2019-07-01

Ultra Low Latency Visual Servoing for High Speed Object Tracking Using Multi Focal Length Camera Arrays

Alexander Steven McCown
Brigham Young University

Ultra Low Latency Visual Servoing for High Speed Object Tracking Using Multi Focal Length Camera Arrays

Alexander Steven McCown

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Dah Jye Lee, Chair
Randal W. Beard
Michael J. Wirthlin

Department of Electrical and Computer Engineering

Brigham Young University

ABSTRACT

Ultra Low Latency Visual Servoing for High Speed Object Tracking Using
Multi Focal Length Camera Arrays

Alexander Steven McCown
Department of Electrical and Computer Engineering, BYU
Master of Science

In high speed applications of visual servoing, latency from the recognition algorithm can cause significant degradation of in response time. Hardware acceleration allows for recognition algorithms to be applied directly during the raster scan from the image sensor, thereby removing virtually all video processing latency. This paper examines one such method, along with an analysis of design decisions made to optimize for use during high speed airborne object tracking tests for the US military. Designing test equipment for defense use involves working around unique challenges that arise from having many details being deemed classified or highly sensitive information. Designing tracking system without knowing any exact numbers for speeds, mass, distance or nature of the objects being tracked requires a flexible control system that can be easily tuned after installation. To further improve accuracy and allow rapid tuning to a yet undisclosed set of parameters, a machine learning powered auto-tuner is developed and implemented as a control loop optimizer.

Keywords: visual servoing, low-latency object tracking, FPGA accelerated vision

ACKNOWLEDGMENTS

I would first like to thank Steve Jensen and Paul Nyholm at Lawrence Livermore National Laboratories, for funding this project in its entirety: none of this would have been possible without them being so generous in sponsoring this research. I am very grateful for them and all of the additional help they provided along the way.

I would like to express my sincerest gratitude for my graduate advisor, Dr. D.J. Lee, for the continuous support and immensely valuable guidance on this project and through my University education. A special thanks is also due to Sam Fuller, Matthew Heydorn and Taylor Simons, for countless times digging through ideas with me and sharing priceless insights.

I would also like to thank my wife, Lucy, and my daughter Catherine, for their unfailing support throughout and for giving me the inspiration to put forth my best in all things.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1. INTRODUCTION

1.1 Statement of Problem

The Flight Test Group at Lawrence Livermore National Laboratories is in charge of collecting data on high speed tests of rapidly moving airborne objects. This data is both extremely valuable, and extremely sensitive. Most of the details and virtually all numbers associated with these objects are heavily classified. After working on the team for multiple years, the author of this work still isn't allowed to know any of the following: the speeds at which these objects move, their flight paths, the exact locations at which they will be at test time, or the distance that these objects to be tracked will be from the test equipment.

This information is classified in order to protect the secrets of some of the United States' most critical defense systems. That being the case, with many of these numbers unavailable, a plethora of obstacles arise in the task of accurately tracking these targets in real-time. Without knowing the size, mass, velocity, or even distance from the vantage point, it's impossible to model their behavior mathematically.

Historically, these tests have been performed using multi-camera arrays built with some of the worlds fastest high speed cameras. These cameras are often in the Phantom line from Vision Research, and are capable of capturing over 500,000 frames per second. These cameras were placed strategically in vicinity of the test site, and equipped with very wide angle lenses to ensure that regardless of the exact flight path, the object to be tracked would at least be somewhere in the frame.

The task of this project is to eliminate the need for wide angle lenses by aiming the high speed cameras directly at the target as it flies by. This represents a massive improvement in the caliber of data that can be recorded, as it maps the entire resolution of the high speed cameras more tightly onto the object of interest, thereby providing the small region of interest with a much higher resolution.

Many traditional methods, such as modeling and predicting the flight path, or modeling the target and predicting its interactions with its environment, would be far too inaccurate. One feasible solution is to introduce a wide field of view camera, and use extremely low latency real time object tracking as the backbone for high-performance visual servoing.

1.2 Background

Visual Servoing is the process of using visual sensor information to control robotic actuators and adjust position or orientation in response. Object recognition algorithms are used to detect an object of interest, and a control loop calculates responses and sends commands to mechanical actuators, often to center the object in frame.

Plaguing the performance of this type of control system are two major factors: latency in processing the visual sensor data, and latency in response of the mechanical actuation. The latter of these two problems can be mitigated by using modern high speed camera gimbals with faster response times and greater angular accelerations. The former is the primary focus of this work.

In many vision processing applications, large portions of the recognition algorithms are performed in software. This means that the camera frames are read in and stored in memory where the processor can read them at its own speed. This in turn necessitates that information from the camera be stored in the interim, which introduces a measurable delay. Additionally, camera interfaces are often simplified by the use of buffers, which store one or more frames internally, so as to allow a communications chip to request data before its transmitted, thus making data transmission more manageable. This further introduces delays, and results in the processor being multiple frames behind events happening around it. Further compounding this issue is the fact that many cameras and interfaces require compression, which means that the data must first be compressed before it can be transmitted, and then uncompressed after the fact. This introduces additional latency.

This paper explores the idea of implementing recognition algorithms in a Field-Programmable Gate Array, or FPGA, as well as managing the capture of pixel information directly within the FPGA itself, so as to minimize the delay between an event occurring and the start of the corresponding reaction to that event. One major strength of a system with direct access to the camera

sensor is that it eliminates all buffers and interface delays: the visual information is no longer several frames behind real-time.

Another advantage is that with this type of system, the FPGA is then able to pull in visual information and locate the object of interest very rapidly, much sooner than traditional recognition systems allow. Implementing the control system in the same FPGA allows for output (going to the actuators) to adjust nearly instantly after movement of the object of interest.

1.3 Definition of Terms

Visual Servoing – while not technically an English word, servoing is the process of obtaining and maintaining a target position or orientation. Visual servoing is doing so using feedback from visual sensory input.

Colorspace – a method of digital representation of color and light information. Many colorspaces are widely used, and vary greatly in their method of representing colors.

Grayscale – a pseudo colorspace, with zero information representing color. Grayscale is exactly what it sound like: a scale representing shades of gray.

1.4 Literature Survey

From a scholarly standpoint, Visual Servoing is the concept of using visual sensor information to control motion or orientation of robotic systems. Often a set of visual features are extracted from the sensor data and used to extrapolate a multidimensional modeling of the environment around the robot [1]. This representation is used to approximate the error in position and orientation, which is then used to compute the desired action [2].

The term ”Visual Servoing” has been around for decades, and was likely first coined by Hill and Park in 1979 [3]. Their work is distinguished as being a major break from previous designs which essentially alternated between capturing a single image, processing it, and moving accordingly. Hill and Park’s innovation was a more continual approach that provided a more seamless response [3].

In recent years, much focus has been given to coordinate-based modeling, (see [2]) which generally allows for more precision, but has the drawback of requiring more information about the

system and its environment. In most cases, an interaction matrix L_s is usually required, which is difficult to construct when information about the environment is limited.

One recent development that loosely relates to this paper is from Collewet and Marchand's 2011 work on Photometric Visual Servoing [4], which proposed a much more generic system that did away with all geometric modeling of the objects being tracked. This is intriguing of its own right, and lends itself well to environments or objects that are not well modeled.

This approach produced tracking accuracy roughly one order of magnitude better than a model based approach using SIFT feature tracking (0.001 degree error vs 0.041). This method was also particularly effective on objects that either had sharp changes in light intensity, or were of a different level of luminance than the image background. [4]

Another recent work that is perhaps the most similar to the vision portion of this project is the work on FPGA based real-time object detection by Alexander Bochem, Kenneth B Kent and Rainer Herpers [5].

In this work, the authors present an FPGA based implementation of a real-time multiple blob tracking algorithm. They utilize a bounding box and center of mass estimates for approximating the center point of each of the blobs. By checking adjacency of each pixel that met the given threshold, it was possible to combine blobs that had convexity's on their upper most sides.

The thresholding performed was done simplistically, with a single value binarization thresholding the brightness of a given pixel, in fact the author there states that a pixel is considered to be relevant if its brightness value exceeds a specified threshold value [5]. However, it's also stated that the processing was done in RGB, which implies that some conversion is taking to transform the RGB values into at least a gray-scale channel.

This method has several distinct advantages, such as the ability to distinguish cleanly between individual blobs and to combine together blobs that intersect at any point in the frame. One major drawback is the processing time that these algorithms require. Using either the bounding box or the center-of-mass methods, their algorithm could only process up to 12 frames per second at 640x480. This is impressive given the complexity of their algorithm and the amount of resources used, but quickly becomes prohibitive as the latency-sensitivity of a system increases. Later work by these same authors does increase the frame rate of blob-detection to roughly 60

frames per second, which will be discussed in Chapter 9. This work was tracking a white dot on a black background.

On the front of implementing control loops in FPGA hardware, one paper that was referenced in the course of this project is the paper *Design and Implementation of FPGA - digital based PID controller* [6]. This paper provides a great overview of the process of implementing PID controllers in FPGAs.

Last but not least, the work most simliar to the overall layout of this Thesis is the work by Aiman Alabdo, Javier Prez, Gabriel J. Garcia, Jorge Pomares, and Fernando Torres at the University of Alicante, Spain. In their work, *FPGA-based architecture for direct visual control robotic systems* [7], they developed a visual servoing system that utilized FPGA vision processing. This system was demonstrated able to track a white dot on a solid black surface.

To accomplish this, they implemented traditional center-of-mass blob detection. Like the Bochem, et al work, this also used only a single-channel gray-scale image.

1.5 Description of Remaining Chapters

The remainder of this Thesis will discuss in detail the work done to implement an ultra low latency recognition algorithm and the control of the gimbal itself.

Chapter 2 provides some background information on some of the central machine vision concepts that are utilized in this project. Chapter 3 details the hardware chosen for use in this project, along with some discussion behind changes that were made along the way. Chapter 4 expands on this with documentation on how the hardware was connected together.

Chapter 5 provides an overview of the algorithm implemented for the vision tracking portion of this project, along with a brief discussion of the merits of using this level of optimization for hardware acceleration. Chapter 6 discusses the top level system design. Unlike Chapter 4 which discusses the physical connections, Chapter 6 delves into the design internal to the FPGA.

Chapter 7 discusses the custom testing platform that was developed to benchmark and tune this system. The usefulness of this testing platform is furthered by means of the machine learning algorithm implemented to fine tune the gimbal system. This machine learning automated tuning is discussed in Chapter 8, where results are shared for the improvement this brought over manually tuning methods.

Chapter 9 gives some performance comparisons to related works. Chapter 10 provides some insights into future improvements on this project, as well as additional applications where portions of this work could be implemented.

CHAPTER 2. VISION BACKGROUND INFORMATION

This chapter serves as an introduction to the elements of computer vision that are pertinent to and utilized in this project.

2.1 Colorspaces

One of the most fundamental elements in any machine vision application is the representation of the image data. There are many methods for representing image data, all of which have their own unique strengths and drawbacks. A few of these that were either considered or utilized in this project will be discussed here.

2.1.1 RGB Colorspace

In many instances, an image is represented in terms of the red, green, and blue light contents. This representation is known as the RGB colorspace, and is the easiest to display on television and computer monitor screens, which are generally designed to display images in the RGB format.

Many camera sensors are designed to capture raw images in this format, often using 8 bit representations for each of the three color components (red, green, and blue), producing 24 bits per pixel. A major strength to this method of representation is simplicity: because the human eye readily recognizes this format, displaying the intermediated images in this format is trivial.

The major drawback to this method is that it is incredibly luminosity sensitive. Often, computer vision algorithms are used to look for objects of known colors, such as yellow lines on a road, or products on a conveyor belt, which requires isolating objects of the desired color from their surroundings. When the overall brightness in the environment cannot be directly controlled, the RGB representation becomes very difficult to accurately distinguish between colors. This is

because as the overall brightness increases, the value of each of the color component channels can be altered drastically.

For example, supposing all color channels are integer values from 0 to 255, with 255 meaning that color component is full brightness and 0 meaning that it's absent, a naive approach to searching for orange cones in a daylight environment might be to represent the desired color as blue less than 10, red greater than 150 and green between 75 and 150. This breaks down when the ambient light changes, for example in a dark environment the blue will still be less than 10 but the red might now be only 50, and the green might need to be between 25 and 30.

Because of this property, isolating specific colors can require nonlinear estimations, which can make the RGB colorspace very difficult to use in hardware accelerated applications.

2.1.2 HSV Colorspace

Another color space that is very commonly used in Machine Vision applications, particularly when color detection is important, is known as the Hue Saturation Value, or HSV, representation.

In this representation, there are similarly three color component channels, which are often also represented with 8 bits each. However, rather than representing light components how the human eye sees them, this representation stores the color information very differently.

The value channel is best thought of as a grayscale representation of the image. The Hue and saturation channels are slightly more difficult to grasp conceptually. Essentially, each pixel is represented as being somewhere on a color wheel, with nearly colorless graytones being close to the center, and sharper more solid colors being around the edges.

The hue and saturation are essentially polar coordinates on this color wheel. The saturation component describes how far from the center, and the Hue describes which color is being represented.

A visual depiction of the HSV colorspace can be seen in Figure 2.1, which illustrates how the saturation and hue determine the color, regardless of the value channel. Of course, whenever value is close to zero, all colors appear black regardless of the hue and saturation channels.

HSV has several major advantages over the RGB space discussed earlier. For one, changes in ambient lighting will generally only affect the value Channel, leaving the saturation and hue

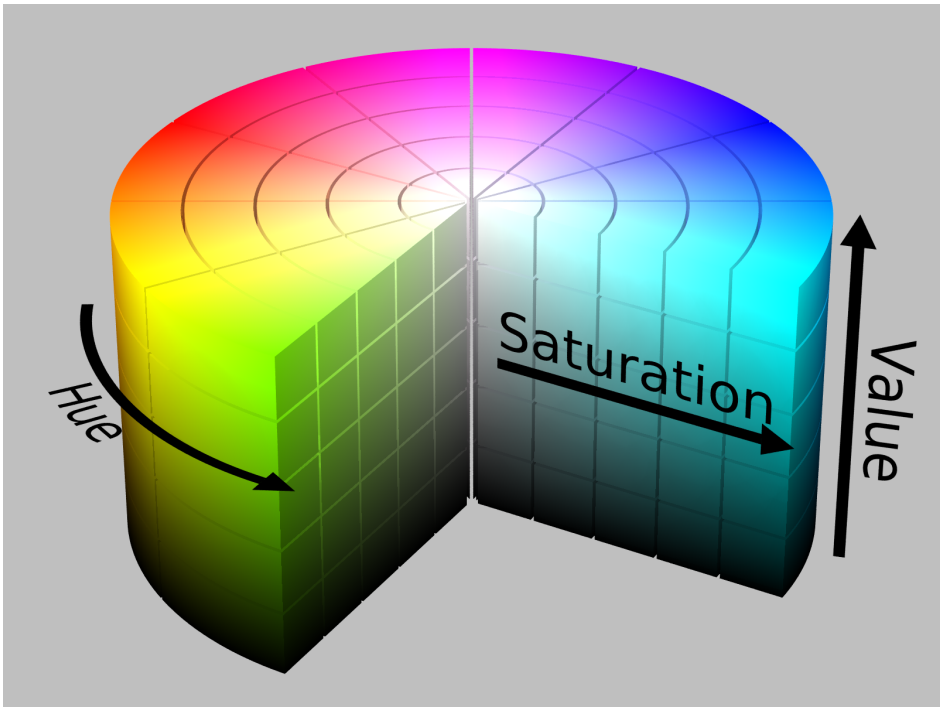

Figure 2.1: The HSV colorspace mapped to a cylinder. [8]

unaltered. This means that in our simplistic example earlier of looking for orange cones, The target color might be represented as having a hue between 150 and 175, a saturation above 80, and the value channel can almost be completely ignored. Often this channel is simply threshold such that it's not completely dark, as black and can be represented as having any Hue and any saturation in the HSV representation. As changes in the overall brightness in the scene will only affect the value channel, assuming it doesn't rail in either direction, we can generally leave the target hue and saturation levels the same.

One major drawback to this color representation is the complexity in converting from RGB. The equations to convert from RGB to HSV are as follows:

$$R' = R/255 \tag{2.1}$$

$$G' = G/255 \tag{2.2}$$

$$B' = B/255 \tag{2.3}$$

$$Cmax = max(R', G', B') \tag{2.4}$$

$$Cmin = min(R', G', B') \tag{2.5}$$

$$\Delta = Cmax - Cmin. \tag{2.6}$$

$$Hue = \left\{ \begin{array}{ll} 60^\circ \times \dfrac{G'-B'}{\Delta} \mod 6), & \text{for } Cmax = R' \\ 60^\circ \times \dfrac{B'-R'}{\Delta} + 2), & \text{for } Cmax = G' \\ 60^\circ \times \dfrac{R'-G'}{\Delta} + 4), & \text{for } Cmax = B' \end{array} \right\} \tag{2.7}$$

$$Saturation = \left\{ \begin{array}{ll} 0, & \text{for } Cmax = 0 \\ \Delta/Cmax, & \text{for } Cmax \neq 0 \end{array} \right\} \tag{2.8}$$

$$Value = Cmax \tag{2.9}$$

The primary disadvantage to this method is that, as seen in Equation 2.7, calculating the hue requires division of two non constants. As division with a non constant divisor is a very resource intensive operation, converting to HSV is not as hardware-friendly as other approaches.

2.1.3 YUV/YCbCr Colorspace

Another alternative colorspace is the YUV Colorspace. It is worth noting here that while YUV is technically an analog standard used in old television broadcasts, and YCrCb is a digital approximation, much of the documentation and implementation available will use the two names interchangeably. To avoid further confusion, this work will refer to this colorspace as YCbCr only.

The general equations for converting from RGB to YCbCr are defined by the NTSC standard:

$$Y = 0.299R + 0.587G + 0.114B \tag{2.10}$$

$$Cb = -0.147R - 0.289G + 0.436B \tag{2.11}$$

$$Cr = 0.615R - 0.515G - 0.100B. \tag{2.12}$$

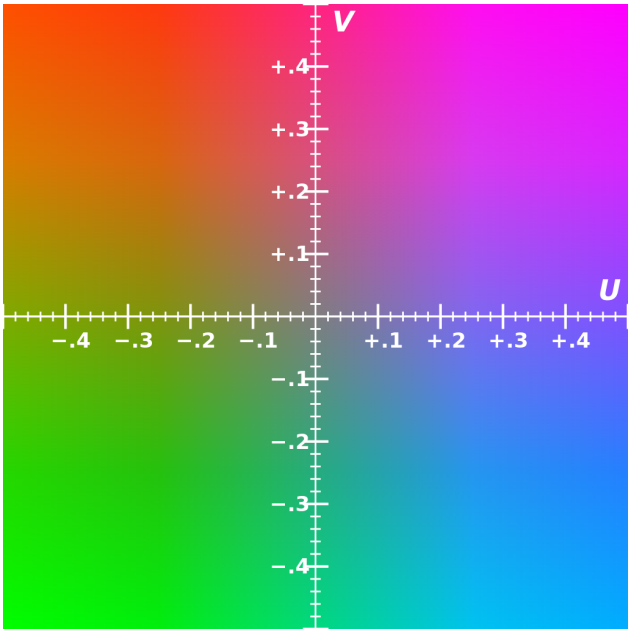

Figure 2.2: The UV (or Cb Cr) plane in YCbCr colorspace shows an approximate representation of Cb and Cr values required to produce a given color. [9]

The main advantage of this colorspace is that it maps the color information onto a plane such that it groups similar colors into regions, identifiable by their Cb and Cr values, while being almost entirely irrespective to ambient lighting. This allows selecting specific colors to be recognized even when the system is used in a different lighting situation.

Because this application requires hardware acceleration as well as accurate color tracking, YCbCr was chosen for the vision recognition algorithm here.

CHAPTER 3. HARDWARE SELECTION

3.1 Camera Sensor

Several sensor types were purchased and used in this project. Each had its own strengths and shortcomings, but all were intended to be used in a parallel direct interface.

A few of these sensors were made by OmniVision. The one that was eventually chosen for this project was the OV 5642. This sensor was chosen mainly because it is capable of outputting at 120fps, and operates at a 3.3V logic level, making interfacing with standard FPGA GPIO very straightforward.

The OV5642 is a 1/4 inch sensor with a max resolution of 5 megapixels (2592x1944) that uses the OmniBSI technology. It supports high frame-rates at lower resolutions, including a QVGA mode that is designed to run at 120 fps. The OV5642 takes an input clock between 6 - 27 MHz, and has an onboard Phased Locked Loop (PLL) that can upscale the clock input for higher data-rates. It provides parallel 8 bit data output, along with a pixel clock for direct synchronization.

Another strong advantage to this sensor is that it's readily available in a convenient package that utilizes a CS-lens mount, which is one of the most common lens styles used in security cameras. This features was a great asset, in that it allowed the use of the extensive market for CS mount lenses used in CCTV security cameras.

This sensor also has a built-in I2C-like interface for setting register values and restarting the chip. To maintain a truly hardware-friendly implementation, a lightweight I2C sender was implemented in VHDL for loading pre-determined values into these registers at startup.

The configuration details of the OV5642 as required for this project can be seen in the register values in the source code, but some notable features that were utilized are the manual exposure and gain settings, and the configuration of the PLL to multiply the input clock by 4 for the pixel clock.

Figure 3.1: The OV5642 [10]

3.2 Gimbal

The gimbal used in this project was the M$\bar{o}$VI XL made by Freefly Systems. Figure 3.2 shows the MVI XL gimbal mounted from the top. This gimbal supports mounting either above or below, ie in a top-down or hanging configuration.

The M$\bar{o}$VI XL was designed to be an extremely versatile high performance camera system, intended to be used in professional video recording. As such, it sports impressive performance specs, is able to accept a wide variety of control inputs, and comes with a very robust set of mounting brackets. These brackets turned out to be an invaluable resource in adding additional components for testing.

The M$\bar{o}$VI XL supports full 360 °continuous rotation, as well as software adjustible end stops on the tilt and roll axes. According to the specification document, the maximum rotation rates for all three axes is 200°per second.

While only weighing 11kg, the M$\bar{o}$VI XL is able to swing a payload of up to 23kg, and supports mounting in applications that are moving at up to 100mph. The M$\bar{o}$VI XL utilizes three large direct-drive 3-phase brushless DC motors, one for each axis of rotation, each being rated at 1300w of power.

Figure 3.2: The M$\bar{o}$VI XL High-Powered Gimbal [11]

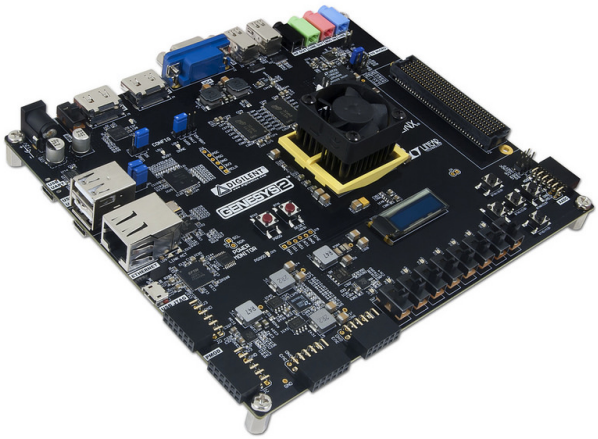

Figure 3.3: Digilent Genesys 2 Board [12]

Some of the interfaces that support input into the M$\bar{o}$VI XL are TSU ports used for both Timecode inputs/outputs, as well as CAN bus inputs and battery voltage sensing. There are also auxiliary ports that support the Futaba SBUS protocol for direct control of the relative movements of the gimbal.

The M$\bar{o}$VI XL gimbal used in this project was provided by the funding party, the Flight Test Group at Lawrence Livermore National Laboratories.

3.3 FPGA Boards

Similarly, several FPGA boards were used in this project. Unlike the camera sensors, however, two were used for more than just initial testing.

Digilent Genesys 2

The first such board was the Genesis 2 made by Digilent. This board had several major advantages over the others, such as the fact that it's built with a Kintex class FPGA and benefits from the smaller process node, as well as a greater number of resources available internally. Another advantage, readily apparent in Figure 3.3, is the fact that it comes with several built-in features that are extremely useful in development and debugging, such as multiple video input and output ports, and a row of switches and LEDs, as well as several buttons.

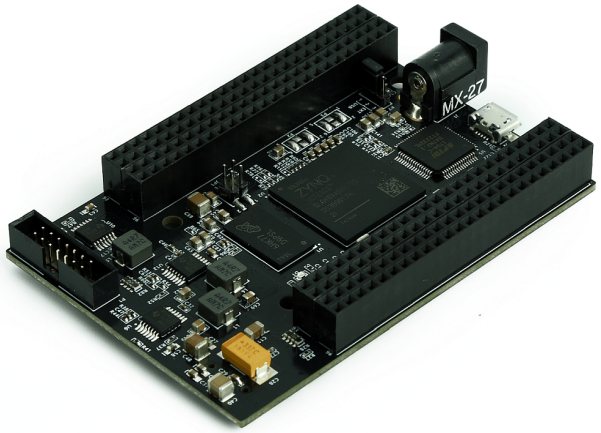

Figure 3.4: The Numato Styx Board [13]

This board was used for initial testing of the video processing algorithm, largely due to the appeal of those built-in video output ports. Eventually, however, it became clear that this board's size was too hard to justify, even given the strength in terms of FPGA fabric size.

Numato Styx

Another board that was used in this project is the Numato Styx. This board is built on one of the 7020 series Zynq chips, leaving the potential to utilize the on-board ARM core for future works. This board was eventually chosen due to its small form factor and how readily the camera sensors lined up with the power, ground and GPIO pins on one of the on-board headers.

Almost too perfectly, the pins of the OV5642 lined up perfectly on one end of the P5 header on the Styx board. Another obvious advantage to this board is the small form factor, seen in Figure 3.4, which is small thanks to the minimal amount of unnecessary hardware, such as the switches, buttons and displays that are common among development board.

CHAPTER 4. HARDWARE CONNECTIONS AND SETUP

There were many portions of this project that required unusual mounts and connections. Many of these necessitated custom fabrication, and a few fit quite fortuitously with neighboring portions of the system. This chapter discusses the physical setup and configuration, and a few of the mounts and connectors that were made in the development process.

4.1 Gimbal Mount

The first step in testing the actuation of the M$\bar{o}$VI XL was ensuring it was secured with enough strength to counteract the massive torque of the motors. The mounting system provided on the M$\bar{o}$VI XL was clearly designed for space conservation rather than ease of mounting. The bolt holes are such that the screws are tightened underneath the gimbal pointing up towards the pan motor.

To facilitate this, a custom mounting plate was designed and milled out in the basement of the Engineering Building, with interior bolt holes milled out to the dimensions provided in the M$\bar{o}$VI XL manual, and outer holes drilled to allow bolts facing downward away from the gimbal. Figure 4.1 shows the completed product, and Figure 4.2 illustrates how the mount attaches to the small plate on the bottom of the M$\bar{o}$VI XL. This plate brought a drastic reduction in the difficulty in removing the bolts that secure the M$\bar{o}$VI XL onto the base, as access to the underside of the base was no longer required.

4.2 Camera Connection

The connection from the camera to the FPGA took on several different styles at various stages of the development of this system. During the initial development of the vision recognition algorithm, which took place using the Genesys 2 board mentioned in Chapter 3, the camera was connected to the Peripheral Module (Pmod) ports using single stranded wires. This worked fine

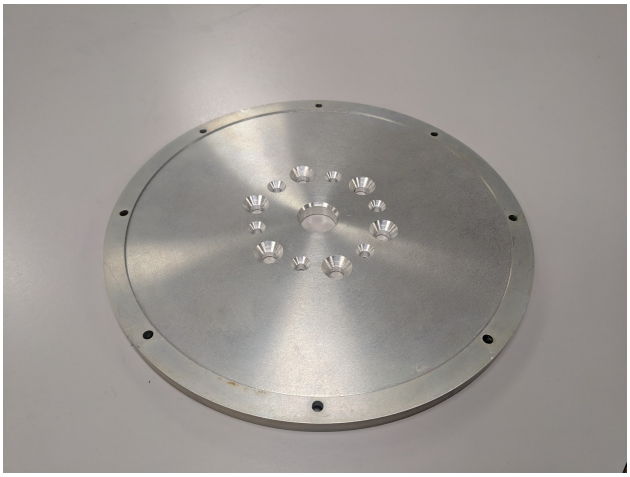

Figure 4.1: Custom mounting plate designed for this project

Figure 4.2: Custom mounting plate bolted on to the M$\bar{o}$VI XL.

at lower frame rates where the clock going to the OV5642, and more particularly the pixel clock coming out of it, could be kept below 25 Mhz. Beyond this point, which was needed to utilize the higher framerates this sensor boasts, these long wires became an issue.

Switching to the Numato Styx board brought several major advantages. Not only was this board smaller and easier to mount on the Gimbal, but as mentioned previously, the pins lined up perfectly where one end of the P5 header had ground and 3.3v supply lines that fit right where the header pins coming out of the OV5642 required them. This allows the camera to plug directly into the FPGA headers, alleviating the need for jumper wires altogether.

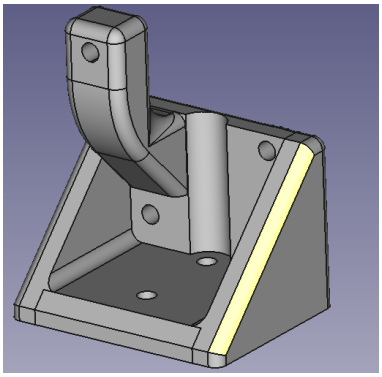

Figure 4.3: Design render of the custom mount fitting camera and FPGA together

This kept the camera offset from the FPGA board roughly 12mm, which was alleviated with brass stand offs. The mounting holes in both the Numato Styx board and the OV5642 breakout board accepted standard 3.5mm stand offs, which were used to mount the two together.

With the goal of leaving the primary payload area open for the high speed cameras, the most logical place to position the camera and FPGA pair was on top of the upper most bracket on the M$\bar{o}$VI XL. As no mount existed prior that could fit the FPGA and camera together, let alone affix them onto the bracket, one was designed and 3D printed on site.

After several revisions, and improvements made to strengthen the mount and reduce flexing under the intense acceleration that the M$\bar{o}$VI XL is designed to produce, the final form of the mounting bracket can be seen in Figure 4.3.

The bottom of this mount was designed such that it mates perfectly onto the holes at either end of the top bracket on the payload area of the M$\bar{o}$VI XL, which also served the purpose of maintaining alignment with the gimbal arm. Figure 4.4 shows this mounted on top of the bracket on the M$\bar{o}$VI XL.

4.3 Control Line Inputs

The S.Bus input was chosen to be used to issue commands to the M$\bar{o}$VI XL. S.Bus is an inverted serial protocol invented by Futaba, and intended to be a replacement for PWM servo control on radio controlled aircraft. This bus works by sending a packet of 25 bytes that map to 16 individual 11 bit values, each of which would traditionally be represented as a stand alone PWM signal.

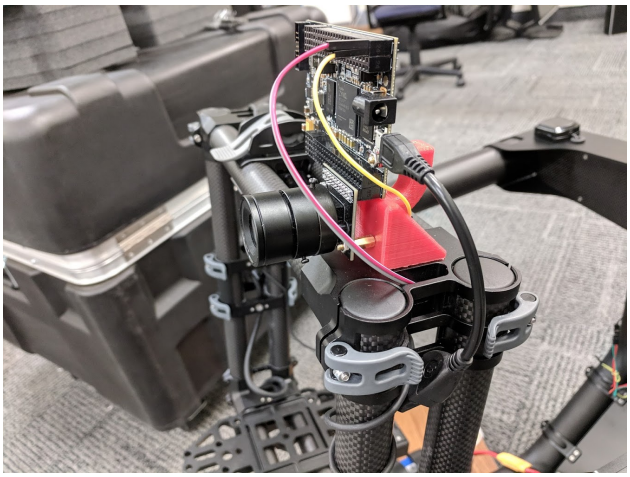

Figure 4.4: Camera and FPGA together mounted in place on the M$\bar{o}$VI XL

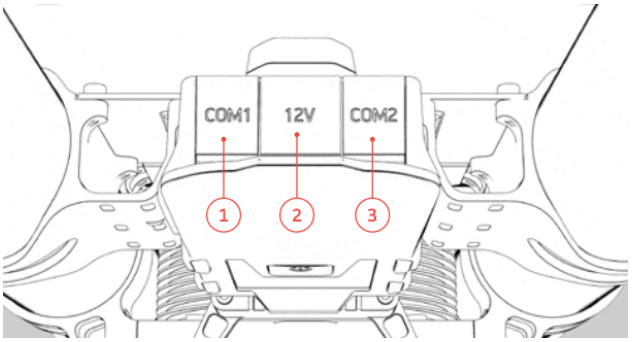

Figure 4.5: Com port locations on the M$\bar{o}$VI XL

The S.Bus input on the the M$\bar{o}$VI XL is wired internally to the Com 1 port on the control box behind the main arm of the pan axis. As this port is located outside the main payload area, it was necessary to run a control line through the hinges, into the Com1 input seen in Figure 4.5. In order to minimize overall latency of the system, the S.Bus generator is implemented in the FPGA, so a custom cable was required to connect the two.

CHAPTER 5. VISION ALGORITHM DESIGN

At the core of this project is the design of the vision recognition system. For purposes of clarity, this will be broken down into three distinct parts: the colorpsace coversion, the thresholding / binarization stage, and the real-time blob detection stage.

5.1 Colorspace Conversion

YCbCr is a simplified color space that allows hardware implementation to be done with fairly low resource use. This color space is discussed in more detail in Chapter 2. The Y channel, the luma component, is essentially a gray-scale representation of the image, while Cb and Cr are the blue-difference and red-difference chroma values. The main advantage of this colorspace is that it groups similar colors into regions identifiable by their Cb and Cr values, while being almost entirely irrespective to ambient lighting. This allows selecting specific colors to be recognized even when the system is used in a different lighting situation.

For the OV5642 sensor, the Y, Cb, and Cr channels are calculated slightly differently than the standard YCbCr colorspace, likely due to unique aspects of the sensor due to the manufacturing process used. In practice, this simply means that for this sensor, the YCbCr conversions require different equations than the NTSC standard equations. In this case, the RGB to YCbCr conversion is done using the following equations:

$$Y = 0.299R + 0.587G + 0.114B \tag{5.1}$$

$$Cb = -0.172R - 0.339G + 0.511B + 128 \tag{5.2}$$

$$Cr = 0.511R - 0.428G - 0.083B + 128. \tag{5.3}$$

However, since division and floating point multiplication isn't very easy to implement in hardware, it is very beneficial to simplify these equations by approximating the values as integers to later be divided by a power of 2 – in this case 256, where $>>$ is the logical bit-shift operator:

$$Y = (77R + 150G + 29B) >> 8 \tag{5.4}$$

$$Cb = ((-43R - 85G + 128B) >> 8) + 128 \tag{5.5}$$

$$Cr = ((128R - 107G - 21B) >> 8) + 128. \tag{5.6}$$

Using these reduced equations above allows the conversions to be multiplication and addition/subtraction only operations, followed by a left shift to compensate for the earlier adjustment.

For example, in the first equation for Y, we see that the R channel must be multiplied by 0.299. To remove the need for decimal calculations, we can substitute in a number scaled by our precomputed scaling factor, in this case 256. Since 0.299 X 256 = 76.54, we simply round to the nearest whole number, which in this case is 77, which is the coefficient for R seen in Equation 5.4, which is how this conversion is currently implemented. Then, after summing up the individual terms, we simply shift right by 8, equivalent to performing a division by 2^8, thereby removing the 256 scaling factor we used earlier.

5.2 Binarization

One immediately obvious requirement for recognition of our airborne object is proper binarization of the pixel data. As each pixel is read in, it must first be thresholded to produce binary data with 1's representing pixels that match the target and 0's representing those that don't.

This can be accomplished in a variety of ways, each bringing their own unique advantages and drawbacks, and often need to be tailored to the specific application. For this project, binarization was done with direct thresholding of the Y, gray-scale, component along with a combination of the color information stored in the Cb and Cr channels, which were similarly thresholded to further reduce false positive pixels.

After the color conversion step, see Section 5.1, the real time video stream has three channels, the Y, Cb, and Cr components, for each pixel.

Since the object to be tracked is of a known brightness and color, the binarization step is implemented by thresholding the three channels around their target values. For this project, since the actual objects to be tracked are sensitive information, the project sponsor, the Flight Test Group at Lawrence Livermore National Lab, requested the demonstrations be performed with the system tracking a colored laser.

Tracking a green laser was accomplished by thresholding the Y channel such that any pixels with a Y below 200, or roughly 78% of the max, were ignored, and also thresholding the Cb and Cr channels such that any pixels below 128, or roughly 50% of the max, on the Cb and Cr channels were ignored.

As is the case in the YCbCr color plane in Figure 2.2, these color components are often represented as being signed decimals ranging between -1 and 1. In this case, values on Cb and Cr that are below 128 represent strictly negative values when scaled between -1 and 1. As can be seen in Figure 2.2, the region on the colorplane where both the Cb and the Cr channels are less than 0 is the green portion of the colorspace.

5.3 Recognition Algorithm

The recognition algorithm used in this project is implemented in real time during the raster scan as pixel data comes directly from the image sensor. This algorithm was aggressively optimized for hardware acceleration.

The algorithm can be thought of as a loose variant of the image histogram representation. Instead of using tonal or pixel values as the independent axis, the line and column numbers themselves are used as the bins.

Center Detection

As each pixel is read in, it must first be thresholded to produce binary data with 1s representing pixels that match the target and 0s representing those that dont. Each time a 1 is found, two counters are incremented: one counter is the bin representing the line number currently being

read, and the other is the one that represents the column number, ie the current pixels location in that line.

As each line is completed, the value in that line's bin will be equal to the number of non-zero values in that line. Similarly, at the conclusion of each frame, the column bins will hold counts which will be equal to the number of non-zero values in their respective columns.

Assuming proper binarization, discussed in Section 5.2, these bins will then hold values equal to, or at least very close to, zero everywhere except in the rows and columns where our object of interest is in the current frame. Then the column with the highest count is selected as the approximate x coordinate of our object, and the row with the highest count is selected as the y coordinate.

This can be thought of as a heavily optimized hardware accelerated version of a projection onto the column and row axis.

Locating these maximums then reveals the location of the center of the blob. To further reduce the latency and optimize the algorithm for hardware, this locating the maximum phase was also implemented during the raster scan. As each pixel is read in, and the counts for the current rows bin updated, that bin's value is simultaneously checked against a stored value – the maximum row value previously encountered in the frame. The same is done for the column bin, which similarly is checked and updated if the current pixel results in a count that is greater than the maximum value encountered thus far.

Tracking of two more values was required: An index of the greatest-so-far values for the row, and also one for the column. These were updated every time the greatest-so-far values were, and they hold the current values for the row or column at the time the update was called.

Figure 5.1 shows an example of this, with a very low binning resolution done here for clarity. Here the black represents the pixels that were below the threshold, and were therefore treated as 0's, and the green represents the pixels that met all the threshold limits and were treated as 1's. As can be seen, the column and row bins are left with 0's everywhere except where the object is. Assuming the object is roughly round, the bin with the greatest count will always be the one at the center of the blob.

For the column bins in this example, the greatest value occurs at the 7th column from the left, which is where the object has its tallest point, and that bin is left with a value of 3 . Similarly,

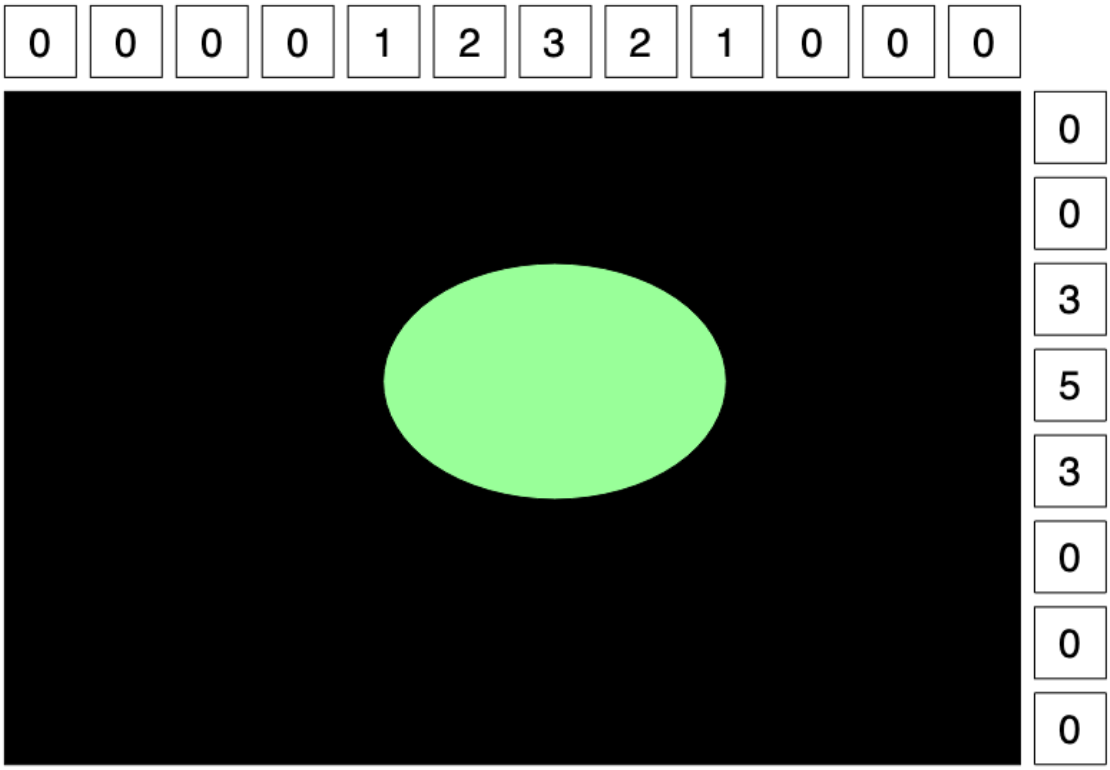

Figure 5.1: CV Algorithm Visualization

the row bins show that the greatest value occurs where the object has its widest point in the frame, which happens at row 4, which is left with a value of 5.

Naturally, in cases intended for more than illustration purposes, the number of bins is set to match the resolution of the image coming in. This results in a very much finer binning arrangement, and produces a very accurate location of the center of a blob.

Strengths and Drawbacks of the Method

It is worth noting that a drawback of this method is that the accuracy will be somewhat lower than traditional blob detection in cases of partial occlusion – where it will center on only the visible portion of the object – and in cases where the object is not round.

For example, in the case of a perfect square, perfectly oriented with sides parallel to the sides of the image frame. The bins for all columns in which the object reside would then hold the same value – ie, the height of the object. Likewise all the bins for all rows in which the object resides would be left with the same value – the width of the object.

As this algorithm has been discussed up to this point, it would choose the last of the equal values as the point to track, which in the case of the square would be the bottom right corner. This results in the primary weakness of this method: dealing with duplicate maximum values.

One solution that was implemented to mitigate this rare but potential issue involves storing a second location point associated with the current maximum. This then allows the two location points to represent the first and the last point at which the maximum was hit, and the two are then averaged. This averaging does not required any division, as the divisor would be 2, requiring only a single bit shift on the result of the adder.

This method has several substantial advantages over traditional blob detection. It does all the processing of any given pixels data once, unlike many methods that must iterate through all detected points in the entire frame and retrace paths to identify convexities. This means that the algorithm can be made without even a single line buffer, which represents a massive improvement over the traditional method of holding the entire image in memory before beginning processing. It also maintains the advantages of the base algorithm, in that it requires no complex operations, such as the division used to locate the center of a blob in software, an operation which is both slow and very expensive in hardware applications. All of these significant strengths allow this method to be implemented with extremely low latency.

As demonstrated and discussed in more detail in Chapter 9, this method can run a full order of magnitude faster than even a hardware accelerated implementation of the traditional blob detection.

5.4 Image Pre-Processing

In order for the location detection to be accurate, it's important to ensure the pixel data is laid out in such a way that the object of interest can be clearly distinguished from any background. This section discusses some additional tricks that make recognition more accurate. Some of the pre-processing techniques that not only lend themselves well to hardware accelerated applications,

but are also very effective at reducing background noise are the Gaussian Blur, morphological erosion.

Both of these methods utilize the kernel concept in Computer Vision, where a small 2D matrix, generally 3x3, is applied as a sliding window over the entire image. As the kernel passes over the image, the value in each index of the kernel is multiplied by the corresponding pixel value in the image.

Gaussian Blur

The Gaussian Blur can be achieved in real time by utilizing a 3x3 kernel as described earlier. The equation for this kernel is as follows:

$$S = 1/16 \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}. \tag{5.7}$$

Where S represents the output value for the current pixel, and the values of the items in the matrix are each multiplied by the original pixel values in coresponding locations in the current position of the sliding window in the input image.

For example, suppose the current region of the image is represented as the following matrix:

$$\begin{bmatrix} A & B & C \\ D & E & F \\ G & H & I \end{bmatrix}. \tag{5.8}$$

The products to be calculated would be:

$$\begin{bmatrix} 1 \cdot A & 2 \cdot B & 1 \cdot C \\ 2 \cdot D & 4 \cdot E & 2 \cdot F \\ 1 \cdot G & 2 \cdot H & 1 \cdot I \end{bmatrix}. \tag{5.9}$$

These products are then summed, producing the sum of products value:

$$1 \cdot A + 2 \cdot B + 1 \cdot C + 2 \cdot D + 4 \cdot E + 2 \cdot F + 1 \cdot G + 2 \cdot H + 1 \cdot I. \tag{5.10}$$

The result of this is then divided by the scaling correction factor, which in the case of a 3x3 kernel is 1/16, as seen in Equation 5.7. As in all cases where division by a constant where the constant divisor is a power of 2, the logical way to implement this in hardware is to perform a bit shift, in this case by 4 bits, which takes zero time in hardware as the wires can be directly mapped.

The result of this scaling reduction is then used as the pixel value in the blurred image at the location of the center entry in the current location of the 3x3 sliding window. Ie, in our example above in Equation 5.8, the output pixel would be processed as being at the same location as E, that is, one less than the current row and one less than the current column.

It's worth noting that the Gaussian Blur can actually be performed on either the pre-thresholded image or on the post-thresholded binary image. In the case of the former, it will produce the same type of colored image that then needs to be binarized. In the case of the latter, the values must either be truncated or else thresholded again, as the coefficients in the Gaussian Blur matrix will produce values other than 0 and 1.

Erosion

The erosion kernel operates slightly differently, and instead of multiplying each item in the sliding window by some coefficient, the erosion operation works by performing a logical AND on all of the bits in the sliding window together. Thus, the center pixel in the output is a 1 if and only if all 9 of the pixels in the binarized image are all 1's.

Since the erosion must be performed on the post-binarized image, we can assume all the values in our current stage of the image are either 0 or 1, and we can treat the erosion operation as a matrix fully populated with 1's:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}. \tag{5.11}$$

Because we know all the pixel values will be single bit binary values, we can perform the operation as if it were a sum of products on the current sliding window, which again is represented as:

$$\begin{bmatrix} A & B & C \\ D & E & F \\ G & H & I \end{bmatrix}. \tag{5.12}$$

Therefore, the result of this sum of products will be:

$$SoP = 1 \cdot A + 1 \cdot B + 1 \cdot C + 1 \cdot D + 1 \cdot E + 1 \cdot F + 1 \cdot G + 1 \cdot H + 1 \cdot I. \tag{5.13}$$

This value is checked against the max, which is $n \cdot m$, where n and m are the dimensions of the kernel. In the case of this 3x3 kernel, the result is that the current pixel is a 1 if and only if the SoP is equal to 9.

Implementation in Hardware

One thing to note is that the addition of either erosion or Gaussian Blur kernels requires the use of two line buffers, resulting a delay equal to the time required to receive two full lines. For the hardware implementation, these 3x3 kernels are applied to a sliding window composed of the the past three pixels in the current line, the last line, and the line before that. In order to accomplish this, the past three lines must be stored in linebuffers, so as to allow the kernel processing module access to those values.

This is done by using 3 three-pixel-wide shift registers, and two one-line wide linebuffers in Block RAMs. Each time a new pixel comes in, it's loaded into the first three-pixel shift register. The pixel exiting the last one in the shift register is pushed into a single-line line buffer of size $width - 3$. The output of this linebuffer then goes into the second three-pixel shift register, which represents the three pixels on the previous line. The output of that shift register is then fed into the second line buffer, also of size $width - 3$, which itself feeds into the final shift register.

Storing the large portions of the line that are not required for the iteration in line buffers, instead of shift registers, allows that major portion of the line buffers to be synthesized as Block RAMs, which alleviates the vast majority of the resource requirements of the line buffers. Only the 9 pixels that are required for the current iteration will be stored in the shift registers, which allow immediate access.

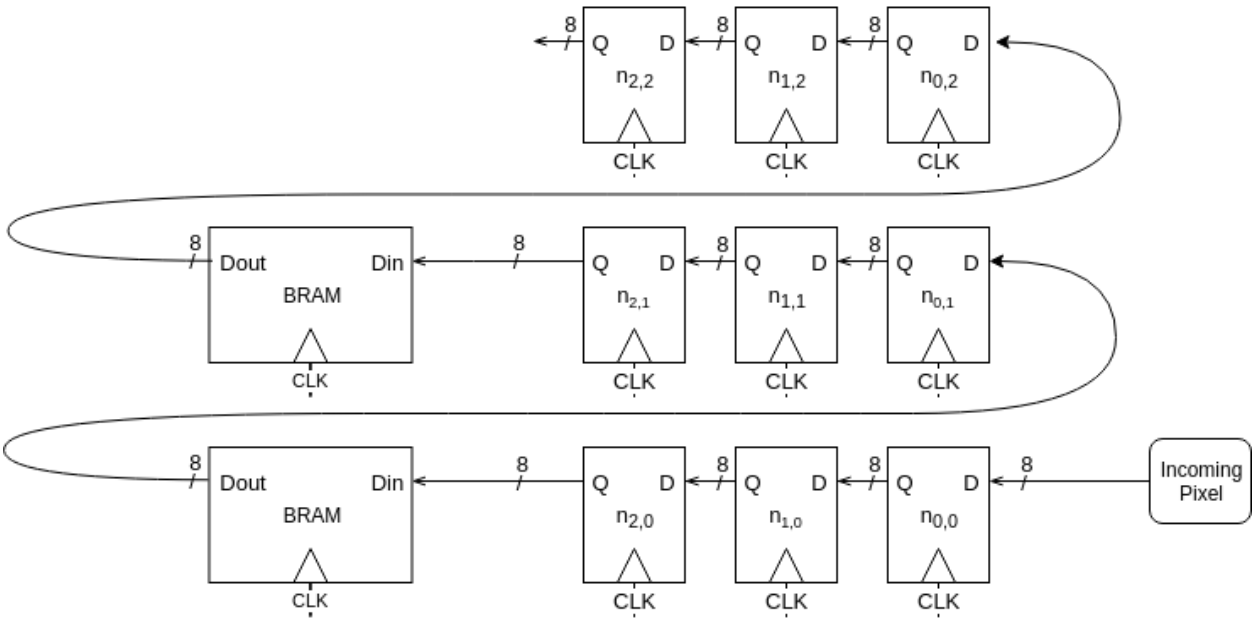

Figure 5.2: BRAM Line Buffer Design

Figure 5.2 illustrates how these shift registers hand off to the BRAMs, which then hand back to the shift registers representing the previous line. Two of these BRAM's are required, along with three sets of shift registers.

CHAPTER 6. SYSTEM DESIGN

This chapter discusses how the system is connected together, and provides an overview of the implementation internal to the FPGA. Resource utilization is also included with each of the lower level component descriptions. The percentage utilization numbers presented in this section are based on the Zynq 7020 Series FPGA. This exact part number is xc7z020-clg484-1.

6.1 System Overview

Figure 6.1 shows the top level overview of the system, as well as the connections between the physical components. Two optional pieces are the Video Output and the Control and Debug I/O. The video output lines are constantly driven, but the pins can be left un-terminated for reduced weight and complexity. The Control and Debug I/O is a standard UART pair, where the TX outputs a data packet each time a frame is completed, and the RX line is used to tune various parameters inside the system. One virtue of this setup is that it allows for remote tuning of the thresholds and gains the recognition and control loop modules.

6.2 FPGA Design Overview

Figure 6.2 shows the top level overview of the portions of the system internal to the FPGA, as well as the connections both between the various internal modules and between those modules and the exterior devices. Several major components themselves contain many smaller modules, such as the Control System and the Computer Vision Recognition System, whose block diagrams will follow.

Table 6.1 shows the overall resource utilization of the entire system, as described in Figure 6.2. Note that the high BRAM usage is due almost entirely to the frame buffer stage, which is used by the video output generator. These blocks are not necessary for operation, and can be left out entirely to further reduce resource usage, as shown in Table 6.2.

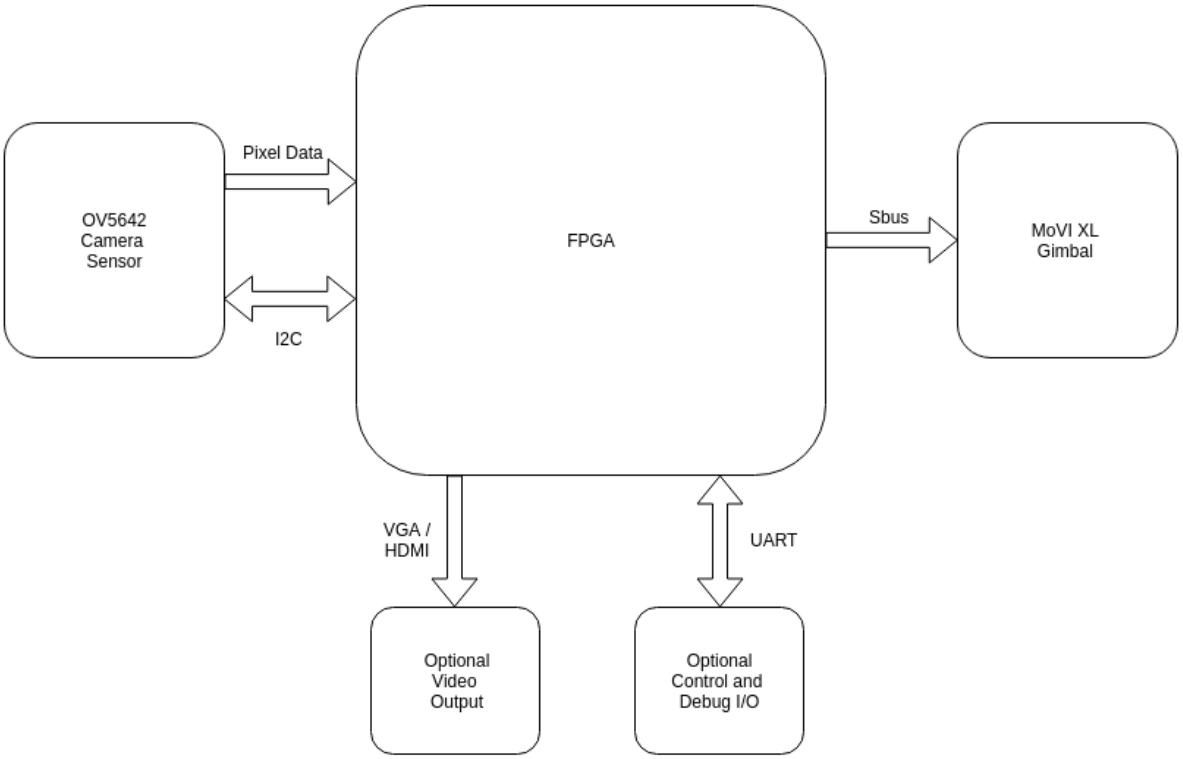

Figure 6.1: High Level Overview of the System Connections.

Table 6.1: Overall resource use of the entire system, including the optional frame buffer and output video generator – not required in operation

Resource	Utilization	Available	Utilization %
LUT	1529	53200	2.87
LUTRAM	203	17400	1.17
FF	886	106400	0.83
BRAM	113	140	80.71
DSP	13	220	5.91
IO	35	200	17.5
BUFG	4	32	12.5
MMCM	1	4	25

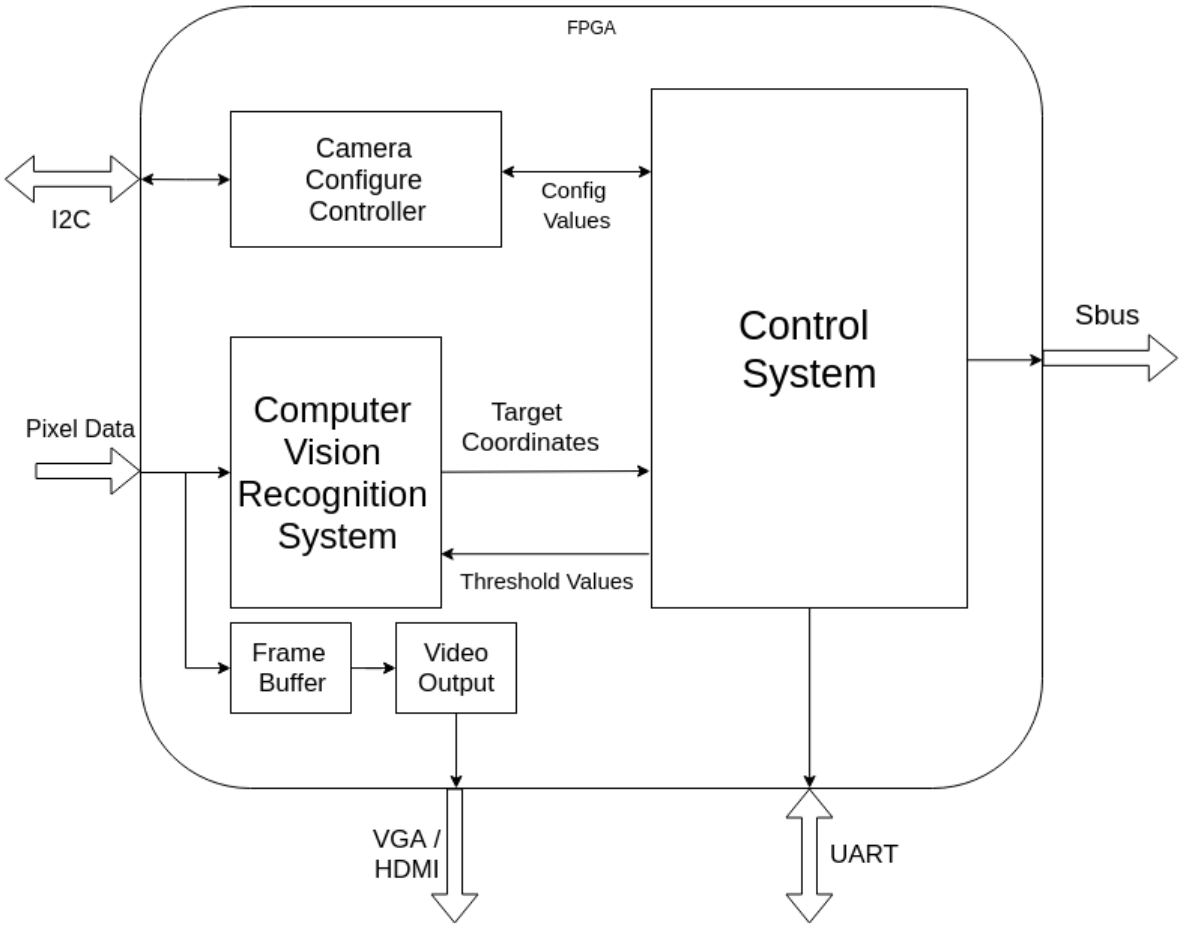

Figure 6.2: High Level Block Diagram of Connections Between FPGA Modules.

Table 6.2: Overall resource use of the entire system without the option frame buffer

Resource	Utilization	Available	Utilization %
LUT	817	53200	1.54
LUTRAM	201	17400	1.16
FF	738	106400	0.69
BRAM	0.5	140	0.36
DSP	8	220	3.64
IO	35	200	17.50
BUFG	4	32	12.50
MMCM	1	4	25.00

Table 6.3: Overall resource use of the additional top level modules in the system, including the optional frame buffer and output video generator – not required in operation

Resource	Utilization		
	Camera Config	Frame Buffer	Video Output Gen
FLOP_LATCH	112	14	115
LUT	105	372	210
CARRY	5	0	23
BMEM	1	114	0
MUXFX	0	130	0
MULT	0	0	4

6.2.1 Vision Recognition System

The Computer Vision Recognition System is composed of a thresholding stage that is discussed in detail in Section 5.2, and a recognition stage that processes the post-thresholded binary pixel data and produces target coordinates based on the same.

The inputs to the computer vision block are the raw camera data outputs from the OV5642 sensor, which are the 8 bits of pixel data, along with a pixel clock, a vertical synchronization pulse and a horizontal reference signal. The remaining lines from the OV5642 are managed by the Control System.

The vertical synchronization pulse is used to determine the end of the frame, and to coordinate when to reset all counters and flags for the next incoming frame. The pixel data is passed immediately into the thresholding stage described in Section 5.2, where it it's converted into the YCbCr colorspace and thresholded to remove any background data and highlight the object to be tracked.

6.2.2 Control System

The Control System itself is composed of a Control Loop, a UART receiver, and a state machine for decoding UART messages and parsing their values into the appropriate places. The primary reason for laying out the control system in this manner was the flexibility born by separating the control loop in its entirety. Keeping this module completely isolated from the rest of the design allows other control loops to be swapped in at will. For example, supposing on another

Table 6.4: Overall resource use of the Vision Recognition System

Resource	Utilization		
	CV System	Thresholding	Detection
FLOP_LATCH	113	16	97
LUT	172	7	165
MUXFX	110	0	110
CARRY	10	0	10
MULT	1	0	1
DMEM	200	0	200

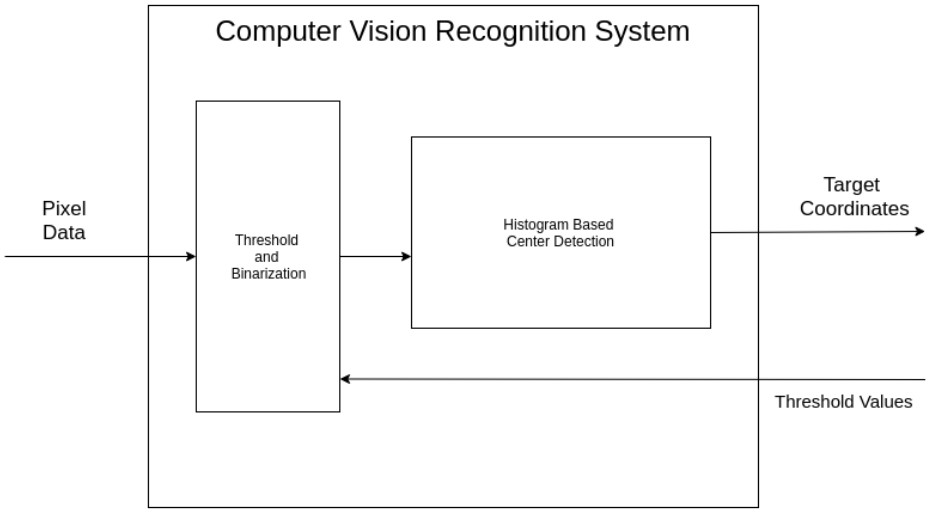

Figure 6.3: Block Diagram of Computer Vision Recognition System Design

gimbal system, a precise model of the gimbal can be approximated. A state space control system would then become very desirable, and the design could be altered to accept it with little to no modification.

Control Loop

The Control Loop currently used in this project is a Proportional-Integral-Derivative (PID) Control loop. PID controllers operate by tracking the current error value, as well as some representation of the previous error values, and use this information to calculate the appropriate control response.

Table 6.5: Overall resource use of the Control System

Resource	Utilization			
	Control System	Control Loop	Input Decoder	UART Rx
FLOP_LATCH	298	67	174	35
LUT	352	256	36	38
CARRY	42	38	0	4
MULT	8	8	0	0

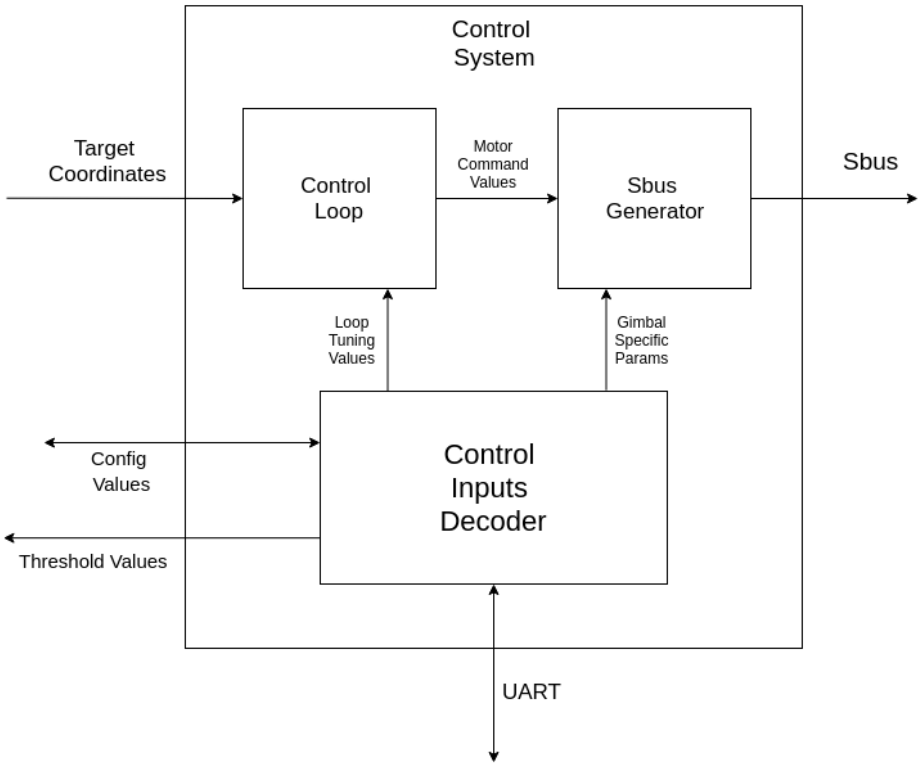

Figure 6.4: Block Diagram of Control System Design

The conventional equation for a PID controller is given in Equation 6.1, where Kp is the proportional gain coefficient, T_i is the integral time constant, and T_d is the derivative time constant. e is the error, that is, the difference between desired position and actual position, and u is the output of the control loop.

$$u(t) = K_p\{e(r) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\frac{de(\tau)}{d\tau}\}. \tag{6.1}$$

Discretizing Equation 6.1 gives:

$$u(k) = K_p\{e(k) + \frac{T}{T_i}\sum_{i=0}^{k-1} e(i) + \frac{T_d}{T}[e(k) - e(k-1)]\}, \tag{6.2}$$

where T is the sampling interval.

Unfortunately, this equation is still rather difficult to implement in hardware. This is due in part to the summation that took place of the integral in Equation 6.1. Hardware accelerated applications often use what is known as the recursive PID algorithm [6]. The recursive PID algorithm is as follows:

$$u(k-1) = K_p\{e(k-1) + \frac{T}{T_i}\sum_{i=0}^{k-2} e(i) + \frac{T_d}{T}[e(k-1) - e(k-2)]. \tag{6.3}$$

If the last control output, that is $u(k-1)$, is known, then the relationship to the current control output then becomes dependant only on the errors from the last three cycles:

$$u(k) - u(k-1) = K_p\{e(k) - e(k-1) + \frac{T}{T_i}e(k-1) + \frac{T_D}{T}[e(k) - 2e(k-1) - e(k-2)]. \tag{6.4}$$

As Equation 6.5 shows, this removes the summation almost entirely, and allows computation of the control output based only on the previous output and the errors from the current cycle as well as the two cycles prior. Solving for the desired output $u(k)$ yields:

$$u(k) = u(k-1) + K_p\{e(k) - e(k-1) + \frac{T}{T_i}e(k-1) + \frac{T_D}{T}[e(k) - 2e(k-1) - e(k-2)]. \tag{6.5}$$

It's worth noting that the two apparent divisions are not divisions at all, but are really just multiplications by a constant. In the first case, $\frac{T}{T_i}$ is simply the sampling interval divided by the integral time constant, neither of which are variable at runtime. The latter case is similar, $\frac{T_D}{T}$ is the derivative time constant divided by the sampling interval, which again is division involving constants for both divisor and dividend. This allows their quotient to be pre-computed and stored as a constant fixed point number. Then, as each sample window arrives, the appropriate error

value is simply scaled by the constant pre-computed number, which can be done with bit shifts and additions – very hardware friendly operations.

CHAPTER 7. AUTOMATED TESTING SYSTEM

As this system is unique in the world, measuring its performance required using an automated testing system to predictably and repeatably provide stimuli. In order to meet this need, a custom actuation platform was designed and built to test the object tracking system. This platform will not necessarily be part of the final rendition of this project, but its usefulness in collecting data has been proven repeatedly over the course of this work, and as such it's only fitting to include a brief discussion of it in the same.

7.1 Design and Fabrication of Testing System

As the closest feasible approximation to the objects to be tracked, a high powered colored laser pointer was selected as a sample target. A high powered green laser with adjustable focus was purchased to serve this role. This provided similarities in that it was a bright object of a known color. In order to generate repeatable test patterns, a two axis actuator was constructed to allow precise control of the direction in which the laser points.

As controlling the pan and tilt axes of the test platform required precise position control, high speed digital servos were the logical choice. The servos chosen are high speed metal-geared servos capable of accelerating from a standstill up to angular speeds of over 600 degrees per second, in under 0.1 seconds.

7.2 Test Patterns

In order to provide several data points to compare response times on, several test patterns were implemented, each designed to test the requested specification of travel speeds around 180 degrees per second. Each of these test patterns reaches those speeds in different circumstances, but all reach it at a minimum of one point in their pattern.

7.2.1 Circles Test Pattern

The *Circles* test pattern moves the laser point in a single large circle at a constant speed. The circle is roughly 90 degrees in diameter, as measured from the perspective of the tracking gimbal. In order to achieve the requested speed of 180 degrees per second, one rotation of this circle pattern is done in roughly 1.5 seconds. This was computed by approximating the overall path distance around the circle as 90π, because the diameter of the field of view of this circle is 90 degrees. Dividing this by the requested speed, 180 degrees per second, gives roughly 1.57 seconds.

7.2.2 Figure Eight Test Pattern

The *Figure Eight* test pattern introduces slightly more complex motions, by moving the tilt servo at twice the rate of the pan servo. This creates a sideways figure eight, and fits where it touches all the edges of the camera sensor's field of view. The width of the figure eight is roughly 110 degrees, and the height is approximately 80, ensuring that much of the figure eight would not be visible from a non tracking point of view.

7.2.3 Sharp Corners Test Pattern

The *Sharp Corners* test pattern introduces what is believed to be a harder situation than this system will be subject to in its final application, but one that provides interesting data nonetheless. This pattern moves the laser as quickly as possible from one corner to the next, in the following order: top right, bottom left, bottom right, top left. Because the servo's are capable of 600 degrees per second, and the M$\bar{o}$VI XL specifications state 200 degrees per second, this creates situations where the pan motion is being pushed as fast as it will respond, as well as times when both pan and tilt are simultaneously being tasked with speed requirements that greatly exceed both the required numbers and the hardware specifications of the gimbal.

7.2.4 Impact Test Pattern

This test pattern is designed to be as close an approximation as is feasible with information that is not classified. This pattern starts out in the top right corner, and then moves down to the

bottom left at a steadily increasing rate of speed, until at the end the servos are moving at full power.

7.3 Remote Control

In order to coordinate this test platform's actions remotely, a wireless message service was setup to send and parse commands to and from this assembly. An ESP8266 board was selected to facilitate this connection. The ESP8266 is a lightweight low-power WiFi-enabled micro-controller that supports most internet protocols. One such protocol that's designed to fit well the needs of this application is Message-Queue Telemetry Transport (MQTT). This protocol operates by having a single host device acting as a 'broker' and other satellite devices that 'subscribe' to pre-defined topics. The broker registers internally which devices have requested notifications on a given topic, and each time a message comes to the broker under the tag of that topic, the message is relayed to all devices that are subscribed to it.

By utilizing this type of messaging service, starting and stopping any given test pattern can be done remotely, and confirmation is sent as a reply when the pattern has completed. This proved to be invaluable in collecting data during these test pattern runs.

To allow further synchronization for data measurements, the gimbal controls were also modified to take advantage of this wireless protocol. By mounting a second ESP8266 board on the gimbal, and tying a pair of UART lines into the FPGA's input controller discussed in Chapter 6, it became possible to remotely activate and tune the gimbal in real time. This allowed for things like power settings, control loop parameters, and even machine vision thresholds to be adjusted while the gimbal was in motion. By coordinating both the gimbal and the test platform on this wireless protocol, it became possible to start a test pattern, and log in real time the measured position of the target in the frame.

CHAPTER 8. MACHINE LEARNING POWERED AUTOMATED TUNING

Often with traditional control loops, tuning is done using the Ziegler Nichols, or Z-N, tuning method. The goal of this method is to minimize response time to a step function, while still preventing large overshoot. This Z-N method is often used, even currently, as it's frequently found to produce the best performance when compared to other tuning methods [14]. Many improvements have been made on this method, such as the Modified Ziegler Nichols method [15], which seeks to further minimize the lag and overshoot response to a step function input.

However, as discovered analytically in this work, it turns out that tuning for a clean response to a step function input does not produce the most accurate solution in many cases. This is particularly evident in applications where minimizing raw error is more essential than a smooth response curve, such as this work's task where the primary goal is maintaining the object of interest center in frame.

To further tighten the raw error between the object to be tracked and the center of frame, thereby allowing an even tighter field of view on the high speed payload cameras, a machine learning automation script was written to allow for rapid custom-tuning of the control loop. This machine learning optimization enables quick and accurate tuning of the control loop parameters to fit more precisely a particular test pattern. This means that future applications can utilize this machine learning tool to custom tailor the control system to meet the exact needs of that particular application. Not only does this reduce the imperfections in the tuning, but it does so fully automatically without requiring any intimate knowledge of the target's path, or even of the gimbal system itself. This is particularly useful in this work, where the classified information regarding these details is hidden from this projects perspective.

8.1 Automated Tuning System Design

As the gimbal controls discussed in Chapter 7 were already wireless, it made sense to take advantage of this already implemented feature in building the automated tuning system.

The automated tuning script operates by first subscribing to the topics where the gimbal is publishing current error data, which is sent over once per frame. This data is then processed to calculate the average error across the entire runtime of the given test pattern.

The script begins by sending the command to the automated testing platform to start a particular test sequence – whichever one is currently being optimized for. Then several counting arrays are initialized: arrays that record the current error on both the pan and tilt axes as the test pattern runs. After the test pattern is finished, as marked by a response from the automated test system, these arrays are averaged to find the mean error value, as well as the maximum instantaneous error. Then the target value-to-be-minimized, generally the average error, is saved as the current best value. Then the machine learning phase begins.

8.2 Simulated Annealing

The machine learning technique that was applied here is known as Simulated Annealing. This algorithm derives its name from a process in metallurgy that improves the ductility of a metal by heating it to a predetermined temperature, generally just above the recrystallization point. The annealing process then involves controlling the cooldown in such as way as to facilitate the movement of atoms into a state of greater equilibrium. As the atoms randomly move, slowly reducing the temperature results in a larger number of atoms that assume a nearly uniform low-energy state. [16]

The simulated annealing algorithm mimics this process by generating a random adjustment to the current design, and testing against the current design's performance. If the randomly generated design performs better, it is accepted as the new current design and the process repeats. When the randomly generated design under-performs the current design, a second random number is generated and compared against the Boltzmann probability factor, which is determined by Equation 8.1 :

$$P = exp(\frac{-\Delta E}{K_b T}). \tag{8.1}$$

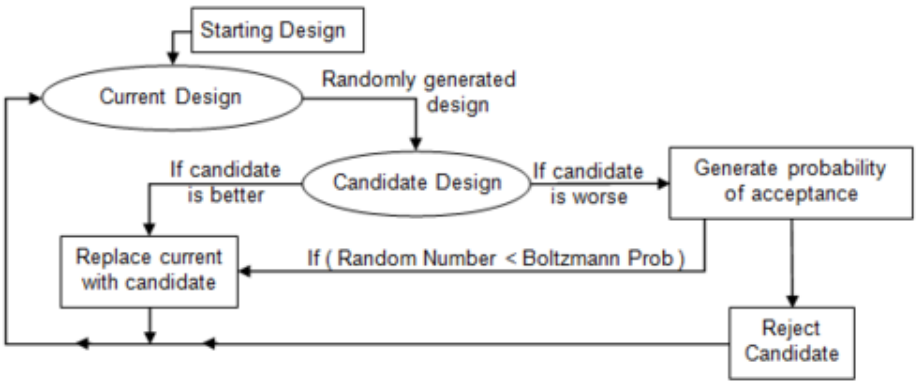

Figure 8.1: Conceptual Flowchart of the Simulated Annealing Optimization Step [17]

Where P is the probability of the inferior design step being accepted, ΔE is the change in performance or energy states between the current and the randomly generated designs, k_b is the Boltzmann constant, and T is a variable called the temperature of the system at the current iteration. As the process iterates, the temperature is slowly reduced, thus lowering the probability that a worse design gets accepted.

Having a high temperature in the early iterations allows this machine learning technique to work its way out of local minima. Reducing the temperature later on ensures that solution works its way toward a solution that has better performance than any neighbors within the random displacement window.

Figure 8.1 shows the decision making process of the simulated annealing algorithm.

8.3 Auto-Tuning Results

The results of this machine learning custom tuning turned out even more impressive than expected. On all of the test patterns, the simulated annealing algorithm was able to produce significantly lower average errors than the traditional tuning methods. These numbers are reported both in pixels and degrees of error.

Table 8.1: Improvements and Results Achieved on the Figure Eights Test Pattern

	Manually Tuned for Step Response	After 50 Iterations of SA
Tilt Error Pixels	25.624	11.989
Pan Error Pixels	25.477	10.483
Tilt Error Degrees	7.207	3.372
Pan Error Degrees	7.165	2.948
Pan Kp	5	13
Pan Kd	20	17
Tilt Kp	9	21
Tilt Kd	25	26

8.3.1 Figure Eight Test Pattern

Table 8.1 shows the quite drastic improvement attained on the *Figure Eight* test pattern. Notice how the average error was reduced by over 50%, 58% on the pan and 53% on the tilt axes, respectively. This brought the average error from roughly 7 degrees down to roughly 3 degrees.

Because the goal of this system is to allow the high-speed payload cameras to use a very narrow field of view lens, these results are extremely promising. For example, going from a 7 degree field of view down to a 3 degree field of view lens means that over 4 times the number of pixels are on the object of interest – twice the resolution-on-target in both the rows and the columns.

8.3.2 Circles Test Pattern

Table 8.2 shows the same data collected for the *Circles* test pattern. This test pattern also showed massive improvement using the simulated annealing automated tuning, with the pan error boasting an incredible 68% reduction over the manual tuning numbers. The tilt accuracy also saw a large improvement, with a 36% reduction in average error. To put that into perspective, the manual tuning's 9.6 degree error on the pan axis would have required a field of view of nearly 10 degrees to capture a reasonable amount of the test pattern motion; with the machine learning improvements, a field of view of only 3.02 degrees would be required to capture the same amount of the target object's path. This means that over 9 times as many pixels would be located on the object of interest, while still maintaining the object in frame as accurately as before.

Table 8.2: Improvements and Results Achieved on the Circles Test Pattern

	Manually Tuned for Step Response	After 50 Iterations of SA
Tilt Error Pixels	13.44	8.582
Pan Error Pixels	34.344	10.742
Tilt Error Degrees	3.780	2.414
Pan Error Degrees	9.659	3.021
Pan Kp	5	19
Pan Kd	20	16
Tilt Kp	9	17
Tilt Kd	25	15

Table 8.3: Improvements and Results Achieved on the Corners Test Pattern

	Manually Tuned for Step Response	After 50 Iterations of SA
Tilt Error Pixels	14.264	12.855
Pan Error Pixels	35.376	34.385
Tilt Error Degrees	4.012	3.615
Pan Error Degrees	9.950	9.671
Pan Kp	5	6
Pan Kd	20	21
Tilt Kp	9	9
Tilt Kd	25	21

8.3.3 Sharp Corners Test Pattern

Table 8.3 shows the same data collected on the *Sharp Corners* test pattern. These results are not nearly as impressive, with only a 9.8% improvement in the tilt average error and even less, 3%, in the pan error. As much as these numbers pale in comparison to the improvements made in the other test patterns, these still represent measurable improvements over the traditional tuning method. Why this particular pattern is more difficult for the simulated annealing to optimize for is beyond the scope of this work, but it's likely that since this test pattern pushes the servos to their max, the mechanical motion of the M$\bar{o}$VI XL is already the bottleneck.

Table 8.4: Improvements and Results Achieved on the Impact Test Pattern

	Manually Tuned for Step Response	After 50 Iterations of SA
Tilt Error Pixels	12.064	6.737
Pan Error Pixels	31.521	19.301
Titlt Error Degrees	3.393	1.895
Pan Error Degrees	8.865	5.428
Pan Kp	5	9
Pan Kd	20	14
Tilt Kp	9	14
Tilt Kd	25	21

8.3.4 Impact Test Pattern

As mentioned in Chapter 7, this test pattern is believed to be the most accurate to the real world application this project is designed for. Table 8.4 shows the numerical improvement achieved by the simulated annealing algorithm on the *Impact* test pattern.

The average error in the pan axis saw a 38% reduction by using the simulated annealing to fine tune the control loop for this test pattern. Even more impressive, the average tilt error was brought down over 44% to a minuscule 1.9 degrees average error.

CHAPTER 9. PERFORMANCE COMPARISON

This machine vision algorithm implemented in this Thesis is extremely well suited for hardware acceleration, and is capable of running at impressive speeds. In order to determine the maximum speeds at which the algorithm itself can operate, the vision processing algorithm was synthesized and implemented expecting various clock rates and resolutions.

In all numbers reported here, timing is being met for all metrics, setup time, hold time and pulse width requirements. For reference, these synthesis runs were done using the Xilinx Vivado Default strategies for Synthesis and Implementation, with the target device being the Zynq-7000 series System-on-Chip.

9.1 Frame Rate Comparison

In high speed real-time processing, one important metric is the maximum frame rate at which incoming video can be processed. The algorithm presented here was demonstrated both in simulation and in an implemented Vivado design to run with a pixel clock rate of 666Mhz. This allows a maximum frame rate of roughly 2170 frames per second, assuming the resolution of 640x480 used in this Thesis. While the camera sensor used in this project is not capable of speeds that high, this means that the algorithm implemented in this Thesis could support frame rates over 2100 fps if paired with a faster camera.

9.1.1 Hardware Acceleration of BLOB Detection for Image Processing

A more traditional approach to blob detection is demonstrated by Bochem, Herpers and Kent in their work Hardware Acceleration of BLOB Detection for Image Processing [18]. In this work, they implement blob detection in FPGA hardware, and provide results for their design's speed and frame rates.

Table 9.1: FPGA Accelerated Blob Detection Frame Rate Comparison of Traditional Blob Detection Implementations to the Algorithm Presented in this Thesis

Resolution		Frames Per Second		
x	y	Bochem, et al [18]	Alabdo, et al [7]	This Algorithm
640	480	61.39	-	2170.14
800	800	47.34	-	1041.67
1024	768	28.9	-	847.71
752	582	-	340	1521.7

Table 9.1 shows the results from [18] at the three resolutions reported in their paper, side-by-side with the results of this algorithm synthesized for the same resolutions. As can clearly be seen, this algorithm is shown running at approximately 30 times faster than traditional blob detection.

9.1.2 FPGA-Based Architecture for Direct Visual Control Robotic Systems

One of the related works that most closely matched this Thesis is the work by Aiman Alabdo, Javier Prez, Gabriel J. Garcia, Jorge Pomares, and Fernando Torres at the University of Alicante, Spain. In their work, *FPGA-based architecture for direct visual control robotic systems* [7], they developed a visual servoing system that utilized FPGA vision processing. This system was demonstrated able to track a white dot on a solid black surface.

To accomplish this, they implemented traditional center-of-mass blob detection, which took only a single-channel gray-scale image. They performed all the vision processing and actuator control inside the FPGA, which was a 7-series Kintex chip, XC7K325T-2. They were able to achieve a framerate of 340 frames per second on a resolution of 752582.

As can be seen in Table 9.1, this is also substantially slower than the algorithm presented in this Thesis, despite the fact that Alabdo, et al [7] leveraged a newer and larger chip, the Kintex-7 XC7K325T-2. The Alabdo, et al design used over 80,000 LUTs, which simply would not fit on the Zynq chip used in this Thesis, which only has 53,000 LUTs available.

It is also worth noting that this Thesis implemented color tracking, using YCbCr to track a colored dot over a wide variety of backgrounds, whereas Alabdo, et al. only implemented gray-scale tracking of a white object on a solid black background. Despite this simplified task and

larger FPGA, the traditional method of blob detection is still only able to achieve less than 1/4 the frame-rates of the algorithm implemented in this Thesis.

Unfortunately, their accuracy was only reported in pixels, which makes it difficult to compare without knowing the exact dimensions of their field of view. However, they do state that they were able to achieve an average error of only 3.1 pixels on their test pattern, when it moved with a max speed of 2 radians per second, or approximately 114 degrees per second [7]. This is an impressive number, but without knowing their field of view, it's impossible to compare accurately to this Thesis method, particularly as this project requires much higher speeds, and utilizes test patterns running at 230 degrees per second.

CHAPTER 10. ADDITIONAL APPLICATIONS AND FUTURE WORKS

10.1 Additional Applications

The low latency blob detection and vision processing algorithm developed in this work lend themselves well to a variety of embedded applications. Here are a few potential applications that have connections with the Robotic Vision Laboratory at Brigham Young University.

The PixelLight is an electronically controllable headlight system that leverages multiple LED's strategically located such that each illuminates a vertical bar on the road ahead. Then, as oncoming traffic is detected, the lights can intelligently turn off on the one specific slot where the oncoming vehicle is located in order to reduce glare and improve visibility for the oncoming drivers.

Thus far, the system uses a camera sensor placed elsewhere on the front of the car, and has a separate embedded processor to determine the location of oncoming traffic in the frame, which is then passed to the headlights to blank the corresponding regions. Figure 10.1 shows the PixelLight Smart Headlamp System, and Figure 10.2 shows the system in operation, illustrating how a small vertical bar would be blanked to avoid blinding the oncoming driver.

The blob detection algorithm presented in this work would fit very well in such an application. Using this work's vision processing system here would allow the camera to be relocated to inside the headlamp assembly, which not only minimizes connections to be tracked, but also makes the entire system fit into a single product. Similarly, as the FPGA would then be doing all the processing, there's no longer any need for the remote processor. This means that the headlamp assembly could be manufactured as a single unit, built to meet the dimensions of stock headlamps that came with mass produced vehicles. The product could then be sold as a drop-in replacement for standard headlamps.

Figure 10.1: The PixelLight Smart Headlamp

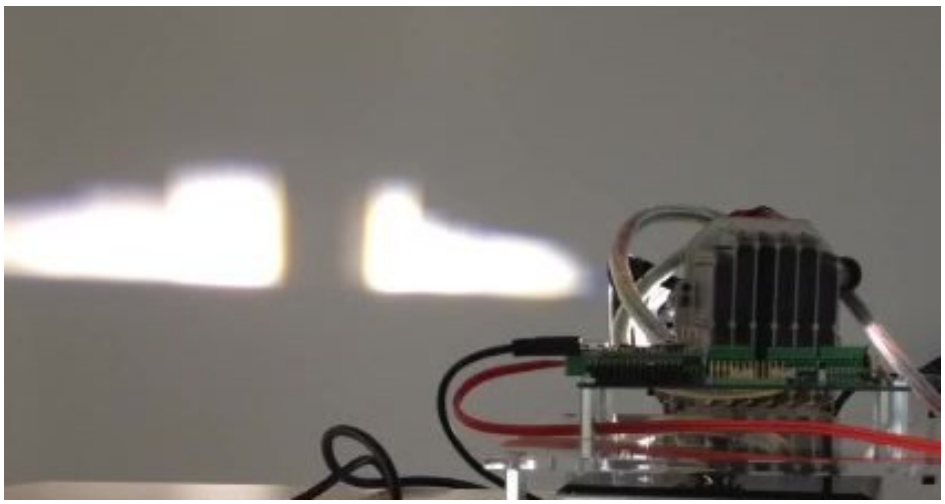

Figure 10.2: The PixelLight Smart Headlamp in Operation

This would drastically reduce the amount of work required to retrofit an existing vehicle with the PixelLight smart headlamps. And as an added benefit, because the cameras would be mounted together with the lights, this would also remove any need for post-installation alignment.

10.2 Future Improvements to the System

One logical course for future extensions of this project could be to employ cameras with support for higher frame rates, which the Machine Vision algorithm would readily support. This would bring a reduction in overall latency, that would make the control system even more accurate.

Another potential future improvement would be to use mechanical hardware with faster actuation and lower response time. Speeding up the response time of the mechanical components could further improve the accuracy of the overall system. A similar improvement would be seen by designing a custom gimbal system where the motor controllers could be directly manipulated by the FPGA, as this would reduce any lag in the current system that stems from time the M$\bar{o}$VI XL takes in processing the S.Bus commands.

If even lower latency is required, then entire gimbal-and-high speed camera could be replaced with a fixed ultra high-resolution camera sensor with electronically selectable real-time region of interest cropping. The FPGA could then process a low resolution frame to locate the object and request a small region around it at higher resolution.

CHAPTER 11. CONCLUSION

This work details the design and construction of an ultra low latency object tracking system. Using a high powered gimbal like the M$\bar{o}$VI XL paired with a low latency FPGA vision processing algorithm allows for extremely short response time and very accurate object tracking.

Using hardware friendly recursive PID loop algorithms can produce very rapid response times, and substantial further improvements can be achieved by leveraging machine learning techniques like simulated annealing. Perhaps the most surprising finding of this work was that the margin of error can be cut in half in some applications by utilizing machine learning automated tuning.

This work centered around a real time blob detection algorithm, which was not only achieved theoretically, but also fully implemented in hardware. This blob detection was employed as the backbone for an ultra low latency tracking platform, which was demonstrated capable of tightly tracking an object moving at over 180 degrees per second.

REFERENCES

[1] K. Hashimoto, *Visual Servoing*, ser. World Scientific series in robotics and automated systems. World Scientific, 1993. [Online]. Available: https://books.google.com/books?id=uMOmoFOPLxUC 3

[2] F. Chaumette and S. Hutchinson, "Visual servo control. i. basic approaches," *IEEE Robotics Automation Magazine*, vol. 13, no. 4, pp. 82–90, Dec 2006. 3

[3] H. J. and W. Park, "Real time control of a robot with a mobile camera," *Proc. 9th ISIR*, pp. 233–246, 1979. 3

[4] C. Collewet and E. Marchand, "Photometric visual servoing," *IEEE Transactions on Robotics*, vol. 27, no. 4, pp. 828–834, 2011, cited By :67. [Online]. Available: www.scopus.com 4

[5] A. Bochem, R. Herpers, and K. B. Kent, "Fpga based real-time object detection approach with validation of precision and performance," *22nd IEEE International Symposium on Rapid System Prototyping*, 2011. 4

[6] M. Kocur, S. Kozak, and B. Dvorscak, "Design and implementation of fpga - digital based pid controller," in *Proceedings of the 2014 15th International Carpathian Control Conference (ICCC)*, May 2014, pp. 233–236. 5, 37

[7] A. Alabdo, J. Prez, G. Garcia, J. Pomares, and F. Torres, "Fpga-based architecture for direct visual control robotic systems," *Mechatronics, 39*, 2016. 5, 49, 50

[8] Wikipedia, the free encyclopedia, "Hsv color solid cylinder," 2019, [Online; accessed April 29, 2019, By SharkDderivative work SharkD]. [Online]. Available: https://commons.wikimedia.org/w/index.php?curid=9801673 9

[9] Wikipedia, the free encyclopedia, By Tonyle - Own work, "Example of u-v color plane, y value = 0.5, represented within rgb color gamut," 2019, [Online; accessed April 29, 2019]. [Online]. Available: https://commons.wikimedia.org/w/index.php?curid=6977944 11

[10] RobotShop, "Ov5642," 2019, [Online; accessed May 7, 2019]. [Online]. Available: https://www.robotshop.com/en/arducam-5-mp-camera-module-ov5642-cs-mount-lens.html?gclid=EAIaIQobChMI6aHhvZCK4gIVBtVkCh01pQNOEAQYAiABEgK8C_D_BwE 13

[11] BH Photo Video, "Mvi xl," 2019, [Online; accessed May 7, 2019]. [Online]. Available: https://static.bhphoto.com/images/images750x750/1491240145000_1330379.jpg 14

[12] Trenz Electronics, "Genesys 2," 2019, [Online; accessed May 7, 2019]. [Online]. Available: https://shop.trenz-electronic.de/media/image/1c/3a/e7/26992_0.jpg 15

[13] Numato Website, "Styx," 2019, [Online; accessed April 29, 2019]. [Online]. Available: https://numato.com/docs/styx-zynq-module/ 16

[14] A. A. Azman, M. H. F. Rahiman, N. N. Mohammad, M. H. Marzaki, M. N. Taib, and M. F. Ali, "Modeling and comparative study of pid ziegler nichols (zn) and cohen-coon (cc) tuning method for multi-tube aluminum sulphate water filter (mtas)," in *2017 IEEE 2nd International Conference on Automatic Control and Intelligent Systems (I2CACIS)*, Oct 2017, pp. 25–30. 42

[15] P. M. Meshram and R. G. Kanojiya, "Tuning of pid controller using ziegler-nichols method for speed control of dc motor," in *IEEE-International Conference On Advances In Engineering, Science And Management (ICAESM -2012)*, March 2012, pp. 117–122. 42

[16] E. Britannica, "Annealing," 2011, [Online; accessed May 16, 2019; published September 25, 2011]. [Online]. Available: https://www.britannica.com/science/annealing-heat-treatment 43

[17] J. D. Hedengren, "Design optimization: Simulated annealing tutorial," 2019, [Online; accessed May 16, 2019]. [Online]. Available: http://apmonitor.com/me575/index.php/Main/SimulatedAnnealing 44

[18] A. Bochem, R. Herpers, and K. B. Kent, "Hardware acceleration of blob detection for image processing," in *2010 Third International Conference on Advances in Circuits, Electronics and Micro-electronics*, July 2010, pp. 28–33. 48, 49