High Frame Rate Marker Detection for Single Camera-Based Localisation

by

Schalk Pretorius

Thesis presented in partial fulfilment of the requirements for the degree of Master of Engineering (Electronic) in the Faculty of Engineering at Stellenbosch University

Supervisor: Dr. A. Barnard
Co-supervisor: Dr. HW. Jordaan

April 2022

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: .....April 2022.....

Abstract

Localisation is a fundamental component of autonomous navigation. Autonomous systems that perform high accuracy tasks require position and orientation information at high rates. Pose information at high rates allows an autonomous system to react faster to changes in the environment and increases the accuracy and stability of the system.

Current implementations of landmark-based localisation by using fiducial markers have a limited utility due to low detection and pose estimation rates. Increasing the frame rate that fiducial marker systems can run at greatly increases the utility of these systems.

The goal of this thesis was to design and implement a fiducial marker detection system that runs on a low-power-, light-weight- and compact hardware platform that can operate at frame rates larger than 60 Frames Per Second (FPS) without compromising pose accuracy.

The project goal was achieved by researching existing marker systems and implementations, designing a marker detection concept, implementing the concept design on the chosen hardware platform and testing and verifying the implemented system.

This thesis presents a high frame rate marker detection system that is designed to run on a ZYNQ 7020. The ZYNQ 7020 is a System on a Chip (SoC) consisting of a Field Programmable Gate Array (FGPA) and a Central Processing Unit (CPU).

The marker detection system achieves a higher frame rate at comparable accuracy to existing systems by utilising the advantages of an FPGA. The FPGA is utilised to parallelise image processing functions and accelerate the data intensive image processing system. By using in-line processing to extract image information without first needing to store the image in memory processing time is reduced.

The presented marker detection system has a comparable accuracy to other fiducial marker detection systems validating the performance of the proposed system. The designed and implemented system is tested and the system is verified to operate at an average framerate of 134 FPS.

Uittreksel

Lokalisering is 'n fundamentele komponent van outonome navigasie. Outonome stelsels wat hoë akkuraatheid take verrig vereis posisie en oriëntasie inligting teen 'n hoë tempo. Posisie en oriëntasie inligting teen 'n hoë tempo laat 'n outonome sisteem toe om vinniger op veranderinge in die omgewing te reageer en verhoog die akkuraatheid en stabiliteit van die stelsel.

Huidige implementerings van landmerk-gebaseerde lokalisering deur die gebruik van merkers het 'n beperkte nut as gevolg van lae opsporing en posisie- en oriëntasie skattingstempo. Die verhoging van die raamtempo van merkerstelsels kan die bruikbaarheid van hierdie stelsels baie verhoog.

Die doel van hierdie tesis was om 'n merker-opsporingstelsel te ontwerp en te implementeer wat op 'n lae-krag-, liggewig- en kompakte hardeware platform werk en wat teen 'n raamtempo groter as 60 FPS kan werk sonder om akkuraatheid op te offer. Die projekdoelwit is bereik deur bestaande merkerstelsels en implementerings van huidige merkerstelsels te identifiseer, 'n merker-opsporingskonsep te ontwerp, die merker-opsporingskonsep op die gekose hardeware platform te implementeer en die sisteem te toets om te verifieer dat die geïmplementeerde stelsel werk.

Hierdie tesis bied 'n hoë raamtempo merker-opsporingstelsel aan wat op 'n ZYNQ 7020 (SoC wat uit 'n FGPA en 'n CPU bestaan) loop. Die merker-opsporingstelsel bereik 'n hoër raamtempo teen 'n vergelykbare akkuraatheid deur die voordele van 'n FPGA te benut. Die FPGA word gebruik om die beeldverwerking funksies te paralleliseer en die data intensiewe beeld verwerking stelsel te versnel. Deur die gebruik van inlynverwerking om beeldinligting te onttrek sonder dat die beeld eers in geheue gestoor moet word kan die verwerking tyd verminder word.

Die voorgestelde merker-opsporingstelsel het vergelykbare akkuraatheid teenoor ander merker-opsporingstelsels wat die prestasie van die verrigting van die voorgestelde stelsel verifieer. Die ontwerpte en geïmplementeerde stelsel is getoets en die stelsel is geverifieer om teen 'n gemiddelde raamsnelheid van 134 FPS te werk.

Acknowledgements

I would like to express my sincere gratitude to the following people and organisations:

- My supervisors Dr. Arno Barnard and Dr. Willem Jordaan for their guidance, encouragement and mentoring throughout this whole process.
- Stellenbosch University for financial aid during my studies and for allowing me to take my workstation computers and equipment home with me during covid lockdowns.
- My family for their financial and moral support during this process. Working through and completing my thesis at home was made possible through their support and encouragement. They provided me a space where I could focus completely on my project.

Contents

List of Figures

List of Tables

Nomenclature

Acronyms

3D	Three-Dimensional
AGV	Autonomous Ground Vehicle
AR	Augmented Reality
AXI	Advanced eXtensible Interface
AXIS	AXI Stream
BRAM	Block RAM
CPU	Central Processing Unit
DMA	Direct Memory Access
DUT	Design Under Test
FAST	Features from Accelerated Segment Test
FF	Flip-Flop
FIFO	First In First Out
FPGA	Field Programmable Gate Array
FPS	Frames Per Second
GPIO	General Purpose Input Output
GPS	Global Positioning System
GPU	Graphics Processing Unit
ID	Identification
IMU	Inertial Measurement Units
IP	Intellectual Property
LUT	Look Up Table
LUTRAM	LUT RAM
MM	Memory Mapped
NMS	Non-maximum suppression
OBC	On-Board Computer
PC	Personal Computer
PL	Programmable Logic
PMU	Performance Monitoring Unit

POI	Point of Interest
POS	Pose from Orthography and Scaling
POSIT	Pose from Orthography and Scaling Iterative
PS	Processing System
RAM	Random Access Memory
SLAM	Simultaneous Localisation and Mapping
SoC	System on a Chip
SVD	Singular Value Decomposition
UAV	Unmanned Aerial Vehicle
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit Program

Chapter 1

Introduction

1.1 Background

Robotics has revolutionised industrial processes by allowing complex tasks to be repeated rapidly without loss of quality. The automation of manufacturing processes using assembly line robots has allowed for a great increase in production while limiting human intervention. These robots that execute complex tasks operate with minimal sensory feedback and are mostly unaware of their surroundings and this lack of awareness prevents them from operating in new or volatile environments. These flaws of conventional robotics are overcome by incorporating autonomous systems [23]. An autonomous system is any electro-mechanical system that has the capability to complete specified tasks automatically while adapting to changes in the environment [24].

Autonomous systems must solve the problem of navigation to achieve mobility. Navigation as a problem can be broken down into three components: pose destination and method. The pose of the system is its current position and orientation in the world or relative to landmarks and is determined through localisation [25]. The destination is a specified objective or goal of that system, while the method handles how the specific objective is achieved. Pose is a precursor to the system's method as a system must first determine its current pose to formulate a plan to achieve its objective. Being a vital component of navigation, the determination of a system's pose is therefore a fundamental challenge regarding the mobility of autonomous systems.

Landmark based localisation by using fiducial markers has an ever-growing utility in various industries. Examples of autonomous systems that use fiducial markers are small autonomous Unmanned Aerial Vehicles (UAV) and Augmented Reality (AR) systems. Small autonomous systems use fiducial markers to identify landmarks in the environment for pose determination and for precision tasks that involve stable position control e.g. auto-docking in recharge stations and maneuvering in spatially restrictive environments. AR systems use fiducial markers to determine camera pose at high frame rates to over-

lay Three-Dimensional (3D) graphics onto the image frame. Tracking systems consist of any autonomous system that uses computer vision to track the movement of objects in the environment to perform tasks.

1.2 Project Scope

Current fiducial marker detection solutions achieve frame rates of around 9-45 Frames Per Second (FPS) at image resolutions of 480p (640×480) or higher. Higher resolutions result in higher processing times. These marker detections systems are implemented on computers or GPUs [26, 21, 2]. Implemented systems have an average fiducial marker pose estimation translation error is 4.36 cm and the average rotational error is 1.5° [1].

Frame rates of <45 FPS are suitable for landmarking where the landmark information is used to supplement an existing pose determination system made up of Inertial Measurement Units (IMU), but the frame rate is too low for UAV stability where only the marker detection system supplies pose information. Small autonomous UAVs have position and altitude controllers that run at frequencies of 100Hz [27, 28]. The faster pose information is sent to the position and altitude controllers the faster the system can react to accomplish higher accuracy related tasks. AR systems require high frame rates to reduce 3D graphic jitter. Humans can perceive up to about 60 FPS and notice lag or jittery graphics at lower frame rates [29]. AR systems therefore require pose information at rates higher than 60 FPS to process the graphical overlay and display it at a fast enough rate to be perceived as smooth and natural.

Pose estimation from fiducial markers are currently limited to <45 FPS while being run on hardware platforms that are not suitable for light-weight applications. The hardware and low frame rate limit the utility of fiducial markers. The utility of fiducial markers for autonomous systems and wearable tech is increased by increasing the framerate and by designing the marker detection platform to be light-weight, compact and low-power.

1.3 Project Definition

This project aims to design a marker detection system that runs on a light-weight-, compact-, and low-power hardware platform that can detect fiducial markers and estimate relative pose at a frame rate higher than 60 FPS. The design aims to operate at an accuracy comparable to existing solutions. The project provides a detailed design and implementation of the marker detection system. Verification of the implemented system is done to characterise its performance and accuracy. To ensure the successful completion of the project the process was broken down into project objectives. The objectives are shown in the next section.

1.4 Objectives

1. Identify existing fiducial marker detection methods.
2. Identify a hardware platform best suited for a mobile marker detection system.
3. Design a high frame rate fiducial marker detection system that operates at comparable accuracies to existing solutions.
4. Implement the designed fiducial marker detection system on the chosen hardware.
5. Test and analyse the functionality and performance of the implemented fiducial marker detection system.

1.5 Thesis Summary

- Chapter 2 is the literature review that details the research and discussion into relevant marker detection methods and implementations to determine the performance of current marker detection systems.
- Chapter 3 details the concept design of the proposed marker detection system. The concept is designed by using the information gathered in the literature review and choosing/modifying the detection methods to suite the chosen hardware platform and proposed solution.
- Chapter 4 details the hardware implementation of the marker detection concept. The hardware usage of the design is also analysed.
- Chapter 5 details the testing process and discusses the results of the marker detection tests. The test results are provided in the form of graphical comparisons between the measured data and test variables.
- Chapter 6 provides a conclusion to the project and discusses future work and system improvements.

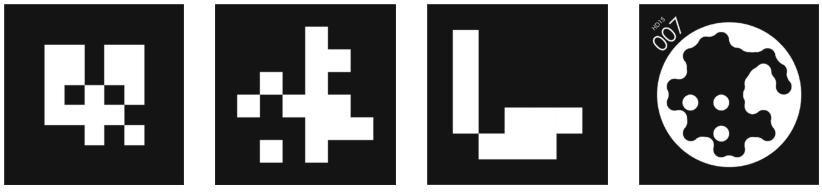
Chapter 2

Literature Review

To further understand the problem stated in Section 1.2 and formulate a solution to meet the set objectives listed in Section 1.4 relevant literature is investigated. Several types of fiducial markers and implementations of marker detection systems are investigated and discussed. The performance, accuracy, system hardware and marker types are identified to provide a starting point of the design and to provide benchmarks to verify the functionality of the completed system.

2.1 Fiducial Markers

Fiducial markers provide recognisable features in a autonomous system's operating environment. Four of the most widely used marker detection systems, examples shown in Figure 2.1, are presented in [1] and are evaluated to compare the different system's operating accuracy.



(a) ARTag (b) AprilTag (c) ArUco (d) STag

Figure 2.1: Examples of four of the most commonly used fiducial markers [1]

The four different marker detection systems are evaluated by comparing pose error test results. The hardware used for the test setup are compact processing platforms used for on-board platforms. The hardware platforms are a Nvidia Jetson TX2 (powerful Graphics Processing Unit (GPU) for compact form factor), a Raspberry Pi 3B+, and an Intel NUC (powerful Central Processing Unit (CPU)).

The position error from the distance tests are shown in Table 2.1. The average position error of all the fiducial markers is 2.14 cm.

	Distance (cm)				
Marker	75	100	150	175	200
ARTag	1.431	1.954	2.617	1.768	2.697
AprilTag	2.142	2.541	3.292	3.110	3.510
ArUco	1.284	1.562	3.632	3.827	3.78
STag	0.737	0.842	0.475	0.093	1.542

Table 2.1: Position error of tested marker detection techniques [1]

The rotation error from the rotation tests are shown in Table 2.2. The average rotation error of all the fiducial markers is 0.39°.

	Angle (°)							
Marker	0	10	20	30	40	50	60	70
ARTag	1.883	1.532	0.062	0.256	0.092	0.440	0.163	0.131
AprilTag	0.649	0.476	0.028	0.035	0.034	0.519	0.022	0.137
ArUco	0.602	1.024	0.138	0.765	0.296	0.134	0.528	0.299
STag	0.627	1.261	0.144	0.494	0.457	0.064	0.064	0.324

Table 2.2: Rotation error of tested marker detection techniques [1]

The summary of the experimental results, from [1], are listed in Table 2.3.

Marker	Pros	Cons
ARTag	Lowest Computational Cost	Low detection rate Extreme outliers
AprilTag	Accurate Pose results Great detection rate	Computationally expensive
ArUco	Accurate Pose results Great detection rate Low computational rate	Computational cost scales with detected marker number Sensitive to smaller marker sizes and larger distances
STag	Accurate Pose results Great detection rate	Sensitive to smaller marker sizes and larger distances

Table 2.3: Comparison summary from experimental results [1]

2.2 Marker Detection Systems

An overview of pose accuracy and system performance is provided in section above, Section 2.1. The discussed research article comparing the main fiducial marker systems does not provide timing analysis of each marker system. Fiducial marker systems are therefore further researched and discussed to provide a better understanding regarding fiducial marker detection timing performance.

2.2.1 ARTag

The ARTag fiducial marker system is designed to lower inter-marker confusion and false positive rates in smaller marker sizes by using digital code theory. The ARTag is a marker containing an encoded Identity (ID) number [30].

The detection processing time of ARTag markers are dependent on hardware platforms and number of detectable markers but the typical performance rate is 10-50 ms (100 - 20 Hz respectively)[31].

2.2.2 AprilTag

AprilTag is an improvement to the ARTag system by implementing marker information encoding system that reduces the false positive rate and makes the marker detectable at any rotation.

The tag detector was tested to verify that the AprilTag detection system works. The tag detector tests are run on an Intel Xeon E5-2640 2.5 GHz CPU. For a 640×480 image the detection time is 22 ms (45.45 Hz) [26].

2.2.3 ArUco

The ArUco fiducial marker system, presenting in [14], is designed specifically for camera pose estimation. The ArUco marker is designed to have encoded information that has the largest inter-marker distance to reduce false positives. The marker detection system is also designed to accurately detect marker corners to provide pose estimation algorithm a more accurate input.

The hardware used to test the marker detection system is an Intel Core 2 Quad processor (2.4 GHz) with 2048 MB of Random Access Memory (RAM). The marker detection system was tested using an image with a resolution of 640×480 and the average detection processing time was recorded as 11.08 ms (90.25 Hz) [14].

2.2.4 STag

The STag fiducial marker system proposed in [32] is designed to provide stable pose estimation. The system is designed to be robust against jitter and to therefore provide sustainably stable pose estimation.

The STag fiducial marker was tested using a single core of an Intel Xeon processor (3.7 GHz) with images with a resolution of 1280×720. The processing time of the STag system is 18.1 ms (55.24 Hz) with the detection algorithm being run on an image containing a single marker [32].

2.2.5 TopoTag

The TopoTag, presented in [2], is a topological-based fiducial marker. The marker systems achieve accurate detection and high robustness by utilising topological information. The topological information used to identify TopoTags is shown in Figure 2.2.

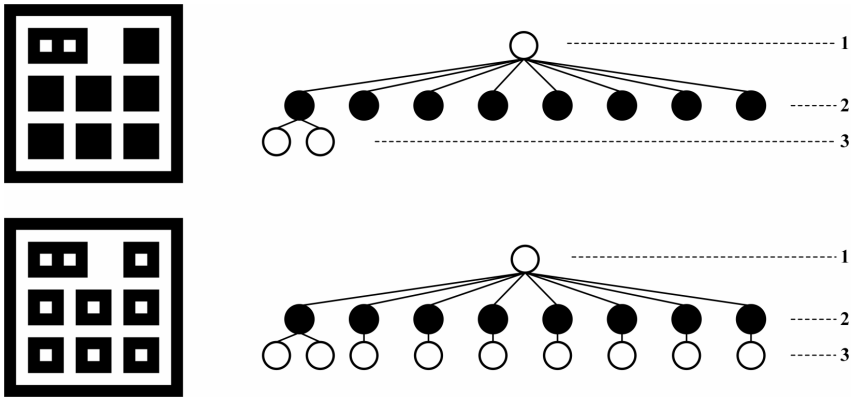


Figure 2.2: Examples of topological information in TopoTags [2]

The TopoTag solution and other current marker detection systems were implemented on a laptop Personal Computer (PC) with an Intel Core i7-7700HQ 2.8 GHz 8 core CPU and 8 GB RAM to test the systems and compare them. The timing results of the TopoTag comparison tests are shown in Table 2.4.

Marker	ArUco	AprilTag	TopoTag
Time (ms)	54.319	15.115	33.638
Rate (Hz)	18.41	66.159	29.728

Table 2.4: Timing results of TopoTag comparison tests [2]

2.2.6 DeepTag

The DeepTag marker detection technique proposed in [21] discusses a general deep learning-based framework that improves the flexibility and robustness of existing marker detection systems.

The DeepTag framework can be used to detect markers and design new fiducial markers. The DeepTag solution was implemented on a Personal Computer (PC) with an Intel i7-11700L 6.6 GHz CPU and 64 GB Random Access Memory (RAM), and a NVIDIA GeForce GTX 3090 GPU.

The timing results of the DeepTag solution is 23 Hz for a single marker at an image resolution 640×480 and 40 Hz at 640×480 regardless of number of markers if the ID of the markers in the image are known [21]. The accuracy results of the DeepTag marker detection solution are shown in Table 2.5.

Marker	Position Error (mm)	Rotation Error (°)
TopoTag	0.889	0.228
ARToolKit	1.010	0.566
ArUco	0.628	0.716
AprilTag	0.937	0.462

Table 2.5: Pose results of DeepTag marker detection system [21]

2.2.7 Concentric Circle Patches

The fiducial marker consisting of detectable concentric circles, presented in [3], provides a high-speed low information fiducial marker system. The presented fiducial marker is shown in Figure 2.3.

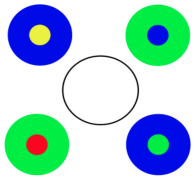


Figure 2.3: The Concentric Circle fiducial marker [3]

The fiducial marker is designed to track moving objects and therefore has a small detection time. The presented marker is simple and easy to detect containing minimal encoded information compared to ArUco markers and AprilTags. Using a computer, an Intel i7 3.6 GHz CPU with 16 GB RAM, the presented system was tested against existing fiducial markers (ArUco and AprilTags). The average computation times for the ArUco marker, AprilTag and presented markers were 26 ms, 30 ms and 5.1 ms respectively (38.5 Hz, 33.33 Hz and 196.1 Hz).

2.3 Implemented Marker Detection Systems

Previously implemented marker detection systems are researched and discussed to highlight the different applications of fiducial marker-based localisation systems. The discussed implemented marker detections systems provide insight into the accuracy, performance and limitations of previously implemented marker detection systems.

Autonomous robot applications that have weight, size and power consumption limitations require compact-, light-weight- and low-power electronic components. Marker detection systems for these applications must therefore be implemented on smaller less powerful hardware.

2.3.1 Automated Recharging for a Quadrotor UAV

The automated recharging system, presented in [4], uses marker-based Simultaneous Localisation and Mapping (SLAM) to autonomously land on the recharge station for automated recharging. The marker detection system is run on a Jetson Nano, a GPU, which acts as the On-Board Computer (OBC).

The marker-based localisation uses a large black square border containing an ArUco marker, as shown in Figure 2.4. The large square border is used to locate the landing region from a further distance. The ArUco marker is then used for more accurate position feedback for accurate landing.

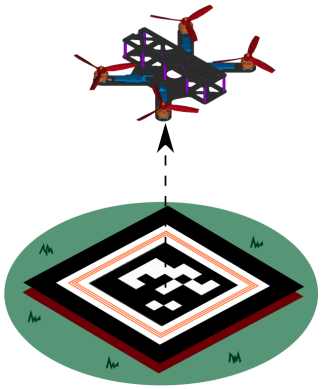


Figure 2.4: Example of the fiducial marker used in the automated recharge station for autonomous landing [4]

The results of the practical landing tests show that the system has an an average landing accuracy of 4 cm.

The automated recharging solution has an average marker detection rate of 17.3 Hz using a GPU on-board a Quadrotor UAV [4].

2.3.2 Automated Landing for Quadrotor UAV

The on-board vision-based system, presented in [33], uses ArUco markers placed on a landing pad to assist the autonomous landing system of a UAV. The vision-based system consists of the ArUco markers, an on-board camera and an on-board processing platform (Intel NUC).

The on-board detection system can detect ArUco markers at a framerate of 25 Hz [33]. The accuracy of the measured position of the drone is 23.2 mm at a hovering height of 1 m away from the marker.

2.3.3 Localisation for Satellite Experiments

The satellite testing facility, presented in [5], estimates the pose of the air bearing vehicle using ArUco markers. An example of the ArUco marker used for pose estimation mounted on the air bearing vehicle is shown in Figure 2.5. The air bearing vehicle is used for space and space system testing by emulating a satellite with three degrees of freedom using low friction air bearings on a glass platform.

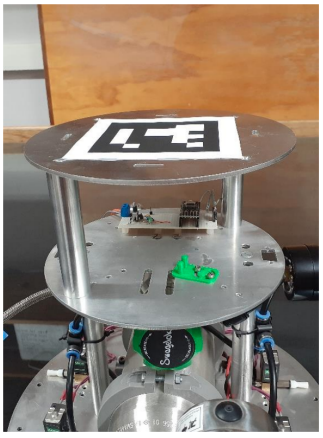


Figure 2.5: Picture of the ArUco marker placed on the air bearing vehicle [5]

A stationary camera is used to capture images to identify the ArUco markers of the air bearing vehicle and origin position to determine the pose of the air bearing vehicle relative to the origin position. The camera has an image resolution of 1440×1080 and has a maximum framerate of 238 Hz using USB 3.1 connected to a camera driver Robot Operating System (ROS) node run on a PC.

The tested average position error is 1.2 cm and the average rotation error of 2.29°. The ArUco marker pose estimation node can output at 60 Hz but 10 Hz is used as the output frequency because that is the required by the payload control system [5].

2.3.4 Vision-Based Localisation for Subsea Intervention

The subsea intervention system, presented in [6], is used to perform underwater localisation for the automation of tasks performed on subsea facilities. The subsea system uses a vision-based localisation method that detects fiducial markers. The camera is mounted on a Seabotic LBV 600 aquatic vehicle. An example of the camera view used to identify fiducial markers is shown in Figure 2.6.

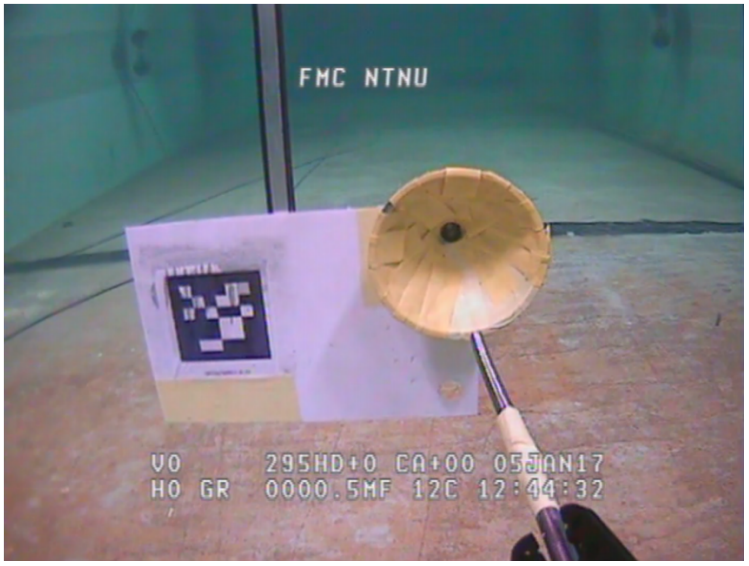


Figure 2.6: Camera view of fiducial marker used for subsea alignment [6]

The system uses a camera sensor with a resolution of 640×480 that streams the video data through an ethernet link to a topside computer. The computer is a laptop with an Intel Core i7 with a 2.7 GHz CPU.

The accuracy of the system was tested and the results of tests are shown in Table 2.6. The rate of detection is not documented.

Direction	Average Error	Standard Deviation
X	0.0186 mm	2.7661 mm
Y	-0.0382 mm	11.4338 mm
Z	-0.0205 mm	4.4481 mm
Yaw	0.0191°	0.8037°

Table 2.6: Accuracy results of Subsea marker detection tests [6]

2.3.5 Vision-Based Flight Control for a Quadrotor UAV

The vision-based control system, presented in [34], is used to perform waypoint navigation and flight control of a Quadrotor UAV. The system was developed to provide the navigation for an inspection drone that can navigate around an inspection target. The system was designed to operate in indoor or Global Position System (GPS) restrictive environments.

The navigational system is built around a vision-based pose estimator. The pose estimation algorithm uses ArUco markers to estimate pose and is run on a companion computer. The camera that is used as the vision sensor has a resolution of 640×480.

The accuracy of the pose estimation system was determined through experiments. The position accuracy of the system is 50 mm and the rotation accuracy is 2.5°. The time taken to process the captured images and estimate the pose is 100 ms giving a detection and pose estimation rate of 10 Hz [34].

2.3.6 Localisation for Cooperative Navigation

The cooperative navigation system, presented in [7], uses two stationary cameras and ArUco marker boxes mounted on Autonomous Ground Vehicles (AGV) to perform cooperative navigation of several AGVS in the same environment. The system identifies the pose of the different AGVs using the ArUco markers and plans each individual AGVs' path to move to its specific locations while avoiding the other AGVs. The AGVs, shown in Figure 2.7, are given movement commands from the system to move to their specified positions while avoiding the other AGVs.

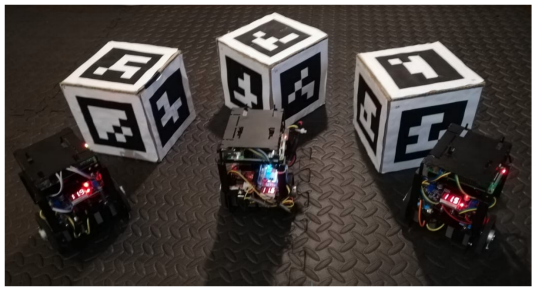


Figure 2.7: The AGVs and ArUco marker boxes used to test the cooperative planning system [7]

The two cameras that are used are smartphone cameras (Sony Xperia Z3 Compact and Huawei P20 Lite). The pose estimations are done on the smartphones and the resulting pose information is sent to the pose ROS topic to be used for the path planning of the system. The smartphones detect and estimate pose at a frequency of 15 Hz. The implemented pose estimation system has a position accuracy of 2 cm and a rotation accuracy of 2° [7].

2.4 Literature Review Conclusion

The investigation into and discussion regarding related work provides a better understanding of marker detection systems. The literature review presents the currently implemented systems performance, accuracy and limitations.

The comparison between the most used markers, presented in Table 2.3, shows that the ArUco marker system is the least computationally expensive while still retaining accurate pose estimation results.

The average detection rate for all the investigated marker detection systems is 45 Hz (45 FPS is achieved with most tests on powerful hardware). The average detection rate for ArUco marker systems is 33.67 Hz (33.67 FPS).

From the discussed marker detection systems and implementations, the average position accuracy is 3 cm and average rotation accuracy is 1.8°. The accuracy of the previously implemented systems are used to verify that the timing information is significant and comparable.

The current marker detection systems are tested and implemented on powerful hardware that is not suitable for small autonomous systems that operate without a companion computer. Autonomous systems that require compact-, light-weight- and low-power electronic components cannot implement detection systems that require heave immobile hardware. Most of the implementations are run on computers with some running on GPUs and smartphones.

Chapter 3

Conceptual Design

To solve the problem posed the steps required to go from the chosen input to the desired output need to be identified and expanded on. The conceptual design of the system breaks down the functional steps of the solution into smaller sub elements that are further discussed.

Background literature, provided in the Appendix A, provides background information and gives a more in detail look into the various methods, techniques and technology used to design the concept of the marker detection solution.

3.1 Design Solution

For the proposed solution an appropriate marker detection system and hardware platform was chosen with performance, accuracy, size, weight, power consumption and system price in mind.

3.1.1 Localisation and Marker Detection

For this work the system is operated in controlled environments which allows for landmark-based localisation to be used. A single camera connected to the hardware platform provides a light-weight-, low-power-, compact and cost-effective sensor that can identify markers placed in the environment.

From Section 2.1 it is shown that ArUco marker is a suitable marker detection system for the proposed solution. More information regarding the ArUco marker is provided in Appendix A.2.

The ArUco marker allows for pose estimation by providing four detectable points in the environment (the four corners of the marker). The four corners are used to estimate pose of the camera with regard to the marker. The pose estimation algorithm must therefore be able to estimate pose by using four coplanar points.

3.1.2 Hardware Platform

Image sensors supply large amounts of data that need to be processed. A suitable hardware platform is therefore required to process the large amount of data, Appendix A.1 provides an overview of the potential platforms.

To process the large amount of data at the required speed a Field Programmable Gate Array (FPGA) was chosen as the hardware platform. An FPGA can process large datasets very quickly just like a GPU can but for a localisation platform that requires a low-power-, light-weight- and compact solution the FPGA was the clear option [35, 36].

The FPGA is well suited for high bandwidth low complexity algorithms, but to calculate the system's pose an iterative Pose from Orthography and Scaling Iterative (POSIT) algorithm is required. A hybrid solution is therefore proposed. A platform that contains an FPGA and CPU is used. The FPGA handles the low-level image processing and feature extraction and the CPU handles the high-level marker identification and pose estimation.

3.2 System Overview

The process of determining the pose and marker ID from a single camera sensor can be broken down into five blocks shown in Figure 3.1. The output of the system is the pose information of the system relative to the marker and the ID of that marker for landmarking purposes. The input of the system is a single monocular camera stream.

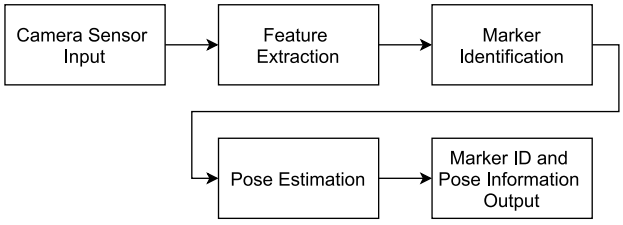


Figure 3.1: Functional flow of the proposed system

Using the required output and the chosen form of input the process required to manipulate the data is determined. To extract information from an image, points of interest in that image need to be identified. These points of interest are referred to as features. The extracted features are evaluated to determine the group of features that represent a potential marker. The potential markers are assessed and valid markers are identified. The valid marker information (corner points and orientation) is used to estimate the system's pose.

The marker detection concept is designed using the ArUco marker detection algorithm, provided in Appendix A.2, and algorithms that are suited for FPGA hardware acceleration and parallelisation.

3.3 Camera Sensor Input

A camera sensor contains a series of sensors that each capture the intensity of light. An 8-bit depth grayscale image consists of pixel intensity values in a two-dimensional array shown in Figure 3.2. To extract useful information from an image the pixels must be evaluated or manipulated. By working from image processing first principles and existing marker detection algorithms marker information can be extracted from an image.

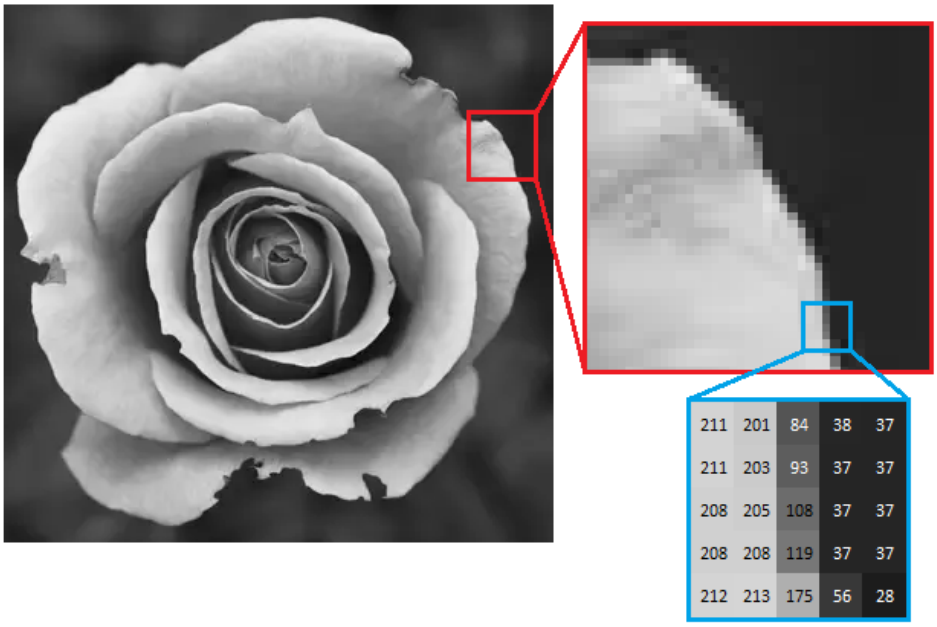


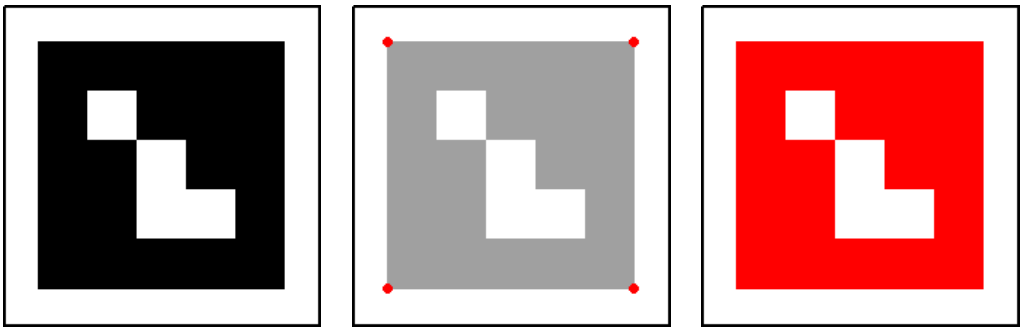
Figure 3.2: Pixel level image breakdown [8]

Computer vision systems capture the digital image and store it in memory. The memory is then accessed and the image is processed. To increase the performance of the marker detection solution the camera sensor input is optimised. The camera sensor information is streamed into the FPGA and processed inline. The pixel data is therefore processed as it is streamed into the FPGA removing the requirement to store the whole image to internal memory first.

By removing the requirement to store a whole image in internal memory and processing the image as it is streamed into the hardware the overall processing time of the system can be reduced significantly.

3.4 Feature Extraction

The marker that is placed in the environment to be detected is a black square surrounded by a white border, shown in Figure 3.3a. The two identifiable features of the marker are the marker's outer corners, shown in Figure 3.3b, and the marker's black body shown in 3.3c. These two features can easily be identified using existing image processing techniques (common techniques discussed in A.3) and are the basic building blocks of the marker.



(a) Basic Marker (b) Outer Corners (c) Body of Marker

Figure 3.3: Features of interest of the marker (a), namely the corners (b) and the body (c), are indicated in red

3.4.1 Feature Extraction Overview

The process to extract the required features from an image is shown in the Figure 3.4. Corner Detection and Blob Detection operate independently from one another. The Corner Detector detects corners from a grayscale image stream while the Blob Detector detects blobs from a binary image stream and therefore the grayscale image stream must first be thresholded. The extracted features are then passed on to the next system functional block to be evaluated in the next step.

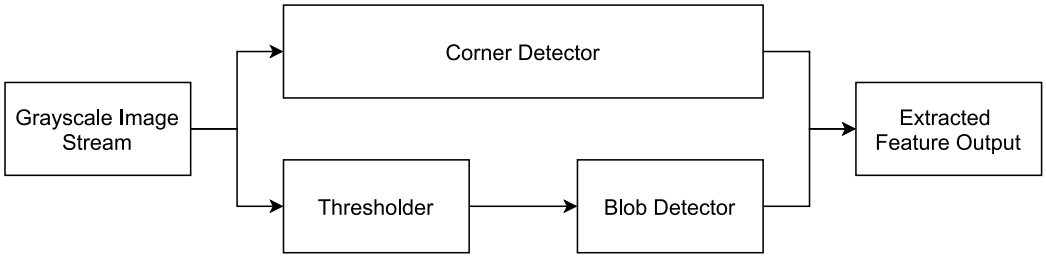
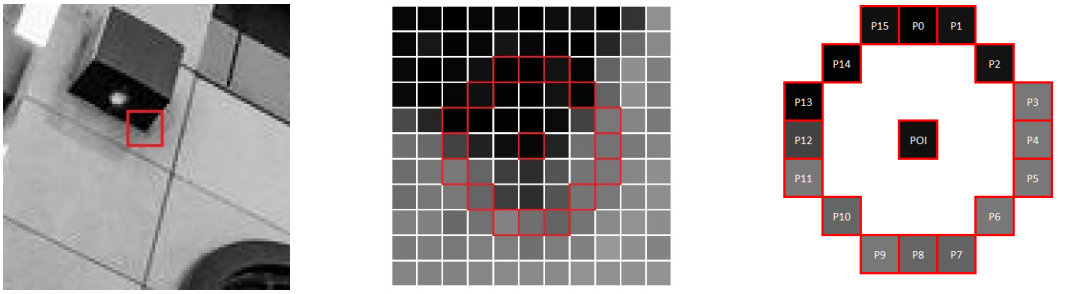


Figure 3.4: Functional flow of Feature Extraction algorithm

3.4.2 Corner Detection

To detect the corners of a marker the Features from Accelerated Segment Test (FAST) algorithm is used. Corner detection algorithms are discussed further in Appendix A.3.2.1. FAST is chosen over the Harris Corner Detector as it provides an algorithmically simple solution that is fast and accurate. FAST detects corners in the image, but it also provides additional corner information that is used to narrow down the potential corner and marker candidates to reduce processing requirements for the next steps. FAST labels all pixels in an image that fulfils the corner criteria as a corner. Figure 3.5 shows an example of a corner in an image fulfilling the required criteria.



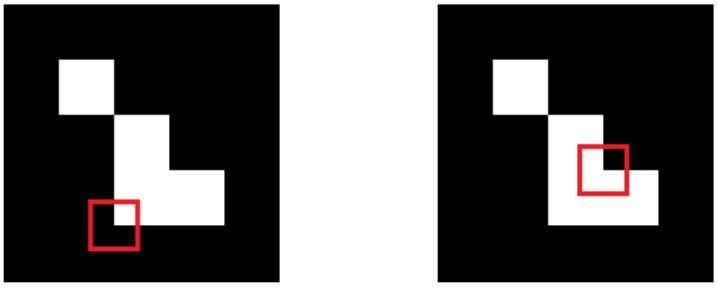
(a) Detectable corner in image [37]

(b) Pixel level breakdown of corner

(c) Bresenham circle of evaluated pixels

Figure 3.5: Example of pixel level evaluation region for Corner Detection

Corner Detector Modification The ArUco marker has a dark square body on a light background and therefore only the four outer dark corners need to be detected for a potential marker to be identified. The FAST algorithm is therefore modified to only identify dark corners reducing the complexity of the algorithm and number of detected corners in the image. With the modified algorithm all the bright corners, Figure 3.6a, are ignored and the dark corners, Figure 3.6b, are detected.



(a) Bright corner

(b) Dark corner

Figure 3.6: Types of corners indicated by a red square around the corner

Figure 3.7 is used as an input image for the unmodified corner detection algorithm. The corners that are detected in the sample image are shown in Figure 3.8. The red pixels are dark corners and the green pixels are bright corners. The image shows that the modification made to the original FAST algorithm greatly reduces the number of corners that can be detected. By detecting only the dark corners the processing time to evaluate all the corners is reduced.

Figure 3.7: Sample grayscale image [9]

Figure 3.8: Dark and bright corners detected

Corner Strength Multiple pixels in adjacent locations can all be classified as corners. To filter out and find the best corner each corner that is detected has a corner score calculated using Equation 3.1. Non-maximal suppression is used on a group of corners and all the weaker corners are discarded leaving a single strong corner for feature evaluation.

The corner score is also used to determine the most suitable corner in each quadrant during feature evaluation. The corner score equation was modified because the corner detector is only looking for dark corners and therefore only half the corner score equation used by FAST is required (Equation A.4). The corner score is given by the modified equation: Equation 3.1.

$$C_s = \sum_{x \in S_{\text{Bright}}} |I_{p \to x} - I_p| - t \tag{3.1}$$

where:

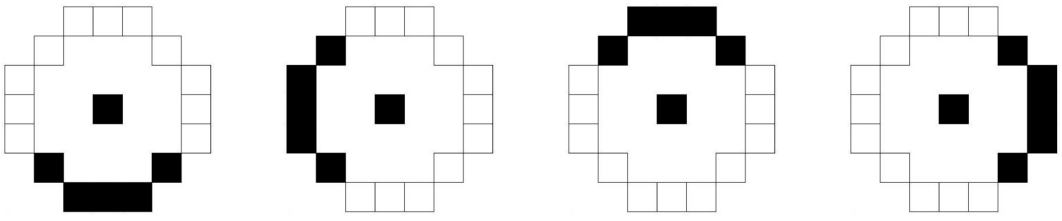
C_s is the Corner Strength,

$S_{\textbf{Bright}}$ are all the pixels in the Bresenham Circle that are bright,

I_p is the Pixel Intensity of the Point of Interest (POI),

t is the Threshold Value (chosen value).

Corner Direction The corner direction refers to the direction in which the tip of the corner is pointing. Feature evaluation only requires the four cardinal directions (north, south, east and west). Corner directions are determined by counting the number of dark pixels in the four directions of the Bresenham Circle as shown in Figure 3.9. The region with the most dark pixels is chosen as the corner's direction. In the case of a corner with two equal dark regions (north-west direction) North is chosen, because the feature evaluation step looks for two directions in each quadrant and choosing one arbitrarily does not change the outcome of the evaluation.



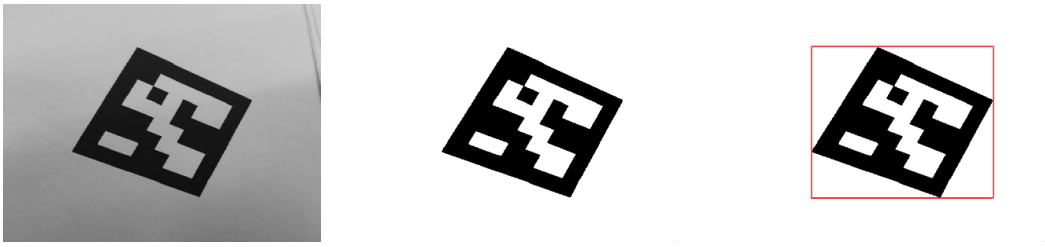
(a) North (b) East (c) South (d) West

Figure 3.9: The four directions a corner can be determined to have

3.4.3 Blob Detection

Blobs are any group of connected dark pixels in a binary image. After thresholding an image all dark objects in the image are seen as blobs. The proposed marker has a black square body bordered by white and is therefore detected as a blob.

Blobs can be found by using connected component analysis. All the connected dark pixels are grouped together under the same label and the blobs information is gathered. Figure 3.10 shows an example image before and after thresholding and the result of the blob detector. The output is a blob encased in a bounding box.



(a) Example image of marker

(b) Binary image after thresholding

(c) Detected blob in bounding Box

Figure 3.10: Example of thresholding (b) a sample image (a) and the output of the blob detector which is a bounding box (c)

The following subsections provide more details on each of the block detection phases.

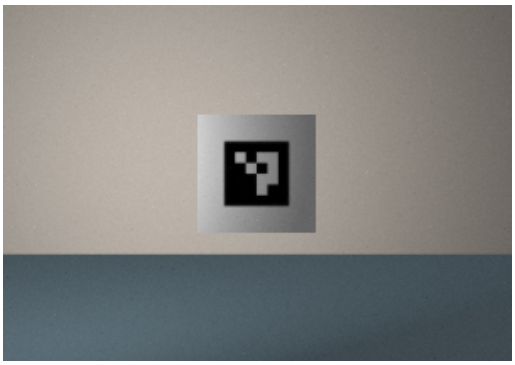
Grouping Grouping (discussed in Appendix A.3.2.2) is chosen as the method of blob detection because it can be implemented without requiring multiple passes through the image. Border Following requires the image to be stored to follow the border of the blob. The grouping method is the most suitable solution for inline processing run on an FPGA.

There are more advanced blob detection techniques that merge labels for complex blob shapes, but a square marker is a simple shape that is easy to detect and label. The whole marker might not be labelled correctly as information stored in the marker can cause multiple detected regions, but it still detects the complete outer square of the marker. For feature evaluation only the outer region is required to be detected. For the connected component labelling each pixel is labelled by looking at the adjacent pixels and grouping them together.

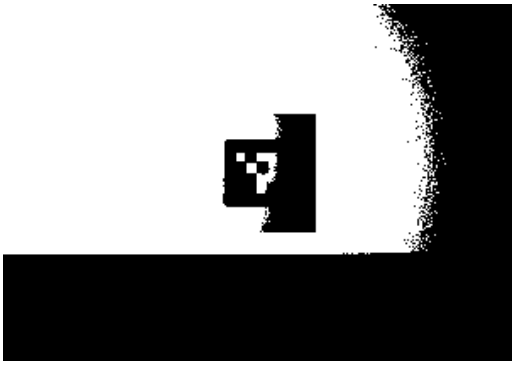
Thresholding The Blob Detector requires the image to be binarised and therefore thresholding is used, discussed in Appendix A.3.1. Global thresholding is used to threshold the image. Global thresholding uses the intensity histogram of the entire frame and calculates the threshold value that best divides the foreground (dark objects) from the background. Local adaptive thresholding does not work for the grouping method discussed in the previous paragraph (Section 3.4.3)

Global thresholding requires the complete image to be processed, but by using the data from the previous frame the threshold value can be calculated for the next incoming frame. The assumption that the previous frame is not that much different from the current frame is made. Using a static threshold value has a higher chance of causing erroneous blob detection results.

Figure 3.11 shows the difference between the static and global thresholding methods used on sample Figure 3.11a. Using a static threshold value of 127 the marker blob is merged with the shadow (Figure 3.11b). The global threshold value of 82 is determined by examining the complete image frame and outputs a clearer marker blob shown in Figure 3.11c.



(a) Shadow across simulated marker



(b) Threshold using static threshold value

(c) Threshold using dynamic threshold value

Figure 3.11: Difference between static and global thresholding

Bounding Box Every blob that is detected in the image is given a unique label and is enclosed in a bounding box. The bounding box shown in Figure 3.12 is the box that visually represents the stored coordinate limits of the blob. The start (Sx, Sy) and end (Ex, Ey) coordinates of the bounding box are stored along with the unique blob label for further evaluation.

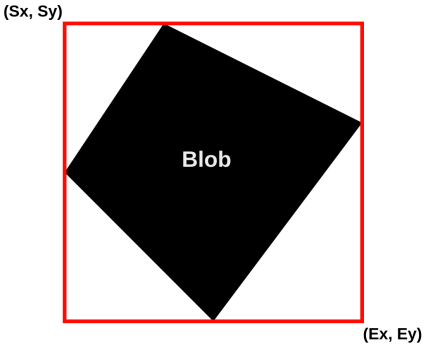


Figure 3.12: Example of bounding box around detected blob

3.5 Marker Identification

The functional flow of marker identification is shown in Figure 3.13. To identify markers in the image the detected features are evaluated to determine all the potential markers. Information must be extracted from the markers to determine if the potential markers are valid. The potential markers can have perspective distortion and must first be rectified to allow for information extraction.

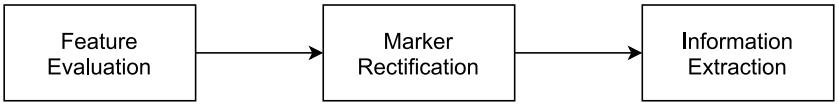


Figure 3.13: Functional flow of Marker Identification algorithm

3.5.1 Feature Evaluation

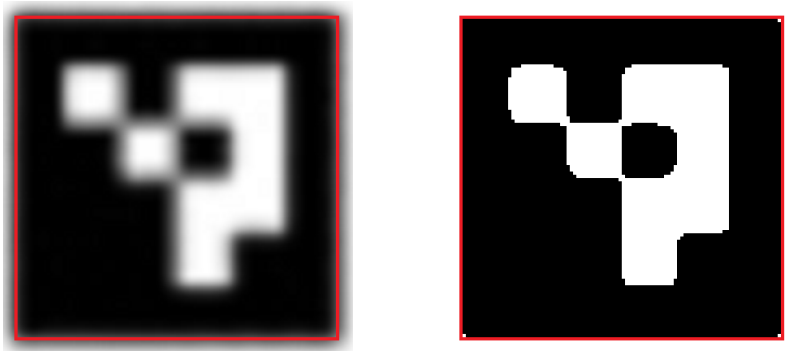
A bounding box contains a blob that is identified in the image. If a marker is visible in the image the body of the marker is classified as a blob and is labelled. Other objects in the environment can also be labelled as blobs. For a blob to be classified as a potential marker the blob must have 4 corners that fall in each of the four quadrants of the bounding box. To filter out blobs that are not potential markers the corners detected within the region of the bounding box is evaluated. Algorithm 1 describes the steps taken to evaluate the features.

Algorithm 1 Feature Evaluation

Require: Information of all corners and bounding boxes in the image
Ensure: Candidate markers with four corners

```
Set all corners to unassigned
Set all quadrant corner scores to zero
repeat
    repeat
        if Corner is unassigned then
            repeat
                if Corner is within quadrant and corner direction is suitable then
                    Calculate current corner score
                    if Current corner score is greater than quadrant corner score
                    then
                        Set quadrant corner score to current corner score
                        Set current corner to assigned
                        Set previous corner that was assigned as quadrant corner to
                        unassigned
                    end if
                end if
            until All quadrants have been evaluated
        end if
    until All corners have been evaluated
until All bounding boxes have been evaluated
```

Marker Area Reduction If a marker is blurry some of the actual area of the marker can be reduced due to thresholding. The thresholding removes the blurry edges and reduces the overall area as shown in Figure 3.14.



(a) Original image with bounding box overlay

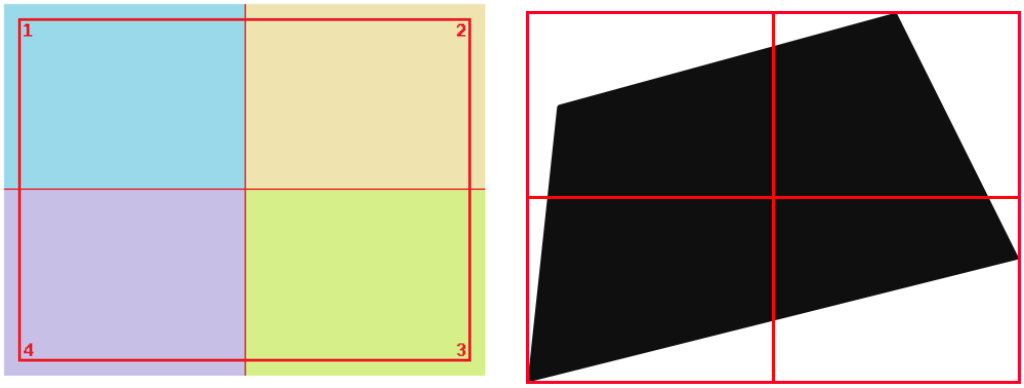
(b) Thresholded image with detected bounding box

Figure 3.14: Example of how thresholding removes the blurry edges of a marker reducing the marker area

Bounding Box Quadrants The bounding box is used to evaluate and identify the marker corners shown in Figure 3.15. The bounding box is shown in figure 3.15a. The region on the outside of the bounding box accounts for the reduction of actual marker area due to thresholding. This prevents corners that were detected that lie outside the bounding box from being discarded.

The internal error region accounts for the perspective distortion of a marker where not all four corners of the marker are in contact with the border of the bounding box shown in 3.15b.

The four quadrants allow for the four corners of the marker to be found and ordered in the correct order for rectification.



(a) Quadrants of the bounding box (b) Perspective distorter marker

Figure 3.15: Bounding box used for Feature Evaluation

Corner Evaluation The information gathered for each corner detected in the image is used to find the most suitable corner in each quadrant to determine if the blob is a potential marker. The corner's X and Y coordinates are used to determine which quadrant the corner is located in. The direction of the corner determines if the corner located in a specific quadrant is facing a suitable direction (Table 3.1) relative to that quadrant.

Quadrant	Suitable Corner Direction
1	North/East
2	North/West
3	South/West
4	South/East

Table 3.1: Suitable corner directions by quadrant

Corner Score The corner score is calculated by taking the strength of the corner and multiplying it by a scaling factor that is dependent on the corner's location (Cx, Cy) within a quadrant. Figure 3.16 shows an example of a corner detected in quadrant 1 (top left quadrant).

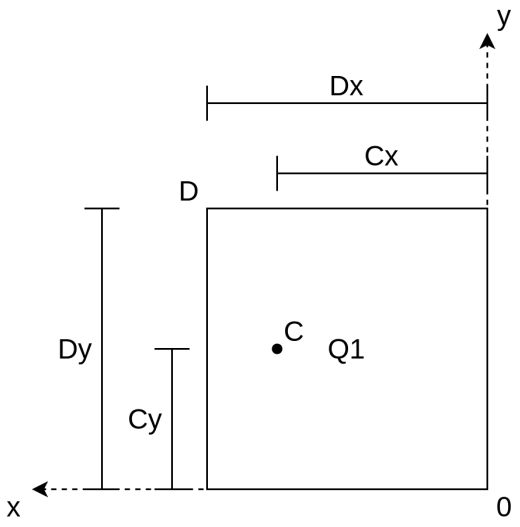


Figure 3.16: Examples of dimensions used for corner score calculated in Q1

The scaling factor is $sf = \frac{Cx}{Dx} + \frac{Cy}{Dy}$ (the need for the scaling factor is further discussed in Section 3.5.1). Multiple corners can be detected in the area of the bounding box and the scaling factor is used to favour the outermost corner points. Favouring the outer most corners filter out all the internal detected corners and false corners and extracts only the outer corners of the marker. The equation for the corner score is described by Equation 3.2.

$$CS = sf \times Cs \tag{3.2}$$

where:

CS is the Corner Score

Cs is the Corner Strength

sf is the Scaling Factor

Scaling Factor The scaling factor accounts for corners that are detected along the marker edge or inside the marker that could have a larger corner strength than the actual corner on the vertex of the marker. Figure 3.17 shows the border of the marker in quadrant 2 (top right quadrant) with detected corners on the border. Corner 1 (circle) represents the actual best corner and the other shapes represent false corners.

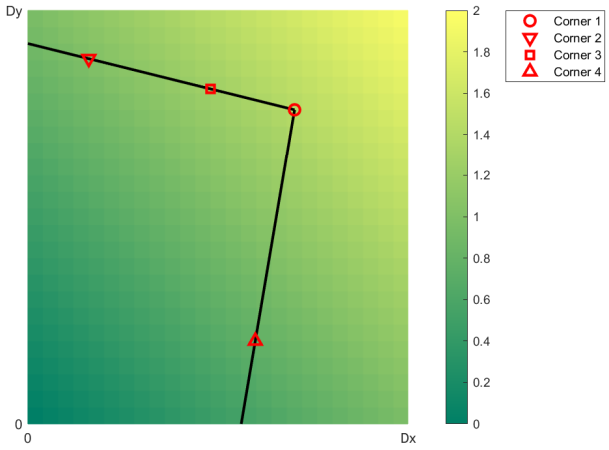


Figure 3.17: Top view of marker border overlaid onto scaling factor plane

Figure 3.18 shows how the scaling factor favours the actual corner and increases the accuracy of the feature evaluation step by reducing the chance that a false corner with a larger corner strength is chosen as the marker corner.

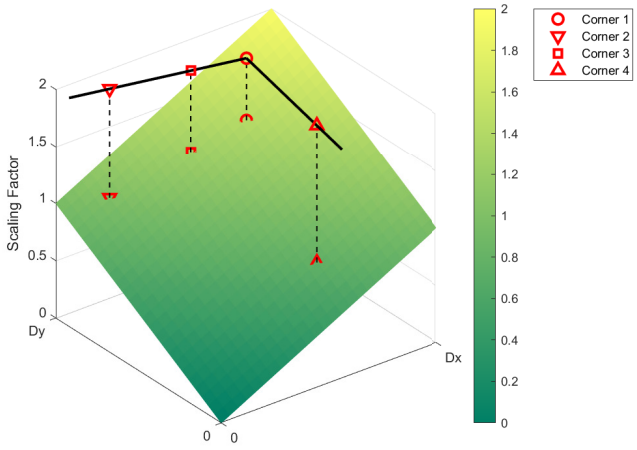
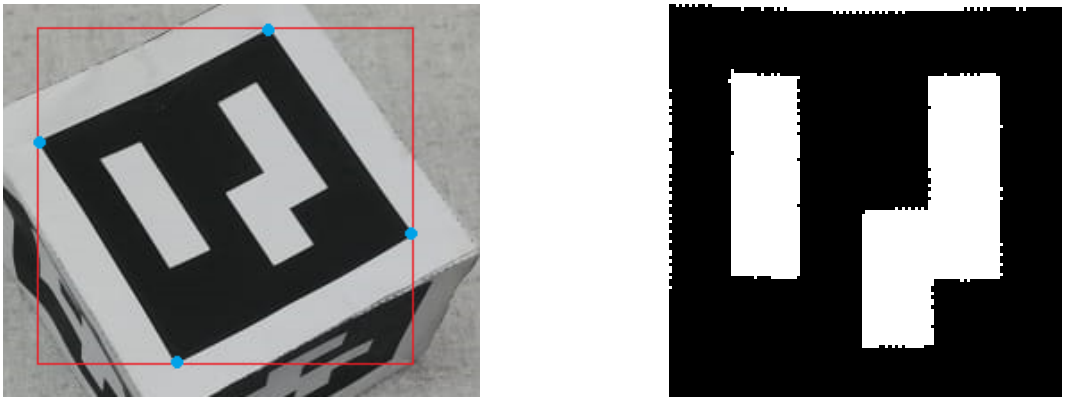


Figure 3.18: Detected corners projected onto scaling factor plane

3.5.2 Marker Rectification

The potential markers that are detected in an image need to be rectified to extract the information stored in the marker, discussed in Appendix A.3.3. The marker rectification (Figure 3.19b) is achieved by using the image plane coordinates of the four outer corners of the detected marker shown in Figure 3.19a to calculate the homography transform matrix.



(a) Detected marker candidate (b) Rectified marker candidate

Figure 3.19: Example of marker rectification (b) using the detected marker candidate (a)

Homography Transform The coordinates of the four marker corners are used to populate the homography matrix. The homography matrix is used to transform the marker detected in the image plane to another predefined plane to rectify the image. The homography transform equations calculate the image coordinates that are equivalent to the rectified image coordinates. The rectified image can then be populated by fetching the equivalent pixel value in the image. The homography transform equation is shown in Equation 3.3.

$$x_i = \frac{h_1 x_r + h_2 y_r + u_1}{h_7 x_r + h_8 y_r + 1} \qquad y_i = \frac{h_4 x_r + h_5 y_r + v_1}{h_7 x_r + h_8 y_r + 1} \tag{3.3}$$

where:

(x_i, y_i) are the x and y coordinates on image plane

$h_1 - h_8$ are the homography matrix elements

(x_r, y_r) are the x and y coordinates on predefined plane

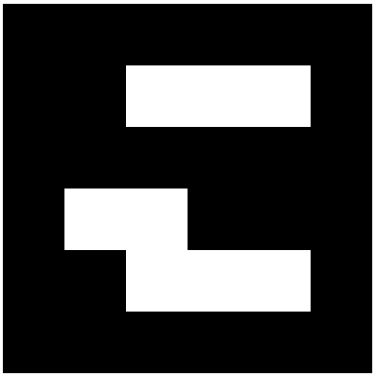
(u_1, v_1) are the x and y coordinates of the first marker corner on the image plane

3.5.3 Information Extraction

Binary square fiducial markers are square markers that contain information for identification. The inner binary codification makes the marker especially robust, allowing the possibility of applying error detection and correction techniques. The binary information matrix can be extracted by dividing the rectified marker into a grid.

ArUco Marker ID ArUco markers (from the 4x4 ArUco Dictionary example shown if Figure 3.20a) are designed with a unique ID in the form of a 4x4 binary matrix (Figure 3.20b). ArUco marker encoded information is discussed in Appendix A.2.1.

The ID of an ArUco marker is found by extracting the binary matrix and finding its corresponding ID in the ArUco marker dictionary. The full marker is divided into a 6x6 grid. The 4x4 binary matrix is found in the center of the marker and it is surrounded by a boundary of black squares. The information is denoted by 1 for a black square and 0 for a white square.



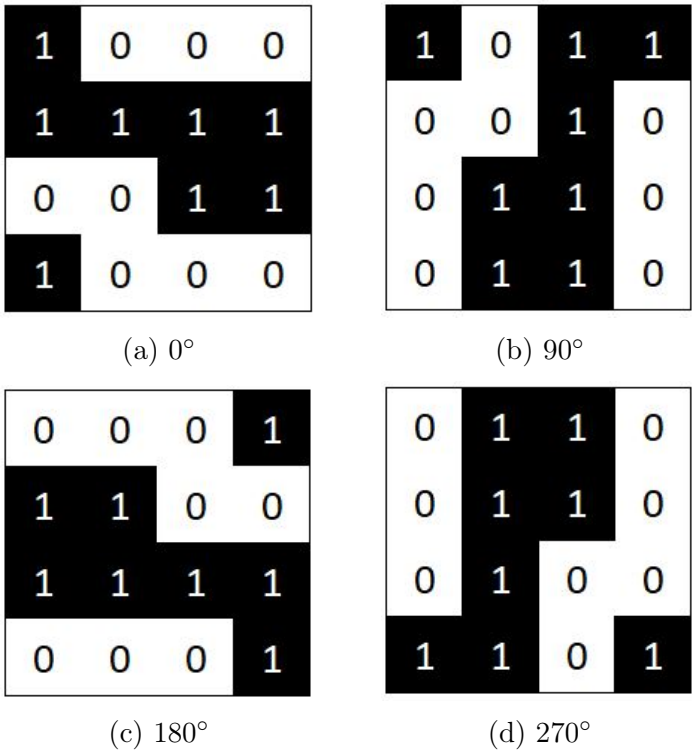
(a) ArUco Marker ID: 99

1	0	0	0
1	1	1	1
0	0	1	1
1	0	0	0

(b) Marker 99 Binary Matrix

Figure 3.20: ArUco marker information matrix (b) extracted from sample marker a

Accounting for Rotation To prevent false positives and to allow for a marker to be detected while in any orientation (Figure 3.21) the ArUco marker dictionary is designed to have a unique binary matrix even when rotated.



(a) 0° (b) 90°

(c) 180° (d) 270°

Figure 3.21: All 90° rotations of the ArUco marker binary matrix

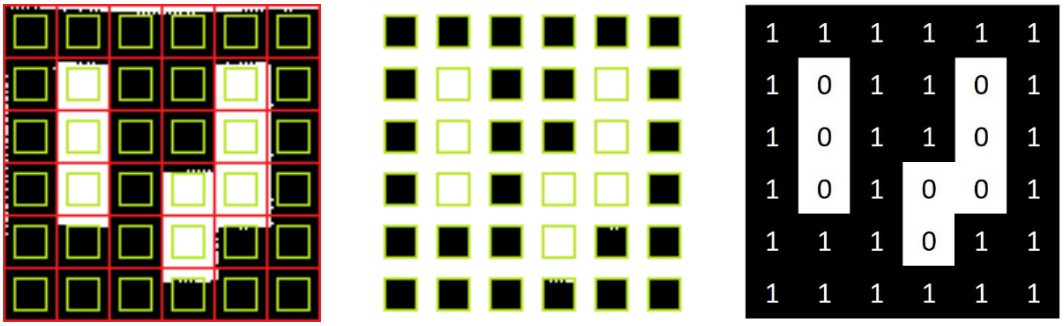
The original binary matrix of an ArUco marker can be rotated 90° 3 times and each rotation and the original orientation's information array (Table 3.2) does not match with any other marker in the same dictionary.

Rotation	16 Bit Information Array
0°	1000 1111 0011 1000
90°	1011 0010 0110 0110
180°	0001 1100 1111 0001
270°	0110 0110 0100 1101

Table 3.2: Information Arrays for Each Rotation

Extracting Binary Information The rectified image is divided into a 6x6 grid (Figure 3.22a) to start extracting the marker's information. A smaller central section of each grid element is evaluated to reduce erroneous data (Figure 3.22b). The grid elements are then denoted as a 1 if the square is majority black (more than half of the pixels in the evaluation grid are black) or 0 if the square is majority white (Figure 3.22c).

The outside border of the marker is first evaluated and if the outside border is not completely black the marker candidate is discarded. The 4x4 binary matrix is populated by evaluating the internal 4x4 grid of the candidate marker.



(a) 6x6 Grid (b) Removed Error (c) Binary Matrix

Figure 3.22: Extraction of information from marker

Decoding Marker ID The marker ID is determined by comparing the four rotations of the detected marker's binary matrix with the ArUco marker dictionary. If no comparison is found the potential marker is discarded. The rotations of the marker are compared to the dictionary elements by calculating the Hamming distance between the rotation information arrays and the marker dictionary information arrays (Table 3.3). The rotation and dictionary pair with a hamming distance of zero indicates that the marker has the ID of the dictionary marker and that the rotation is the upright rotation of the marker. Knowing the actual orientation of the marker allows for the pose of the system to be calculated.

Marker Candidate		Marker ID - 99 1000 1111 0011 1000	Marker ID - 42 1100 1101 1000 1100
0°	0110 0110 0100 1101	10	8
90°	1000 1111 0011 1000	0	6
180°	1011 0010 0110 0110	10	12
270°	0001 1100 1111 0001	8	10

Table 3.3: Hamming distances of detected marker vs example dictionary marker information arrays

3.6 Pose Estimation

The pose estimation is the final step of the algorithm. The pose estimation is used to determine the position and orientation of the system relative to the marker. Knowing the marker's location in the environment allows the system to know where it is in its surrounding environment.

The POSIT algorithm, discussed in Appendix A.4, is used to estimate the pose of the camera relative to the marker.

The four corners of the marker are the 4 coplanar points used in the POSIT algorithm. The orientation of the marker is determined in the previous section and therefore the order of the corners is known.

The POSIT algorithm outputs the rotation matrix and the translation vector for the object. The pose of the camera and in turn the pose of the system is determined by the rotation matrix and the translation vector. The relationship between the camera coordinate system and world coordinate system is shown in Figure 3.23.

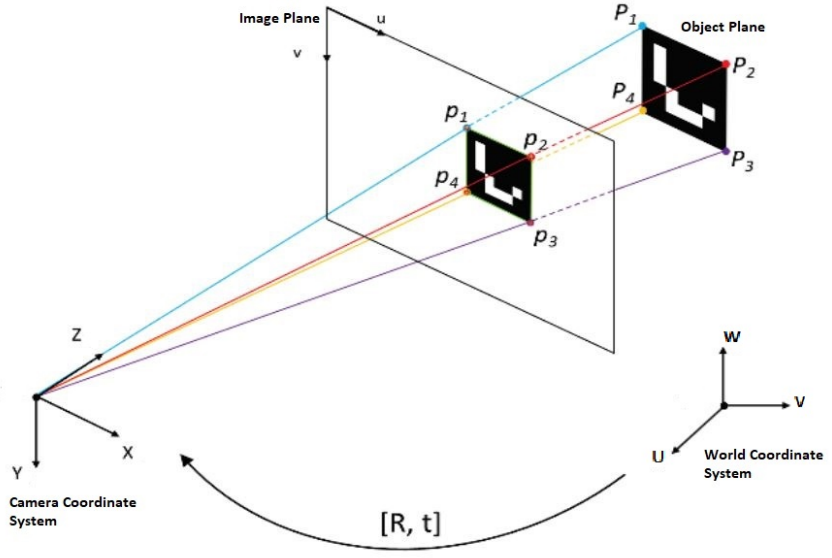


Figure 3.23: Relationship between camera and marker coordinate systems [10]

The model points are the points of the object on the object plane in the world coordinate system. The marker is the object and the markers' corners are the model points. All markers used for the proposed system have constant dimensions and therefore the model points stay constant regardless of where the marker is placed in the world or which marker is detected. The image points are the detected corners of a marker in the image. Using the image points and model points the rotation matrix (R) and translation vector (t) are estimated.

3.7 Conceptual Design Conclusion

The output of the required system is the pose information and ID of the marker. To manipulate the input image data from the sensor a marker detection system was designed. The elements of the marker detection system were built off existing marker detection and image processing algorithms (discussed in Appendix A).

A hybrid of a FPGA and CPU was chosen for the marker detection hardware platform. The FPGA can connect directly to the image sensor to access the raw inline pixel stream to accelerate the feature extraction aspect of the marker detection system. The extract features are then evaluated on the CPU to identify potential markers in the image frame and to estimate the pose of the detected marker.

From an image frame a marker is detected and its information is extracted. The extracted information is then used to estimate the pose of the system. The marker detection element consists of feature extraction, feature evaluation and marker identification.

Each element of the marker detection process was designed from the existing detection techniques and modified to suite the required hardware platforms to optimise the process and increase the processing performance of the system.

Chapter 4

Design Implementation

The designed ArUco marker detection concept is implemented on the chosen hardware (FPGA and CPU). The implementation of the hardware is broken down into the elements of the design that are implemented on the FPGA, referred to as the Programmable Logic (PL), and the elements that are implemented on the CPU, referred to as the Processing system (PS). The structure of the system and connections between the PS and the PL is shown in Figure 4.1.

The elements implemented on the PL are designed for inline processing and parallelisation. They are designed to utilise the benefits of using an FPGA and increase the processing time of the data intensive image processing. The elements implemented on the PL are tested through simulation to determine if timing and functional requirements are met.

The elements implemented on the PS are designed to perform iterative and complex functions.

4.1 Implemented System Overview

The implemented system consists of the data processing elements on the PS and PL. The grayscale image frame stored in memory on the PS is written to and stored on the PL to increase the time of complete system testing by removing the need to recompile the PL every time a test image is changed.

The image frame stored on the PL in Block RAM (BRAM) is streamed into the feature extraction modules. The image frame stream is in the form of pixels (8-bit intensity values) and is streamed into the feature extraction modules at the rate of the pixel clock. The feature extraction modules extract the features of interest in the image frame. The features are sent to the PS from the PL through the Data Bridge. The features are evaluated to identify ArUco markers in the image frame. The identified marker is then used to estimate the pose of the system relative to the detected marker.

The block diagram, shown in Figure 4.1, shows the elements of the marker detection system. The 3 main components of the system are:

- The data intensive image processing elements implemented on the PL.
- The Data Bridge, which allows the PS and PL to communicate and transfer data, implemented on the PL.
- The complex data evaluation and system control implemented on the PS.

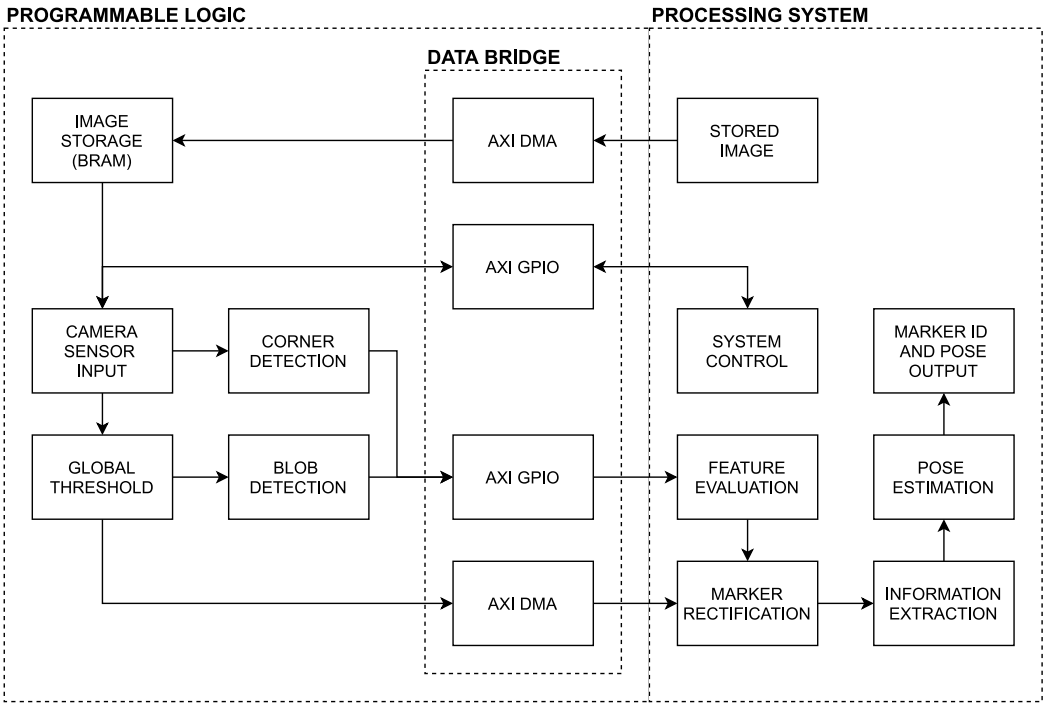


Figure 4.1: Block diagram of implemented marker detection system elements

4.1.1 Hardware Platform

The proposed concept for the marker detection system is implemented on a Xilinx ZYNQ-7020. The ZYNQ is a System on a Chip (SoC) with a dual-core ARM Cortex-A9 processor and FPGA. The ZYNQ was chosen because Xilinx products are popular FPGAs and the SoC provides the required functionality to implement the designed system. The ZYNQ-7020 provides more than enough logic elements and memory to implement the system.

The FPGA and ARM are both used in the implementation of the concept. The FPGA side is referred to as the PL and the ARM side is referred to as the PS. Relevant concepts regarding FPGA design are discussed in Appendix A.5.

4.2 Firmware Functional Breakdown

The Vivado Design Suite (Produced by Xilinx) is used to design and test the marker detection system. The final system schematic, Figure 4.2, consists of custom designed and Xilinx provided Intellectual Property (IP) modules. All the modules listed below are discussed in further detail in this chapter. An enlarged final system schematic is shown in Figure F.1.

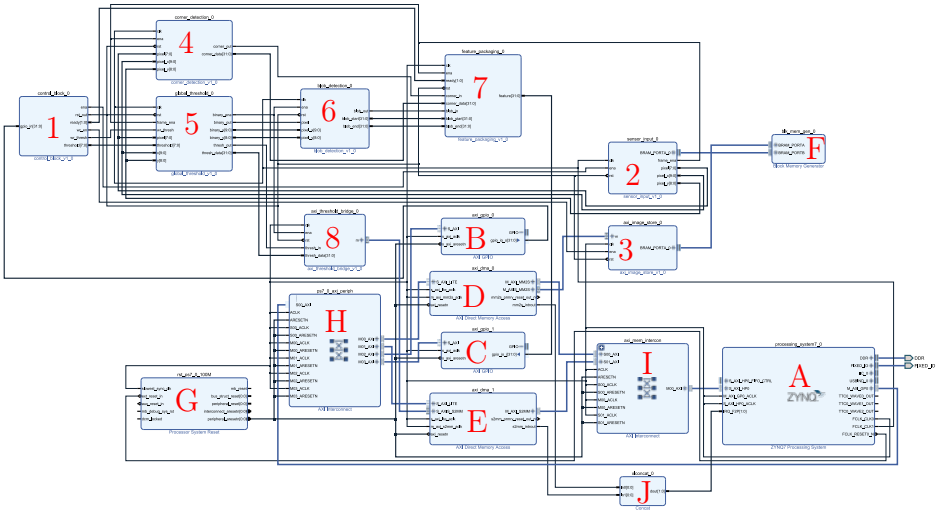


Figure 4.2: Schematic of the marker detection system implemented on the ZYNQ

The custom IP modules are denoted by a number in Figure 4.2. The IP modules, listed in Table 4.1, are discussed in Section 4.3 and Section 4.4. The graphical representation of the custom IP modules and their respective port descriptions are provided in Appendix B.

No.	Module Name	Description
1	Control Block	Receives and distributes control signals.
2	Sensor Input	Simulates camera sensor input.
3	AXI Image Store	Writes received image data to BRAM.
4	Corner Detection	Detects corners in image frame.
5	Global Threshold	Binarises image frame.
6	Blob Detection	Detects blobs in image frame.
7	Feature Packaging	Sends detected features to the PS.
8	AXI Threshold Bridge	Sends the binary image to the PS.

Table 4.1: Descriptions of Custom IP modules

The Xilinx provided IP modules are denoted by a letter in Figure 4.2. The PS interacts with the PL through the Processing System IP module. The PS IP allows communication between the two sides through the Advanced eXtensible Interface (AXI). The PL uses AXI General Purpose Input Output (GPIO) IP modules to send and receive control signals and feature data. The PL uses AXI Direct Memory Access (DMA) IP modules to send threshold data to the PS and to receive image data from the PS that is then stored in BRAM.

Table 4.2 lists the Xilinx provided IP modules and their respective descriptions [38].

Letter	Module Name	Description
A	Processing System	The interface that acts as the logic between the PS and PL. It provides the system clock (100 MHz) and pixel clock (50 MHz).
B/C	AXI GPIO	The GPIO interface to the AXI interface.
E/F	AXI DMA	Allows for high-bandwidth direct access between AXI4-Stream IP modules and PS memory.
F	Block Memory Generator	Automates the creation of block memory.
G	Processor System Reset	System and module reset functionality.
H/J	AXI Interconnect	Connects AXI Memory Mapped (MM) Master modules to one or more AXI MM Slave modules.
J	Concat	Concatenates bus signals.

Table 4.2: Descriptions of Xilinx Provided IP modules

The provided IP modules are customised to suit the proposed solution. The customisations are listed in Table 4.3.

Letter	Module Name	Description
B	AXI GPIO 0	Set to all outputs with a width of 32 bits.
C	AXI GPIO 1	Set to all inputs with a width of 32 bits.
E	AXI DMA 0	Set to read only channel with a MM data width of 32 and a stream data width of 32.
F	AXI DMA 1	Set to write only channel with a MM data width of 32 and a stream data width of 32.
G	Block Memory Generator	Set to simple dual port RAM. Port A is set to a width of 32 bits with a depth of 76800 and Port B is set to a width of 8 bits

Table 4.3: Customised parameters of Xilinx Provided IP Modules

4.3 Programmable Logic

The input from the image sensor and image processing modules run on the PL. The direct connection from the camera sensor allows the image data to be streamed into the system so that inline image processing can be done to increase the performance of the marker detection system. The inline processing of the incoming pixel data removes the need to first store the whole image into memory to later read from the memory to process the data.

The modules shown in Figure 4.3 represent the custom designed components of the feature extraction system that runs on the FPGA.

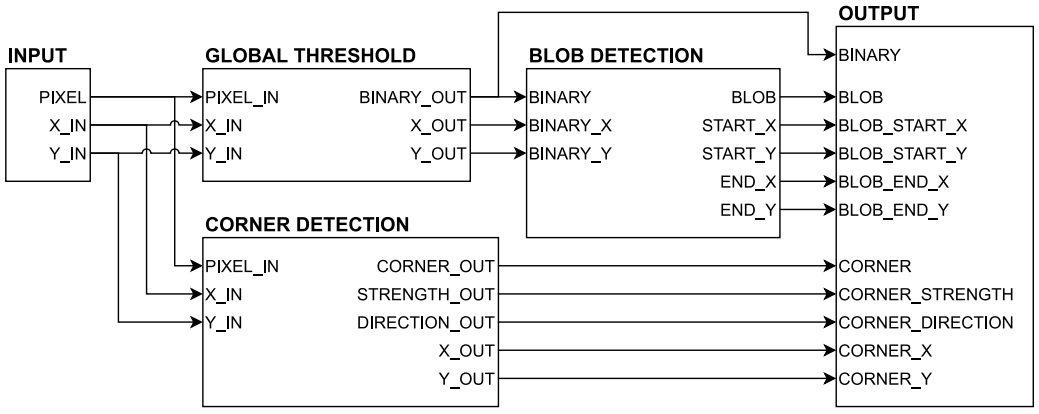


Figure 4.3: Components of Feature Extraction system implemented on the PL

The internal workings of the modules are verified by simulating the components and visually representing the internal signals in waveforms. Examples of component waveforms are provided in Appendix C.

4.3.1 Simulated Camera Sensor

To test the marker detection system a camera sensor is simulated. It is simulated to control the image information to characterise system accuracy without camera induced uncertainties. The simulated camera sensor functionality is shown in Figure 4.4.

The camera simulation design consists of storage and two custom IP modules, one to read data from storage and one to write data to storage. The storage is made up of BRAM that is large enough to store a 640×480 image.

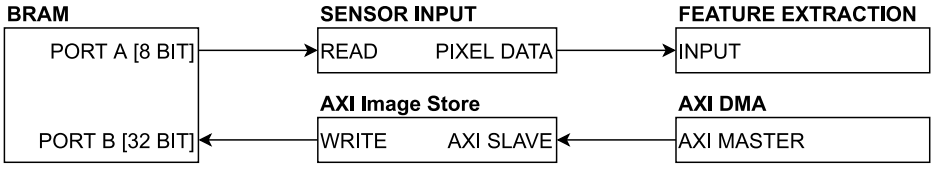


Figure 4.4: Simulated camera sensor functional diagram

4.3.2 Sensor Input

The Sensor Input module, port description in Appendix B.3.1, reads the stored image data from BRAM and simulates input similar to a camera sensor. When enabled the module starts reading from the BRAM and outputs the pixel intensity value (8-bit value) and pixel coordinates (x [10 bit] and y [9 bit]) at the frequency of the pixel clock (50 MHz). The module output is shown in Figure 4.5.

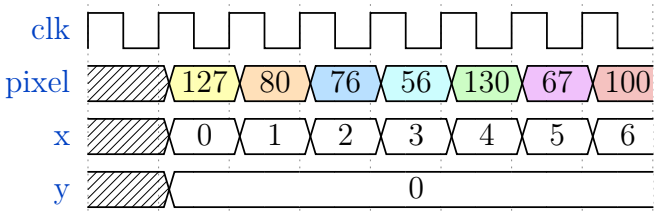


Figure 4.5: Waveform of the Sensor Input module pixel information output

4.3.3 AXI Image Store

The AXI Image Store module, port description in Appendix B.3.2, is used as the interface between the AXI DMA module and the BRAM. The module has an AXI Slave interface which waits for data to be available to write to BRAM. The AXI DMA Master interface connects to the Slave interface and writes image data from PS memory.

4.3.4 Corner Detection

The Corner Detection module, port description in Appendix B.1.1, receives pixel information and outputs corners that have been detected in the image.

The corner detection module buffers rows of the image frame and evaluates multiple POIs in a 9x9 window. The elements of the evaluated window are compared to one another and the best corner, if a corner is present, is chosen. The elements of the corner detection module are displayed in Figure 4.6.

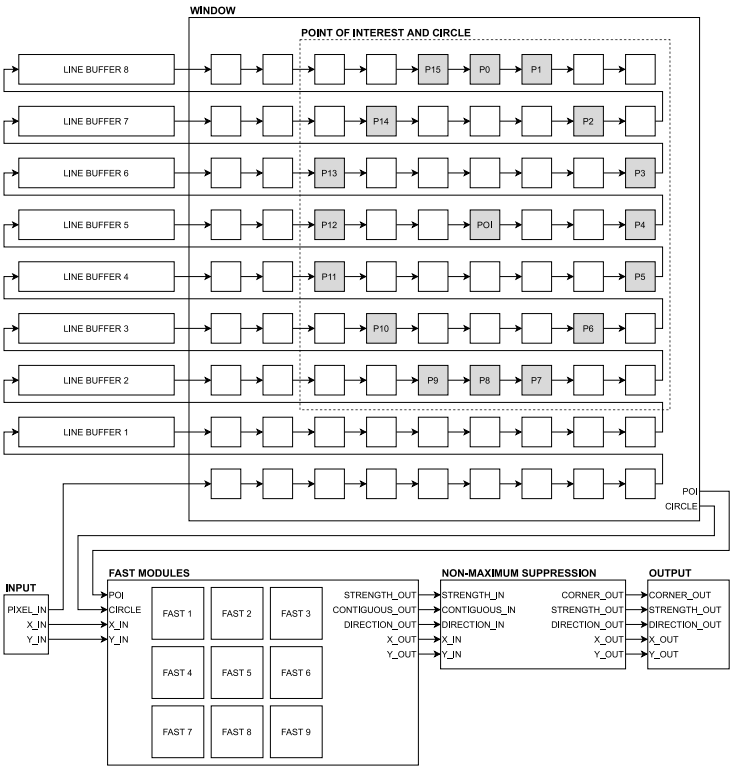


Figure 4.6: Corner Detection functional diagram

The following sections describe the different components of the Corner Detection module in more detail.

4.3.4.1 Line Buffers

A First In First Out (FIFO) that has a write depth of 1024 is used as the line buffer. The line buffer stores a large portion of the image row that is not stored in the window shift register.

Once a buffer is filled with 631 pixel values the buffer is opened and acts as a FIFO for the rest of the image frame. After the 8 line buffers have been filled the window will start shifting along with every pixel read into the module.

4.3.4.2 Window

The window consists of 9 rows of shift registers. The shift registers are made up of 9 registers that have a width of 8 bits that store the pixel values. The center 3 by 3 grid of pixels are the POIs that are evaluated. The rest of the window contains each POI's corresponding circle e.g. the POI and circle shown in gray in Figure 4.6.

4.3.4.3 FAST Module

The FAST Module is used to evaluate a POI and its respective circle and determine if it is a corner. A 3 by 3 grid of FAST modules are run in parallel to allow for non-maximum suppression to be used to reduce the number of detected corners in a single frame. Figure 4.7 shows all the components of a single FAST module. The pixel coordinates are pipelined to store them while the POI is evaluated to output the correct corner data.

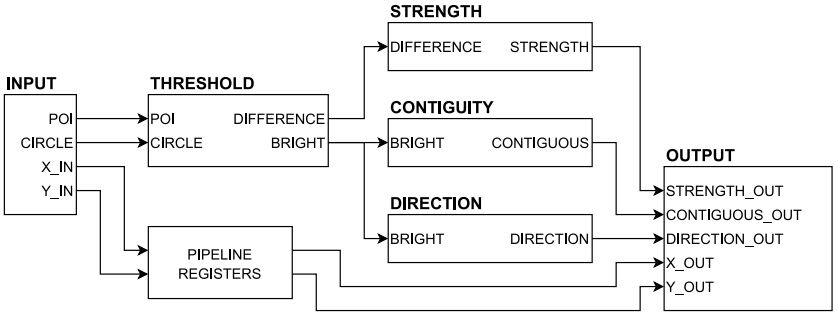


Figure 4.7: Components of the FAST module

The following paragraphs discuss the different components of the Fast Module.

Threshold The threshold component evaluates each pixel in the incoming circle and outputs two 16 element arrays: Difference and Bright. An example of the component functionality is shown in Appendix C.1.1.1. Each pixel of the incoming circle array is compared to the POI plus a threshold value. The difference array is determined by using Equation 4.1 and the bright array is determined by using Equation 4.2.

$$Difference_n = \begin{cases} Circle_n - (POI + t) & \text{if } Circle_n > (POI + t) \\ 0 & \text{else if } Circle_n \leq (POI + t) \end{cases} \tag{4.1}$$

$$Bright_n = \begin{cases} 1 & \text{if } Circle_n > (POI + t) \\ 0 & \text{else if } Circle_n \leq (POI + t) \end{cases} \tag{4.2}$$

Strength The strength component uses the difference and calculates the corner strength of the current POI. An example of the component functionality is shown in Appendix C.1.1.4.

The corner strength is the summation of the difference array shown in Equation 4.3.

$$Corner\ Strength = \sum_{n=0}^{15} Difference_n \tag{4.3}$$

Contiguity A POI is a corner if at least 9 contiguous circle pixels are brighter than the POI plus threshold value. An example of the component functionality is shown in Appendix C.1.1.2.

To determine if a POI is a corner 16 AND operations are performed where the 9 element AND gate is shifted along the circle looping back around until all potential contiguous elements are assessed e.g. one 9 element of adjacent pixels is shown in Figure 4.8. All 16 of the 9 element combinations are AND-ed concurrently. The outcome of AND operations are OR-ed and the output of the 16 element OR operation determines if the POI is a corner.

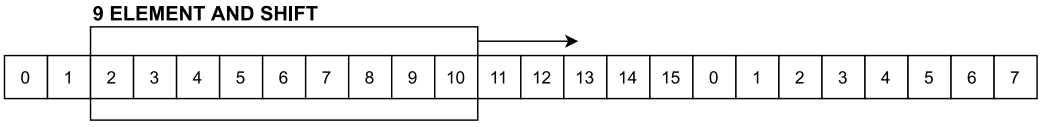


Figure 4.8: 16 arrangements of 9 adjacent circle pixels

Direction The direction of the corner is determined by finding the direction with the most dark squares. An example of the component functionality is shown in Appendix C.1.1.3.

The dark elements are counted and the largest direction is chosen. The directions are indicated in the example shown in Figure 4.9.

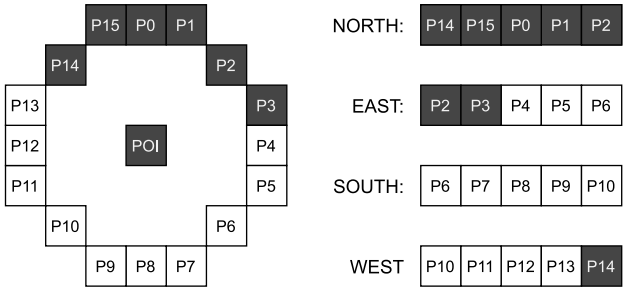


Figure 4.9: Example of a south facing corner

4.3.4.4 Non-Maximum Suppression

Multiple adjacent POIs on the same corner can be identified as corners. Non-maximum suppression (NMS) is therefore used to determine the best corner in the group of identified corners.

NMS determines the best corner by comparing all the POIs in the 3 by 3 grid together and only choosing the corner with the highest strength value. An example of the component functionality is shown in Appendix C.1.2.

The NMS component is pipelined to only compare 3 corner strengths in a single clock period because timing was not met when comparing 9 corners in one clock period (F_{max} of non-pipelined solution is 36.63 MHz)

4.3.5 Global Threshold

To separate the marker from the background the Global Threshold module, port description in Appendix B.1.2, is used convert the input pixel intensity values into binary values.

To account for dynamic light changes global thresholding is used. Frames from an image sensor are streamed in quick succession and therefore the previous frame can be used to determine a suitable threshold value. The module can thus dynamically change with lighting changes while keeping the processing inline. The first frame, when starting the system, is thresholded with a default thresholding value of 127.

The Global Threshold module consists of two components shown in Figure 4.10.

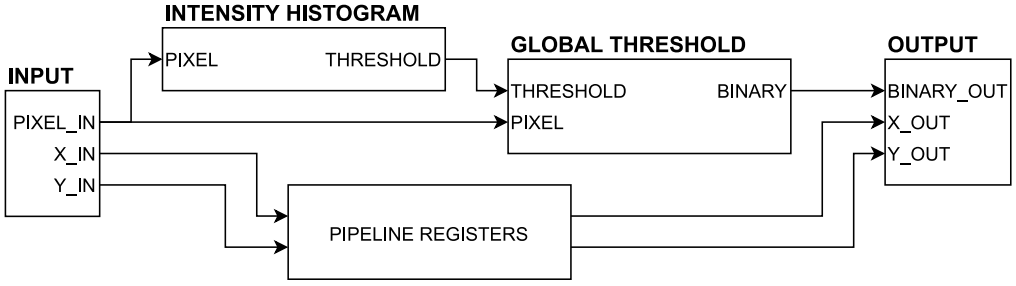


Figure 4.10: Components of the Global Threshold module

The two components of the Global Threshold module are discussed in the paragraphs that follow. Examples of the Global Thresholding module functionality are shown in Appendix C.2.

4.3.5.1 Threshold

The threshold component evaluates the incoming pixel value and determines if the pixel is part of an object or part of the background. The evaluation process is shown in Equation 4.4.

$$binary_{out} = \begin{cases} 1 & \text{if } pixel_{in} \leq threshold \text{ (dark)} \\ 0 & \text{else if } pixel_{in} > threshold \end{cases} \tag{4.4}$$

4.3.5.2 Intensity Histogram

The intensity histogram, e.g. shown in Figure 4.11, is populated as the pixel values are streamed into the module. For each possible pixel intensity (0 to 255) the number of pixels with that intensity are tallied. The new threshold value is the average between the pixel intensity peaks that is calculated using Equation 4.5. The peak in the second half of the histogram (P2) represents the background and the peak in the first half (P1) represents the object.

$$threshold = \frac{P2 - P1}{2} \tag{4.5}$$

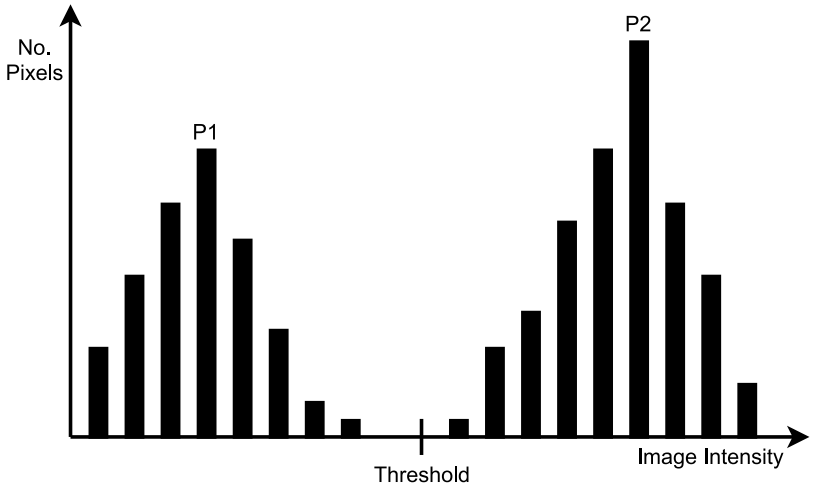


Figure 4.11: Intensity histogram of image frame showing two peaks that represent the object (P1) and the background (P2)

4.3.6 Blob Detection

Potential markers in the image are identified by the Blob Detection module, port description in Appendix B.1.3. The module labels and grows blobs as the pixel values are streamed in. When a suitable blob is detected the module will output the relevant blob information.

The blob detection module can be broken down into 4 main components as shown in Figure 4.12. The incoming binary pixel values are labelled as they are read into the module and when a suitable blob is detected it is selected as output.

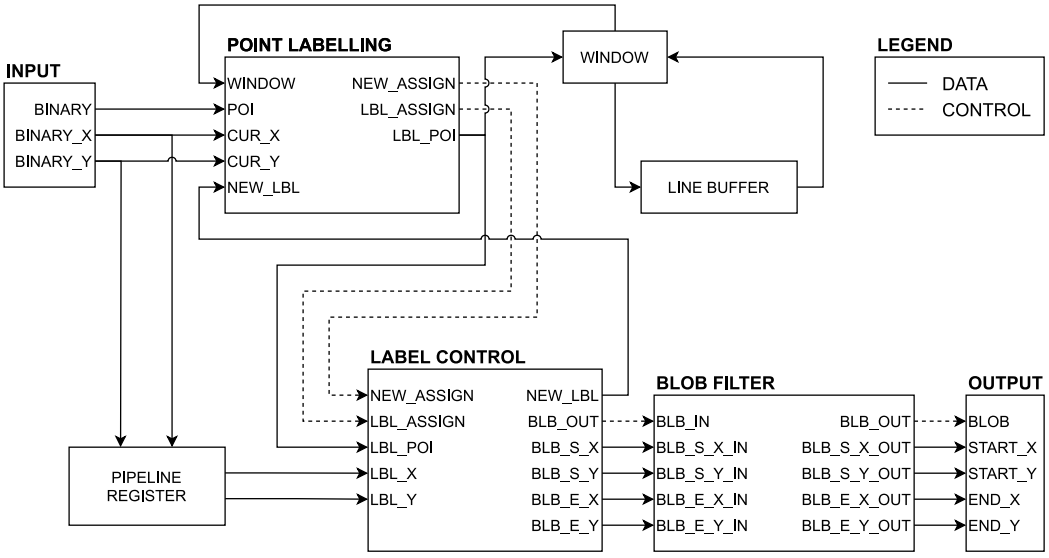


Figure 4.12: Components of the Blob Detection module

4.3.6.1 Neighbouring Window

The neighbouring window, shown in Figure 4.13, consists of shift registers that contain the pixels above and to the left of the POI. The labelled POI is shifted along the shift register until it is fed into a line buffer that stores a single row of the image frame.

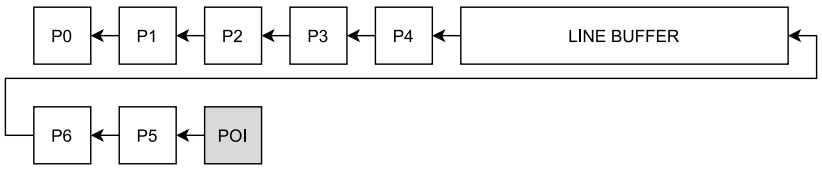


Figure 4.13: Neighbouring window and line buffer

4.3.6.2 Point Labelling

The incoming binary value is labelled as it is read into the component. Examples of the Point Labelling component are shown in Appendix C.3.1.

If the POI is 1 (dark) and has no labelled neighbouring pixels a new label is assigned to the POI and a new blob started.

If the POI is 1 (dark) and it has labelled neighbouring pixels the neighbours are assessed to determine the most suitable label. The most suitable label is the label that has the largest X axis blob width. The reason for the most suitable label being the widest blob is shown in Figure 4.14. An ArUco marker has an internal visual information matrix which can be detected as individual blobs. To avoid the internal blobs from causing erroneous blob detections the wider blob is favoured.

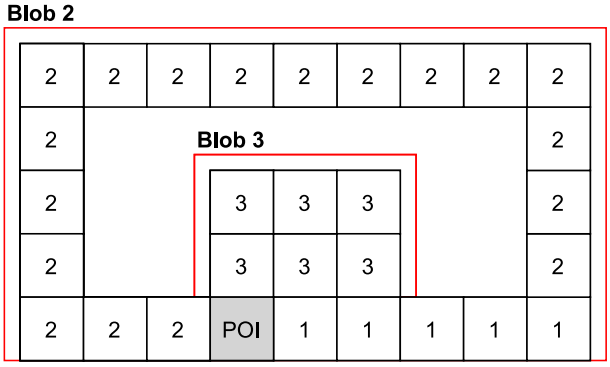


Figure 4.14: Example of blob width assessment

The white border of the marker prevents larger environment blobs from having a negative impact on the marker blob detection. The neighbour window has a left/right reach of 2 pixels and an upper reach of 1 pixel. The white border width is therefore chosen to be larger than the neighbour window reach at detectable marker distances.

4.3.6.3 Label Control

The Label Control component manages the labels and the information associated with them. Examples of the Label Control component are shown in Appendix C.3.3.

A FIFO is used to store the set of available labels. When a label is used it is read from the FIFO and when the label is deactivated it is written to the FIFO. The FIFO is instantiated with all the available labels.

Each label has a bounding box associated with it. The relevant bounding box information is the start and end coordinates of the box.

New Label Assigned When a new label is assigned to a POI the next label is read from the FIFO and stored to be used for the next new assignment as shown in Figure 4.15. When a new label is assigned the bounding box start and end coordinates are set to the coordinates of the assigned POI.

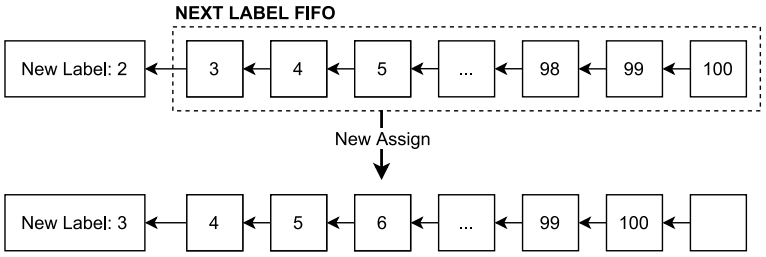


Figure 4.15: Change in FIFO data as label is read from FIFO

Assigned Existing Label If the POI is assigned to an existing label the bounding box coordinates are updated to match the change in the blob shape.

Inactive Label Deactivated If the current coordinate of the POI is out of range of a label (the end coordinates of the bounding box of that label indicates the label is no longer in range of the Neighbouring Window) the label is seen as inactive and is deactivated. The deactivated label is written back into the FIFO (Figure 4.16) to be used again and the bounding box information is sent to the blob filter component.

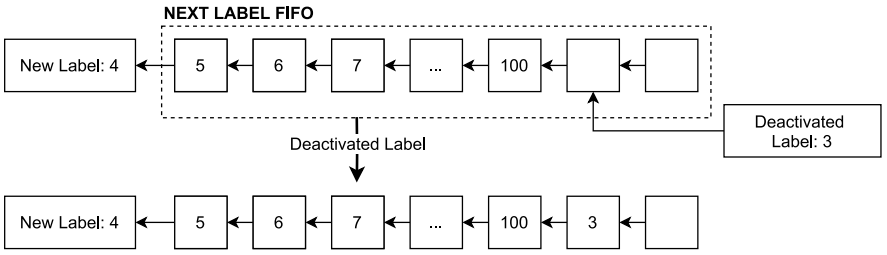


Figure 4.16: Change in FIFO data as label is stored into the FIFO

4.3.6.4 Blob Filter

The blob filter component filters out blobs with sides smaller than 24px. The ArUco marker has a 4x4 information matrix within the marker. If the detected marker is 24px wide the information squares are 4px wide as shown in Figure 4.17.

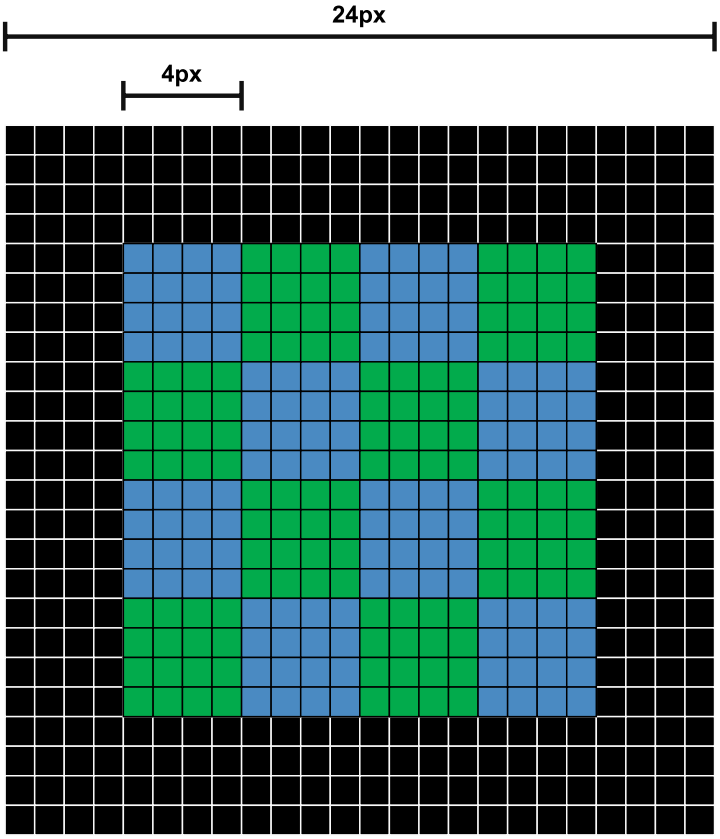


Figure 4.17: The pixel level breakdown of the information squares in the ArUco marker

A square width of 4px is chosen as the smallest detectable information square. The minimum 4px width provides the smallest number of pixels needed to extract the binary information matrix.

The smaller region that is evaluated in each information square, discussed in Section 3.5.3, is 1px width smaller than the detected square. The binary information stored in the 4px wide square is therefore 2px wide. The 4px is therefore the smallest theoretical width that allows for information extraction.

With an information square width of 4px the minimum width of the detectable blob is 24px.

4.4 Data Bridge

Communication between the PS and the PL is required for the system to function as intended. The required data transfer between PS and PL is achieved by using AXI capable modules. The AXI modules and support modules enable the control signals and image data to be communicated between the PS and PL.

Control signals are sent to the PL from the PS and feature data is sent to the PS from the PL by using AXI GPIO IP modules. The AXI GPIO IP modules allow for 32 bits of data to be sent between the PS and the PL.

The transfer of image data is facilitated by the AXI DMA IP module. The AXI DMA IP Module is used to read test image data from PS memory and send it to the Image Store module and to receive binary image packets from the PL and write them to PS memory. The AXI DMA communicates with custom modules through an AXI Stream (AXIS) interface.

The AXIS Interface consists of 5 signals. These signals, shown in the timing diagram in Figure 4.18, are used to communicate with the AXIS interface to transfer data to and from the PS. TREADY is an incoming signal sent from the AXI DMA to indicate if the interface is ready to receive data. TDATA is the data that is being transferred, TVALID is the signal that indicates if the data on TDATA is valid to transfer and TLAST indicates the end of the data stream that is being transferred.

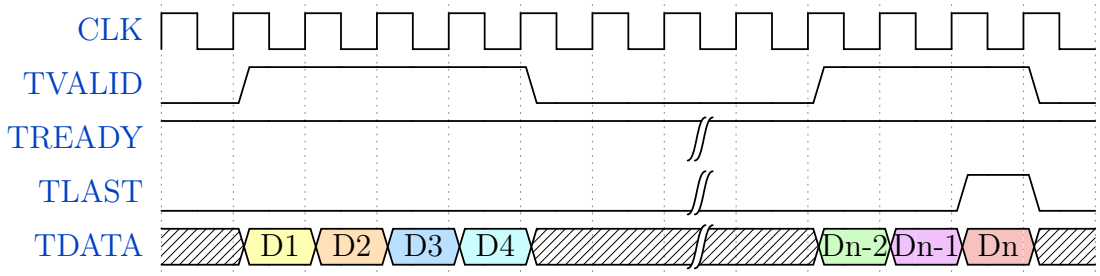


Figure 4.18: Timing diagram of AXIS interface signals

The custom IP modules used to facilitate communication between the PS and PL are discussed in the following sections (except for the AXI Image Store module that is discussed in Section 4.3.3).

4.4.1 Control Block

The Control Block module, port description in Appendix B.2.1, is used to send control signals and data to the PL. The signals and data that are sent to the PL are packaged together into a 32-bit packet. The data format of the package 32-bit packet is shown in Figure 4.19. The packaged 32-bit packet is received by the module. The module then splits the 32-bit packet into their respective individual signals to be distributed to the different parts of the marker detection system.

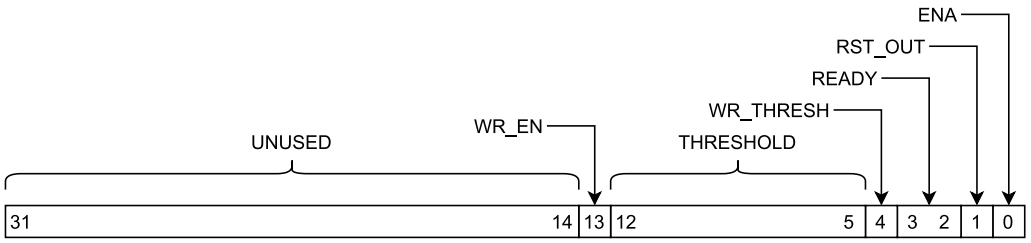


Figure 4.19: Data format of 32-bit control packet

4.4.2 AXI Threshold Bridge

The AXI Threshold Bridge module, port description in Appendix B.2.2, is used to send binary image data to the PS. The module receives 32-bit data packages that contain binary pixel values. The packaged binary pixel values are sent to the PS by using an AXIS Master interface. The binary package is sent from the PL to the PS via AXI DMA in bursts of 32 packets.

An example of the data transfer process is shown in Figure 4.18. Each data package sent contains 32 binary pixel values. The TLAST signal is triggered on the 32nd package to indicate the end of the data burst.

To transfer the complete binary image 300 bursts of 32 packets with a width of 32 bits are sent from the PL to the PS.

4.4.3 Feature Packaging

The Feature Packaging module, port description in Appendix B.2.3, is used to send detected feature information (corner and blob information) to the PS.

To differentiate between the two different features and their data formats bit 31 is used to indicate whether the sent packet is corner or blob information (1 indicates corner information and 0 indicates blob information).

The corner information is packaged in a 32 bit array with the data format shown in Figure 4.20.

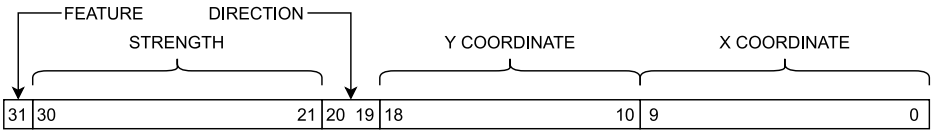


Figure 4.20: Data format of 32 bit corner data packet

The blob information is packaged into 2 separate 32-bit arrays with the data format shown in Figure 4.21. The blob is split into 2 packets because the bounding box start and end coordinate information can not fit into a single packet. Bit 30 in the packet indicates whether the blob data that is sent is the start or end coordinates. The blob counter is used to group start and end coordinates together if they are sent out of order (1 indicates start coordinates and 0 indicates end coordinates).

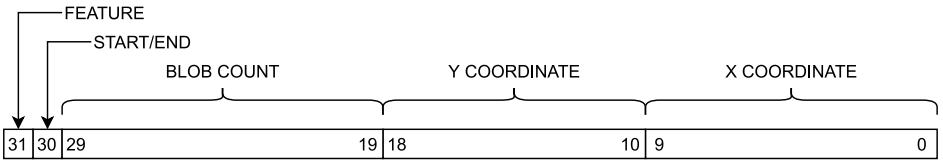


Figure 4.21: Data Format of 32 bit blob data packet

The packaged information is sent when the PS changes the ready signal to indicate that the PS is ready to receive another packet. While the packets are not ready to be sent the incoming feature information is stored in buffers that have a width of 32 bits and a depth of 100 packets.

The end of the feature extraction system is indicated by sending a packet filled with ones. The end packet is sent when the image frame has been completely processed and the feature buffers are empty. The end packet filled with ones is chosen because no corner packet can accidentally be completely filled with ones e.g. the X coordinate segment of the packet has a width of 10 bits giving a maximum unsigned integer value of 1024 where the max X coordinate value is 639.

4.5 Processing System

The PS controls the PL by sending control signals and data to the PL. The reset and enable signals are used to start and restart the hardware logic. The PS receives, decodes and evaluates the extracted information from the image and identifies if the test image contains an ArUco marker and extracts the information from it.

Figure 4.22 shows the flowchart of the marker detection process that runs on the PS.

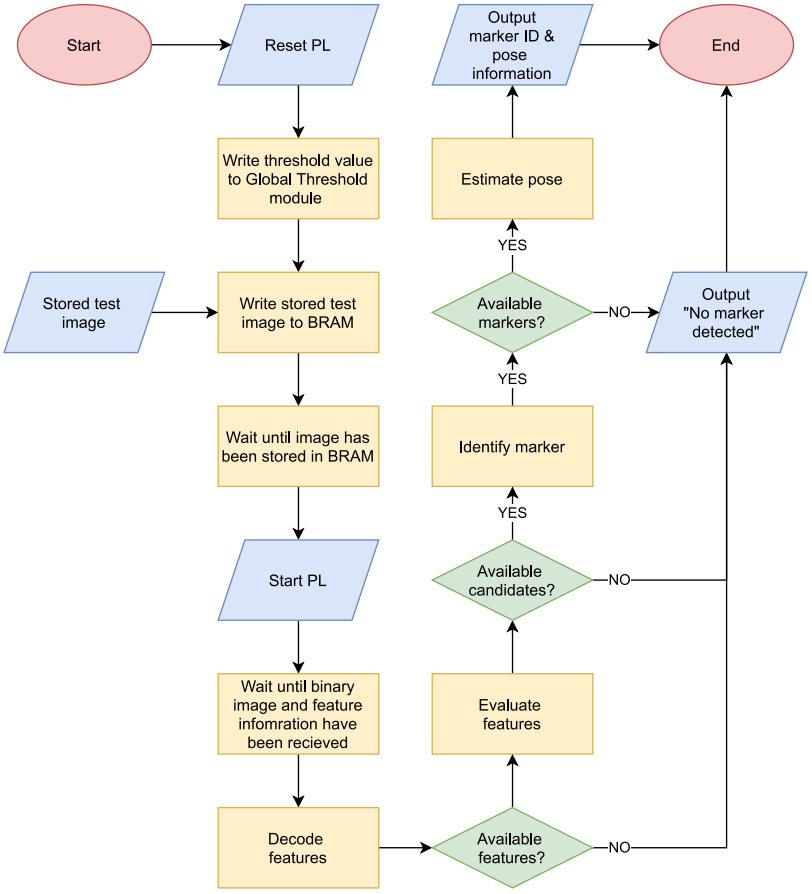


Figure 4.22: PS marker detection flowchart

The sections that follow describe the main functions of the PS marker detection system in more detail.

4.5.1 Data Transfer

Communication between the PS and PL is achieved through the use of the AXI. For large data transfer AXI DMA is used and for smaller packets of data AXI GPIO is used.

AXI DMA is used to transfer the test image data to the PL and to receive binary image data from the PL. The AXI DMA is initialised and left to read-/write directly from/to memory in the background. An interrupt is triggered when a burst of packets has been sent. The callback function that runs when the interrupt is called starts another burst of packets and the function runs in the background until all packets have been sent/received.

AXI GPIO is used to send control signals and data to the PL and to receive feature information from the PL.

4.5.1.1 Test Image Write

The test image is preloaded into memory at a specific address. When the image transfer starts the pixels are read in groups of 4 to fill up the 32-bit data packet (grayscale pixel intensity value is 8 bit). Figure 4.23 shows an example of a packaged test image packet.

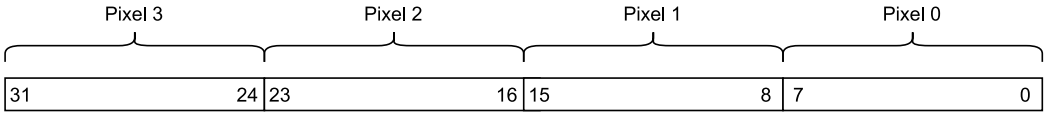


Figure 4.23: Layout of test image 32-bit packet

To transfer the complete test image to BRAM 76800 packets are sent. The packets are sent in bursts of 32 bursts and therefore 2400 bursts of packets are sent through the AXI DMA interface.

4.5.1.2 Binary Image Receive

The binary image packets that are sent from the PL to the PS are stored in memory starting at a predefined memory address. The received packets consist of 32-bit binary pixel values. Figure 4.24 shows an example of the binary image packet.

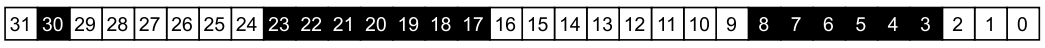


Figure 4.24: Layout of binary image 32-bit packet

4.5.1.3 Feature Information Receive

The Feature Receive function runs concurrently with the Binary Receive function. The PS receives the detected features and binary image at the same time. While the binary image is being received in the background the Feature Receive function is constantly polling the incoming AXI GPIO register. When a change is identified in the GPIO register the contents of the register are written to memory and the feature received counter is incremented. When a feature is received the PL is informed that the PS is ready to receive another feature.

The Feature Receive function stops polling for new packets when the end packet is received (packet full of ones).

4.5.2 Feature Decoding

The received feature packets are decoded into usable integer formats so that the information can be evaluated to detect markers in the test image. Using bitwise operators the 32-bit packet can be split into its various components. An example of decoding the feature packet is shown in Figure 4.25 where the feature indicator is determined.

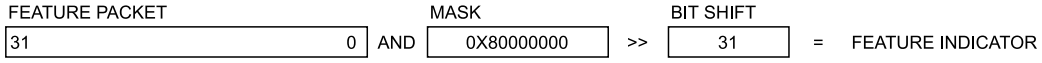


Figure 4.25: Layout of binary image 32-bit packet

Table 4.4 shows the list of bitwise operations to extract all the information from the feature packet.

Information	Mask	Bit Shift
Feature Indicator	0x80000000	Left Shift by 31
Corner X Coordinate	0x3FF	None
Corner Y Coordinate	0x7FC00	Left Shift by 10
Corner Direction	0x180000	Left Shift by 19
Corner Strength	0x7FE00000	Left Shift by 21
Blob S/E Indicator	0x40000000	Left Shift by 30
Blob Number	0x3FF80000	Left Shift by 19
Blob Start X Coordinate	0x3FF	None
Blob Start Y Coordinate	0x7FC00	Left Shift by 10
Blob End X Coordinate	0x3FF	None
Blob End Y Coordinate	0x7FC00	Left Shift by 10

Table 4.4: Bitwise operations to decode feature packets

4.5.3 Feature Evaluation

The feature evaluation process, discussed in Section 3.5.1, is implemented in the Feature Evaluation function. Each detected blob is evaluated to determine if it is a potential marker candidate. To be classified as a potential marker candidate the blob must contain four corners that meet the criteria discussed in the previously mentioned section.

To determine all potential markers in the image each blob is evaluated against all the corners detected.

4.5.4 Marker Rectification

To extract information from the marker candidate and determine if it is an ArUco marker the candidate must first be rectified. To rectify the candidate the components of the homography matrix, discussed in Appendix A.3.3, must be determined. The components of the matrix are used in the homography transform equation, discussed in Section 3.5.2.

The candidate corners and canonical marker image width are used to determine the homography matrix components. The equations for the homography components are shown in Appendix D. The component equations are determined by using Singular Value Decomposition (SVD) [39]. Using Matlab the components were solved symbolically so that the equations can be used in the C implementation. The homography components are dependent on the detected corners.

Using the homography transform equations shown in Equation 3.3 the X and Y coordinates of the equivalent image frame point can be calculated. The equivalent point can then be read from the binary image stored in memory. Only the smaller central section of the 6x6 grid, discussed in Section 3.5.3, is populated to reduce processing time.

4.5.5 Information Extraction

The binary information matrix of the ArUco marker is extracted by evaluating the rectified grid components produced by the Marker Rectification function. The information evaluation is discussed in Section 3.5.3. The extracted marker information matrix is rotated 3 times and the 4 rotations of the matrix are compared to all the ArUco marker IDs in the 250 4x4 ArUco marker library. If any orientation of the matrix corresponds to an ArUco marker ID the ID of that marker is set as the detected markers ID. The orientation that the ID was identified at is used to rearrange the detected corners in the appropriate order.

4.5.6 Pose Estimation

To determine the pose of the camera relative to the marker the algorithm discussed in Appendix A.4 and Section 3.6 is implemented in C code.

The POSIT algorithm requires the model points, the detected marker corners, the image pixel width, the image pixel height and the focal length.

The model points (m) are represented by the matrix in Equation 4.6.

$$m = \begin{bmatrix} P_{0x} & P_{0y} & P_{0z} \\ P_{1x} & P_{1y} & P_{1z} \\ P_{2x} & P_{2y} & P_{2z} \\ P_{3x} & P_{3y} & P_{3z} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ L_{side} & 0 & 0 \\ L_{side} & L_{side} & 0 \\ 0 & L_{side} & 0 \end{bmatrix} \tag{4.6}$$

Each row represents a marker corner point (P) and the columns represent the 3D Axes ($[x, y, z]$). The top left corner of the marker is set as the origin point and therefore has coordinate points of zero. The other marker points are determined by the length of the side of the square marker (L_{side}). All the Z coordinates are set to zero because the marker corners are coplanar.

The pose of the camera relative to the marker is estimated by iterating estimation function until the error is reduced to smaller than 2px or if 100 iterations have passed. The error is the Euclidean distance between the detected marker corners and the corner projected onto the image frame using the estimated camera pose.

During the testing of the POSIT implementation the number of iterations for the error to be reduced to less than 2px was around 1-2 iterations. The small number of iterations required to converge to a suitable error is also mentioned by the article [20] stating that only a few iterations are required to attain a suitable result. A conservative limit of 100 iterations was therefore chosen. If the estimation iterates 100 times and the error is not reduced to less than 2px the pose estimation function will stop iterating and output the last available results.

4.6 Implemented Design Analysis

The aspects of the process to detect markers using an FPGA/CPU hybrid solution are broken down into components and have been analysed. The three main aspects of the implemented design that are analysed are hardware usage, performance and power usage.

The hardware utilisation of the FPGA can be broken down into 5 components. The components are Look Up Tables (LUT), Flip-Flops (FF), BUFGs (global clock buffer), LUT RAM (LUTRAM), and BRAM.

The performance of the custom IP modules is characterised by the maximum clock frequency the hardware design can run at while meeting the timing requirements. The performance of the data bridge is characterised by the data rate and required data that is transferred from PL to PS.

4.6.1 Feature Extraction Modules Analysis

The custom features extraction IP modules were implemented individually to determine their individual functional characteristics. Figure 4.26 shows the hardware utilisation of the individual functions.

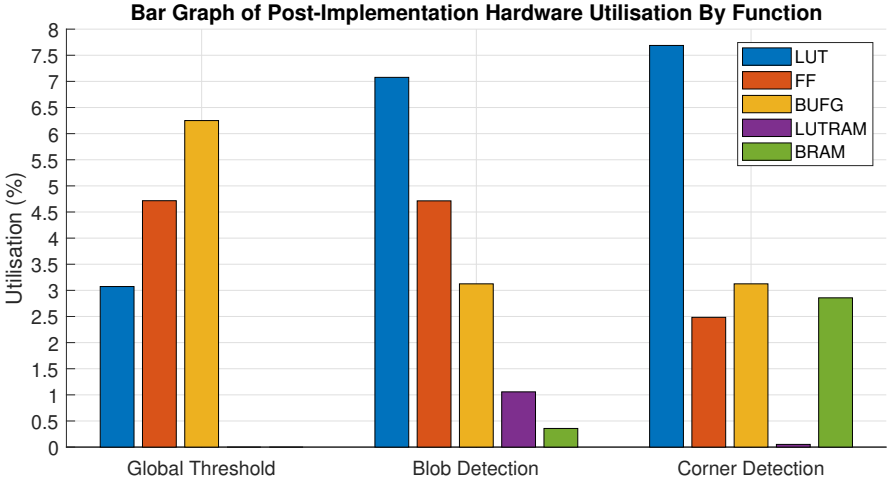


Figure 4.26: Hardware utilisation by feature extraction function

The maximum clock frequencies the feature extraction modules can operate at are shown in Figure 4.27. The Corner Detection and Global Threshold modules can run comfortably at a clock frequency of 100 MHz. The blob detection module has a F_{max} of just about 55 MHz which is only 10% above the 50 MHz pixel clock.

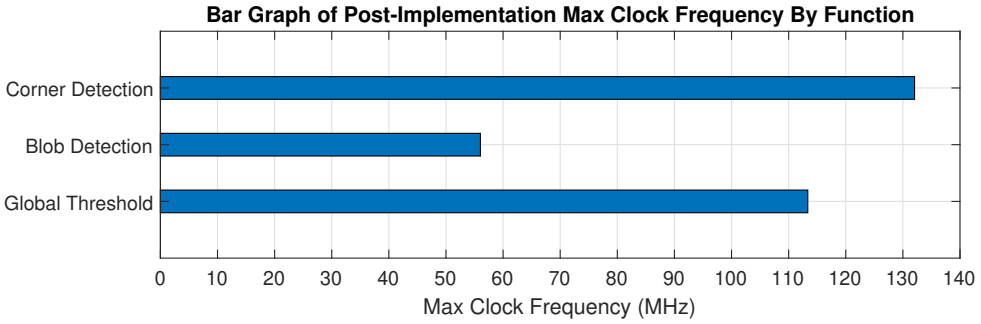


Figure 4.27: F_{max} by function

Figure 4.28 shows the estimate power consumption of the feature extraction modules. The blob detection module uses the most estimated power at a power usage value of about 0.4 W.

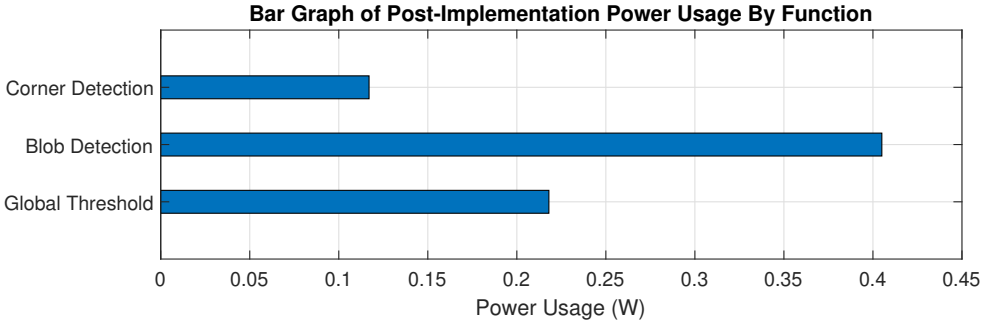


Figure 4.28: Function power usage estimates

From the functional analysis of the feature extraction modules the Blob Detection module is shown to have the highest resource usage with the lowest clock rate.

4.6.2 Programmable Logic Analysis

To test the feature extraction PL system including the Data Bridge the custom IP modules were implemented together, shown in Figure E.1, and evaluated using a test bench. The test bench was used to verify that the modules work together. The implemented design also provided analysis of the PL functionality.

The hardware usage, shown in Figure 4.29, is comparable to the individual function implementations put together.

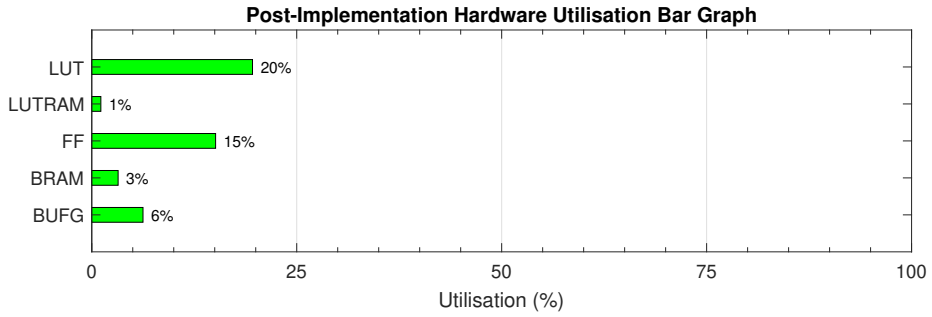


Figure 4.29: Hardware utilisation by PL implementation of feature extraction elements

F_{max} for the PL design was determined to be 55 MHz showing that the Blob Detection module is the bottleneck of the custom IP modules.

Figure 4.30 shows the breakdown of the on-chip power usage estimate of the design. The PL power usage estimate is close to the sum of the feature extraction modules' power usage estimate.

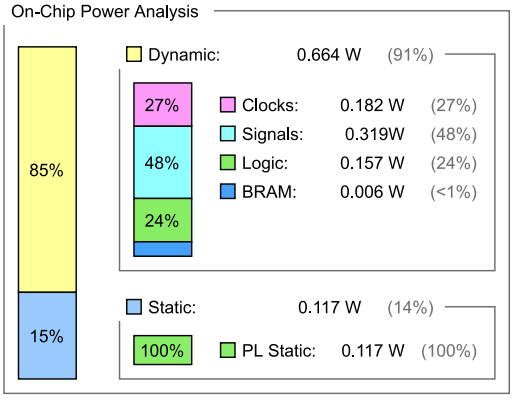


Figure 4.30: On-chip power usage estimate by PL implementation

4.6.3 Complete System Analysis

The complete design, shown in Figure 4.2, implemented on the FPGA was analysed.

The hardware utilisation, shown in Figure 4.31, shows a large increase in hardware usage. The overall usage increases due to the additional implementation of Xilinx provided IP modules to facilitate system functionality. BRAM usage is greatly increased because a complete test image must be stored in hardware to simulate an image sensor frame.

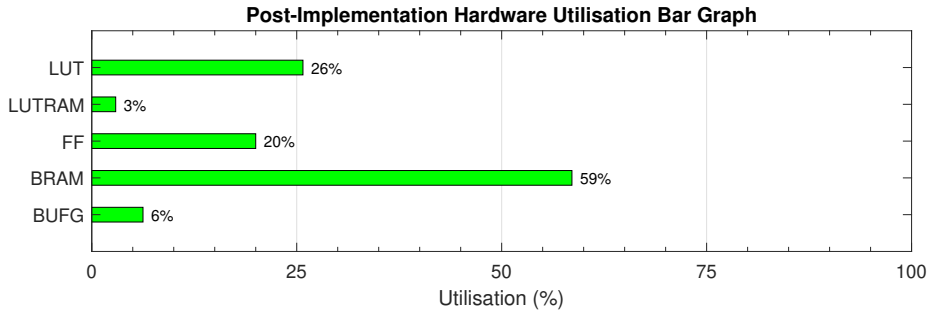


Figure 4.31: Hardware utilisation by complete design

Figure 4.32 shows that the CPU power usage is estimated at 1.5 W.

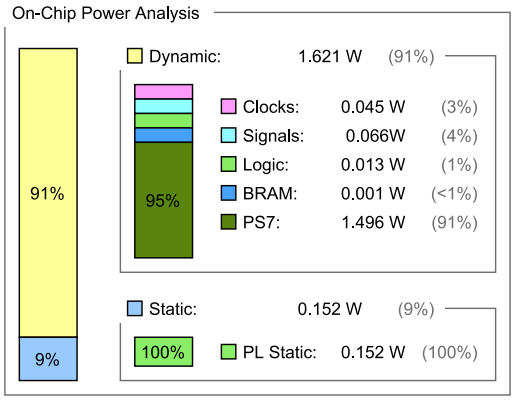


Figure 4.32: On-chip power usage estimate by complete design

Chapter 5

System Verification and Testing

As stated in Section 1.4 the system is designed to detect ArUco markers in images sampled from a single camera sensor and output the marker ID and pose information of the marker relative to the camera at a fast rate. The implemented system was tested to verify that the design works as required by the stated goals. The two main components, shown in Figure 5.1, that were tested are the performance and accuracy of the system. Each component was further broken down into sub-components to test all aspects of the implemented system.

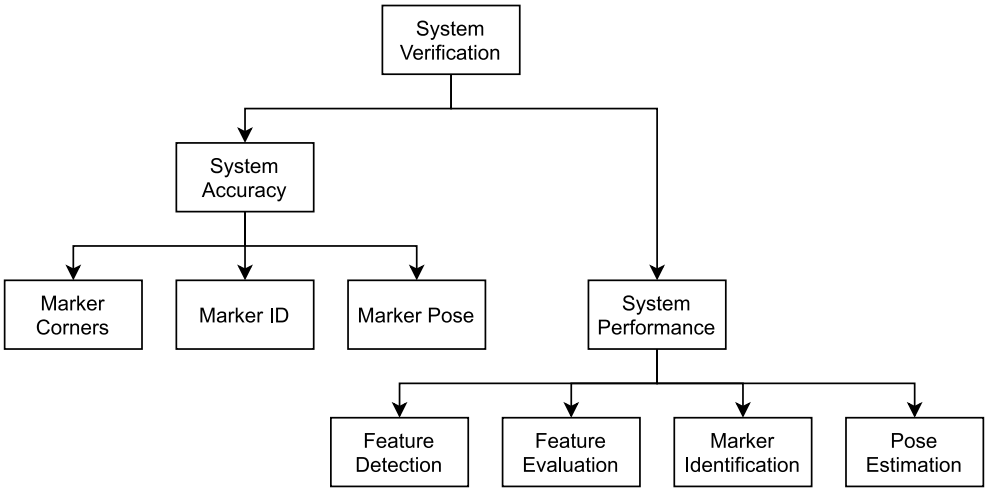


Figure 5.1: Diagram of system testing components (accuracy and performance) and their sub-components

The accuracy of the system was tested to verify that the implemented system operates at an accuracy similar to previously implemented systems. The verification of the accuracy provides the foundation of the performance tests. The increase in system performance is only significant if the accuracy of the system is kept similar to other implemented marker detection systems.

5.1 Test Setup

To test the accuracy and performance of the implemented system a test environment is setup in Gazebo (Robot simulation environment). Gazebo is an open-source 3D robotics simulator. Gazebo is used to design and test robotic implementations in a simulated environment.

The test environment, shown in Figure 5.2, consists of a marker, the camera, environment lighting and walls and a floor to provide visual depth and realism. The camera and marker are placed in the environment at different positions and angles to measure the accuracy of the system.

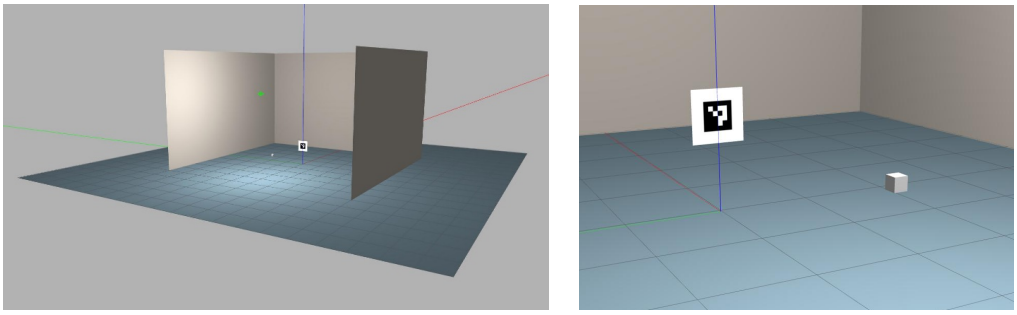


Figure 5.2: Simulated testing environment containing the marker and camera

The simulated camera captures images that are stored on BRAM on the PL. The stored test image simulates an incoming image frame from a camera sensor and is read into the logic to be processed to detect features. The test images provide various marker positions and orientations relative to the camera to be used to verify the functionality and performance of the implemented system.

The camera coordinate system (X, Y, Z) and the marker coordinate system (U, V, W) are shown in Figure 5.3. The origin of the camera coordinate system is the aperture of the camera (ideal pinhole camera model). The origin of the marker coordinate system is the top left corner of the marker.

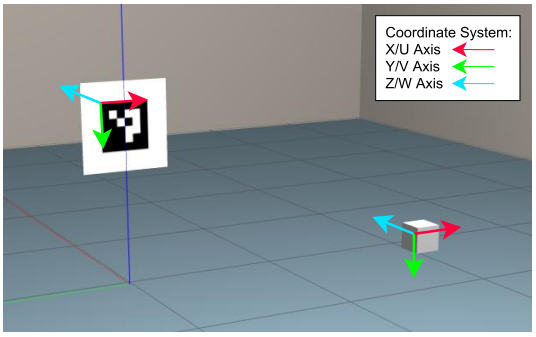
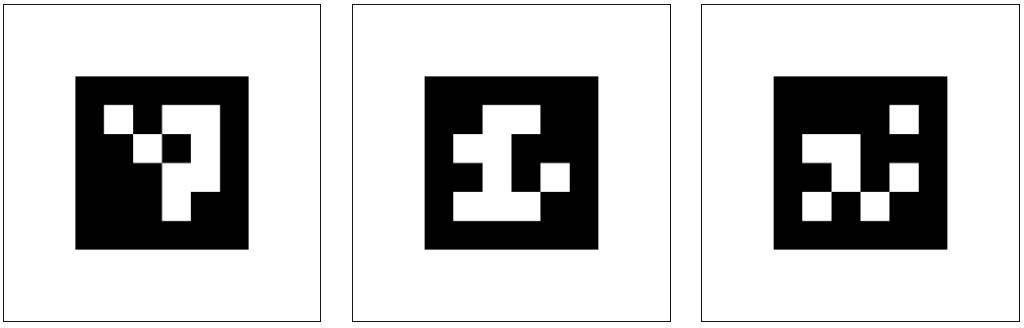


Figure 5.3: Camera and marker coordinate systems

5.1.1 Simulated ArUco Marker

The marker that the system is designed to detect is an ArUco marker that contains a 4 by 4 information matrix. The marker is surrounded by a white border to separate the black marker from its surroundings and to provide high contrast between the marker and white border. Three examples of the marker are shown in Figure 5.4. The square marker has a side length of 30 cm and the width of the white border is 12.5 cm.



(a) Marker ID 0 (b) Marker ID 18 (c) Marker ID 39

Figure 5.4: Examples of ArUco markers

5.1.2 Camera Sensor

The camera sensor used in the simulation is a Robot Operating System (ROS) plugin that simulates a camera sensor and outputs the camera data in the chosen format. The sensor plugin parameters, listed in Table 5.1, are kept to their default values.

Parameter Name	Parameter Value
Width	640px
Height	480px
Focal Length	381.36px
Noise	Gaussian (Standard Deviation = 0.007)

Table 5.1: Camera Parameters

The camera image is stored in BRAM with each pixel represented by its grayscale intensity value (8-bit integer). The pixels are streamed into the image processing system at a rate of 50 MHz to simulate the pixel clock of a physical camera sensor [40].

5.1.3 System Test Variables

To verify that the system elements work as intended certain system parameters are measured when testing.

To measure the accuracy of the implemented design the following test variables, shown in Table 5.2 are measured.

System Elements	Test Variable
Marker Corners	The image frame coordinates of the four corners of the detected marker
Marker Identification	The detected marker ID from the ArUco 4x4 Library
Marker Pose	Translation = [X, Y, Z] and Rotation = [Pitch, Yaw, Roll]

Table 5.2: System test elements and descriptions

To measure the performance of the system each individual function in the marker detection system is measured to determine its processing duration. The timing tests are executed on the PS (the ARM processor on the ZYNQ).

The duration of a function is measured by using the xtime_l library. The xtime_l library provides access to the Performance Monitoring Unit (PMU) global counter which increases every two processor cycles [41].

The start and end of the function global counter value is recorded. The time taken to complete the function is then calculated from the difference between the start and end of the function.

5.2 Verification Tests

Multiple tests are run to determine the accuracy and performance of the implemented system. The time each element takes to complete, marker corner information, marker ID and marker pose are sampled for every test.

The rotation, translation, marker identification and increased feature timing have specific tests to determine their accuracy.

The simulated image stored in BRAM removes any external errors that real world testing would introduce into the system. The system was found to be very consistent during testing. The implemented system is deterministic with constant accuracy results for individual tests. Timing variances of 20 μs for the PS processing functions and 1 μs for the PL were recorded during testing. The system is consistent and therefore each individual test was only repeated 20 times.

The marker identification tests are the only tests where the marker ID is changed to verify if the marker identification system functions correctly.

5.2.1 Marker Identification

The marker detection algorithm is first tested by providing the algorithm with each marker in the ArUco marker 4x4 library that consists of 250 markers. This test, as shown in Figure 5.5, is setup in C code on PC using the Eclipse IDE to quickly verify that the correct markers are detected. The markers are read into the algorithm and the detected marker ID and expected marker ID are compared.

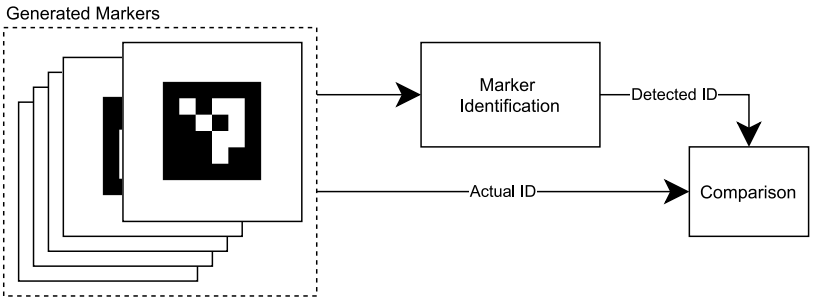
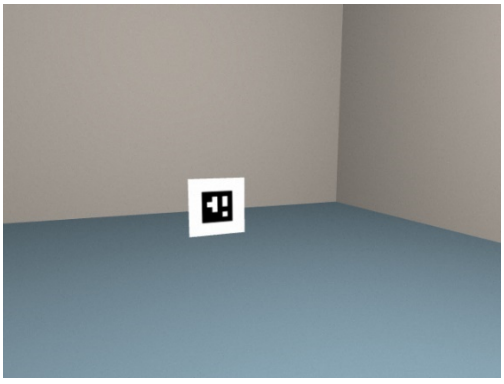
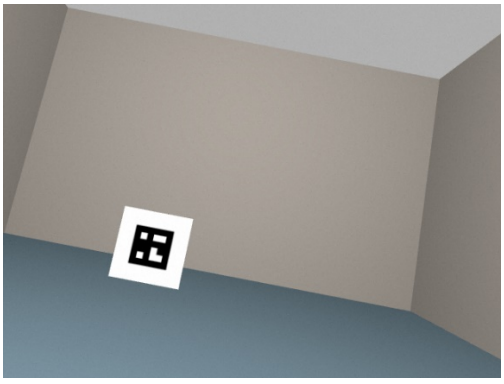


Figure 5.5: Marker identification test

The completed system's marker ID identifying accuracy is also tested by placing 10 different markers in the simulated environment and moving the camera to different positions and orientation. Examples of the test are shown in Figure 5.6. The markers that are used are Marker IDs 0, 7, 15, 21, 36, 43, 52, 76, 88 and 99. The different camera angles and positions provided various perspectives to detect the markers from.



(a) Marker ID 52 with set of camera translations and rotations

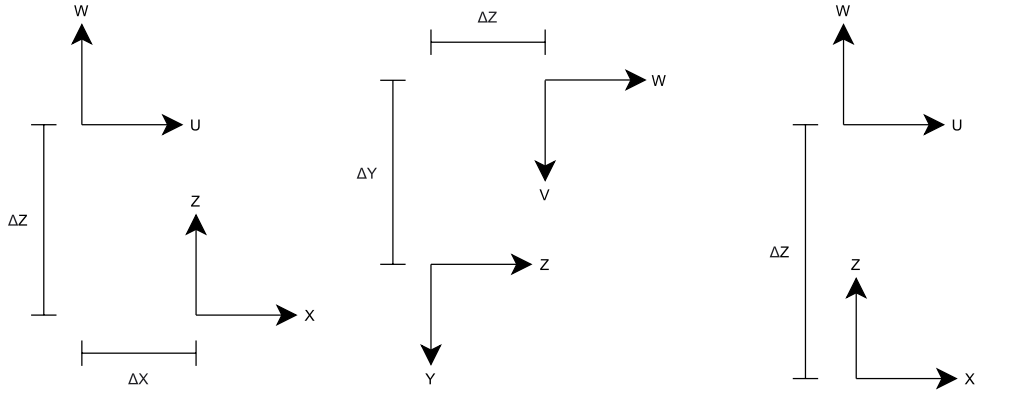


(b) Marker ID 21 with differen set of camera translations and rotations

Figure 5.6: Simulated marker identification test

5.2.2 Translation Tests

To test the implemented system's ability to estimate translation displacement between the camera and marker the camera is moved along each axis taking snapshots at various intervals. The displacement along the X and Y axes is coupled with a fixed Z axis displacement to allow the marker to be in view of the camera. The top right corner of the marker is the origin point of the marker's coordinate system. The displacement is measured from the point of origin. The translation tests are depicted in the diagrams shown in Figure 5.7.

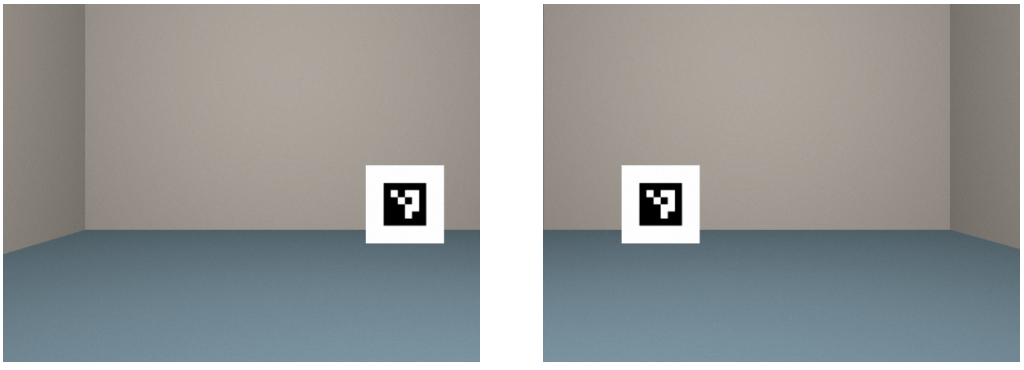


(a) X Axis Displacement (b) Y Axis Displacement (c) Z Axis Displacement

Figure 5.7: Diagrams of axis translation

5.2.2.1 X Axis Displacement

The camera is moved along the X axis and image snapshots are taken at intervals of 0,5 m starting at -1 m and ending at 1 m. The camera is also placed 2 m along the Z axis away from the marker. Examples of X axis displacement are shown in Figure 5.8.

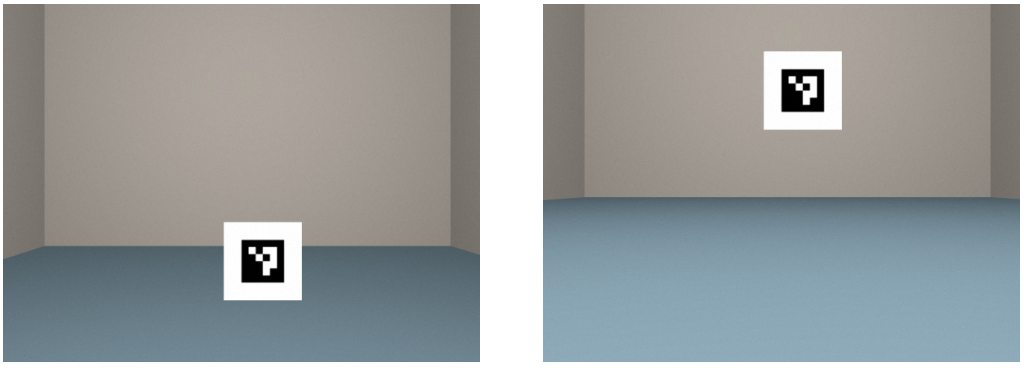


(a) Camera translated 1 m left (b) Camera translated 1 m left

Figure 5.8: X axis displacement test images

5.2.2.2 Y Axis Displacement

The camera is moved along the Y axis and image snapshots are taken at intervals of 0,4 m starting at -0,8 m and ending at 0,8 m. The camera is also placed 2 m along the Z axis away from the marker. Examples of Y axis displacement are shown in Figure 5.9.

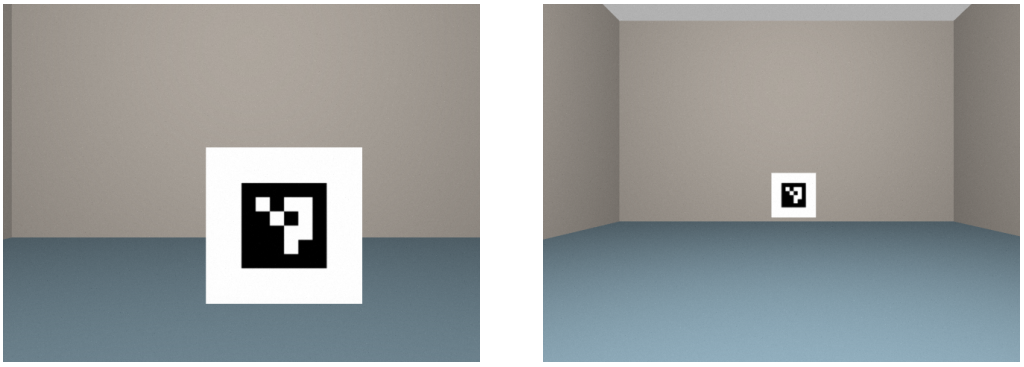


(a) Camera translated 0.4 m up (b) Camera translated 0.8 m down

Figure 5.9: Y axis displacement test images

5.2.2.3 Z Axis Displacement

The camera is moved along the Z axis and image snapshots are taken at intervals of 0,5 m starting at 0,5 m and ending at 4.5 m. The camera points directly at the origin of the marker and the only displacement is along the Z axis. Examples of Z axis displacement are shown in Figure 5.10.

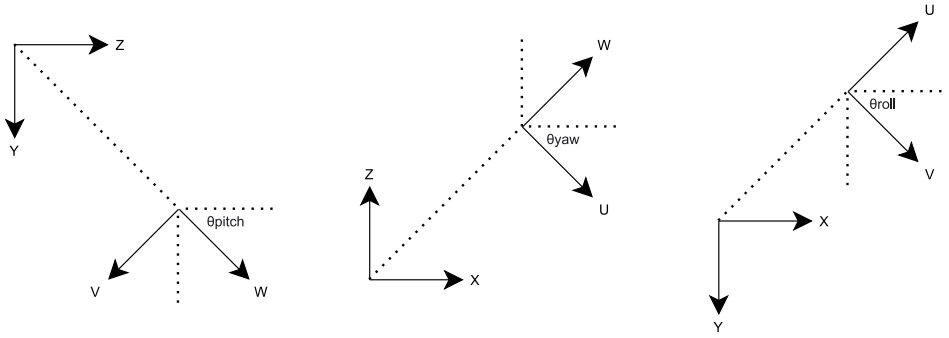


(a) Camera translated 1 m away (b) Camera translated 3.5 m away

Figure 5.10: Z axis displacement test images

5.2.3 Rotation Tests

To test the implemented system's ability to estimate rotation change between the camera and marker the camera is rotated around each axis taking snapshots at various angular intervals. The camera is rotated at certain angles and is moved to point directly at the origin of the marker coordinate system. The rotation tests are depicted in the diagrams shown in Figure 5.11.



(a) Pitch Rotation (b) Yaw Rotation (c) Roll Rotation

Figure 5.11: Diagrams of axis rotation

5.2.3.1 Pitch Rotation

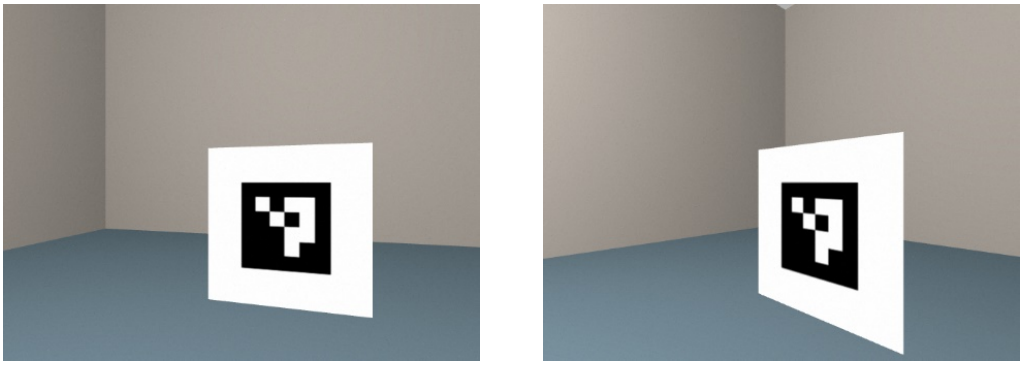
The camera is rotated around the X axis and image snapshots are taken at intervals of 15° starting at 0° and ending at 60°. The camera is kept at a constant distance of 1 m away from the point of origin of the marker. Examples of pitch rotation are shown in Figure 5.12.

(a) Camera rotated 30° in pitch axis (b) Camera rotated 60° in pitch axis

Figure 5.12: Pitch rotation test images

5.2.3.2 Yaw Rotation

The camera is rotated around the Y axis and image snapshots are taken at intervals of 15° starting at 0° and ending at 60°. The camera is kept at a constant distance of 1 m away from the point of origin of the marker. Examples of yaw rotation are shown in Figure 5.13.



(a) Camera rotated 15° in yaw axis (b) Camera rotated 45° in yaw axis

Figure 5.13: Yaw rotation test images

5.2.3.3 Roll Rotation

The camera is rotated around the Z axis and image snapshots are taken at intervals of 30° starting at 0° and ending at 330°. The camera is kept at a constant distance of 1 m away from the point of origin of the marker. Examples of roll rotation shown are in Figure 5.14.

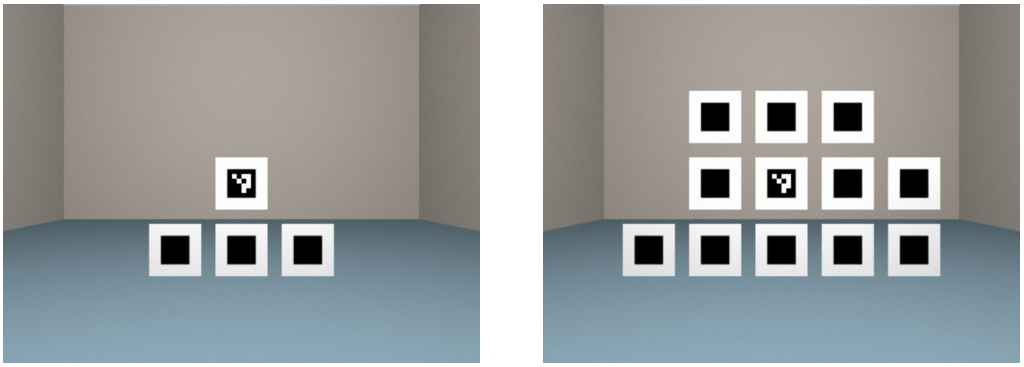
(a) Camera rotated 30° in roll axis (b) Camera rotated 300° in roll axis

Figure 5.14: Roll rotation test images

5.2.4 Increased Feature Testing

To measure the effects of the number of corners and blobs on the completion time of the different marker detection elements a test is run that has an increasing number of squares in each test image.

The center square is an ArUco marker to give the system a marker to detect. Each subsequent image has a black square with a white border (similar to the ArUco marker) added to it. The test starts at 1 square (the ArUco marker) and ends at 15 squares added in the environment. Each added square adds a single detectable blob and multiple detectable corners to the environment. Examples of the increased feature test images are shown in Figure 5.15.



(a) Test image with 4 squares (b) Test image with 12 squares

Figure 5.15: Increased feature test images

5.3 System Accuracy Analysis

The accuracy of the system is determined by the accuracy of the marker pose and marker ID. The accuracy of the system has dependencies, shown in Figure 5.16. The accuracy of the marker ID is dependent on the marker identification algorithm which is in turn dependent on the accuracy of the detected marker corners. The marker pose is dependent on the accuracy of the pose estimation algorithm which is in turn dependent on the marker identification algorithm and the detected marker corners.

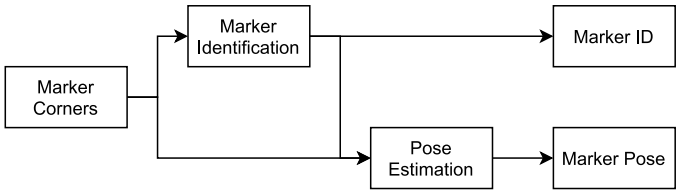


Figure 5.16: Diagram of function accuracy dependencies

5.3.1 Marker Corner Accuracy

The accuracy of the marker corners can be determined by comparing the detected corner coordinates to the absolute corner coordinates. The absolute corner coordinates are known for all the test images that do not have perspective distortion by yaw or pitch rotations. The side dimensions of the marker are known and the pixel width of the marker can be determined by examining the Z axis displacement test images.

The width of the marker (0.3 m) and the pixel width of the marker in the image at 0.5 m is 229 px shown in Figure 5.17.

The pixel width at 0.5 m away from the marker on the marker plane is 1.31 mm.

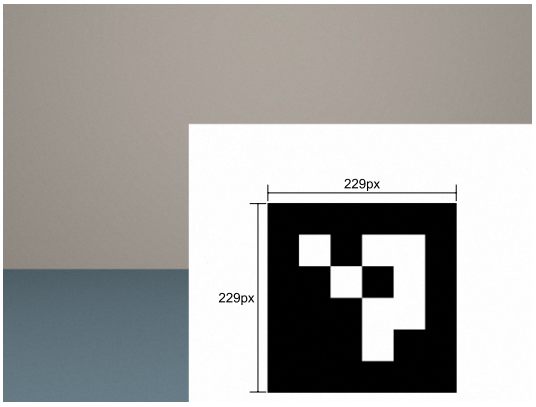


Figure 5.17: Marker width in pixels

By measuring the pixel width of the marker at each test distance the pixel width at each distance can be calculated. The distance dependent pixel widths are shown in Table 5.3.

Distance (m)	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5
Marker Width (px)	229	114	76	57	46	38	33	29	26
Pixel Width (mm)	1.31	2.63	3.95	5.26	6.52	7.9	9.09	10.34	11.53

Table 5.3: Pixel width at test distances

The corner detection algorithm determines the corner as a set of X and Y coordinates in the image plane. The center of the pixel is the corner that has been detected. As the pixel width increases so does the error between the detected corner (if the detected corner is the pixel on the exact vertex) and the absolute corner. The corner error is shown in Figure 5.18.

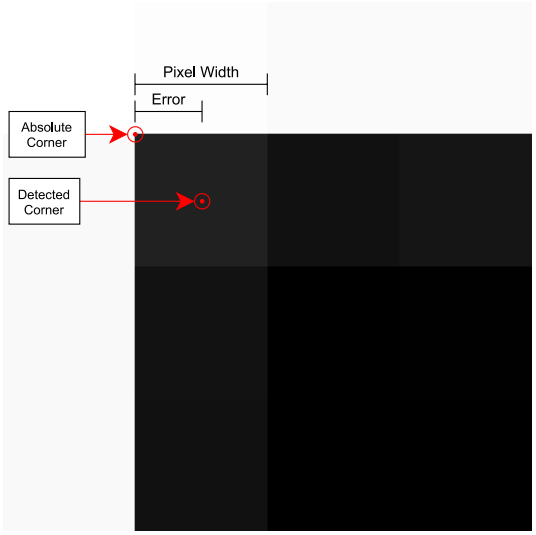


Figure 5.18: Pixel level breakdown of corner error

To determine the accuracy of the detected corners the absolute corners for a test are compared to the detected corners. The absolute corners are represented in image dimensions (m) e.g. the coordinates (320px, 420px) of the center pixel at 1 m away are (0.8416 m, 0,6312 m). The detected corners are converted to the image dimensions by multiplying the corner pixel coordinate by the pixel width. The center of the pixel is then determined by adding half of the pixel width at that distance. The corner error is the average of the four corner Euclidean distances between the detected and absolute corners.

5.3.1.1 Z Axis Translation Corner Accuracy

The Z Axis translation test results are shown in Figure 5.19. As the pixel width increases so does the error between the detected corner and absolute corner. The corner error plot also shows an increase in detected corner error as distance is increased. The detected corner error is dependent on the corner detection algorithm and camera distance from marker. By plotting the corner error versus the pixel width, the linear relationship is made evident.

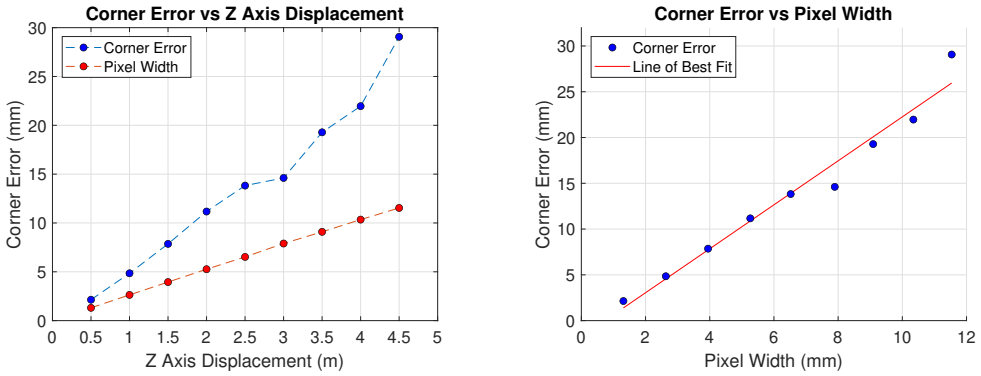


Figure 5.19: Z Axis translation corner accuracy test results

5.3.1.2 X/Y Translation and Roll Corner Accuracy

The X and Y axis translation tests are 2 m away from the marker and from the relationship between corner error and Z displacement the approximate corner error should be 12 mm. The tested corner error average is 10.348 mm with a standard deviation of 0.671 mm.

The roll test is 1 m away from the marker and the expected corner error at 1 m away is 5 mm. The tested roll corner error average is 4.14 mm with a standard deviation of 0.649 mm.

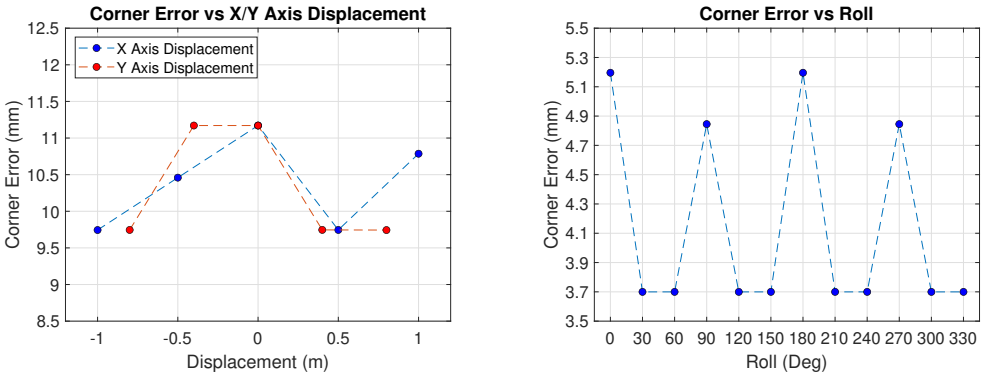


Figure 5.20: Translation and rotation corner accuracy test results

5.3.2 Marker Pose Accuracy

The accuracy of the marker pose is determined by comparing the estimated pose with the actual pose of the marker. The actual pose of the marker is known as the simulated environment provides the exact pose of each component.

The pose estimation algorithm uses an error value to iterate and estimate the pose. The error value is the average of the Euclidean difference between the detected marker corners and the projected corners. The higher the error the higher the uncertainty of the pose estimation output.

5.3.2.1 Translation Tests

The camera is translated along each axis and at each interval the pose information was recorded. The pose information estimated by the implemented system is compared to the known pose information.

X Displacement

The camera is translated along the X axis while keeping the Z displacement 2 m away from the origin point of the marker. From the test results, shown in Figure 5.21, the average translation error is 63.6 mm with a standard deviation of 35.61 mm. The average rotation error is 13.79° with a standard deviation of 7.79°. The pose estimation algorithm has an average uncertainty of 7.34 mm and an average corner error of 10.83 mm.

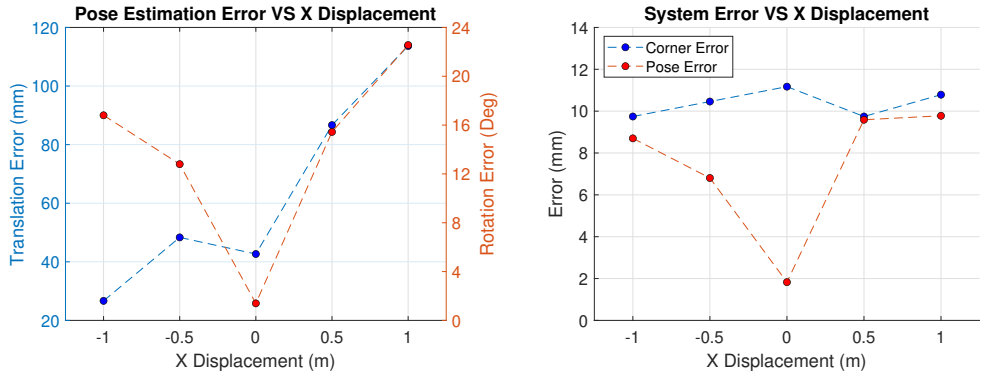


Figure 5.21: Pose estimation results for X axis displacement test

Y Displacement

The camera is translated along the Y axis while keeping the Z displacement 2 m away from the origin point of the marker. From the test results, shown in Figure 5.22, the average translation error is 62.13 mm with a standard deviation of 28.74 mm. The average rotation error is 12.15° with a standard deviation of 6.71°. The pose estimation algorithm has an average uncertainty of 6.06 mm and an average corner error of 10.38 mm.

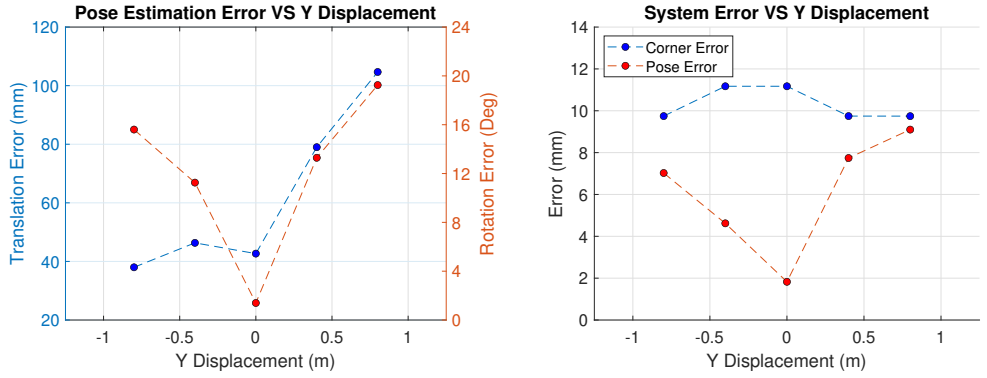


Figure 5.22: Pose estimation results for Y axis displacement test

Z Displacement

The camera is translated along the Z axis while keeping it pointing towards the origin of the marker. From the test results, shown in Figure 5.23, the average translation error is 68.7 mm with a standard deviation of 58.71 mm. The average rotation error is 2.34° with a standard deviation of 1.58°. The pose estimation algorithm has an average uncertainty of 2.43 mm and an average corner error of 13.86mm.

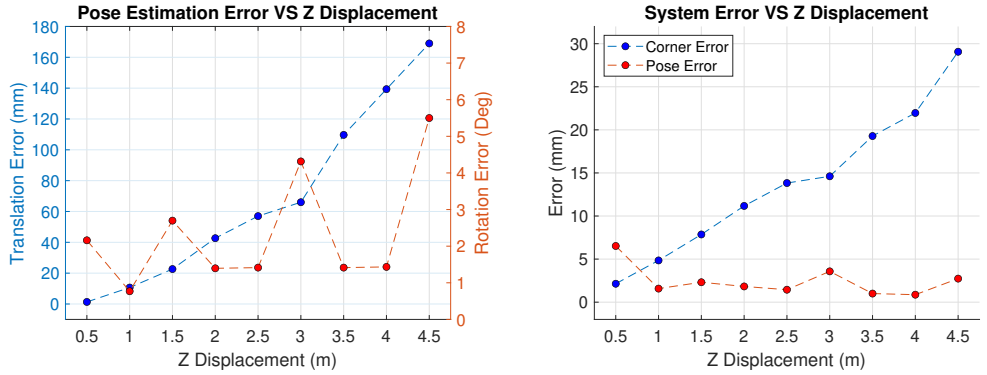


Figure 5.23: Pose estimation results for Z axis displacement test

Translation Test Analysis

From the test results the average translation error is 65.63 mm and the average rotation error is 7.9°. The average error is the result of the corner error and pose error.

The X and Y translation error has a larger error on one side of the marker showing that the pose estimation algorithm has a higher accuracy when the marker is viewed from one side. The rotation error for the X and Y displacement tests have a higher correlation with the pose error than the corner error.

The translation error of the Z displacement test has a high correlation with the corner error. The linear correlation can be seen in Figure 5.24.

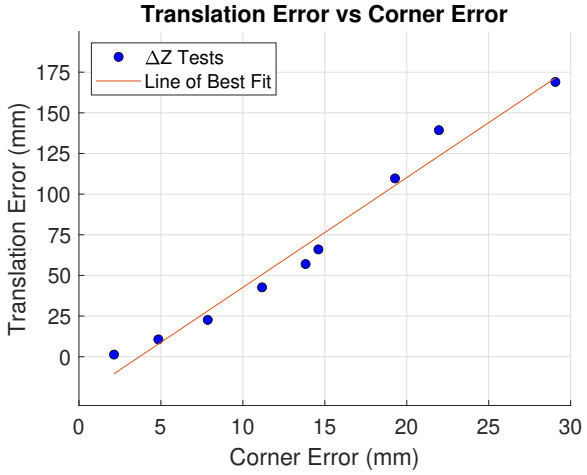


Figure 5.24: Relationship between corner error and translation error

The average translation error of 6.5 cm is very close to the previously implemented marker detection systems' average position error of 3 cm. The average rotation error of 7.9° is close to the previously implemented marker detection systems' average position error of 1.8°.

The results of the translation accuracy test indicate that the results of the timing analysis for tests consisting of translation displacement are significant and comparable to previously implemented marker detection systems.

5.3.2.2 Rotation Tests

The camera is rotated around each axis and at each interval the pose information is recorded.

Pitch Rotation

The camera is rotated around the X axis while keeping it 1 m away from the origin point of the marker. From the test results, shown in Figure 5.25, the average translation error is 10.73 mm with a standard deviation of 3.36 mm. The average rotation error is 0.85° with a standard deviation of 0.79°. The pose estimation algorithm has an average uncertainty of 2.52 mm.

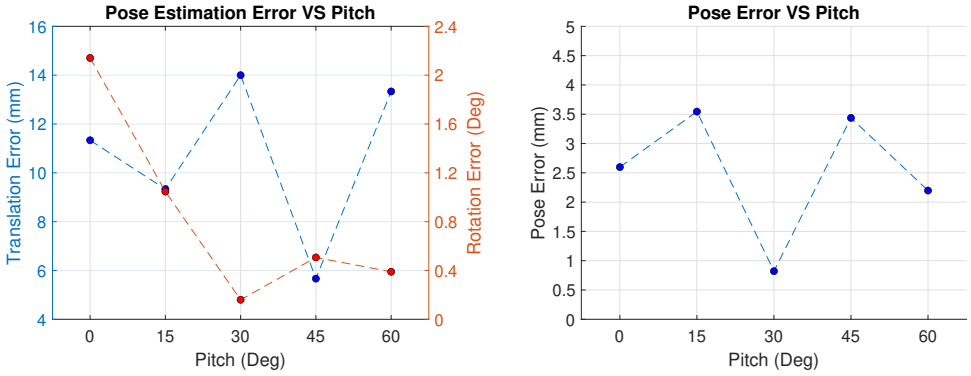


Figure 5.25: Pose estimation results for pitch rotation test

Yaw Rotation

The camera is rotated around the Y axis while keeping it 1 m away from the origin point of the marker. From the test results, shown in Figure 5.26, the average translation error is 15.93 mm with a standard deviation of 10.76 mm. The average rotation error is 2.18° with a standard deviation of 2.72°. The pose estimation algorithm has an average uncertainty of 4.27 mm.

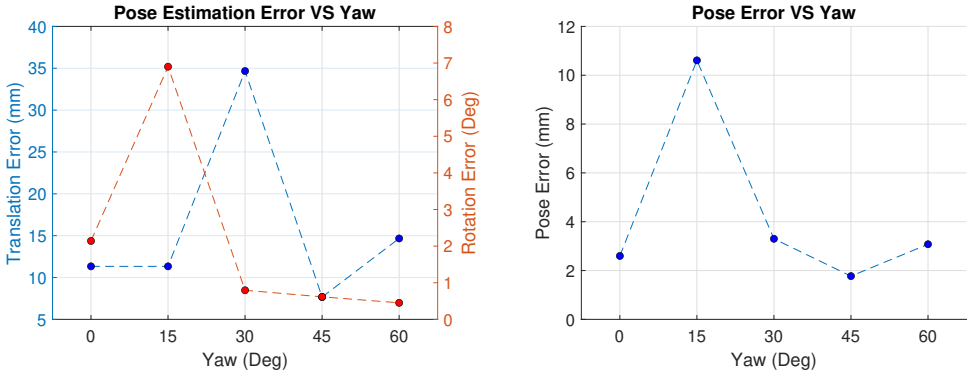


Figure 5.26: Pose estimation results for yaw rotation test

Roll Rotation

The camera is rotated around the Z axis while keeping it 1 m away from the origin point of the marker. From the test results, shown in Figure 5.27, the average translation error is 9.33 mm with a standard deviation of 1.59 mm. The average rotation error is 2.53° with a standard deviation of 0.71°. The pose estimation algorithm has an average uncertainty of 4.27 mm.

Roll does not add perspective distortion to the marker therefore the corner error can be measured. The average corner error is 4.19 mm.

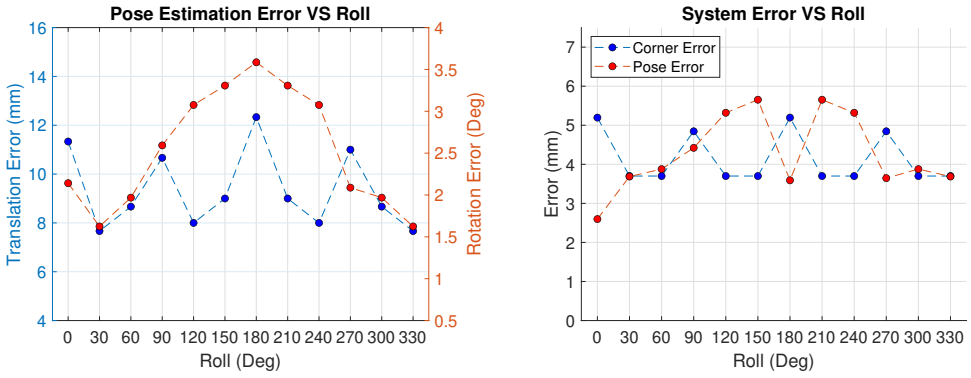


Figure 5.27: Pose estimation results for roll rotation test

Rotation Test Analysis

The pose estimation accuracy is dependent on the accuracy of the corners provided as input and the accuracy of the pose estimation algorithm. From the test results the average translation error is 11.15 mm and the average rotation error is 2.06°. The average error is the result of the corner error and pose error.

Detected corners for markers that are 1 m away have an estimate error of 5 mm. With roll the corner error is closer to the estimate of 5 mm, but with pitch and yaw the error can change depending on the perspective distortion of the marker.

The roll line plot shows a stronger correlation between corner error and translation error than with the pose error. All the rotation test result plots show a stronger correlation between the pose error and rotation error than with the corner error.

The average translation error of 1.1 cm is more accurate than previously implemented marker detection systems' average position error of 3 cm. The average rotation error of 2.06° is close to the previously implemented marker detection systems' average position error of 1.8°.

The results of the rotation accuracy test indicate that the results of the timing analysis for tests consisting of rotational displacement are significant and comparable to previously implemented marker detection systems.

5.4 System Timing Analysis

To provide perspective to the significance of the timing results of the implemented system an equivalent marker detection system was implemented on the processing system that serves as a baseline to compare the results to.

The Hybrid system (the implemented solution) consists of the PL and the PS. The PL is made up of the hardware image processing algorithms that detect features and binarises the image. The rest of the marker detection processing is run on the PS.

The processing system only equivalent (referred to as PS-Only) has all the components of the image processing and marker detection system all run sequentially on the processing system. The PS-Only implementation is similar to traditional marker detection systems. The PS-Only timing also incorporates an image capture time offset to account for the time taken to read a 640×480 image at a rate of 50 MHz (offset of 6144 μs) into memory.

The visual timing breakdown of the two systems and their respective durations to process a single marker in a test image is shown in Figure 5.28.

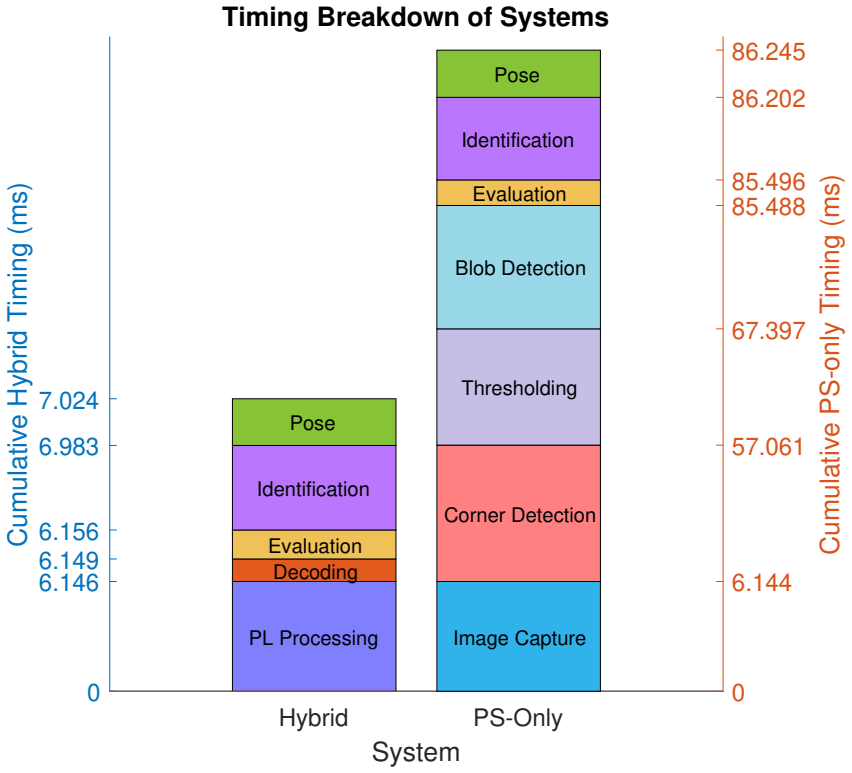


Figure 5.28: Time consumption by marker detection functions of both systems

5.4.1 Increased Feature Analysis

By increasing the black squares visible in the image, the detectable blobs and corners are increased.

From the increased feature test results, shown in Figure 5.29, the average time difference between the PS-Only system and Hybrid system is 80.23 ms. The Hybrid system is significantly faster than the PS-Only system, because 3 of the PS-Only functions are run in parallel on the FPGA and are sped up due to the FPGAs high throughput capabilities.

The step pattern shown on the graph is due to the marker identification function duration increasing, further discussed in Section 5.4.1.6.

The system speed-up starts at 12.3× and lowers to 7×. The decrease in speed-up is discussed in Section 5.4.1.1.

On average the Hybrid system is 9.87× faster than the PS-Only system.

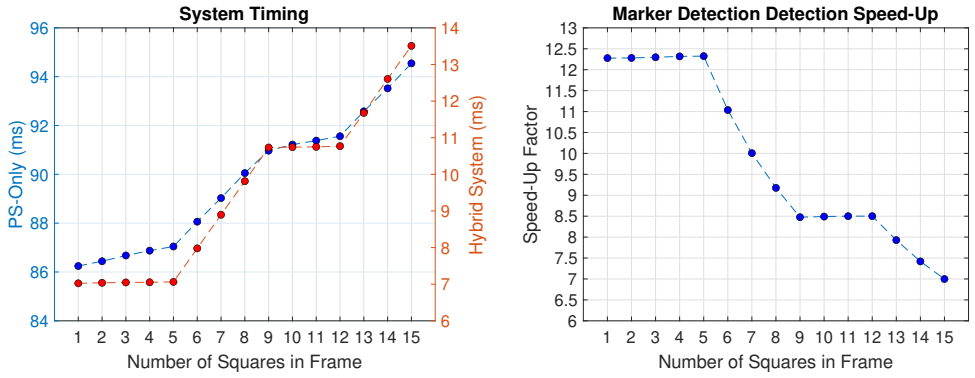


Figure 5.29: Increased feature test results for complete system

5.4.1.1 Speed-Up Decrease

The decrease in system speed-up is due to the increase in the processing time of the marker identification function. The identification function runs on the processing system and the time it takes increases as more marker candidates are evaluated.

At 1 square the identification function takes up a small part, only 12%, of the overall system time. At 15 squares the identification takes up 53% of the overall time. The increase in percentage time taken in shown is Figure 5.30.

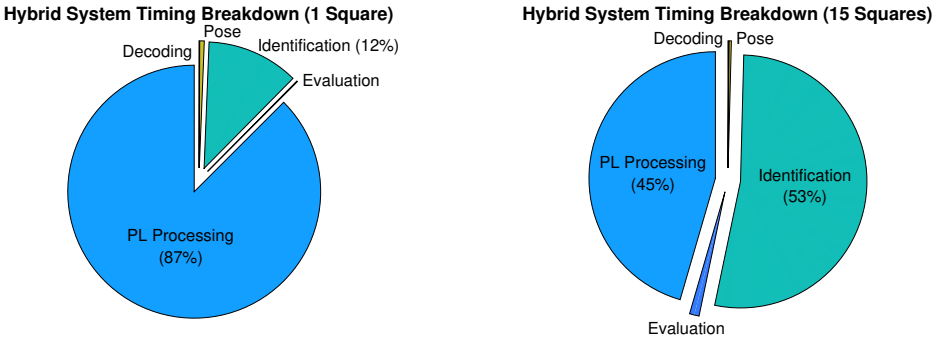


Figure 5.30: Timing breakdown of Hybrid design

The identification function has a smaller impact on the overall PS-Only timing. At 1 square the identification function takes up only 1% and at 15 squares it takes up 7%. The smaller impact on system timing is shown in Figure 5.31.

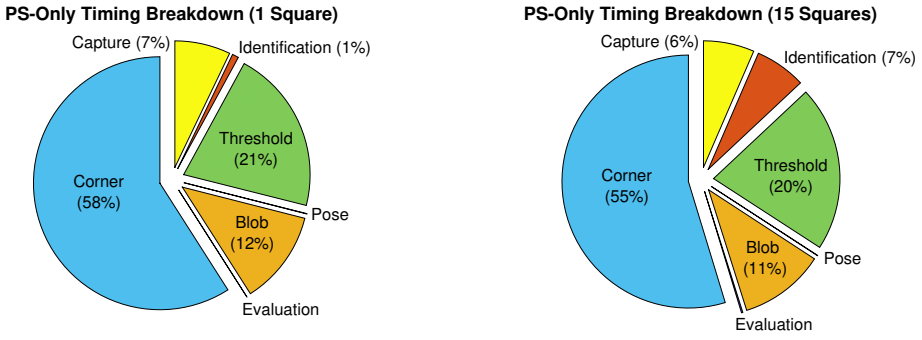


Figure 5.31: Timing breakdown of PS-Only design

The overall speed-up of the Hybrid system compared to the PS-Only system decreases because the impact of the increase of marker identification time is larger on the Hybrid system than on the PS-Only system.

5.4.1.2 Corner Detection

Both the PS-Only and Hybrid corner detection functions have a linear time increase dependent on number of detectable corners. As the squares increase, the corners increase and as the corners increase the processing time increases.

From the test results, shown in Figure 5.32, the average difference between the PS-Only and Hybrid system is 45.16 ms with a standard deviation of 0.249 ms.

The corner detection function run on the FPGA has a near constant duration of 6144 μs (the average duration is 6144.63 μs with a standard deviation of 0.076 μs). The increase in the Hybrid system corner detection time is dependent on the processing time required to decode the received corner packages sent from the PL to the PS.

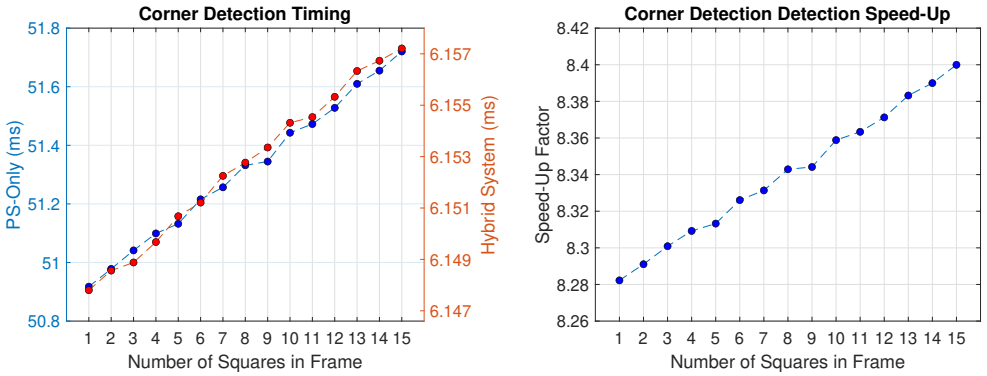


Figure 5.32: Increased feature test results for corner detection

The increase in Hybrid system duration is smaller than the increase in PS-Only duration. The Hybrid corner detection time increases from 1 square to 15 squares at a scale of 1.002× while the PS-Only corner detection time increases at a scale of 1.018×.

The difference in timing increase causes an increase in function speed-up as the detectable corners increase. The average speed-up is 8.34×.

5.4.1.3 Blob Detection

Both the PS-Only and Hybrid blob detection functions have a linear time increase dependent on the number of detectable blobs. As the squares increase the detectable blobs increase which increases processing time.

From the results, shown is Figure 5.33, the average difference between the PS-Only and Hybrid system is 12.9 ms with a standard deviation of 0.599 ms.

The blob detection function run on the FPGA has a near constant duration of 6144 μs (the average duration is 6144.63 μs with a standard deviation of 0.076 μs). The blob detection and corner detection functions have the exact same duration because they run in parallel and the function duration is measured by how long the system takes to read in all the detected features from the PL.

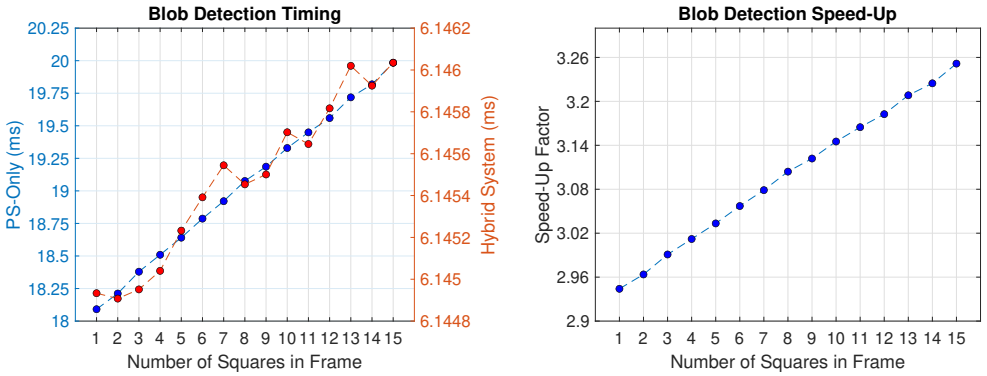


Figure 5.33: Increased feature test results for blob detection

The increase in the Hybrid system blob detection time is dependent on the processing time required to decode the received blob packages sent from the PL to the PS.

The increase in Hybrid system duration is smaller than the PS-Only duration. The Hybrid blob detection time increases from 1 square to 15 squares at a scale of 1.0002× while the PS-only blob detection time increases at a scale of 1.1049×.

The difference in timing increase causes an increase in function speed-up as the detectable blobs increase. The average speed-up is 3.1×.

5.4.1.4 Threshold

Threshold timing is constant for both the PS-Only and Hybrid systems, shown in Figure 5.34, because the processing system evaluates each individual pixel.

The average duration for the PS-Only system is 10.34 ms with a standard deviation of 0.0033 ms. The average duration of the Hybrid system is 6.145 ms with a standard deviation of 0.09 μs. With the consistency in the threshold duration of each system the function speed-up is also constant. The average speed-up is 1.68× with an average difference between the PS-Only and Hybrid system of 4.19 ms.

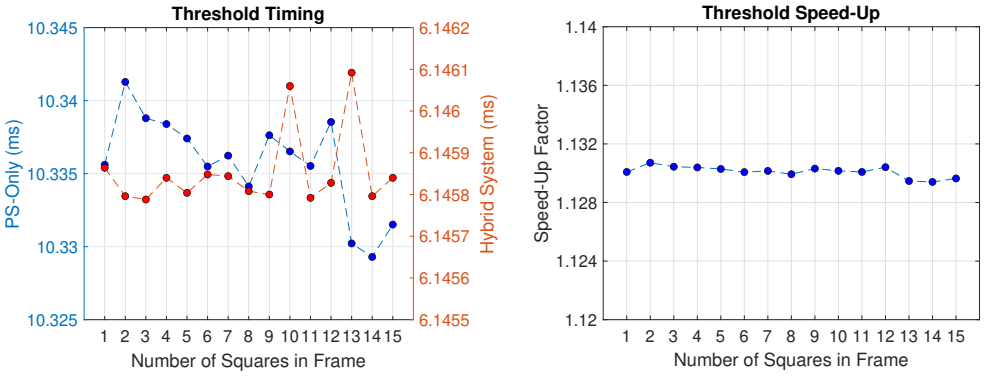


Figure 5.34: Increased feature test results for thresholding

5.4.1.5 Feature Evaluation

The feature evaluation evaluates the exact same features and runs on the processing system. The increase in feature evaluation duration is due to the increase in detected corners and blobs. The increase in features increases the feature evaluation duration, as shown in Figure 5.35 for both implementations.

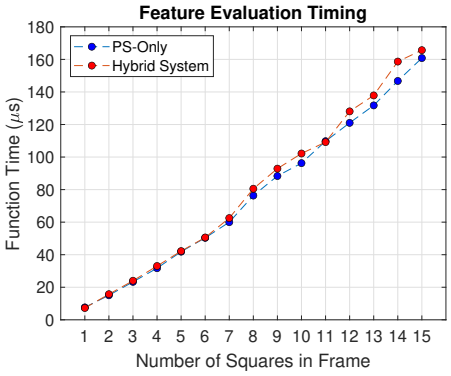


Figure 5.35: Increased feature test results for feature evaluation

5.4.1.6 Marker Identification

Marker identification evaluates each marker candidate (a blob with four corners) that is detected in the image frame. Each candidate is evaluated in the order the blobs are detected in the image frame. Each candidate is rectified and evaluated to determine if it is an ArUco marker. If a marker is found the function is stopped and the marker information is sent to the pose estimation function.

The large time increase on the graph in the form of steps seen in Figure 5.36 is due to the system evaluating the squares before the actual marker. As the tests progress and squares are added to the image all the squares that are detected before the ArUco marker increase the function duration and all the square detected after the ArUco marker are not evaluated.

The average duration for the Hybrid system marker identification is 3.24 ms with a standard deviation of 2.17 ms. The Hybrid system takes longer to process than the PS-Only. The Hybrid system uses DMA to write the binary image to memory while the PS-Only system has the binary image stored in an array that is stored on the stack. The marker identification function takes longer to read from DDR than from the stack. The increase in memory read time means that the average speed-up of the function is 0.858×.

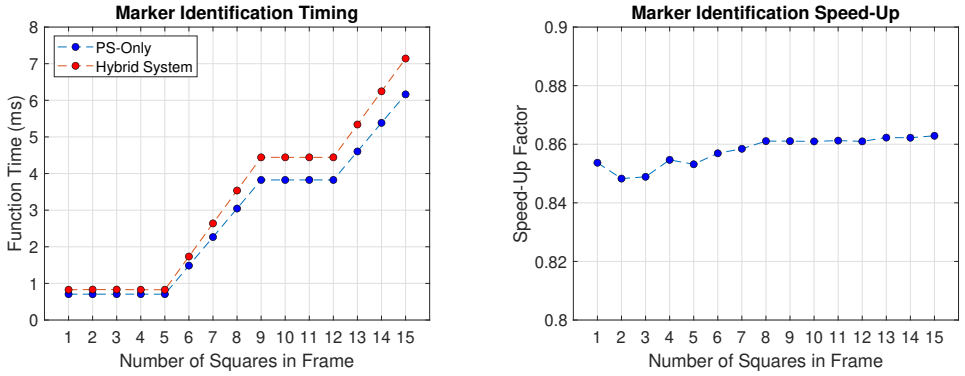


Figure 5.36: Increased feature test results for marker identification

5.4.1.7 Pose Estimation

The pose estimation function for the Hybrid and PS-Only system are the same and and both run on the PS. The duration of the function is dependent on the algorithm iterating enough to produce an output with a small error. The average duration the pose estimation function takes is 41.35 μs with a standard deviation of 0.21 μs.

5.4.2 System Consistency

The Hybrid system is compared to the PS-Only system to determine how consistent the system is as features increase and test variables change.

5.4.2.1 Corner Detection Consistency

The corner detection data was gathered from all the tests that were conducted. The corner detection data for the Hybrid and PS-Only systems is shown in Figure 5.37 and Figure 5.38 respectively.

The scatter plot of the Hybrid system shows a clear linear relationship between number of detectable corners and function duration (indicated by the line of best fit). As the detectable corners increase so does the function duration increase.

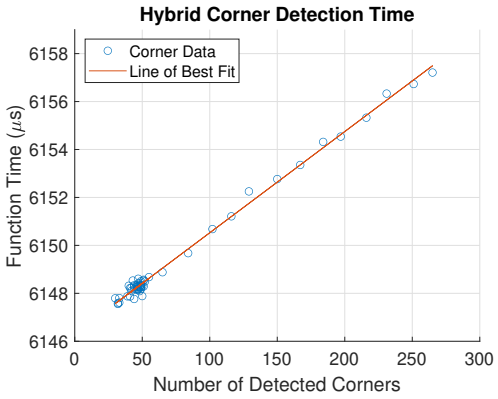


Figure 5.37: Hybrid system corner detection timing analysis

The PS-Only scatter plot shows that the duration of the corner detection function is not only dependent on the number of detectable corners, but that it is also dependent on the rotation of the marker.

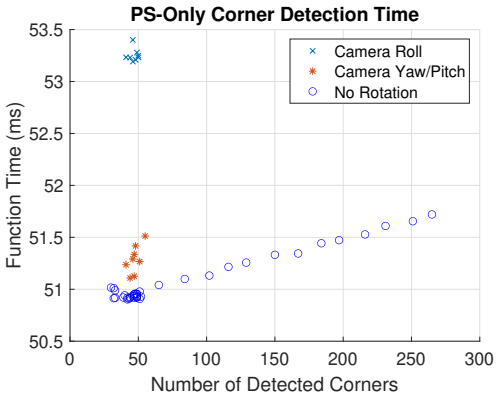


Figure 5.38: PS-Only system corner detection timing analysis

The PS-Only implementation of the FAST algorithm evaluates all pixels that pass certain criteria, discussed in Appendix A.3.2.1.

As the angle of the marker corner deviates from being perpendicular to the X and Y axis of the image frame the number of pixels that must be evaluated increases. The increase of evaluated pixels increases the processing time of the corner detection function.

The relationship between the number of evaluated pixels and marker rotation is shown in Figure 5.39.

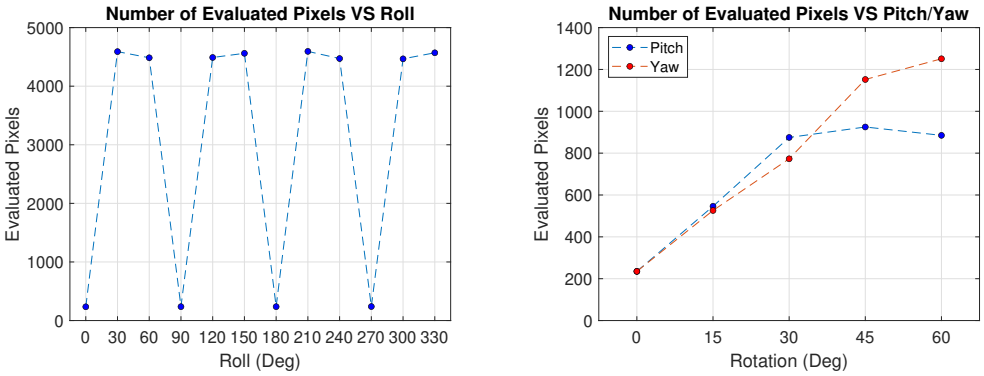


Figure 5.39: Relationship between the number of evaluated pixels and marker rotation for PS-only implementation

5.4.2.2 Blob Detection Consistency

The blob detection data was gathered from all the tests that were conducted. The blob detection data for the Hybrid and PS-Only systems is shown in Figure 5.40 and Figure 5.41 respectively.

The scatter plot of the Hybrid system shows a clear linear relationship between number of detectable blobs and function duration (indicated by the line of best fit). As the detectable blobs increase so does the function duration.

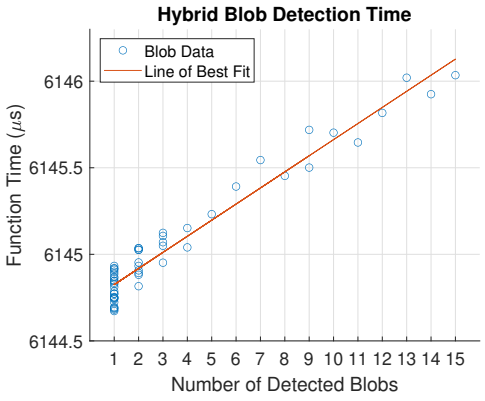


Figure 5.40: Hybrid system blob detection timing analysis

The PS-Only scatter plot shows that the duration of the blob detection function is not only dependent on the number of detectable blobs, but that it is also dependent on the number of black pixels in the binary image.

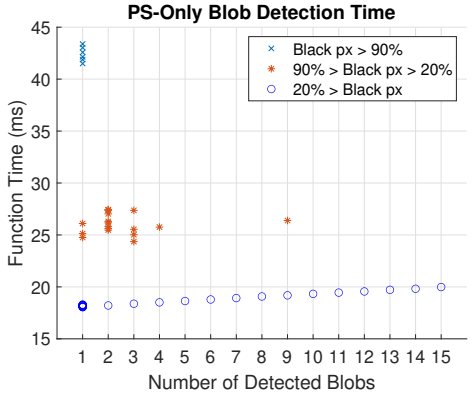
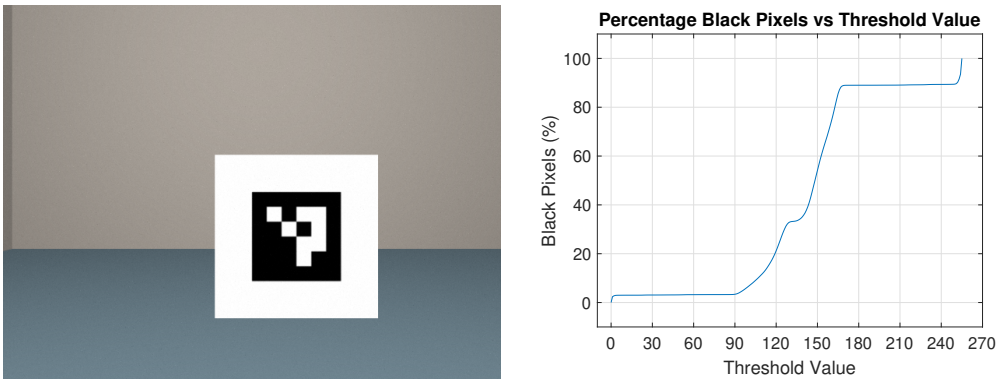


Figure 5.41: PS-Only system blob detection timing analysis

The threshold value is the value that is used to threshold an image. From the test image, shown in Figure 5.42a, a plot is generated to show how the threshold value affects the number of black pixels in the image, shown in Figure 5.42b. The plot shows that a high threshold value produces a binary image with many black pixels.



(a) Test image (b) Line plot of test image

Figure 5.42: The relationship between threshold value and number of black pixels

The scatter plot of the blob detection function that is implemented on the PS shows that there is a linear correlation between the number of black pixels and function duration. The more pixels that must be evaluated the longer the function takes to complete. The linear relationship between the blob detection function duration and the number of black pixels is shown in Figure 5.43.

Figure 5.43: Relationship between number of black pixels and PS-Only blob detection duration

5.5 Testing and Verification Conclusion

5.5.1 Accuracy Analysis

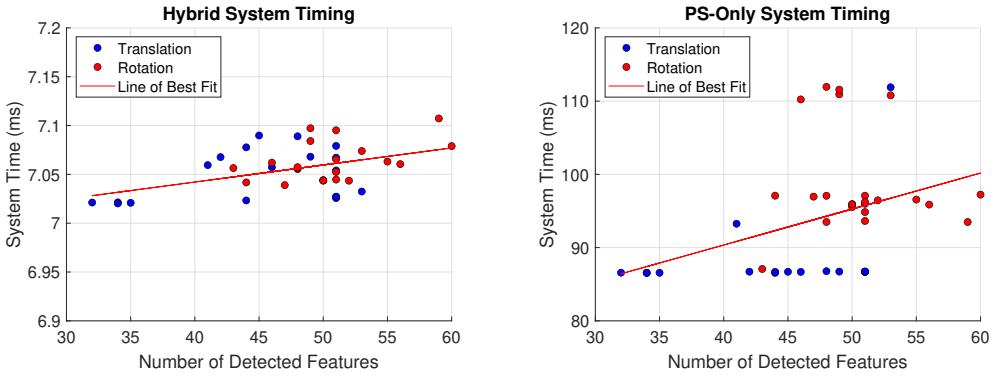
The tests results show that the implemented marker detection system can identify markers accurately and estimate the pose of the marker relative to the camera pose. The accuracy results of all the tests are:

- Average translation accuracy of 4.96 cm.
- Average rotation accuracy of 4.28°.

The accuracy of the implemented system is comparable to the previously implemented marker detection systems discussed in Chapter 2. Comparable accuracy indicates that the implemented systems' timing results are significant and can be compared to the previously implemented systems.

5.5.2 Performance Analysis

The results of the complete system timing of the translation and rotation tests are shown in Figure 5.44. The timing results of the Hybrid system are more consistent than the PS-Only system. Rotation results of the PS-Only are more inconsistent showing the impact the rotation of the marker has on the overall system performance.



(a) Hybrid Timing Results

(b) PS-Only Timing Results

Figure 5.44: Timing results comparison between the Hybrid System (a) and the PS-Only System (b)

The speed-up of the processing time between the Hybrid implementation and the PS-Only is large, as shown in Figure 5.45. The average speed-up for the translation tests is 12.6× and the average speed-up for the rotation tests is 14×.

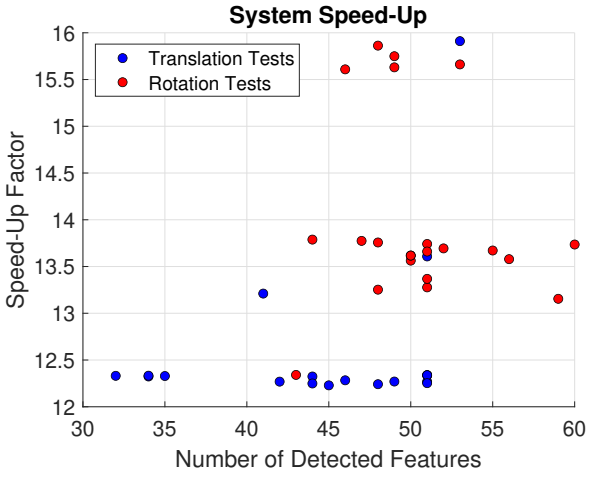


Figure 5.45: Speed-up of Hybrid system compared to the PS-Only

With the increased feature test average speed-up of 9.87×, from Figure 5.29, the average speed-up of all the marker tests is 12.16×.

The large speed-up of the Hybrid system compared to the PS-Only shows that the utilisation of the FPGA to accelerate and parallelise the data heavy image processing functions is successful in decreasing overall processing time. The processing time is also decreased by in-line processing the image data to extract information without the need to store the image into PS memory.

The Hybrid implementation that utilises the FPGA and the CPU to accelerate the fiducial marker-based localisation system runs at an average rate of 134 FPS.

Chapter 6

Conclusion and Recommendations

6.1 Overview of Investigation

The aim of the project was to design a fiducial marker detection system that runs on a light-weight-, compact- and low-power hardware system that can detect markers and accurately estimate pose at high frame rates.

To better understand the problem to be able to formulate a solution, various types of fiducial markers and implementations of fiducial markers were investigated and discussed in the literature review. The literature review into previously implemented marker detection systems showed the current applications of fiducial marker-based localisation methods are limited by immobile processing systems (PCs). To achieve high frame rates of 45 FPS marker detection systems are run on relatively powerful hardware platforms that are not suitable for light-weight-, compact- and low-power autonomous systems.

The previously implemented marker detection systems' limitations highlighted in the literature study were addressed by designing a concept for a high frame rate marker detection system that runs on a light-weight-, compact- and low-power hardware platform. The high-level functionality of the system was designed to process image information to identify ArUco markers and calculate the pose of the camera relative to the marker. To achieve the proposed functionality the components of the design were identified. The identified components were designed to achieve the goals of the proposed system. The technology and algorithms discussed in the literature background were used and modified to specifically achieve these goals.

The designed concept was implemented on a ZYNQ-7020 (SoC containing an FPGA and CPU). The data intensive image processing was implemented on the programmable logic and the feature evaluation and information extraction was implemented on the processor system. To verify that the implemented logic functions as required and achieves the timing requirements the individual components and system as a whole were tested using different test benches. The communication between the PS and PL was achieved using AXI enable

modules. The PS was designed to control the PL and to process the incoming extracted image information to identify markers and determine the system's pose.

Test images were created to test the accuracy and performance of the implemented system in different scenarios. The test images were created by simulating a camera and marker and taking snapshots at different camera positions and orientations. Along with the translation and rotation tests the system was also tested to measure its response to the increase in detectable features in the image frame. To provide context as to how the use of the FPGA to parallelise and accelerate the image processing aspect of the design speeds up processing time an equivalent marker detection system was implemented exclusively on the processing system. The processing speed-up was contextualised by comparing the timing of the implemented solution (Hybrid of PL and PS) and the PS-Only implementation.

6.2 Significant Findings

The test results were analysed and various graphs were created to visually present the relationships between the system's performance- and accuracy and the different testing variables.

The results of the accuracy verification tests show that the proposed marker detection has comparable accuracy to previously implemented systems. Comparable accuracy validates the timing results. The accuracy of the proposed marker detection system at a variety of distances and viewing angles is:

- Average translation accuracy of 4.96 cm.
- Average rotation accuracy of 4.28°.

The accuracy of the system has a linear dependency on the distance between the camera and the ArUco marker with an expected accuracy of 0.5 cm at 1 m away and an expected accuracy of 3 cm at 4.5 m away.

The performance of the proposed marker detection system is significantly faster than the PS-Only implementation of the system. The proposed system has an average speed-up of 12.16×. The large speed-up shows that accelerating and running the data heavy image processing functions on the FPGA removes the largest bottleneck in PS-Only systems and greatly decreases the overall processing time for the system. The average frame rate of the proposed system is 134 FPS which is 3 times larger than the average frame rate of the previously implemented marker detection system.

6.3 Recommendations

The recommended improvements that can be made to the system are related to the improvement of system functionality.

The implemented system provides a working solution that can be built on to produce a low-power-, light-weight- and compact marker detection platform.

6.3.1 Functionality Improvement

The two main aspects of the solution that can improve the system functionality are performance and accuracy.

6.3.1.1 Increase in System Performance

The overall processing time of the system can be reduced by increasing the performance that the PL detects features and the PS processes them.

PL Timing The feature detection algorithms all run on the pixel clock of 50 MHz which is the current performance limitation of the PL. To increase the performance of the feature extraction on the PL the clock frequency can be increased. If the clock frequency is increased to 100 MHz 2 of the 3 feature extraction modules meet the increased timing demand with the Blob Detection module ($F_{max} = 55$ MHz) becoming the new bottleneck. To address the new bottleneck the Blob Detection module can be optimised by altering the point labelling component or optimised by pipe-lining the logic that does not meet the timing requirement.

PS Timing The marker identification function has the largest impact on the processing duration on the PS. The time taken to evaluate each marker candidate must be reduced to reduce the overall function time. Different methods to extract marker information and filter out non marker candidates before they are rectified can be investigated to improve the processing time of the marker identification method.

6.3.1.2 Increase in System Accuracy

The testing of the implemented system shows that the accuracy of the system is dependent on the accuracy of the detected corners and the accuracy of the pose estimation algorithm. The accuracy of the system can therefore be improved by improving one or both of those components.

Another way to increase system accuracy is by altering the solution to detect multiple markers.

Corner Acuracy The corner error, discussed in Section 5.3.1, is the result of image resolution. By increasing image resolution the accuracy of corners that are detected will increase. Image resolution also increases the view angle and distance that the marker can be detected at, by increasing the number of pixels available that store marker information. The corner accuracy can also be increased by replacing the non-maximum suppression of corners detected in groups with sub-pixel corner extraction. By introducing sub-pixel accuracy, a more accurate representation of the marker corner can be determined.

Pose Acuracy The POSIT algorithm was used in the solution to estimate pose because it is a low complexity algorithm that does not have a long processing time. Through testing it was discovered that the Pose Estimation function has a very small impact on overall processing time and therefore there is room to introduce a more accurate pose estimation algorithm that has a larger processing time impact.

Multiple Markers The current solution evaluates all detected candidates in the order they are detected in and when the first ArUco marker is detected the function stops and sends the information to the next function. By altering the Marker Identification function to detect multiple markers the accuracy of the system can be increased by providing the pose estimation algorithm with more corner points (that have known positions in the environment) to estimate a more accurate system pose.

6.3.1.3 Data Bridge Consideration

The data transfer between PL and PS will need to be improved to accommodate the increased required data rate from a decrease in PL processing time. The current data transfer rate can handle the current theoretical maximum number of features that can be detected, but if the required data transfer rate is increased the transfer method will need to be improved. Switching from AXI GPIO transfer to AXI DMA transfer will increase the system's capability to transfer large amounts of feature data.

6.3.2 Real World Implementation

The implemented system acts as a proof of concept that can be used to design and implement a real-world solution.

6.3.2.1 Camera Sensor

Interfacing a camera sensor directly to the PL will remove the need to store test images in BRAM to test the system. The removal of testing logic and memory will give a better representation of the hardware requirements of the marker detection system.

By adding the camera sensor, the system accuracy and timing will be able to be tested in a real-world setting providing better test results and allowing for more insight into how the system can be modified to improve its functionality and robustness.

6.3.2.2 Smart Sensor Platform

A hardware platform can be designed after the system has been tested, modified and proven to work with a camera sensor. The exact hardware requirements will be able to be determined after the camera sensor is tested and an optimised smart sensor is designed.

6.4 Conclusion

This project successfully demonstrated the design and implementation of a marker detection system that processes the data of an image frame and outputs the pose information and marker ID. The implemented logic on the hardware meets the timing requirements with potential to be up scaled to run at faster frequencies.

The implemented design is proven suitably accurate with a translational accuracy of a few centimetres (average of 4.96 cm) and a rotational accuracy of a few degrees (average of 4.28°). The design has a maximum theoretical detection distance of 5 m due to information extraction limitations and has a proven detection distance of 4.5 m. The accuracy of the system can also be up scaled by future improvements.

The complete system detects markers at an average framerate of 134 FPS which is twice the stated goal of 60 FPs and three times larger than the previously implemented marker detection systems that were discussed.

The implemented marker detection solution is a suitable design for a marker detection platform that performs high frame rate localisation from a single camera sensor.

Appendices

Appendix A

Background Literature

A.1 Hardware Platforms

For the required solution to perform autonomous localisation a hardware platform is needed. The hardware platform acts as the brain of the system. It controls the sensors and processes the sensor data to determine the system's pose. The three most common hardware platforms are CPU, GPU and FPGA.

A.1.1 CPU

The CPU takes instructions and performs a task. The CPU is a sequential processing unit that fetches instructions, decodes the instruction to determine what needs to be done and then executes the instruction. CPUs are best suited for high algorithmic complexity or highly iterative linear functions [42].

A.1.2 GPU

GPUs were initially designed to accelerate the rendering of 3D graphics but have evolved and are now also used in high performance computing, deep learning and other computation heavy workloads. GPUs are designed for parallel processing and is made up of thousands of identical processing cores [43].

A.1.3 FPGA

An FPGA is a semiconductor processing device that is made up of a matrix of configurable logic blocks (shown in Figure A.1). The configurable logic blocks are connected to one another through configurable interconnects. The device consisting of configurable components and connections allows the FPGA to be reprogrammed to complete a variety of tasks. FPGAs provide high throughput with low latency and are best suited for low complexity algorithms with high data bandwidths [43].

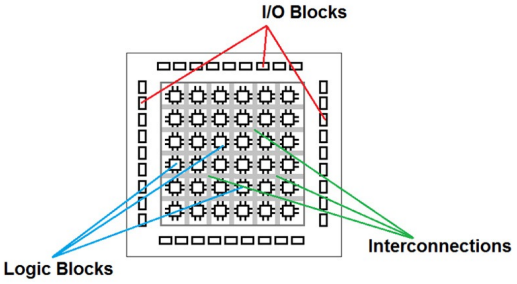


Figure A.1: Basic Structure of an FPGA [11]

A.1.4 Platform Comparison

Table A.1 shows the most relevant strengths and weaknesses of the hardware platforms compared to each other.

Platform	Strengths	Weaknesses
CPU	Versatility, Multitasking	Optimised for sequential processing
GPU	Large processing power for video/image processing	High-power consumption
FPGA	Built for parallel operation, High performance per watt	Poor performance for sequential operations

Table A.1: Comparison Between Hardware Platforms [22]

A.2 Marker Detection

Markers are a set of patterns detectable by a computer vision system. Placing markers in known locations in the environment provides the detection system visual cues for localisation. Detected markers provide a relative pose between the marker and the camera and marker information in the form of patterns [44]. Examples of different markers are shown in Figure A.2.

(a) Intersense (b) ARToolkit (c) RUNE-129

Figure A.2: Example of Markers [12]

One of the most commonly used computer vision markers is the ArUco marker. ArUco markers are used in a variety of computer vision applications in the OpenCV library (OpenCV is a popular open-source computer vision library).

A.2.1 ArUco Marker

ArUco markers are a fiducial marker system designed for camera pose estimation. Appropriate applications for the use of ArUco markers are robot localisation and AR. ArUco markers are used for pose estimation because they have 4 detectable corners and a detectable orientation by decoding the markers unique ID which is visually stored in binary within the markers detectable area [14]. An example of a ArUco marker is shown in Figure A.3.

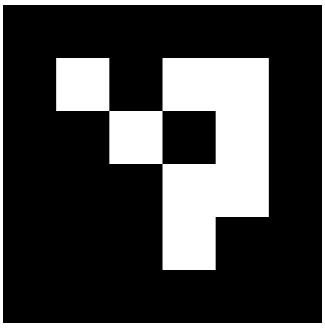


Figure A.3: ArUco Marker ID 0

For the marker ID and relative pose to be determined markers must be detected in the image and decoded. Candidate markers are made up of features and these features are extracted from the image using image processing techniques. The candidates are then evaluated to determine if they are ArUco markers and then decoded to identify the markers ID.

A.2.1.1 Marker Candidate Detection

To detect candidate markers the unique features of the marker must be detected. The process of detecting a candidate marker in an image is [14]:

1. Local Thresholding
2. Contour Detection
3. Polygonal Approximation

The detection process utilises image processing techniques that extract certain features from the image.

A.2.1.2 Marker Identification

The binary information encoded in the ArUco marker is unique to each individual marker ID regardless of rotation the marker is detected in. ArUco markers are also designed to maximise the hamming distance between the encoded binary information to reduce the chance of identifying the incorrect marker [14].

Hamming Distance The Hamming distance is way of comparing two binary strings and setting a metric to identify how different the strings are. By comparing the equivalent bit in each string and counting the bits that are different the Hamming distance can be determined. The larger the Hamming distance the bigger the difference between the binary strings [45].

After the candidate has been detected the 4 square corners are used to rectify the image to uniformly access the encoded information in the marker. The rectification of the candidate is accomplished by using homography. The extracted binary information string is then compared to the library of marker IDs and the ID of the binary string with the lowest hamming distance is chosen. The output of the marker identification is the marker ID and the orientation of the marker [44].

A detailed list of the ArUco marker identification steps [14]:

1. Homography - To remove perspective projection distortion
2. Threshold - Binarise the rectified canonical image
3. Divide into Grid - The binarised canonical image is divided in grid
4. Extract Binary Matrix - The grid elements are assigned the value 0 or 1 depending on pixel values
5. Rotate Matrix - Create the 4 rotations of the information matrix
6. Search Marker Library - Compare all 4 rotations of the information matrix with the marker library to find the valid marker ID

The identification process utilises image processing techniques that extract information from the image.

A.3 Image Processing

Image processing is the manipulation of digital images to perform specific tasks. Tasks that include but are not limited to information extraction and image alteration. Image processing is made up of digital signal processing techniques and processing techniques that are unique to digital images. Grayscale digital images consist of pixel intensity values, Figure A.4, that indicate the pixels grayscale intensity (8-bit pixel depth intensity values range from black which has an intensity value of 0 to white which has an intensity value of 255).

Figure A.4: 8 Bit Grayscale Range [13]

To detect marker candidates and identify ArUco markers, image analysis must be done on the image frames from the camera sensor. Image analysis consists of segmentation, feature extraction and texture analysis [46].

A.3.1 Thresholding

Image segmentation can take the form of binarising an image. By thresholding an image, the image is transformed into a binary image that can be used for further data extraction. Thresholding uses a threshold value to evaluate a pixel. If the grayscale pixel intensity value is larger than the threshold value then the pixel is set to a binary value of 0, indicating white. If the intensity value is smaller than the threshold value, the pixel is set to black with a binary value of 1 [47].

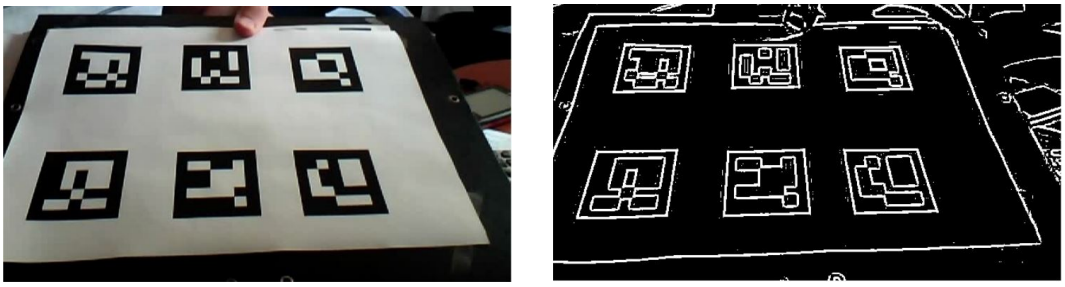
Two different techniques that determine the threshold value are local adaptive thresholding and global thresholding

A.3.1.1 Local Adaptive Thresholding

The local adaptive thresholding method evaluates a window of pixels to determine the thresholding value that will be used to threshold a smaller window within the evaluation window. The thresholding window slides across the image changing the thresholding value as the pixels in the evaluation window change.

The thresholding value is calculated by determining the average of the maximum and minimum pixels in the evaluation window. The evaluation window encompasses the 4×4 thresholding window by incorporating a surrounding border that has a width of three pixels. The added border prevents artefacts from arising in between the thresholding windows [26].

The local adaptive thresholding technique manipulates the original image into an image high contrast edges are represented as lines. The thresholding method input and output are shown in Figure A.5.



(a) Original Image (b) Local Adaptive Threshold

Figure A.5: Example of Local Adaptive Thresholding [14]

A.3.1.2 Global Thresholding

Global thresholding determines the threshold of the whole image and then binarises the image by evaluating every pixel in the image. The global thresholding method is based on the assumption that the intensity histogram of the digital image is bimodal. The bimodal nature of the histogram allows for the differentiation between the object and the background. The object being the black marker and the background being the white border and the rest of the environment. The threshold value is calculated by averaging the two peaks of the histogram [48].

An example of a pixel intensity histogram is shown in Figure A.6. The histogram shows the two peaks in the chart that represent the object and the background.

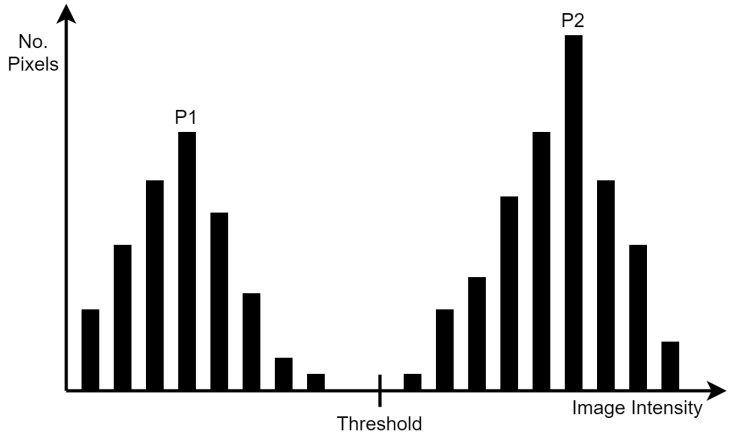


Figure A.6: Intensity Histogram

A.3.2 Feature Extraction

To reduce incoming data size and find useful information in a digital image feature extraction is performed. Feature extraction transforms the input image into a vector of feature information. To detect ArUco markers local geometric features must be extracted from the image. Local features including corners and blobs [49].

A.3.2.1 Corner Detection

Corners are the vertices of shapes. Corner information is useful to extract because it can be used to detect markers in the image as ArUco markers are square and have four corners. Two common image processing techniques that detect corners are the Harris Corner Detector and FAST.

Harris Corner Detector Harris Corner Detector is a corner detection operator that is commonly used in computer vision algorithms to extract corners. The detector evaluates the gradient of an image across a window and computes a corner score [50]. The Sobel Operator (shown in Equation A.1) is a discrete differentiation operator, computing an approximation of the gradient of the image intensity function.

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \times A \qquad G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \times A \tag{A.1}$$

The 3×3 kernel can be used to determine the intensity gradient of the image. The X and Y gradient functions can be combined to calculate the gradient magnitude and gradient direction [15]. An example of using the Sobel Operator on a sample image is shown on Figure A.7

(a) Sample Image (b) Vertical Mask (G_x) (c) Horizontal Mask (G_y)

Figure A.7: Sobel Operator on Sample Image [15]

The Harris Corner Detector Algorithm [50] consists of the following steps:

1. Convert colour image to grayscale image
2. Spatial derivative calculation
   a) I_x and I_y are calculated using the Sobel Operator
   b) Populate a matrix of the image gradients
3. Structure tensor setup where M is the structure tensor

$$M = \sum_{(x,y)\in W} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \tag{A.2}$$

4. Harris response calculation (R)

$$R = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2 \tag{A.3}$$

   a) If R $<$ 0 then the point is an edge
   b) If R $\gg$ 0 then the point is a corner
   c) Else the point is flat
5. Non-maximum suppression

FAST FAST is used to detect corners by evaluating a circle of pixels around the pixel of interest. The evaluation circle (referred to as the Bresenham Circle which has a radius of 3 pixels) is shown in Figure A.8.

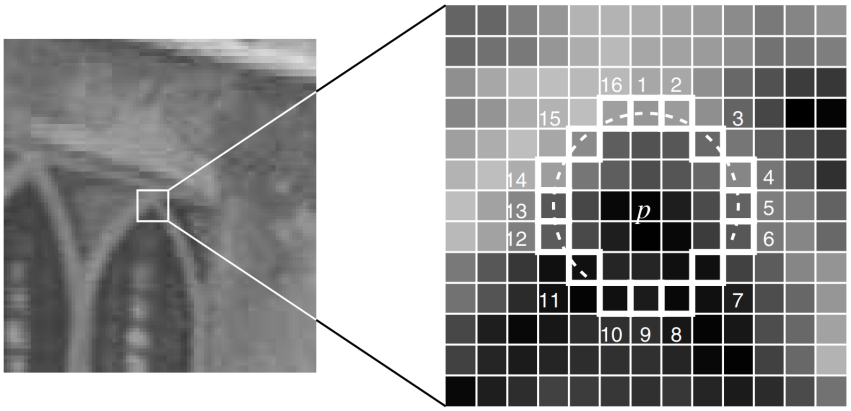


Figure A.8: Evaluation Circle used in FAST [16]

The FAST Algorithm [16] consists of the following steps:

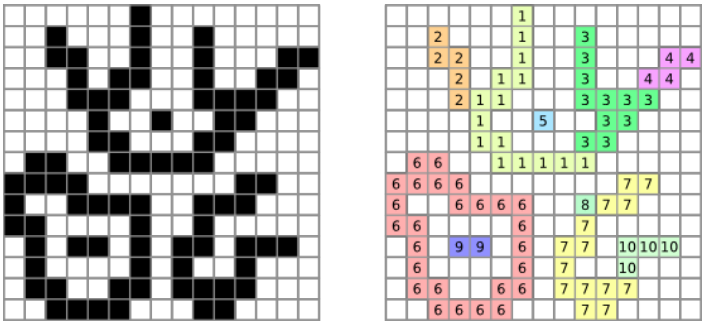
1. Select a pixel p in the image which is to be evaluated. Let the POI intensity be I_p
2. Select an appropriate threshold value t
3. Consider a circle of 16 pixels around the POI. The POI is a corner if there is a set of n contiguous pixels in the circle which are brighter than $I_p + t$, or darker than $I_p - t$
4. To increase the processing speed, first compare the intensity of pixels 1, 5, 9 and 13 of the circle with I_p. At least three of the four pixels should satisfy the threshold criterion
5. If at least three of the four pixels (I_1, I_5, I_9 or I_{13}) are not above or below $I_p + t$, then the POI is not a corner.
6. Else if at least three of the pixel values are above or below $I_p - t$, then check for if the criterion in step number 3 is met.
7. Repeat the procedure for all the pixels in the image.

To reduce detected corner blobs into single corner points non-maximum suppression is performed. To perform non-maximum suppression the corner strength (corner strength equation is provided in Equation A.4) is calculated and used to determine the strongest corner in the blob [16].

$$C_s = max(\sum_{x \in S_{\text{Bright}}} |I_{p \to x} - I_p| - t, \sum_{x \in S_{\text{Dark}}} |I_p - I_{p \to x}| - t) \tag{A.4}$$

A.3.2.2 Blob Detection

Blobs are groups of connected black pixels. After thresholding objects are separated from the background. To detect the separated objects, blob detection is performed on the binary image. Marker candidates can be detected and identified by extracting blob information from the image, as ArUco markers are black squares with a white border [51]. Two common methods to detect blobs and extract blob related information are border following and grouping. A basic example of blob detection is shown in Figure A.9.



(a) Binary Image (b) Labelled Blobs

Figure A.9: Basic Blob Detection and Labelling [17]

Border Following Border following is a blob detection method where the border of every blob in the image is identified. The border following method performs a raster scan until the start of a blob is found. The method then follows along the border of the blob, documenting the border information and relationship with the previously discovered parent blob. After the complete border has been followed, the raster scan is continued on from where it had stopped [18]. An example of border following and the resultant topological analysis is shown in Figure A.10.

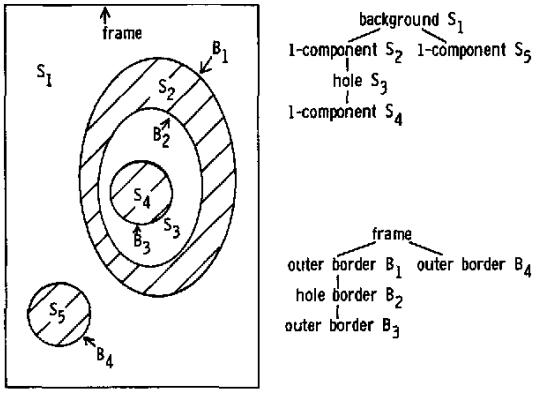
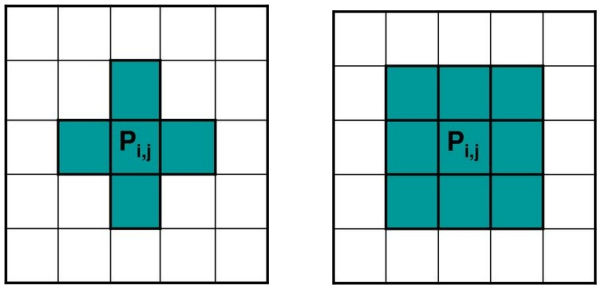


Figure A.10: Border Following and Topological Analysis [18]

Grouping Grouping is a blob detection method where all connected black pixels are grouped together under the same label. An evaluating window is shifted along the image and if black pixels are present in the window they are grouped under the same label. The window used to group connected pixels can vary and more complex grouping algorithms exist that merge two labels together if they are determined to be connected further down the image [52].

The two most common neighbour connectivity evaluation windows are shown in Figure A.11.



(a) 4 Neighbour (b) 8 Neighbour

Figure A.11: Grouping Evaluation Windows [17]

A.3.3 Homography

ArUco Markers detected in a digital image can have perspective distortion preventing accurate information extraction. To remove perspective distortion homography transform can be used to rectify the marker into a canonical image. The rectified canonical image can then be used to extract the marker information [26]. Figure A.12 shows a door with perspective distortion being rectified.

Figure A.12: Rectified Perspective Distorted Image of Door [19]

Homography Transform Homography is a transformation between two planes. It is a mapping between the image plane and the canonical marker plane [53]. The Homography Transform is represented by:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{A.5}$$

A.4 Pose Estimation

Using pose estimation, the position and orientation of the camera relative to the ArUco can be determined. The Coplanar POSIT Algorithm uses the coordinates of the 4 marker corners to iteratively calculate the rotation matrix and the translation vector that represents the change in pose between the camera and marker [20]. The relationship between the coordinates of the camera (X, Y, Z) and the coordinates of the marker (U, V, W) is shown in equation A.6.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = R \begin{bmatrix} U \\ V \\ W \end{bmatrix} + T = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} U \\ V \\ W \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} \tag{A.6}$$

The POSIT algorithm (shown in Figure A.13) estimates the pose iteratively. The Pose from Orthography and Scaling (POS) algorithm produces two pose estimation results when only coplanar points are provided. The POSIT algorithm refines the pose estimation output by supplying a measure of error (the average of the Euclidean distances between the camera corners and the projected corners) and iteratively choosing the pose that has the smallest error. The pose that has the smallest error is used to calculate the new correction factors and the process is repeated until the error meets the threshold requirement.

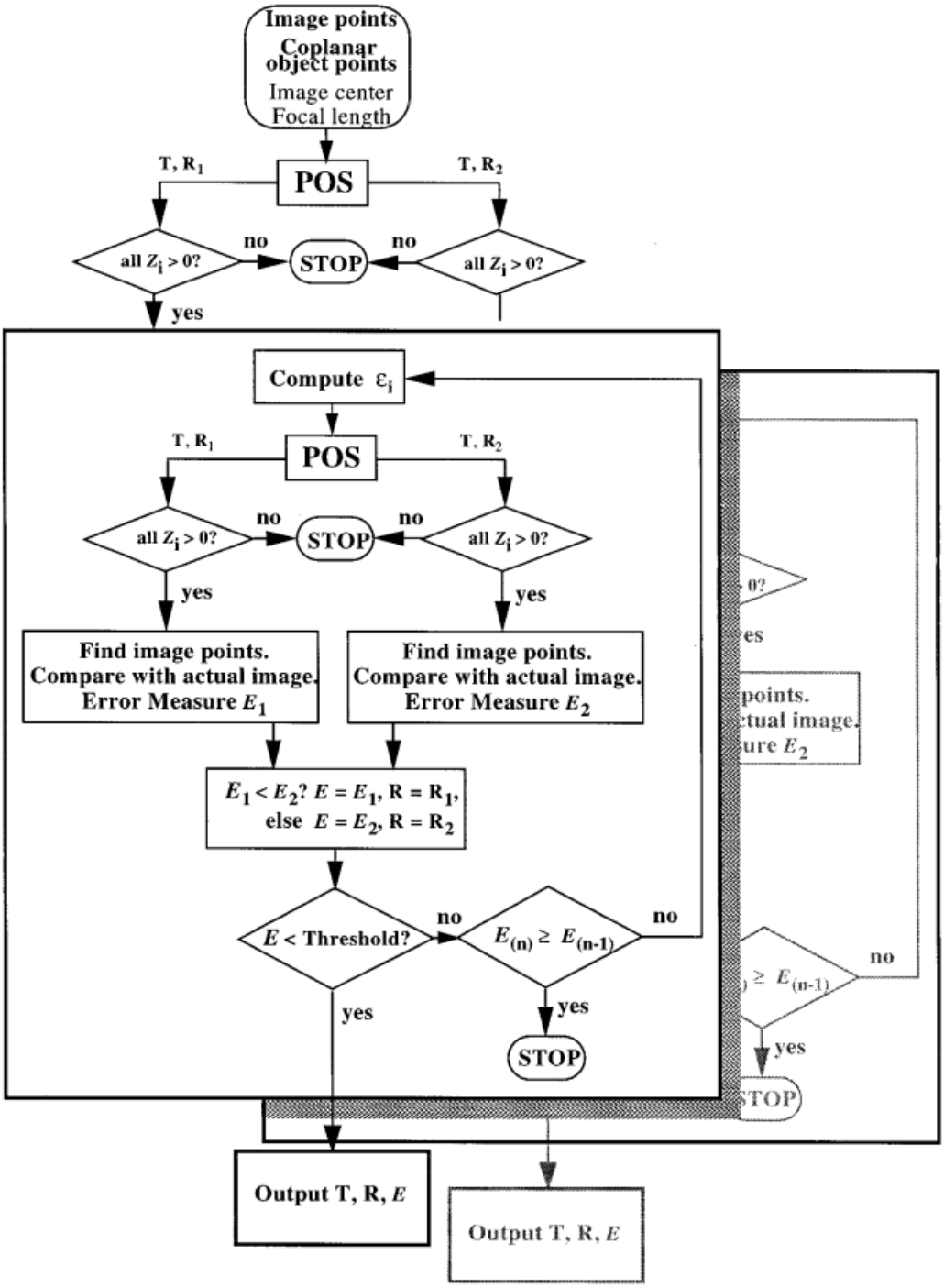


Figure A.13: POSIT Algorithm [20]

A.5 FPGA Design

FPGA design using the Vivado design suite is done by using Xilinx provided IP modules along with custom written IP modules.

All the custom IP modules are written in Very High Speed Integrated Circuit Program (VHSIC) Hardware Description Language (VHDL).

A.5.1 Functional Analysis

The functionality of the modules are verified by creating a test bench to supply each Design Under Test (DUT) with its required input and checking the accuracy of the output. A simple test bench and DUT setup is shown in Figure A.14.

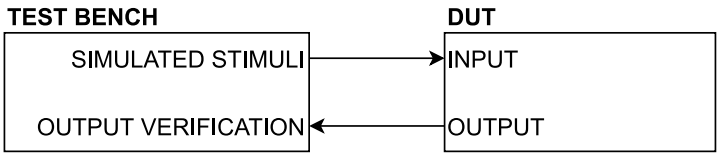


Figure A.14: Simple test bench and DUT representation

The test bench provides input signals that simulate signals the DUT would receive during operation. The internal functionality of each module can by simulated and evaluated by analysing the waveforms of the internal signals.

A.5.2 Timing Analysis

The theoretical maximum clock frequency (F_{max}) a module can operate at is calculated if the design can meet the timing requirements. The equation used to calculate (F_{max} [54]) is shown in Equation A.7.

$$F_{max} = \frac{1}{period - WNS} \times 1000 \tag{A.7}$$

where:

F_{max} is the maximum theoretical clock frequency,

$period$ is the target clock period,

WNS is the worst negative slack of the clock signal in the Intra-Clock Paths section,

Appendix B

Custom IP Modules

B.1 Feature Extraction Modules

B.1.1 Corner Detection

The Corner Detection module, shown in Figure B.1, detects corners in the incoming image frame stream and outputs any detected corner information.

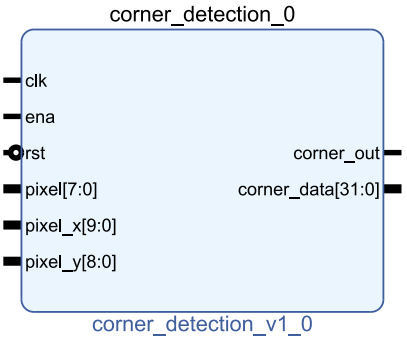


Figure B.1: Corner Detection IP module

The ports of the custom Corner Detection module are described in Table B.1.

Port	I/O	Description
clk	IN	The input clock signal (50MHz pixel clock).
ena	IN	The enable signal.
rst	IN	The reset signal.
pixel	IN	The 8 bit pixel intensity value.
x	IN	The x coordinate of the pixel in the image plane (10 Bits).
y	IN	The y coordinate of the pixel in the image plane (9 Bits).
corner_out	OUT	The corner detected and corner information ready signal
corner_data	OUT	The 32 bit corner data output that contains the corner information.

Table B.1: Corner Detection module port descriptions

B.1.2 Global Threshold

The Global Threshold module, shown in Figure B.2, binarises the incoming image frame and calculates the global threshold value for the next frame.

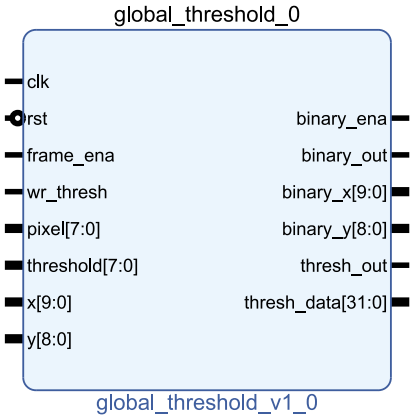


Figure B.2: Global Threshold IP module

The ports of the custom Global Threshold module are described in Table B.2.

Port	I/O	Description
clk	IN	The input clock signal (50MHz pixel clock).
rst	IN	The reset signal.
frame_ena	IN	The enable signal.
wr_thresh	IN	Signal used to write a given threshold to the module.
pixel	IN	The 8-bit pixel intensity value.
threshold	OUT	The 8-bit threshold value.
x	IN	The x coordinate of the pixel in the image plane.
y	IN	The y coordinate of the pixel in the image plane.
binary_ena	OUT	The enable signal for the global threshold dependent modules.
binary_out	OUT	The binary pixel value.
binary_x	OUT	The x coordinate of the binary pixel in the image plane.
binary_y	OUT	The y coordinate of the binary pixel in the image plane.
thresh_out	OUT	The binary packet ready to send signal
thresh_data	OUT	The 32-bit threshold data output that contains 32 binary pixel values packaged together.

Table B.2: Global Threshold module port descriptions

B.1.3 Blob Detection

The Blob Detection module, shown in Figure B.3, detects blobs in the incoming image frame stream and outputs any detected blob information.

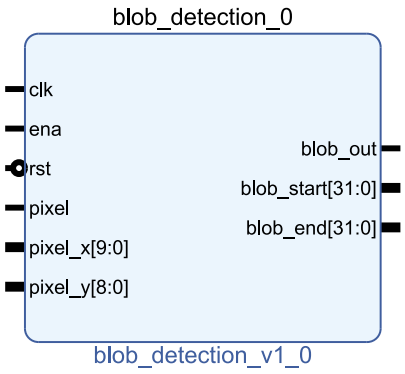


Figure B.3: Blob Detection IP module

The ports of the custom Blob Detection module are described in Table B.3.

Port	I/O	Description
clk	IN	The input clock signal (50MHz pixel clock).
ena	IN	The enable signal.
rst	IN	The reset signal.
pixel	IN	The binary pixel value.
x	IN	The x coordinate of the binary pixel in the image plane (10 Bits).
y	IN	The y coordinate of the binary pixel in the image plane (9 Bits).
blob_out	OUT	The blob detected and corner information ready signal
blob_start	OUT	The 32-bit blob data output that contains the bounding box start coordinates.
blob_end	OUT	The 32-bit corner data output that contains the bounding box end coordinates.

Table B.3: Blob Detection module port descriptions

B.2 Data Bridge Modules

B.2.1 Control Block

The Control Block module, shown in Figure B.4, receives the control packet from the PS and splits the packet into control signals that control the functionality of the PL IP modules.

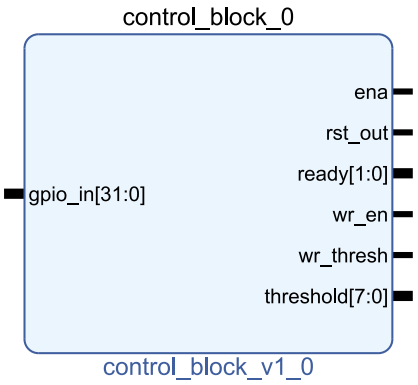


Figure B.4: Control Block IP module

The ports of the custom Control Block module are described in Table B.4.

Port	I/O	Description
gpio_in	IN	The 32-bit input signal that contains the PL module control signals from the PS.
ena	OUT	The enable signal sent from the PS to start the system.
rst_out	OUT	The reset signal to set all the modules to their default states.
ready	OUT	The toggling 2-bit signal used by the Feature Packaging module to indicate a new package can be sent.
wr_en	OUT	The write enable signal to start the AXI Image Store module.
wr_thresh	OUT	The write enable signal to write the new threshold to the Global Threshold module.
threshold	OUT	The threshold value written to the Global Threshold module.

Table B.4: Control Block module port descriptions

B.2.2 AXI Threshold Bridge

The AXI Threshold Bridge module, shown in Figure B.5, receives the binarised image frame and sends it to the PS through the AXI DMA IP module.

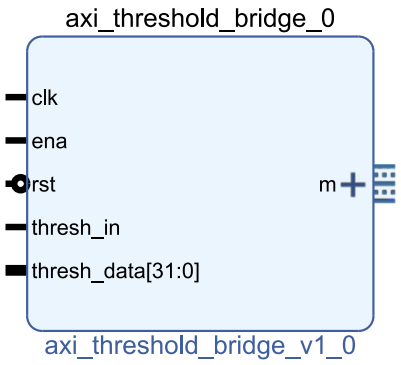


Figure B.5: AXI Threshold Bridge IP module

The ports of the custom AXI Threshold Bridge module are described in Table B.5.

Port	I/O	Description
clk	IN	The input clock signal (100MHz system clock).
ena	IN	The enable signal.
rst	IN	The reset signal.
thresh_in	IN	The thresh package is ready signal.
thresh_data	IN	The thresh data input data.
m	IO	The AXI4-Stream master interface used to interact with the AXI DMA to write data to PS memory.

Table B.5: AXI Threshold Bridge module port descriptions

B.2.3 Feature Packaging

The Feature Packaging module, shown in Figure B.6, receives the corner and blob information and sends it to the PS through the AXI GPIO IP module.

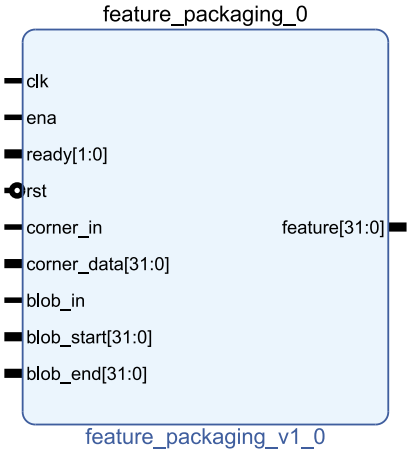


Figure B.6: Feature Packaging IP module

The ports of the custom Feature Packaging module are described in Table B.6.

Port	I/O	Description
clk	IN	The input clock signal (100MHz system clock).
ena	IN	The enable signal.
ready	IN	The toggling signal that indicates a new package can be sent.
rst	IN	The reset signal.
corner_in	IN	The corner detected signal.
corner_data	IN	The corner data input data.
blob_in	IN	The blob detected signal.
blob_start	IN	The bounding box start X and Y coordinates.
blob_end	IN	The bounding box end X and Y coordinates.
feature	OUT	The 32-bit feature package that will be sent to the PS.

Table B.6: Feature Packaging module port descriptions

B.3 Simulation Modules

B.3.1 Sensor Input

The Sensor Input module, shown in Figure B.7, reads the stored image frame data from BRAM and outputs the data in a stream into the feature extraction modules. The module simulates the ideal output of a physical camera sensor.

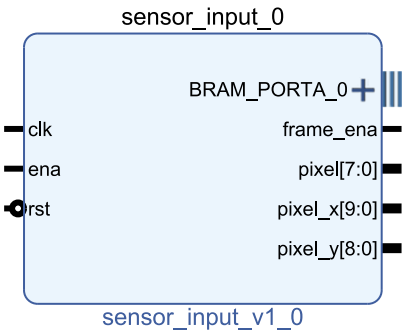


Figure B.7: Sensor Input Module

The ports of the custom sensor module are described in Table B.7.

Port	I/O	Description
clk	IN	The input clock signal (50MHz pixel clock).
ena	IN	The enable signal.
rst	IN	The reset signal.
BRAM_PORTA_0	IO	The BRAM interface used to read pixel data from BRAM.
frame_ena	OUT	The frame start signal.
pixel	OUT	The 8-bit pixel intensity value read out to be processed by the system.
pixel_x	OUT	The x coordinate of the pixel in the image plane (10 bit).
pixel_y	OUT	The y coordinate of the pixel in the image plane (9 bit).

Table B.7: Sensor Input Module Ports

B.3.2 AXI Image Store

The AXI Image Store module, shown in Figure B.8, receives the test image from the PS via the AXI DMA IP module and stores it in BRAM.

Figure B.8: AXI Image Store IP module

The ports of the custom AXI Image Store module are described in Table B.8.

Port	I/O	Description
m	IO	The AXI4-Stream slave interface used to interact with the AXI DMA to read data from PS memory.
clk	IN	The input clock signal (100MHz system clock).
ena	IN	The enable signal.
rst	IN	The reset signal.
BRAM_PORTA_0	IO	The BRAM interface used to write pixel data to BRAM.

Table B.8: AXI Image Store module port descriptions

Appendix C

Simulation Waveforms

C.1 Corner Detection

C.1.1 FAST

C.1.1.1 Threshold

Figure C.1 shows the circle being evaluated against the POI (value = 139) and the empirically chosen threshold of value (value = 30). The difference between the individual circle pixels and the sum of the POI and threshold value are displayed in the difference array.

All the pixels in the circle above the sum of the POI and threshold are also denoted as 1 in the bright bit array. The bright array is shown in decimal format (value 24588) and when converted to 16 bit binary it is "0110 0000 0000 1100". The binary values show that pixels 2, 3, 13 and 14 are larger than the sum of the POI and threshold value.

It takes 1 clock period to calculate the output of the FAST Threshold component.

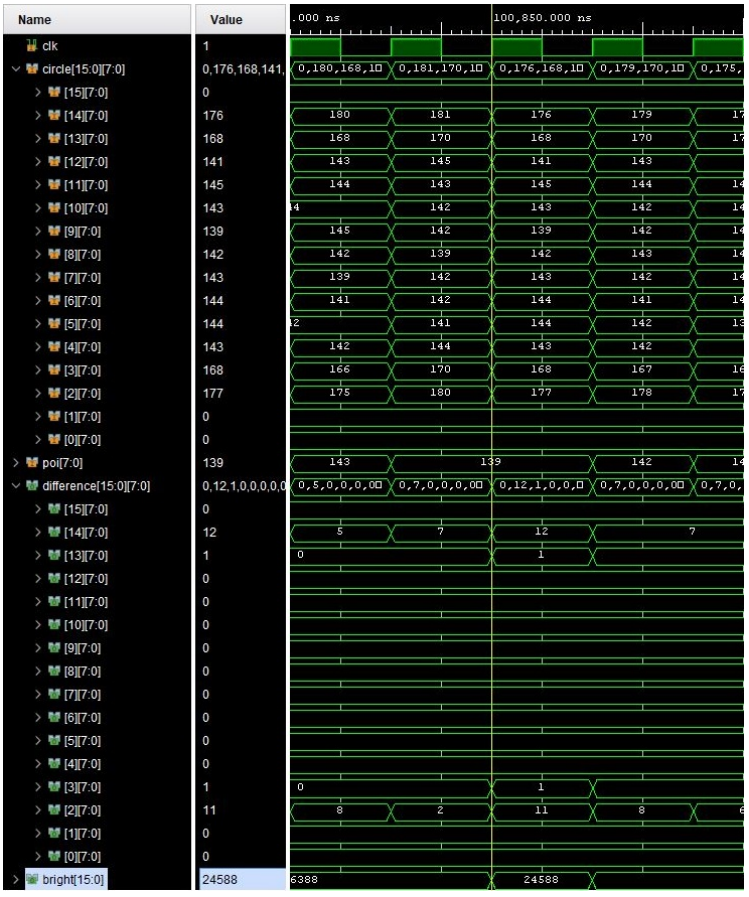


Figure C.1: Waveform of Threshold component in the Corner Detection module

C.1.1.2 Contiguity

Figure C.2 shows the 16-bit bright array being evaluated to determine if there is a contiguous string of nine bright pixels.

The Contiguity component takes 1 clock period to calculate to output and the example shows that the bright array with decimal value of 33023 has a contiguous string of 9 bright pixels. The contiguous string is determined and the contiguous signal is set high to indicate that the criterion has been met.

Figure C.2: Waveform of Contiguity component in the Corner Detection module

C.1.1.3 Direction

Figure C.3 shows the 16 bit bright array being evaluated to determine what the direction of the potential corner is. The example shows that the east blocks contain the most bright pixels and therefore the DIR value is set to 2.

The Direction component takes 1 clock period to calculate the output.

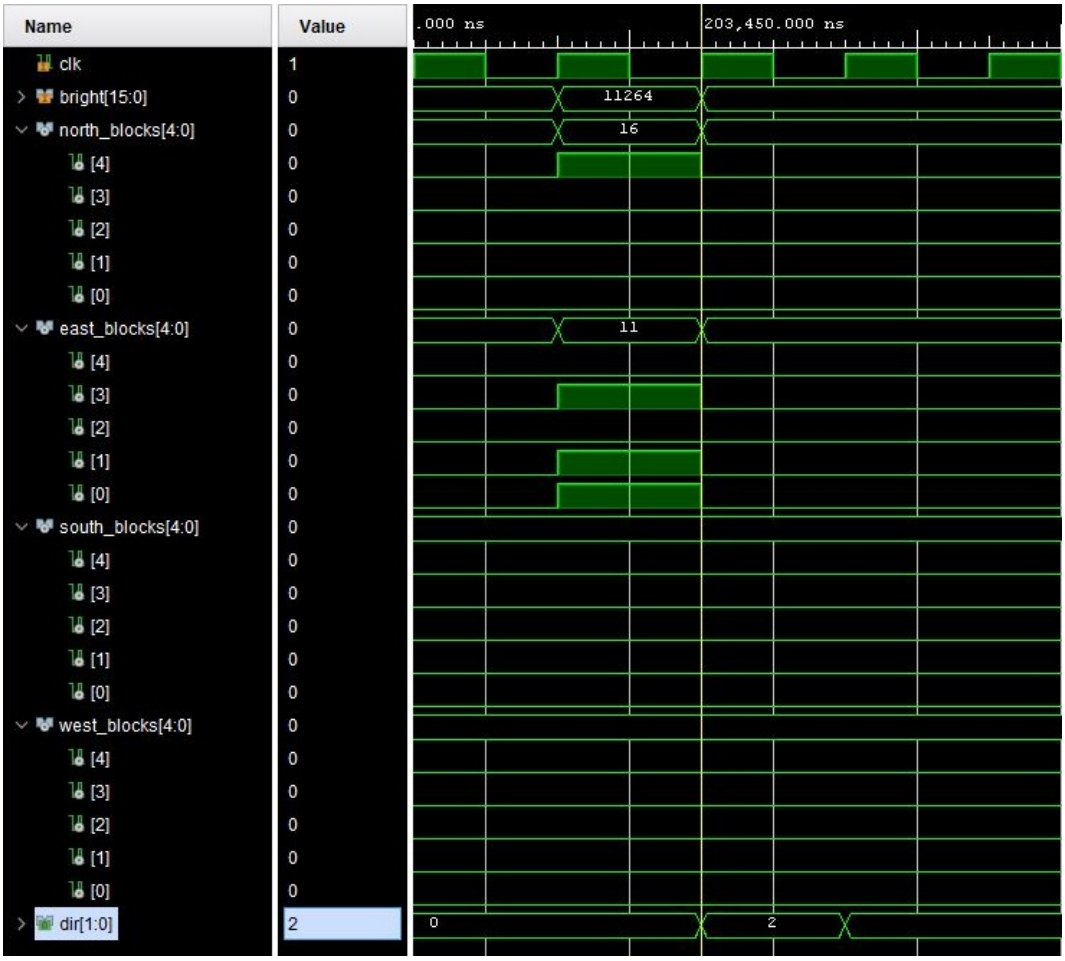


Figure C.3: Waveform of Direction component in the Corner Detection module

C.1.1.4 Strength

Figure C.4 shows the summation of the difference 8-bit vector array. Only the bright pixels in the circle will have a difference value and therefore the summation of the difference array will calculate the strength of the corner for only dark corners.

The Strength component takes 1 clock period to calculate the output.

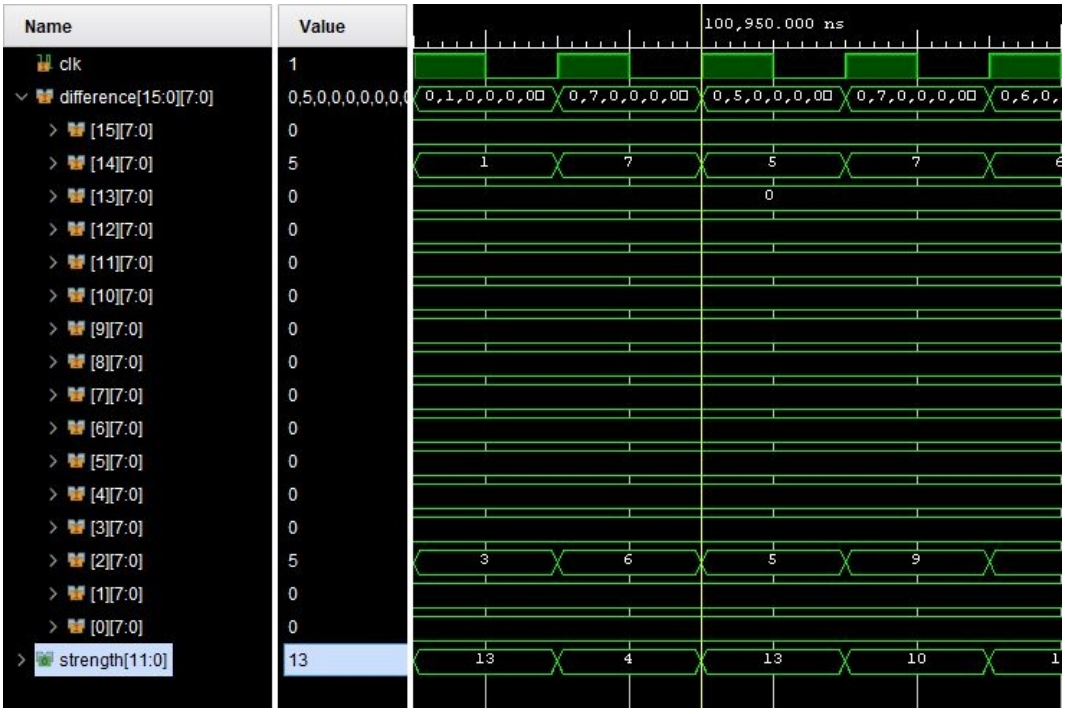


Figure C.4: Waveform of Strength component in the Corner Detection module

C.1.2 Non-Maximum Suppression

Figure C.5 shows the process of determining the strongest corner in the 3x3 corner array. The first clock period determines the strongest corner in each row of 3 corners and the second clock period determines the strongest corners out of those 3. Only corners that have a contiguous value of 1 are considered. From the example the only corner on the first clock period that is contiguous is the corner with the strength value of 1372. The corner strength value is then bit shifted by 2 to fit into the feature package size limit.

The Non-Maximum Suppression component takes 2 clock period to calculate the output.

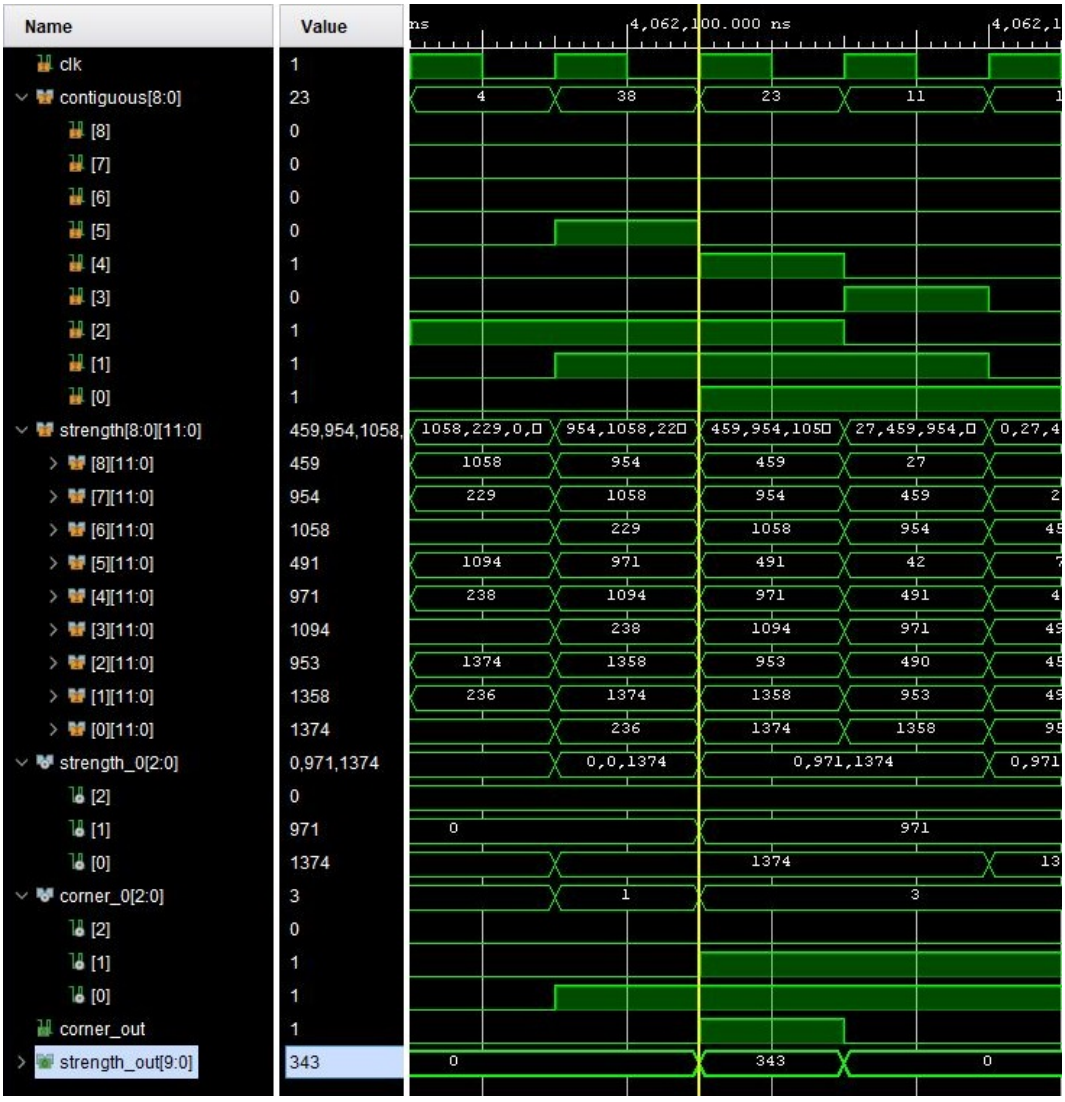


Figure C.5: Waveform of Non-Maximum Suppression component in the Corner Detection module

C.2 Global Threshold

Figure C.6 shows an example an input pixel intensity value (80) that is smaller than the threshold value (100) being given the binary value of 1 which indicates a black pixel. The example waveform also shows the intensity histogram being added to. The pixel count of pixels with an intensity value of 80 is incremented from 15 to 16 pixels.

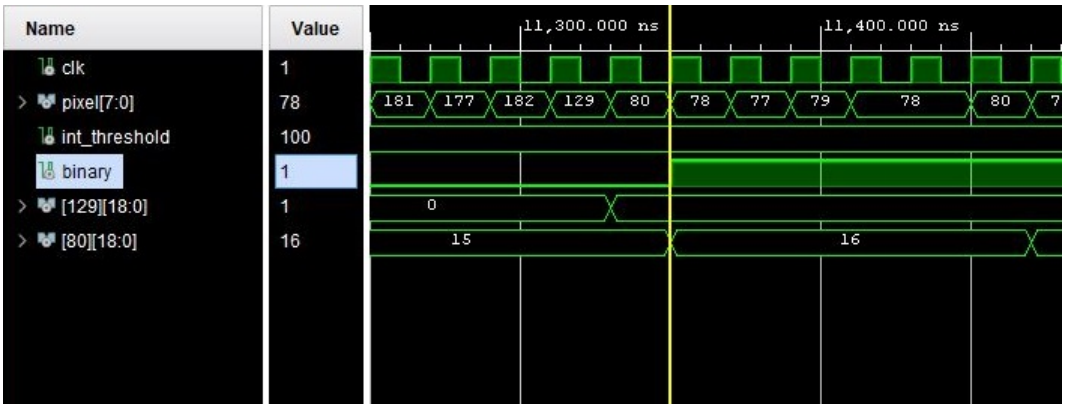


Figure C.6: Waveform of Global Threshold module thresholding a pixel and added to the intensity histogram

Figure C.7 shows how the new threshold for the next frame is constantly being determined. The first half of the histogram's peak (black peak) is shown to be at the intensity value of 79 and the second half's peak (white peak) is at the intensity value of 178. The threshold value is the average of the two peaks and in this example is 128.

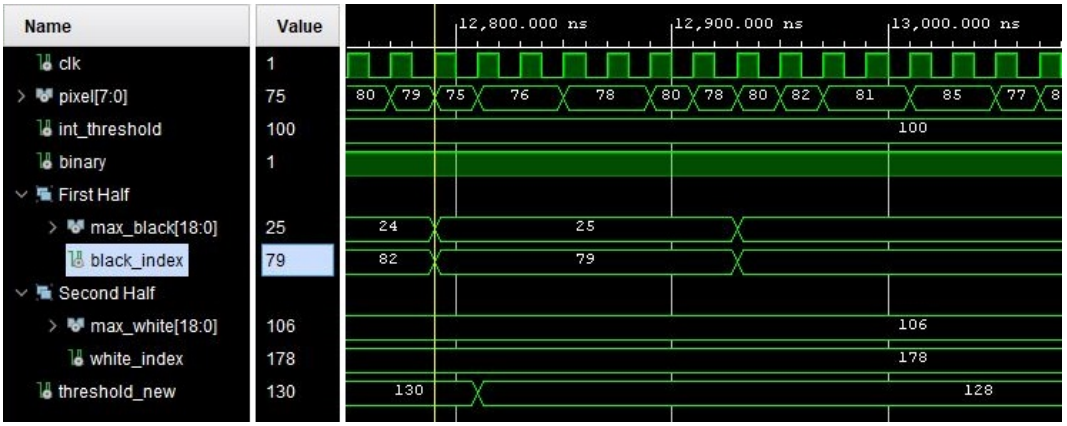


Figure C.7: Waveform of Global Threshold module determining the next threshold value

C.3 Blob Detection

C.3.1 Point Labelling

Figure C.8 shows an example of a new label being assigned to the start of a new blob. The first pixel in the blob is assigned the label of 2. The next pixels connected to that first pixel are also labelled under the same label of 2.

The assessment of the neighbouring window and assignment of label to pixel takes 1 clock period.

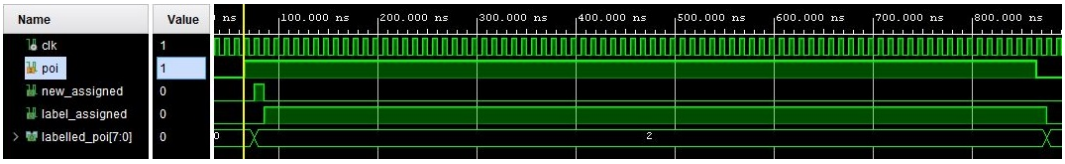


Figure C.8: Waveform of Point Labelling component assigning a new label

Figure C.9 shows an example of a pixel being added to an existing blob. At the clock period where the POI is set high the potential labels in the neighbouring window are 7 and 12. Label 12 is determined to be the label with the larger blob X Axis width.

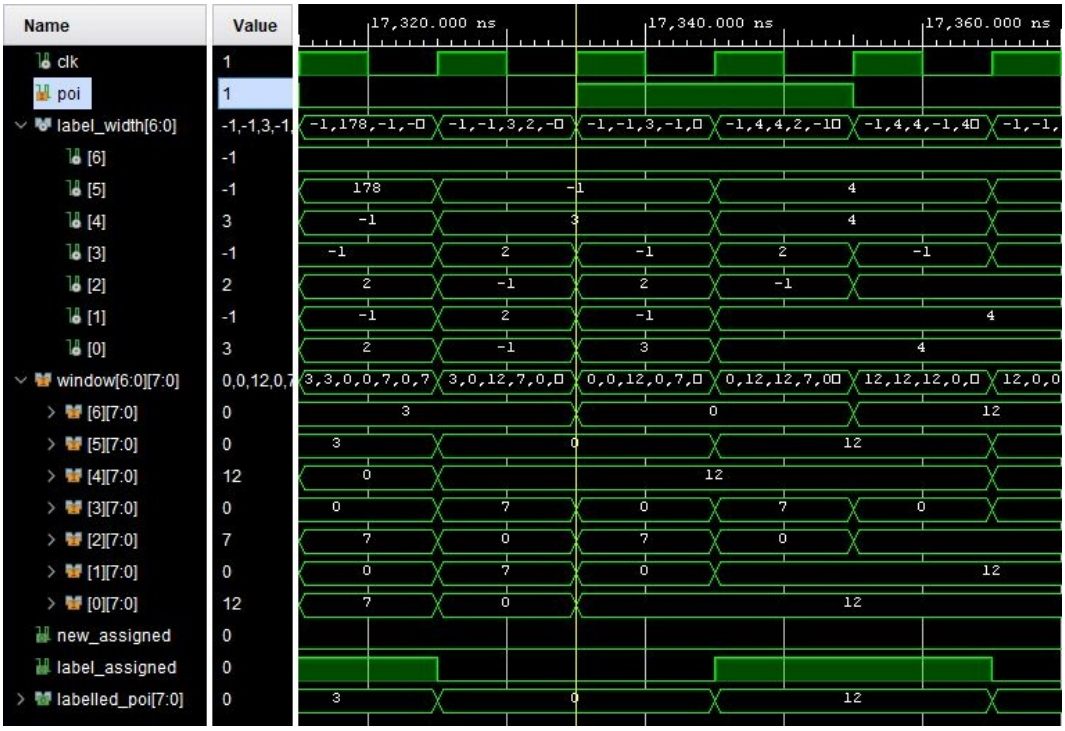


Figure C.9: Waveform of Point Labelling component assigning an existing label

C.3.2 Blob Detection

C.3.3 Label Control

Figure C.10 shows an example of a newly assigned bounding box (label 3) being started with the start and end coordinates starting from the same point. The bounding box is also shown growing as more pixels are detected in that blob. After the label (label 3) has been assigned to a new blob a new label is chosen for the next blob (label 4).

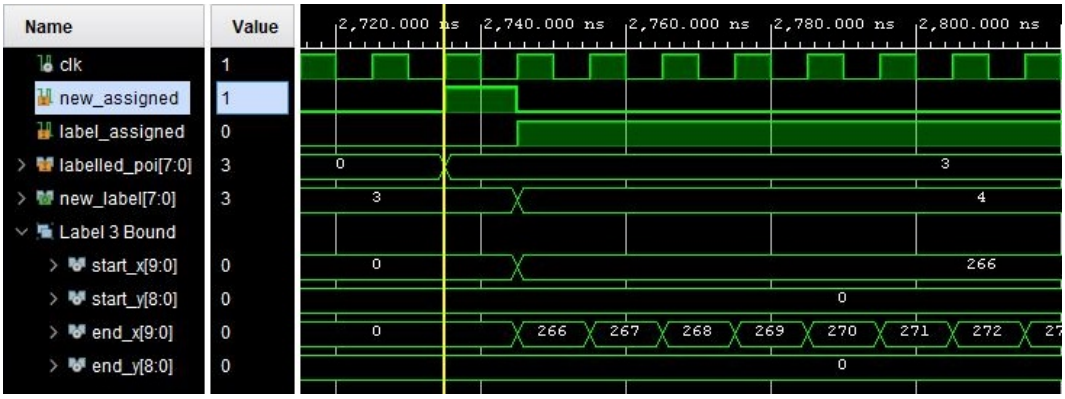


Figure C.10: Waveform of the Label Control component starting and growing a bounding box

Figure C.11 shows an example of a deactivated label (label 4) being output so that the Blob Filter component can evaluate the detected blob.

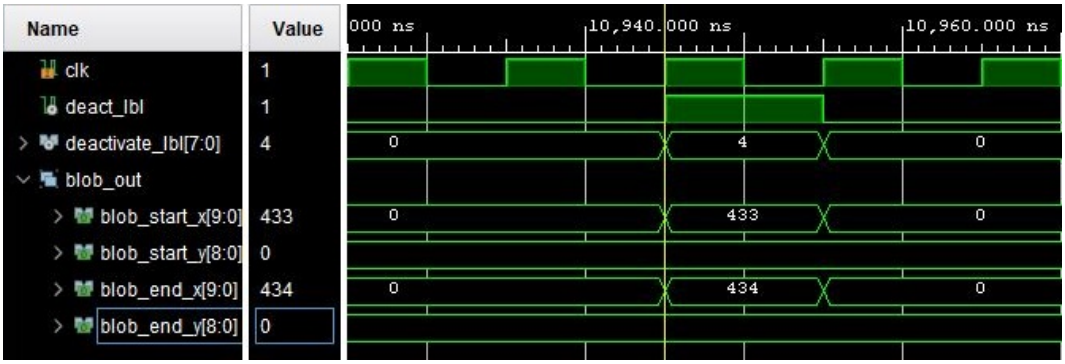


Figure C.11: Waveform of the Label Control component outputting a detected blob

C.3.4 Blob Filter

Figure C.12 shows an example of a detected blob being accepted by the Blob Filter component. The detected blob passes the side width criterion of 24 pixels. The X axis side is longer than 24 pixels (X Axis side length = Ex - Sx = 79 - 0 = 79) and the Y axis side is longer than 24 pixels (Y Axis side length = Ey - Sy = 45 - 0 = 45). The start and end coordinate data and a blob detected signal are output.

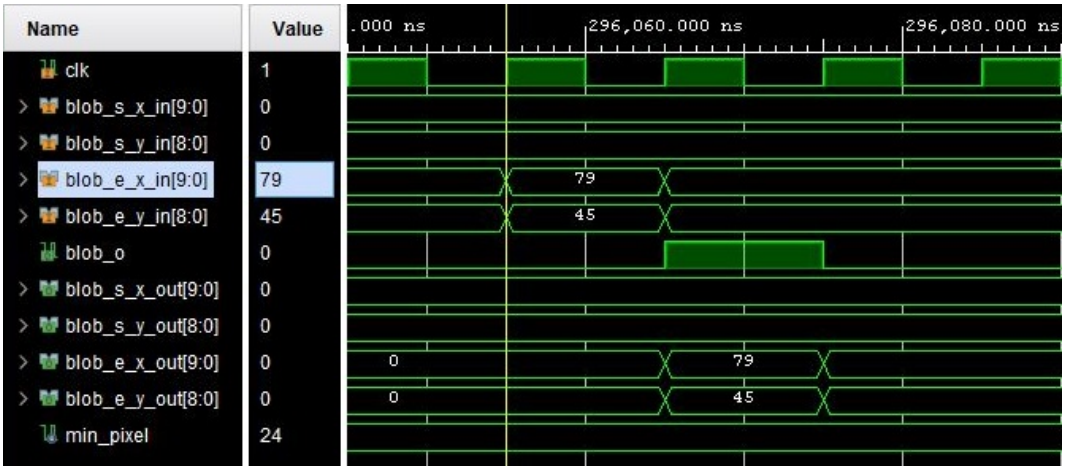


Figure C.12: Waveform of the Blob Filter component in the Blob Detection module

Appendix D

Homography Equations

D.1 Components of Homography Matrix

The equations listed in this section are derived to calculate the homography matrix. The variables in the equations are the four corners of the marker candidate ($[u_1, v_1]$, $[u_2, v_2]$, $[u_3, v_3]$, $[u_4, v_4]$) and the width of the canonical marker (x_4).

$$h_1 = \frac{u_1u_3v_2 - u_2u_3v_1 - u_1u_4v_2 + u_2u_4v_1}{x_4(u_2v_3 - u_3v_2 - u_2v_4 + u_4v_2 + u_3v_4 - u_4v_3)} + \frac{-u_1u_3v_4 + u_1u_4v_3 + u_2u_3v_4 - u_2u_4v_3}{x_4(u_2v_3 - u_3v_2 - u_2v_4 + u_4v_2 + u_3v_4 - u_4v_3)} \tag{D.1}$$

$$h_2 = \frac{-u_1u_2v_3 + u_2u_3v_1 + u_1u_2v_4 - u_1u_4v_2}{x_4(u_2v_3 - u_3v_2 - u_2v_4 + u_4v_2 + u_3v_4 - u_4v_3} + \frac{u_1u_4v_3 - u_3u_4v_1 - u_2u_3v_4 + u_3u_4v_2}{x_4(u_2v_3 - u_3v_2 - u_2v_4 + u_4v_2 + u_3v_4 - u_4v_3} \tag{D.2}$$

$$h_4 = \frac{u_1v_2v_3 - u_2v_1v_3 - u_1v_2v_4 + u_2v_1v_4}{x_4(u_2v_3 - u_3v_2 - u_2v_4 + u_4v_2 + u_3v_4 - u_4v_3} + \frac{-u_3v_1v_4 + u_4v_1v_3 + u_3v_2v_4 - u_4v_2v_3}{x_4(u_2v_3 - u_3v_2 - u_2v_4 + u_4v_2 + u_3v_4 - u_4v_3} \tag{D.3}$$

$$h_5 = \frac{-u_1v_2v_3 + u_3v_1v_2 + u_2v_1v_4 - u_4v_1v_2}{x_4(u_2v_3 - u_3v_2 - u_2v_4 + u_4v_2 + u_3v_4 - u_4v_3} + \frac{u_1v_3v_4 - u_3v_1v_4 - u_2v_3v_4 + u_4v_2v_3}{x_4(u_2v_3 - u_3v_2 - u_2v_4 + u_4v_2 + u_3v_4 - u_4v_3} \tag{D.4}$$

$$h_7 = \frac{u_1v_3 - u_3v_1 - u_1v_4 - u_2v_3}{x_4(u_2v_3 - u_3v_2 - u_2v_4 + u_4v_2 + u_3v_4 - u_4v_3} + \frac{u_3v_2 + u_4v_1 + u_2v_4 - u_4v_2}{x_4(u_2v_3 - u_3v_2 - u_2v_4 + u_4v_2 + u_3v_4 - u_4v_3} \tag{D.5}$$

$$h_8 = \frac{-u_1v_2 + u_2v_1 + u_1v_4 - u_2v_3}{x_4(u_2v_3 - u_3v_2 - u_2v_4 + u_4v_2 + u_3v_4 - u_4v_3} + \frac{u_3v_2 - u_4v_1 - u_3v_4 + u_4v_3}{x_4(u_2v_3 - u_3v_2 - u_2v_4 + u_4v_2 + u_3v_4 - u_4v_3} \tag{D.6}$$

Appendix E

Feature Extraction DUT Schematic

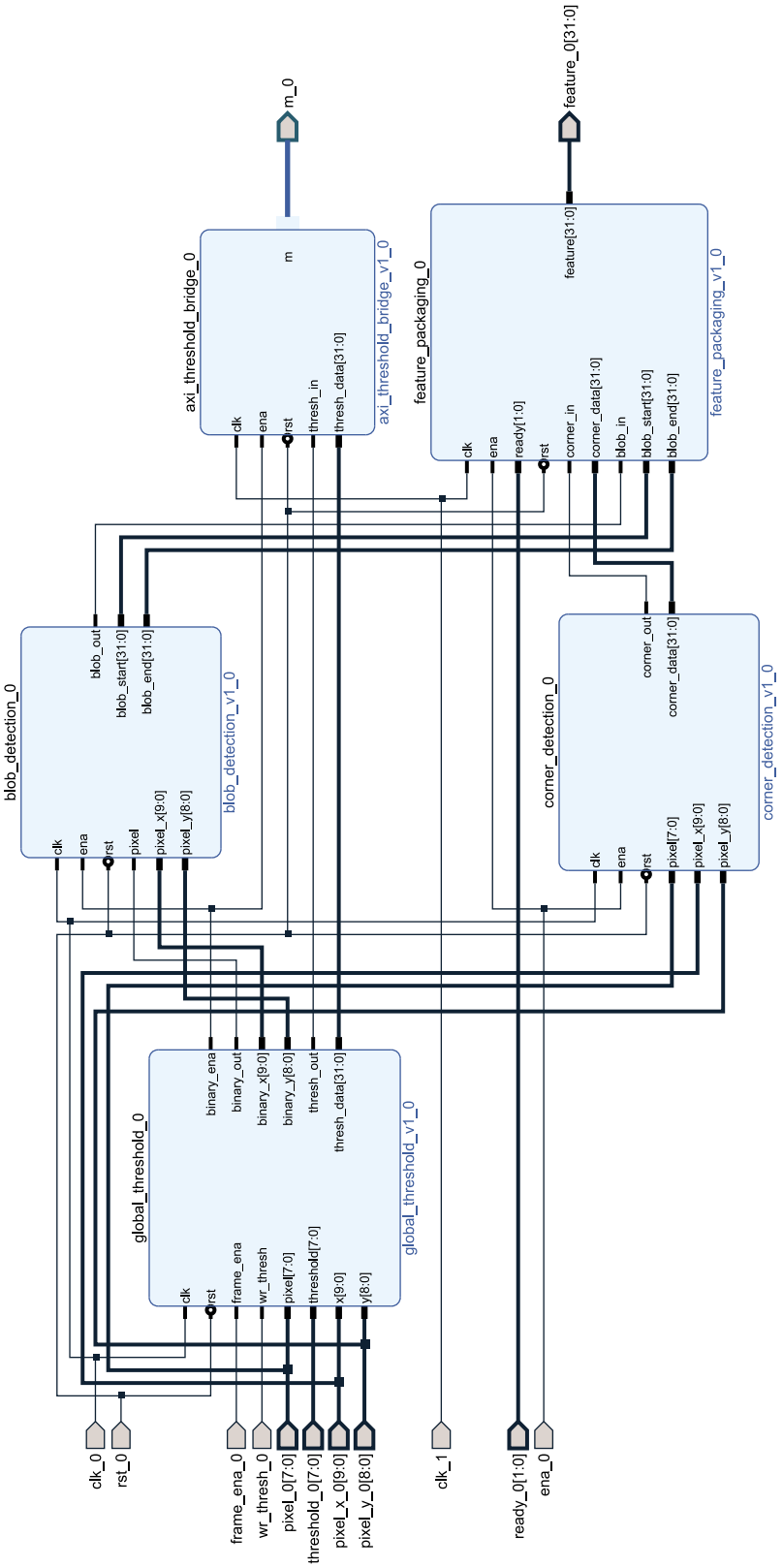


Figure E.1: Schematic of the feature extraction PL modules including the Data Bridge

Appendix F

System Firmware Schematic

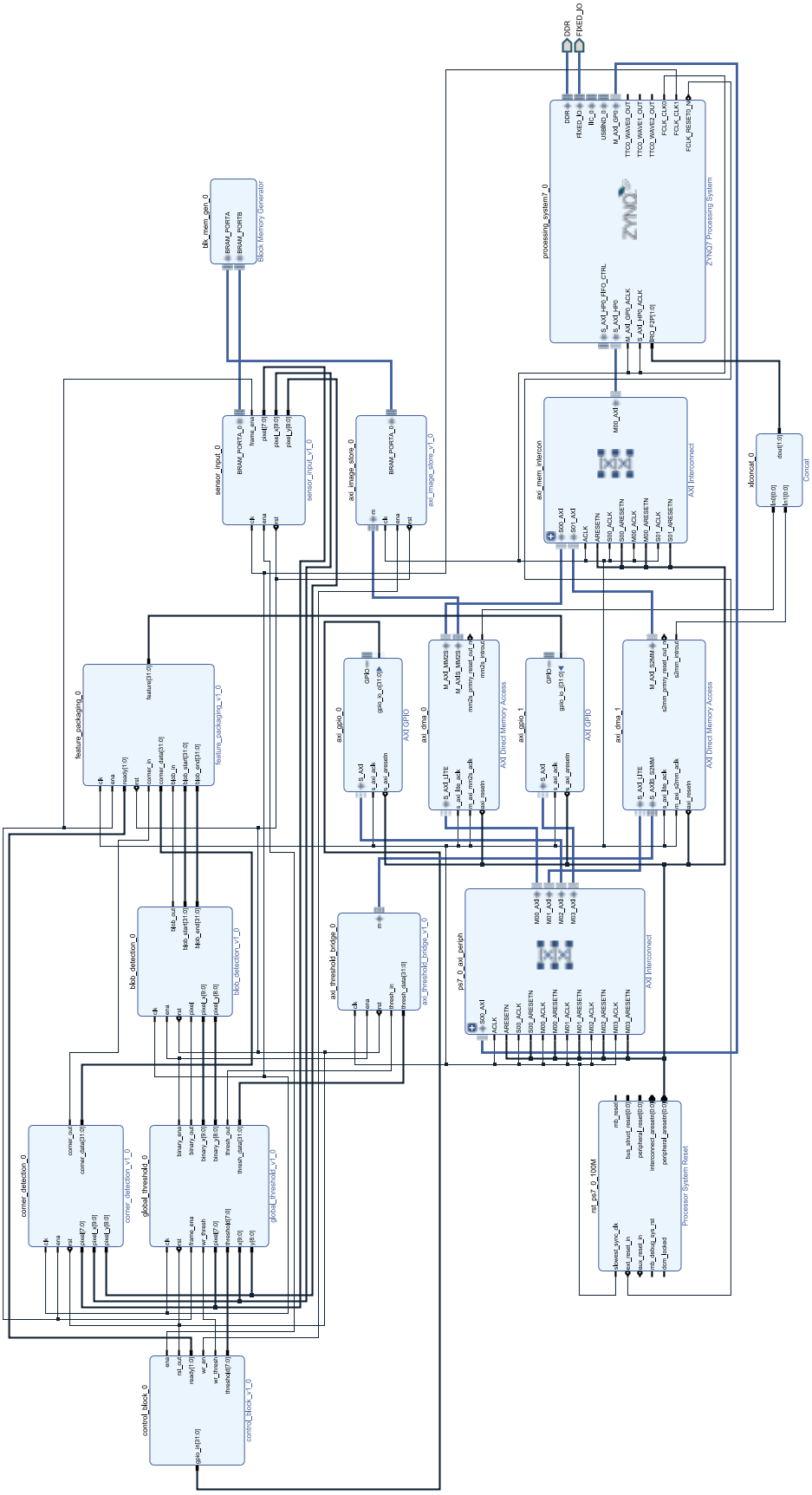


Figure F.1: Enlarged schematic of the marker detection system implemented on the ZYNQ

List of References

[1] M. Kalaitzakis, B. Cain, S. Carroll, A. Ambrosi, C. Whitehead, and N. Vitzilaios, "Fiducial markers for pose estimation: Overview, applications and experimental comparison of the artag, apriltag, aruco and stag markers," *Journal of Intelligent & Robotic Systems*, vol. 101, Apr 2021.

[2] G. Yu, Y. Hu, and J. Dai, "Topotag: A robust and scalable topological fiducial marker system," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, pp. 3769–3780, 2021.

[3] A. Singh, Q. Ha, D. Wood, and M. Bishop, "Low-latency vision-based fiducial detection and localization for object tracking," in *34th International Symposium on Automation and Robotics in Construction*, Jul 2017.

[4] R. Grobler, "Automated recharging and vision-based improved localisation for a quadrotor uav," Master's thesis, Stellenbosch University, Mar 2021.

[5] D. Jansen, "The development of a test facility for satellite experiments," Master's thesis, Stellenbosch University, Mar 2021.

[6] E. Henriksen, I. Schjølberg, and T. Gjersvik, "Vision based localization for subsea intervention," in *ASME 2017 36th International Conference on Ocean, Offshore and Arctic Engineering*, Jun 2017.

[7] R. M. Viljoen, "Cooperative navigation for multiple autonomous ground vehicles (agvs) with kinematic constraints," Master's thesis, Stellenbosch University, Dec 2020.

[8] J. Hawly, "Grayscale photography of rose." [Online]. Available: `https://www.pexels.com/photo/grayscale-photography-of-rose-57905/`.

[9] C. Classroom, "Monochrome vs grayscale photography: What are the differences?." [Online]. Available: `https://www.colesclassroom.com/grayscale-vs-monochrome-photography-what-are-the-differences/`.

[10] F. Cosco, *Visuo-Haptic displays for Interactive Mixed Prototyping*. PhD thesis, University of Calabria, Feb 2011.

[11] S. Imaging, "Introduction to fpga acceleration." [Online]. Available: `https://www.stemmer-imaging.com/en/technical-tips/introduction-to-fpga-acceleration/`.

[12] F. Bergamasco, A. Albarelli, E. Rodolà, and A. Torsello, "Rune-tag: A high accuracy fiducial marker with strong occlusion resilience," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 113 – 120, Jul 2011.

[13] M. J. McNamara, "The deep dive on bit depth." [Online]. Available: `https://www.projectorcentral.com/All-About-Bit-Depth.htm?page=Understanding-Bit-Depth-Specs-When-Shopping`.

[14] S. Garrido-Jurado, R. Muñoz Salinas, F. Madrid-Cuevas, and M. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognition*, vol. 47, pp. 2280 – 2292, Jun 2014.

[15] tutorialspoint, "Sobel operator." [Online]. Available: `https://www.tutorialspoint.com/dip/sobel_operator.htm`.

[16] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Computer Vision – ECCV 2006* (A. Leonardis, H. Bischof, and A. Pinz, eds.), (Berlin, Heidelberg), pp. 430–443, Springer Berlin Heidelberg, 2006.

[17] T. L. B. Pages, "Blob detection." [Online]. Available: `http://www.labbookpages.co.uk/software/imgProc/blobDetection.html`.

[18] S. Suzuki and K. Abe, "Topological structural analysis of digitized binary images by border following," *Comput. Vis. Graph. Image Process.*, vol. 30, pp. 32–46, 1985.

[19] E. C. Vision, "L4: Perspective projection and homography." [Online]. Available: `https://maciek-slon.github.io/ecovi/2018/11/27/L4/`.

[20] D. Oberkampf, D. DeMenthon, and L. Davis, "Iterative pose estimation using coplanar feature points," *Computer Vision and Image Understanding*, vol. 63, pp. 495–511, May 1996.

[21] Z. Zhang, Y. Hu, G. Yu, and J. Dai, "Deeptag: A general framework for fiducial marker design and detection," *ArXiv*, May 2021.

[22] Arrow, "Fpga vs cpu vs gpu vs microcontroller: How do they fit into the processing jigsaw puzzle?." [Online]. Available: `https://www.arrow.com/en/research-and-events/articles/fpga-vs-cpu-vs-gpu-vs-microcontroller`.

[23] Deloitte, "Using autonomous robots to drive supply chain innovation." [Online]. Available: `https://www2.deloitte.com/us/en/pages/manufacturing/articles/autonomous-robots-supply-chain-innovation.html`.

[24] K. L. Moore, "A tutorial introduction to autonomous systems," *IFAC Proceedings Volumes*, vol. 41, no. 2, pp. 11720–11731, 2008. 17th IFAC World Congress.

[25] S. Huang and G. Dissanayake, *Robot Localization: An Introduction*, pp. 1–10. American Cancer Society, 2016.

[26] J. Wang and E. Olson, "Apriltag 2: Efficient and robust fiducial detection," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4193–4198, 2016.

[27] M. Mardan, M. Esfandiari, and N. Sepehri, "Attitude and position controller design and implementation for a quadrotor," *International Journal of Advanced Robotic Systems*, vol. 14, May 2017.

[28] N. Xuan Mung and S.-K. Hong, "Improved altitude control algorithm for quadcopter unmanned aerial vehicles," *Applied Sciences*, vol. 9, May 2019.

[29] CaseGuard, "How many frames per second can the human eye see?." [Online]. Available: `https://caseguard.com/articles/how-many-frames-per-second-can-the-human-eye-see`. Last Modified: 2021-03-21.

[30] M. Fiala, "Artag, a fiducial marker system using digital techniques," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2, pp. 590–596, 2005.

[31] M. Fiala, "Designing highly reliable fiducial markers," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 7, pp. 1317–1324, 2010.

[32] B. Benligiray, C. Topal, and C. Akinlar, "Stag: A stable fiducial marker system," *Image and Vision Computing*, vol. 89, Jul 2017.

[33] X. Liu, S. Zhang, J. Tian, and L. Liu, "An onboard vision-based system for autonomous landing of a low-cost quadrotor on a novel landing pad," *Sensors*, vol. 19, no. 21, 2019.

[34] J. Rademeyer, "Vision-based flight control for a quadrotor uav," Master's thesis, Stellenbosch University, Mar 2020.

[35] L. Jurgen, "Sub-pixel image translation estimation on a nanosatellite platform," Master's thesis, Stellenbosch University, Apr 2019.

[36] C. L. Von Wielligh, "Fast star tracker hardware implementation and algorithm optimisations on a system-on-a-chip device," Master's thesis, Stellenbosch University, Dec 2019.

[37] Dreamstime, "Grayscale low angle photography of high rise building." [Online]. Available: `https://www.dreamstime.com/grayscale-low-angle-photography-high-rise-building`.

[38] Xilinx, "Intellectual property." [Online]. Available: `https://www.xilinx.com/products/intellectual-property.html`.

[39] D. J. Kriegman, "Lecture notes in computer vision i - cse252a," 2007.

[40] OmniVision, *Color CMOS QSXGA (5 megapixel) image sensor with OmniBSI technology*, 5 2011. Rev. 2.03.

[41] M. Sayyed, "embeddedsw." [Online]. Available: `https://github.com/Xilinx/embeddedsw/blob/master/lib/bsp/standalone/src/arm/cortexa9/xtime_l.h`. Last Modified: 2021-04-11.

[42] D. Trends, "What is a cpu." [Online]. Available: `https://www.digitaltrends.com/computing/what-is-a-cpu/`.

[43] Intel, "What is a gpu." [Online]. Available: `https://www.intel.com/content/www/us/en/products/docs/processors/what-is-a-gpu.html`.

[44] J. Köhler, A. Pagani, and D. Stricker, "Detection and identification techniques for markers used in computer vision.," in *Visualization of Large and Unstructured Data Sets - Applications in Geospatial Planning, Modeling and Engineering*, vol. 19, pp. 36–44, Jan 2010.

[45] tutorialspoint, "Hamming distance." [Online]. Available: `https://www.tutorialspoint.com/what-is-hamming-distance`.

[46] E. A. da Silva and G. V. Mendonça, "4 - digital image processing," in *The Electrical Engineering Handbook* (W.-K. CHEN, ed.), pp. 891–910, Burlington: Academic Press, 2005.

[47] F. A. D. G. Initiative, "Term: Thresholding." [Online]. Available: `http://www.digitizationguidelines.gov/term.php?term=thresholding`.

[48] J. Rogowska, "Chapter 5 - overview and fundamentals of medical image segmentation," in *Handbook of Medical Image Processing and Analysis (Second Edition)* (I. N. BANKMAN, ed.), pp. 73–90, Burlington: Academic Press, second edition ed., 2009.

[49] G. Kumar and P. K. Bhatia, "A detailed review of feature extraction in image processing systems," in *2014 Fourth International Conference on Advanced Computing Communication Technologies*, pp. 5–12, 2014.

[50] C. Harris and M. Stephens, "A combined corner and edge detector," in *In Proc. of Fourth Alvey Vision Conference*, pp. 147–151, 1988.

[51] T. R. Society, "Blob detection." [Online]. Available: `https://medium.com/image-processing-in-robotics/blob-detection-309226a3ea5b`.

[52] L. He, X. Ren, Q. Gao, X. Zhao, B. Yao, and Y. Chao, "The connected-component labeling problem: A review of state-of-the-art algorithms," *Pattern Recognition*, vol. 70, pp. 25–43, 2017.

[53] M. Maulion, "Homography transform - image processing." [Online]. Available: `https://mattmaulion.medium.com/homography-transform-image-processing-eddbcb8e4ff7`.

[54] Xilinx, "57304 - vivado timing - where can i find the fmax in the timing report?." [Online]. Available: `https://support.xilinx.com/s/article/57304?language=en_US`.