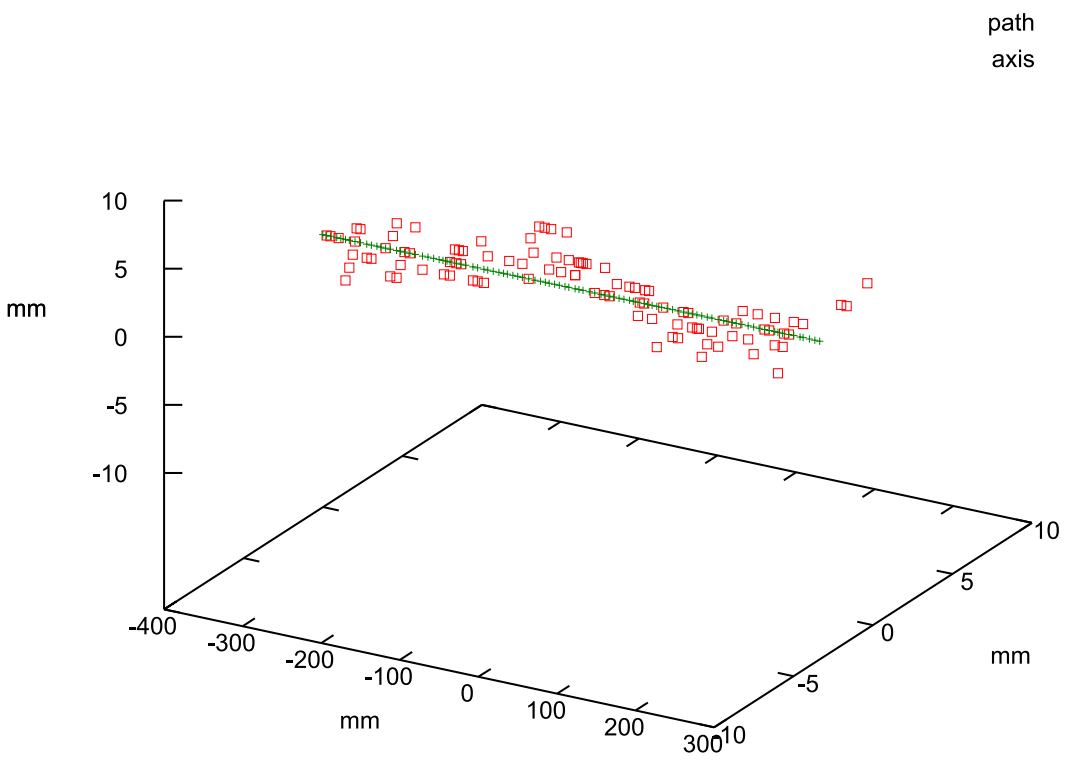


Figure 4.8: The red boxes indicate the recovered path of the linearly moving Angel model along with the best fit line, indicated by green crosses.

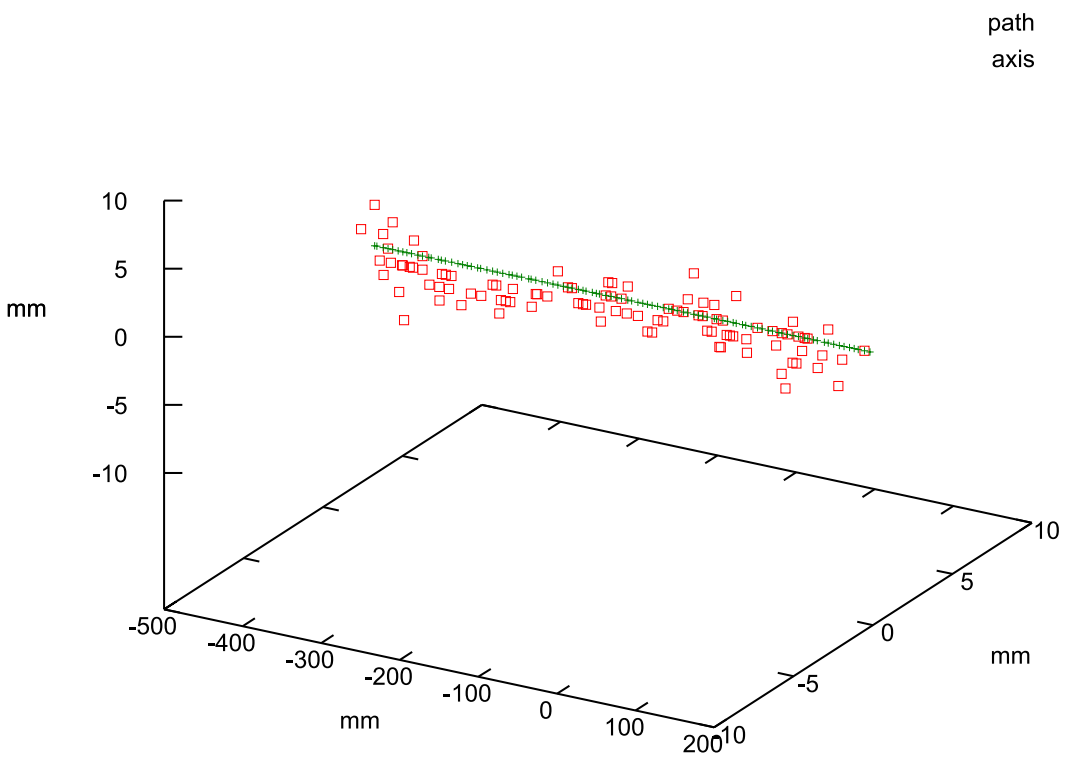


Figure 4.9: The red boxes indicate the recovered path of the linearly moving Big Bird model along with the best fit line, indicated by green crosses.

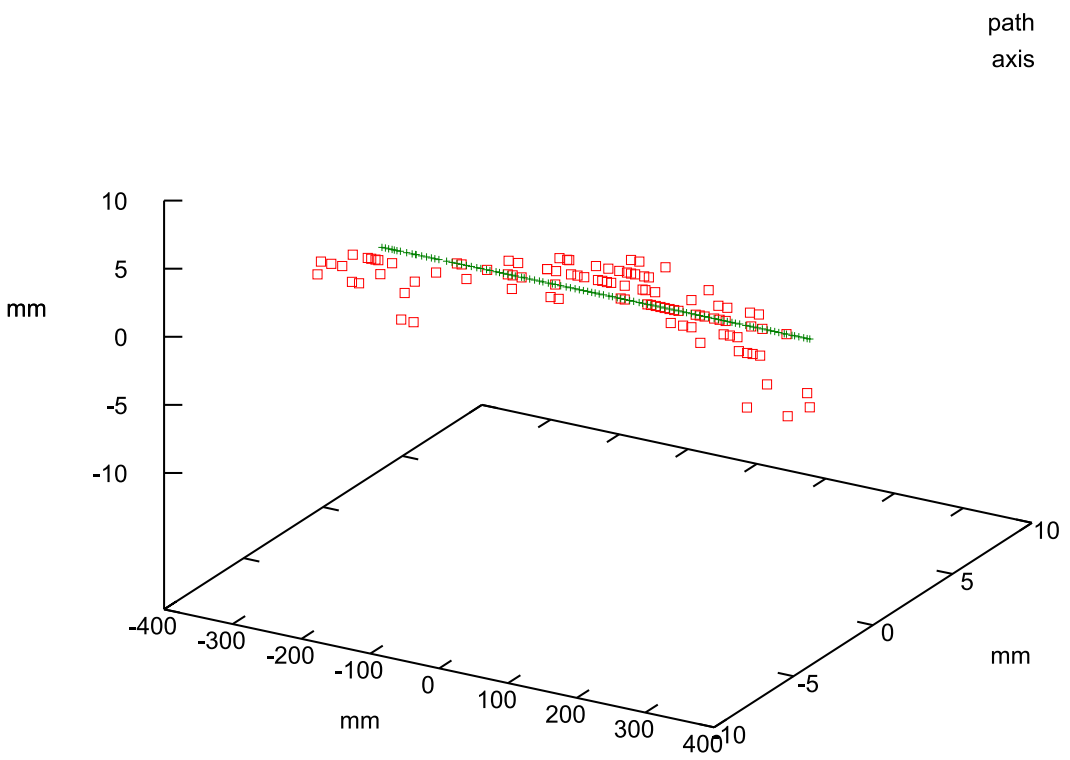


Figure 4.10: The red boxes indicate the recovered path of the linearly moving cube model along with best fit line, indicated by the green crosses.

Rotational Test					
		Angel	**Big Bird**	**Cube**	**Average**
Hardware	Average Error of Axis(deg.)	0.27	0.23	0.68	0.39
	Standard Dev of Axis (deg.)	0.15	0.12	0.30	0.19
Software	Average Error of Axis(deg.)	0.27	0.23	0.69	0.40
	Standard Dev of Axis (deg.)	0.15	0.12	0.30	0.19

Table 4.3: The statistics of the Angel, the Big Bird and the Cube, following a rotational path using both hardware and software.

4.4.4 Discussion of Results of Quantitative Tests

Tests were performed to determine how well the hardware tracker performed, as well as how the external components (stereo sensors and stereo extraction) performed. The sensors and stereo extraction generated noisy data and slightly warped disparities. Although the noisy data did not noticeably affect the system, the warped disparities did affect the recovered path. The quantitative tests verify the correctness of the tracker even under considerable sensor and stereo extraction noise. Paths were followed with low path errors and no loss of tracking during sequences.

4.5 Tracker Testing - Qualitative

4.5.1 Free-Form Path

The three test objects were put through a long free-form path test to ensure that the tracker could track unpredictable paths. Each sequence was created by manually moving the hand-held object for a duration of 1000 frames.

All three objects were tracked from start to finish with no obvious loss of object pose. The object path at a greater depth appeared to have a less accurate pose estimate. However, no loss of tracking occurred despite any abrupt changes in rotation and translation. Figures 4.11, 4.12, and 4.13 show the estimated tracked path for the 1000-frame sequences. Figure 4.14 shows sample frames of the Big Bird model with pose estimate overlays added post processing.

4.5.2 Occlusion

The Big Bird object was used for the occlusion test. During this test, the object remained stationary for 200 frames, while a wood panel was moved in front of the object to increase the amount of object occlusion.

At less than approximately 60 percent occlusion, the object tracker appeared to be unaffected when compared to the non-occluded state. However, at more than approximately 60 percent occlusion, the tracker gracefully degraded until the object was fully occluded. That is the tracker stability slowly declined once ideal conditions were exceeded. The previous results were expected with the decrease in the diversity (descriptiveness) of the data.

For points sampled from only one region of an object, there is less spatial diversity than for points sampled from the entire object. This loss of data diversity would directly affect the amount of shape information presented to the ICP algorithm. As a result, this decreased data would be more likely to cause tracking difficulty.

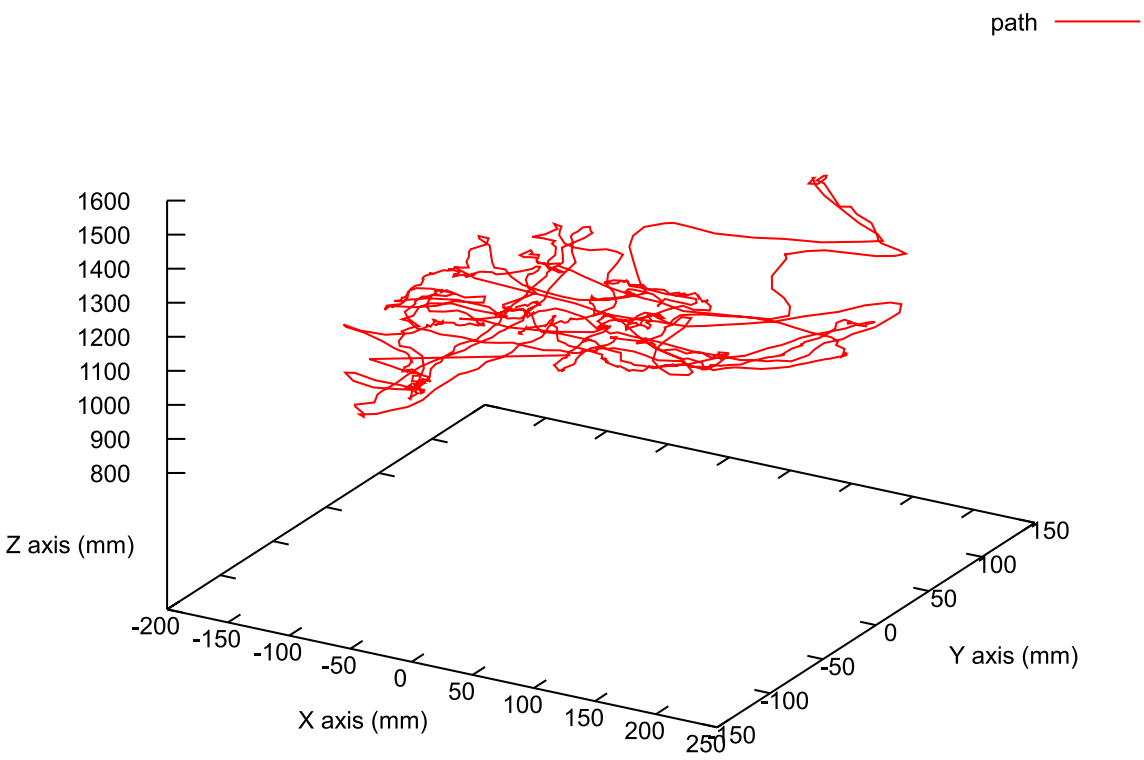


Figure 4.11: The estimated track of the Angel moving in an arbitrary path of 1000 frames.

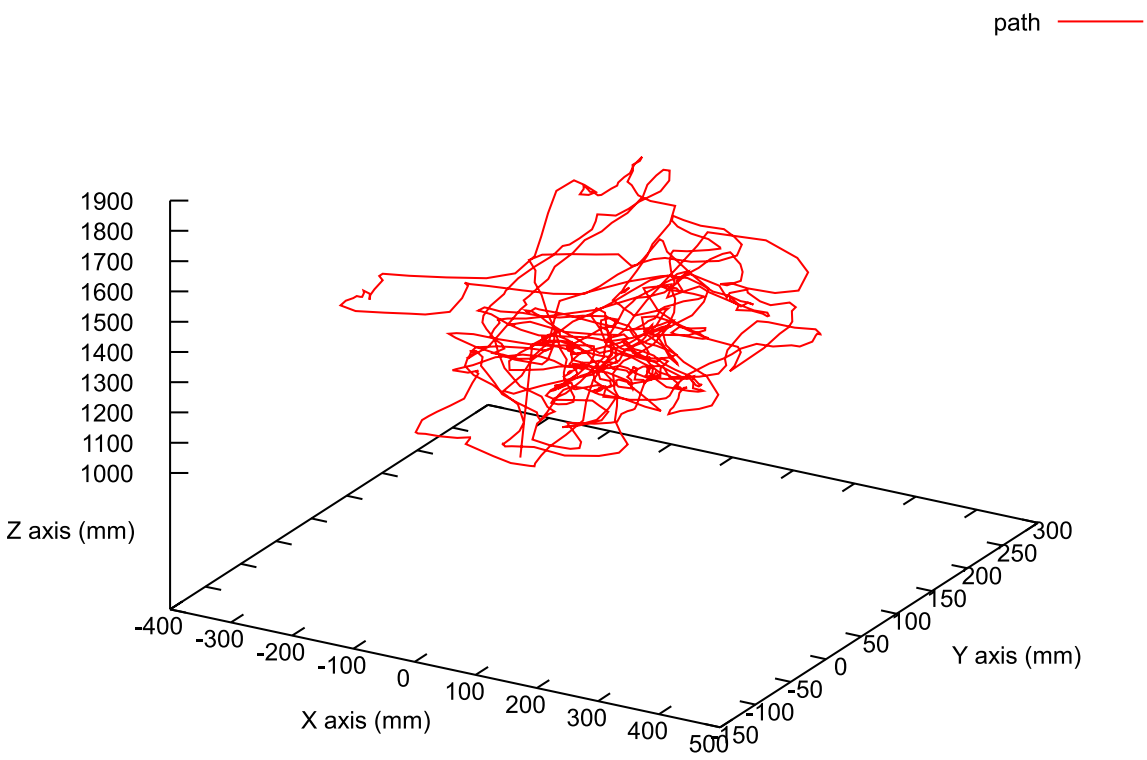


Figure 4.12: The estimated track of the BigBird moving in an arbitrary path of 1000 frames.

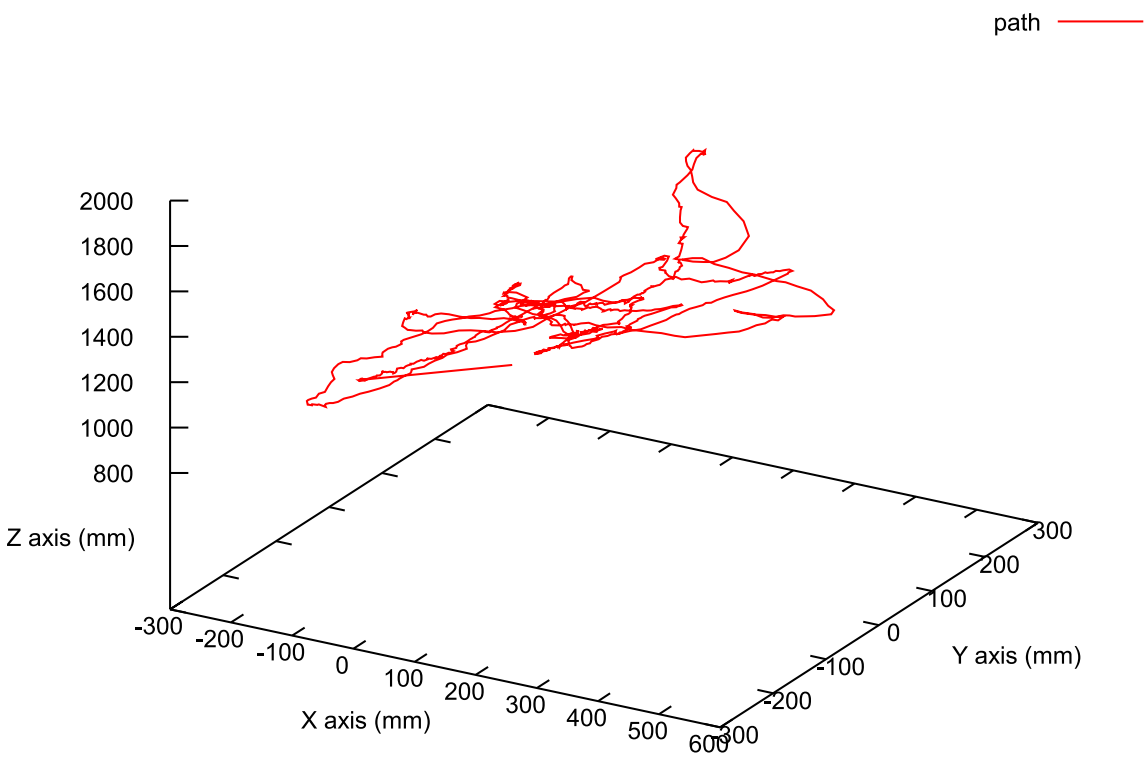


Figure 4.13: The estimated track of the Cube moving in an arbitrary path of 1000 frames.

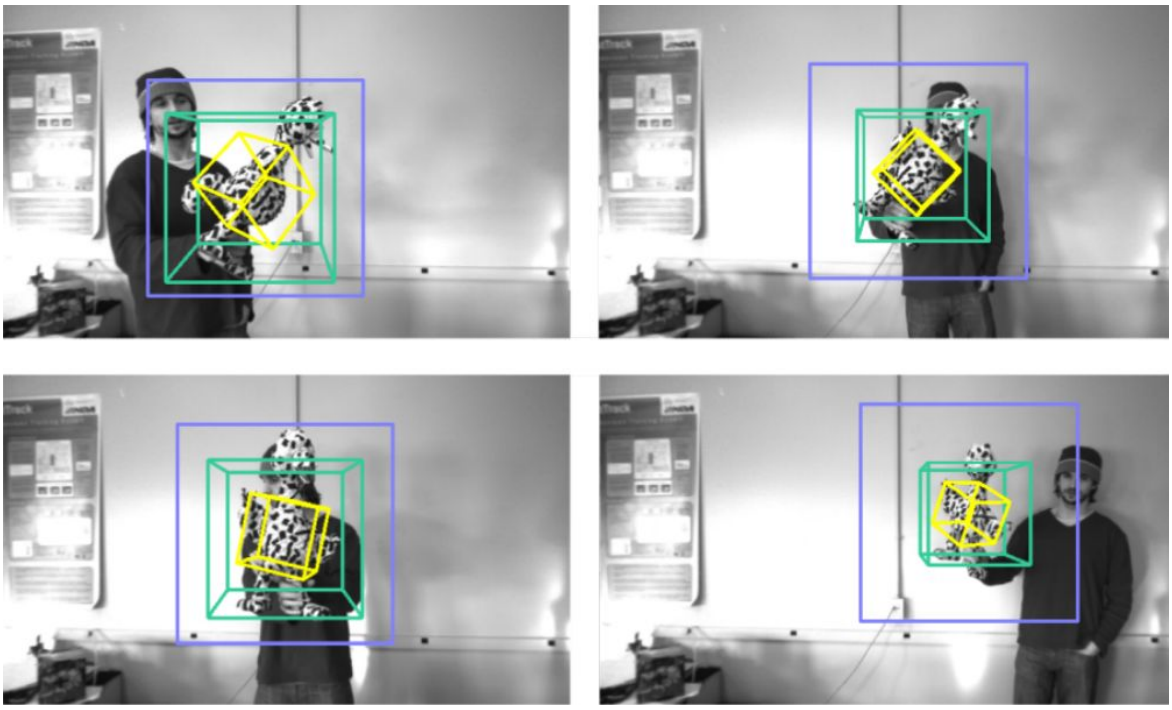


Figure 4.14: Four example frames of the Big Bird model being tracked with the Fast-Track hardware ICP tracker. The images were obtained by sampling the left stereo sensor at various time instances. Various tracker output overlays were then applied to the image. These overlays indicate the tracker's current configuration (the regions of interest and the estimated pose) so that its performance can be observed. The outermost overlay is the disparity region of interest. The middle overlay is the cube region of interest. The innermost overlay represents the estimated pose of the object.

4.5.3 Discussion of Results of Qualitative Tests

The freeform tests verify the correctness of the tracker qualitatively. Freeform paths were followed without any obvious pose error. In addition, occlusion tests indicate a graceful decrease in tracker performance after approximately 60 percent of object occlusion.

4.6 Hardware Details

4.6.1 Clock Speed and Frames-per-second

To integrate with the other components of the FastTrack stereo vision tracking system, the ICP tracker was designed to operate at 125.0 MHz. The Xilinx tool synthesis report indicates that the maximum possible speed was 130.8 MHz. This speed is limited due to a critical path within the Microblaze processor, and not the surrounding hardware implementation.

For the unoptimized tracker used during testing (with buffering between stages, no double buffering for the pre-filter), the tracker was set to operate with eight iterations per frame and 16 parallel nearest neighbor units. This configuration of the circuit can operate at $234.3 - S$ fps, where S is the relatively small, variable time for pre-filtering and transform recovery. Therefore, the tracking system (with incoming frame rate of 200.0 fps) can operate with S as large as 17 percent of the total processing time without missing any frames. The only time S would become large enough to noticeably present a problem would be: if the object was almost fully occluded during tracking, if the cube ROI was extremely small, or if the previous frame's pose estimate was not close enough to the object in the current frame.

For comparison, software ICP tests using both a brute-force nearest neighbor search and a K-D tree nearest neighbor search have been performed. Table 4.4 provides the results of speed tests. The speed tests were executed on a Intel Pentium M760 2 GHz processor. The results are the average of three different frame sequences. Any IO timings have not been included in these speed calculations because the pure processing speed is of interest and the 200 fps stereo sensor was not available. With IO, the software performance would be expected to decreas. Because the hardware architecture and timings have been designed specifically for these IO bandwidths, there will be no performance reduction.

Although the K-D tree ICP performs significantly better than the brute-force ICP in software, the hardware parallel brute-force ICP exceeds its speed by a factor of five.

Speed Comparison			
Technique	**Software Brute-Force**	**Software K-D Tree**	**Hardware Parallel Brute-Force**
Frames-per-second	3.9	39.3	200.0

Table 4.4: The results of ICP hardware and software solutions.

4.6.2 Utilization of Hardware Resources

The hardware resource utilization in the selected Xilinx Vertex II Pro VC2VP100 can be seen in Table 4.5. These FPGA statistics were based on the ICP implementation used in the experimentation. This implementation used 16 bits per component and 16 parallel-processing nearest neighbor units. Because only 21 percent of the total slices in the chip were utilized, there is still 79 percent remaining for integration with

Resource Utilization in Xilinx VC2VP100			
	Used	**Available**	**Percent**
Number of Slices	9309	44096	21%
Number of Slice Flip Flops	11209	88192	12%
Number of 4 input LUTs	9938	88192	11%
Number of BRAMs	77	444	17%
Number of MULT18X18s	78	444	17%
Number of GCLKs	1	16	6%

Table 4.5: The FPGA component utilization of implemented circuit provided by place and route tool.

the remaining FastTrack system components.

4.6.3 Projected Power Consumption

One advantage of FPGAs is the hardware's lower power consumption, compared to general computing systems. The Xilinx tool XPower was used to predict the FPGA's power consumption, as shown in Table 4.6.

Power Consumption		
	Current(mA)	**Power(mW)**
1.50V	0	0
2.50V	167	418
3.30V	2	6
Total Power		**423**

Table 4.6: The predict power consumption of implemented circuit.

By way of example, a current Intel mobile processor running at idle will use approximately 6 W. In comparison, the FPGA implemented here will utilize an estimated 423 mW of power during full processing load. As evident here, the slower clock speeds of the custom FPGA design require less power than the faster clock

speed processor design. This significant reduction in power requirement is one of the key reasons for utilizing hardware solutions.

4.7 Discussion of Results

The tests that were performed to characterize the stereo vision sensor and stereo extraction techniques indicate large errors in the disparity image. Because these errors are present in the disparity image, the path tracking tests must take these errors into consideration when interpreting the results.

The stationary object test indicates that the tracker can indeed provide a consistent track of stationary objects. This is the simplest case. Plots of the data from the stationary tests show a compact region with no outlying data points, which is indicative of a stable track. Further tests, such as the linear and the rotational tests, also exhibit stable tracks with relatively small displacement errors and rotational axis errors, despite noisy disparity data. Notably, the linear path tests exhibit a slight arc that can be attributed to the sensor and stereo extraction errors (that were also evident in initial tests). This arc is not a tracking error; however, all tracked paths will be influenced by this sensor and stereo extraction error. All of these positive tracking results were then further confirmed by tracking the object along free-form paths. No obvious deviation of path was observed. Occlusion testing showed a graceful failure only after approximately 60 percent of the object became invisible.

During all tests, the Big Bird model produced the best tracking results. These results are probably due to the fact that this model was the largest object and also had a highly descriptive shape from all vantages.

The speed tests indicate that the hardware parallel brute-force ICP is over five

times faster than the software K-D tree ICP. Although the software was executed on a processor that represents slightly older technology, the selected FPGA is of the same generation.

This ICP tracker consumes only 21 percent of the logic resources in the selected FPGAs. Therefore, 79 percent of the chip area is available for further development and integration. As a result, this facilitates integration with the stereo sensor and stereo extraction components of the overall FastTrack system that is under development, without having to upgrade to a larger FPGA.

Although the hardware was not optimized for low power utilization, the design tools predict that the tracker only requires 423 mW to operate, making it substantially more power efficient than a general processor ICP tracker. This low power requirement and the fact that the FPGAs are compact and lightweight was the key reasons for using custom hardware processing for tracking in space.

Chapter 5

Summary

5.1 Accomplishment of Objectives

The objective of this work was to design and construct the high-speed hardware tracking component of a three-part team project called 'FastTrack'. The FastTrack project consists of three components which were designed and implemented by three Universities; stereo vision sensor platform (Ryerson University), stereo extraction hardware (University of Toronto), and object tracking hardware (Queen's University). These three components were designed to utilize a single field-programmable gate array (FPGA) in order to address the large bandwidths associated with the communication between components. For example, the input to this tracking component is disparity data with a resolution of 640×480 pixels per frame, with each pixel having 128 disparities. This high-speed realtime operation at 200 fps could not have been accomplished in software. Tests indicate that if IO is ignored, a single processing core is only able to achieve less than a fifth of this goal. Therefore, the tracking component was designed using a hardware based ICP algorithm to track objects at a high frame

rate of over 200 fps.

High-speed tracking is desired for many applications such as: video surveillance, monitoring, innovative medical procedures, and advanced robotics. This tracker was specifically designed for tracking satellites to enable remote satellite capturing for repair and maintenance. Because of the complexity and delicate nature of satellites, high-speed tracking is required in order to correctly position the robotic servicer for grasping in realtime. Presented here is a novel hardware solution for tracking an object through a sequence of stereo vision disparity frames at a high frame rate using an FPGA.

5.2 Hardware Implementation of an ICP Tracker

The ICP hardware consists of four stages: pre-filter, transform, nearest neighbor, and transform recovery. These stages all operate in hardware at 125.0 MHz clock speed in a Xilinx XC2VP100, utilizing 21 percent of the available logic resources in the chip.

The pre-filter stage was designed to reduce the amount of data utilized by the tracking algorithm. It operates by using a disparity region of interest (ROI), cube ROI, and uniform subsampler in order to segment the object of interest from the surrounding clutter, thereby removing most of the unnecessary data. Because the pre-filter buffers the disparity ROI in memory, the pre-filtering algorithm is able to cycle through the ROI and select exactly the same number of disparities for every frame. The selection of these subsampled pixels is governed by the technique of point verification, utilizing a cube ROI. This cube ROI position is given by the tracked object's last estimated position in the camera coordinate frame.

The ability of the pre-filter stage to select exactly the same number of disparities in

each frame simplifies later processing in the nearest neighbor and transform recovery stages. This technique causes the pre-filter to have an unpredictable frame rate. However, because this filter's frame rate is always much higher than the rest of the system, this unpredictability can be safely ignored.

In the transform stage, the data is transformed to the last pose estimate for further processing by the nearest neighbor stage. This stage is a simple matrix multiplier that allows for data points to be processed in a pipeline fashion.

The nearest neighbor stage utilizes the parallel brute-force approach with 16 comparators (which can process 512 data to 2048 model correspondences at 234.3 fps). This approach was ideal for a hardware design because the problem was easily segmented into 16 parallel searches. Although additional multiplier resources were necessary for distance calculations on every parallel path, a distance approximation could be performed without multipliers. Another nearest neighbor variation, referred to as 'low-to-high', was investigated. Theoretically, it provided substantial speed increases when compared to the brute-force approach. In order to obtain this increase in speed, a slightly more involved model and hardware would be required.

Finally, the transform recovery stage determines the estimated transform, using a hardware and software SVD implementation. This stage was designed using hardware with software on a soft-core Xilinx Microblaze processor.

A technique to reduce data communication bandwidth between hardware and software processing was also developed. This data reduction, which pre-computes the $\mathcal{N}$ matrix in hardware, was implemented on the FPGA.

5.3 Experimentation and Results

Testing of the tracker was performed using both simulated and physical hardware tests. In order to accurately determine the tracker characteristics, with minimal bias from the testing environment (sensors and extraction method), the surrounding components were examined. Tests were executed to characterize the stereo sensor and stereo extraction algorithm, that provide the system input. These initial tests indicated that the commercial stereo sensor and available stereo extraction algorithm produced extremely inaccurate data in certain situations. For sensing a plane, an average 58.05 mm displacement error was observed. For sensing a plane when only ideal disparities were selected, an average 4.50 mm displacement error was observed. Also, a warping of disparities was found when sensing planar objects. These results are significant because they greatly affect the performance of the tracker.

Next, the tracker was tested on three physical objects: the Angel, the Big Bird, and the Cube. A robotic arm was used to position these objects along three set paths: stationary, linear and rotational. During the stationary path test, the object was held completely motionless. Results indicated that the tracker was able to estimate the position to within an average of 0.42 mm and 1.42 degrees. During the linear path test, the object was moved along a straight line, utilizing a robotic arm for positioning. The average error of the linear path test was within 1.57 mm and 2.80 degrees. Compared to the results of the stationary path test, the translation and rotation errors were higher due to the object's interframe motion. Also, a comparison to the stereo extraction and sensor test (which involved registering a plane to ideally selected points) resulted in an average translational error of 4.50 mm, which was significantly higher than the translation error of the linear path. During the rotational path test,

the rotation axis of the path was observed to shift only slightly with an average of 0.39 degrees.

Further tests were performed to ensure that the tracker could also track a free-form unpredictable path. The three test objects were moved by hand for 1000 frames. These tests were successful with no loss of tracking during sequences.

As mentioned in Section 4.4, no ground truth data were obtained and only relative paths were considered. However, ground truth data would enable better comparisons to other trackers.

The robustness to occlusion was also tested by tracking an object that was progressively occluded. Results indicated that the tracker was able to handle up to 60 percent occlusion. Further occlusion gracefully degraded the tracker's performance.

Speed tests were performed on the software and hardware implementations of ICP. The hardware implementation, which was designed to run at 200 fps, was over five times faster than the K-D tree software. In fact, the software design operates at even slower speeds if IO is considered.

5.4 Conclusion

This hardware object tracker has demonstrated tracking results comparable to a prototype software version, has the advantage of operates successfully at high-speeds of over 200 fps. Several hardware techniques were explored to design the hardware ICP tracker. A prefilter stage reduced data bandwidth, a transform stage re-aligned scene data points, a parallel nearest neighbor stage enabled high-speed correspondence search, and a hardware software transform recovery stage determined the change in object pose. Experiments demonstrated that the tracker operates correctly when

tracking various paths (and at speeds that are faster than the software implementations). Free-form path tests (in which an object is moved with arbitrary motion) showed no loss of tracking during 1000-frame sequences.

As mentioned previously, this tracker is designed for integration within the larger FastTrack stereo vision tracking system on a single FPGA platform. FPGA utilization reports indicate that the tracking component consumes only 21 percent of the FPGA area, leaving 79 percent of the FPGA area available for the remaining FastTrack components. In addition, power requirements of this tracker are low, requiring only 423 mW of power for operation, as estimated by the FPGA design software.

In conclusion, the tracker is fast, compact, lightweight, has low power requirements, and can be integrated with both a stereo sensor and stereo extraction module in a single FPGA. This tracker is valuable in situations that require object tracking at high-speeds but have limits on size, weight, and power availability, such as tracking satellites in space.

5.5 Future Work

In the future, this system will be merged with the other two components in order to form the complete high-speed FastTrack stereo vision tracking system. The system performance can be improved by implementing the techniques described. By implementing the low-to-high nearest neighbor search, chip utilization could be further reduced or higher speeds could be achieved. Further optimizations to remove unnecessary buffers between tracker components could be accomplished by using the the statistical filtering technique (see Section 3.4.2) as well as the streaming transform recovery technique (see Section 3.8).

Further detailed testing can also be performed. For example, further testing with replica satellite models and appropriate lighting to simulate space conditions would allow for better understanding of the tracker's performance for in-orbit operation. Also, in order to allow for comparisons with other trackers, this system can be tracked with data that have ground truth.

Bibliography

[1] S. Sabihuddin and W. MacLean. Maximum-likelihood stereo correspondence using field programmable gate arrays. In *Proceedings of the 5th International Conference on Computer Vision Systems (ICVS)*, 2007.

[2] A. Yilmaz, O. Javed, and M. Shah. Object tracking: A survey. *ACM Computing Surveys (CSUR)*, 38(4):13, 2006.

[3] P. Besl and N. McKay. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, Feb. 1992.

[4] S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. In *3DIM01: Proceedings of the IEEE Computer Society 3rd International Conference on 3D Digital Imaging and Modeling*, pages 145–152, Quebec City, Quebec, Canada, May 2001.

[5] P. Jasiobedzki, M. Greenspan, and G. Roth. Pose determination and tracking for autonomous satellite capture. In *i-SAIRAS: the Proceedings of 6th International Symposium on Artificial Intelligence and Robotics and Automation in Space*, pages 18–22, Montreal, Quebec, Canada, June 2001.

[6] P. Jasiobedzki, M. Abraham, P. Newhook, and J. Talbot. Model based pose estimation for autonomous operations in space. In *Proceedings of IEEE International Conference on Intelligence, Information, and Systems.*, pages 211–215, Bethesda, MD, USA, Nov. 1999.

[7] H. Li and M. Greenspan. Segmentation and recognition of continuous gestures. In *Proceedings of the IEEE International Conference on Image Processing (ICIP)*, pages 365–368, San Antonio, TX, USA, Sept. 2007.

[8] D. Cobzas and P. Sturm. 3D SSD tracking with estimated 3D planes. In *CRV-05: Proceedings of the 2nd Canadian IEEE Computer Society Conference on Computer and Robot Vision*, pages 129–134, Washington, DC, USA, 2005.

[9] D. Simon, M. Hebert, and T. Kanade. Real-time 3-D pose estimation using a high-speed range sensor. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 2235–2241, San Diego, CA, USA, May 1994.

[10] M. Greenspan, L. Shang, and P. Jasiobedzki. Efficient tracking with the bounded Hough transform. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 520–527, Los Alamitos, CA, USA, June 2004.

[11] M. Fiala. Artag, a fiducial marker system using digital techniques. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 20–25, San Diego, CA, USA, 2005.

[12] B. Taati and M. Greenspan. Satellite pose acquisition and tracking with variable dimensional local shape descriptors. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems - Workshop on Robot Vision for Space Application (IROS05 RVSA)*, pages 4–9, Edmonton, Alberta, Canada, Aug. 2005.

[13] B. Taati, M. Bondy, P. Jasiobedzki, and M. Greenspan. Variable dimensional local shape descriptors for object recognition in range data. In *11th IEEE International Conference on Computer Vision - Workshop on 3D Representation for Recognition (ICCV-3dRR)*, pages 1–8, Rio de Janeiro, Brazil, Oct. 2007.

[14] D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision (IJCV)*, 60(2):91–110, Nov. 2004.

[15] Motion Capture. Optitrack. Natural Point Inc. 2007, available at `http://www.naturalpoint.com/optitrack/products/motion-capture/`.

[16] G. Welch and G. Bishop. An introduction to the Kalman filter. In *SIGGraph-2001 course 8: Computer Graphics, Annual Conference on Computer Graphics and Interactive Techniques*, Los Angeles, CA, USA, Aug. 2001. ACM Press, Addison-Wesley.

[17] D. Simon. Kalman filtering. *Embedded Systems Programming*, 14(6):72–79, 2001.

[18] F. Dellaert and C. Thorpe. Robust car tracking using Kalman filtering and Bayesian templates. In *SPIE: Intelligent Transportation Systems*, volume 3207, Pittsburgh, PA, USA, Oct. 1997.

[19] K. Nummiaro, E. Koller-Meier, and L. Van Gool. Object tracking with an adaptive color-based particle filter. In *Proceedings of the 24th DAGM Symposium on Pattern Recognition*, volume 21, pages 353–360, Zurich, Switzerland, Sept. 2002.

[20] S. Weik. Registration of 3D partial surfaces using luminance and depth information. In *3DIM97: Proceedings of the IEEE Computer Society International Conference on Recent Advances in 3-D Digital Imaging and Modeling*, pages 93–100, Ottawa, Ontario, Canada, May 1997.

[21] G. Turk and M. Levoy. Zippered polygon meshes from range images. In *Proceedings of SIGGraph-94*, pages 311–318, Orlando, Florida, USA, July 1994.

[22] C. Langis, M. Greenspan, and G. Godin. The parallel iterative closest point algorithm. In *3DIM01: Proceedings of the IEEE Computer Society 3rd International Conference on 3D Digital Imaging and Modeling*, pages 195–204, Quebec City, Quebec, Canada, May 2001.

[23] T. Masuda, K. Sakaue, and N. Yokoya. Registration and integration of multiple range images for 3-D model construction. In *ICPR-96: Proceedings of the IEEE Computer Society International Conference on Pattern Recognition*, volume 1, page 879, Washington, DC, USA, 1996.

[24] G. Godin, M. Rioux, and R. Baribeau. Three-dimensional registration using range and intensity information. In *Proceedings of the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference, Videometrics III*, volume 2350, pages 279–290, Oct. 1994.

[25] M. Greenspan and M. Yurick. Approximate k-d tree search for efficient ICP. In *3DIM03: Proceedings of the IEEE Computer Society 4th International Conference on 3D Digital Imaging and Modeling*, pages 442–448, Banff, Alberta, Canada, Oct. 2003.

[26] T. Ziner, J. Schmidt, and H. Niemann. Performance analysis of nearest neighbor algorithms for ICP registration of 3-D point sets. In *the 8th International Fall Workshop of Vision, Modeling, and Visualization*, Stuttgart, Germany, 2003.

[27] T. Jost and H. Hügli. A multi-resolution ICP with heuristic closest point search for fast and robust 3D registration of range images. In *3DIM03: Proceedings of the IEEE Computer Society 4th International Conference on 3D Digital Imaging and Modeling*, pages 427–433, Banff, Alberta, Canada, Oct. 2003.

[28] Y. Chen and G. Medioni. Object modelling by registration of multiple range images. *Image and Vision Computing*, 10(3):145–155, 1992.

[29] G. Blais and M. Levine. Registering multiview range data to create 3D computer objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):820–824, 1995.

[30] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical recipes in C: the art of scientific computing*, chapter Singular Value Decomposition. Cambridge University Press, 2002.

[31] W. Ma, M. Kaye, D. Luke, and R. Doraiswami. An FPGA-based singular value decomposition processor. In *Canadian Conference on Electrical and Computer Engineering*, pages 1047–1050, Ottawa, Ontario, Canada, May 2006.

[32] C. Bobda and N. Steenbock. Singular value decomposition on distributed reconfigurable systems. In *RSP-01: Proceedings of the IEEE Computer Society 12th International Workshop on Rapid System Prototyping*, page 38, Washington, DC, USA, 2001.

[33] F. Mühlbauer and C. Bobda. A dynamic reconfigurable hardware/software architecture for object tracking in video streams. *EURASIP: Journal on Embedded Systems*, pages 8, ID 82564, 2006.

[34] P. Ashenden. A comparison of recursive and repetitive hardware models in VHDL. In *Proceedings of the VHDL International Users Forum. Spring Conference*, pages 80–89, Oakland, CA, USA, May 1994.

[35] A. Glassner and J. Ritter. *A Fast Approximation To 3D Euclidian Distance.* Academic Press, 525 B Street, Suite 1900, San Diego, CA, USA, 1990.

[36] B. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 4(4):629–642, Apr. 1987.

Appendix A

Bumblebee Camera Specifications

Permission to reprint by Point Grey Research.

digital stereo vision camera

- Two progressive scan CCDs
- High speed IEEE 1394 digital communication
- Accurately precalibrated
- Compatible with all Point Grey software
- Ideal for application prototyping

bumblebee™

POINT GREY RESEARCH

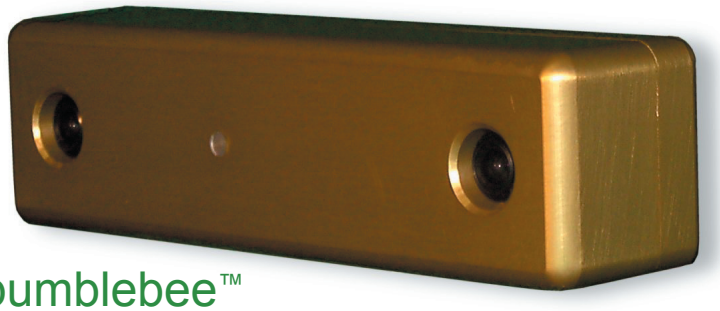
Bumblebee™ is Point Grey's new two lens stereo vision camera. It provides a balance between 3D data quality, processing speed, size and price. The camera is ideal for applications such as people tracking, gesture recognition, mobile robotics and other computer vision applications.

Bumblebee is compatible with the Triclops™ library and works with all of Point Grey's application level software, such as Censys3D® people tracking SDK and Multiclops™ inter camera calibration system. Users with Digiclops® camera can convert their existing application level software with minimal source code changes.

Bumblebee is precalibrated for lens distortions and camera misalignments. It doesn't require in-field calibration and is guaranteed to stay calibrated. The left and right images are aligned within 0.05 pixel RMS error. The calibration information is pre loaded on the camera, allowing the software to retrieve the image correction information. This allows seamless swapping of the cameras, or retrieving the correct information when multiple cameras are on the bus.

Bumblebee allows inter-camera synchronization. By virtue of being on the same Firewire bus, all Bumblebee cameras synchronize themselves. This is particularly useful for acquiring 3D data from multiple points of view, such that it is exactly registered in time and space. Multiclops software allows inter camera calibration and integration.

Bumblebee is supplied as a full development kit, including the camera head, interface card, 4.5m cable, device driver, image acquisition software, Triclops library, user manuals and 1-year technical support. As always, application level software, such as Censys3D and Multiclops is supplied for download free of charge.



bumblebee™

Specifications:

- 640x480 Option: two Sony ICX084 Color or BW CCDs
- 1024x768 Option: two Sony ICX204 Color or BW CCDs
- 1/3 inch, progressive scan CCDs
- 640x480 square pixels at 30 Hz frame rate
- 1024x768 square pixels at 15 Hz frame rate
- 10 bit A/D
- Shutter speed: 1/8000s to 1/30s @ 30hz at 640x480;
  1/6000s to 1/15s @ 15hz at 1024x768
- Baseline: 12cm
- Choice of 2mm, 4mm or 6mm focal length lenses (100°, 70°, and 50° HFOV respectively)
- Size: approx. 160 X 40 X 50 mm
- Weight: approx. 375g
- Signal to noise ratio: TBD
- Automatic/manual gain and shutter control with adjustable frame rate
- Independent control of CCD gains for image intensity balancing
- Color models provide Bayer tiled images for reconstruction on the host
- One IEEE-1394 6-pin connector
- Power supplied by IEEE-1394
- Power consumption: 2.1 W

point grey RESEARCH

www.ptgrey.com

305-1847 West Broadway, Vancouver, B.C., Canada V6J 1Y6 T: +604-730-9937 F: +604-732-8231

Point Grey Research (PGR) is a worldwide leader in the development of advanced digital camera technology products. Based in Vancouver, British Columbia, PGR designs, manufactures and distributes IEEE-1394 cameras, stereo vision cameras and spherical digital video cameras to a broad spectrum of industries. Through a close working relationship wtih its customers, PGR continues to be at the forefront of innovation.

Revised February 14, 2005