ACTIVE SELF-LANDMARKING TO MOBILE ROBOT CAMERA CALIBRATION
THROUGH WIRELESS COMMUNICATIONS

A Thesis
Submitted to the Faculty
of
Purdue University
by
Glenn Harden

In Partial Fulfillment of the
Requirements for the Degree
of
Master of Science in Engineering

May 2013
Purdue University
Fort Wayne, Indiana
ACKNOWLEDGMENTS

The author would like to thank the National Science Foundation for providing funding for this project, the Indiana Space Grant Consortium for providing the base robotic platforms, and the IPFW Wireless Technology Center for funding all other hardware components of this project. The author would also like to thank Dr. Liu, Dr. Pomalaza-Ráez, Dr. Thompson, and Dr. Wang for the valuable advice and the (probably undeserved) patience throughout this process. The author would also like to thank some friends who provided much needed inspiration, motivation, and, on a couple of occasions, assistance (in the form of soldering, metal fabrication, or an extra set of hands to move equipment around during testing). Thank you all very much.
TABLE OF CONTENTS

LIST OF TABLES .............................................................................................................. v

LIST OF FIGURES ......................................................................................................... vi

ABSTRACT ...................................................................................................................... ix

1. INTRODUCTION ....................................................................................................... 1

1.1 Active Landmarking .............................................................................................. 2
1.2 Robotic Platform ................................................................................................... 3
1.3 Calibration Model ................................................................................................... 6

2. ROBOTIC PLATFORM ............................................................................................. 7

2.1 Base Robotic Platform .......................................................................................... 7
2.2 Active Landmarking Subsystem ............................................................................ 9
2.3 Wireless Communication Subsystem .................................................................. 11
2.4 Power Supply Subsystem .................................................................................... 12
2.5 Camera Subsystem ............................................................................................... 13
  2.5.1 Initial camera design ..................................................................................... 14
  2.5.2 Custom camera design .................................................................................. 15
  2.5.3 Camera integration ......................................................................................... 18
  2.5.4 Cameraboard testing ..................................................................................... 18
  2.5.5 Cameraboard debugging ............................................................................... 19
  2.5.6 FIFO debugging ............................................................................................. 20
  2.5.7 Camera testing ............................................................................................... 23
  2.5.8 Temporary camera design ............................................................................. 27
  2.5.9 Revisiting the initial camera design ............................................................... 28
  2.5.10 New camera design ...................................................................................... 29

3. CAMERA CALIBRATION ALGORITHM ................................................................ 31

3.1 Active Landmarking .............................................................................................. 32
3.2 Blob Processing ..................................................................................................... 32
  3.2.1 Binary image generation ............................................................................... 33
  3.2.2 Slab and blob generation ............................................................................... 33
### LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Uninitialized output data stored in FIFO</td>
<td>22</td>
</tr>
<tr>
<td>2.2</td>
<td>Showing the significance of each data bit</td>
<td>26</td>
</tr>
<tr>
<td>5.1</td>
<td>Landmark locations and corresponding image coordinates</td>
<td>61</td>
</tr>
<tr>
<td>5.2</td>
<td>Intrinsic values for the given sets of data from the first test</td>
<td>68</td>
</tr>
<tr>
<td>5.3</td>
<td>Landmark locations and corresponding image coordinates</td>
<td>69</td>
</tr>
<tr>
<td>5.4</td>
<td>Intrinsic values for the given sets of data from the second test</td>
<td>70</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>The base robotic platform, DaNI</td>
<td>5</td>
</tr>
<tr>
<td>2.1</td>
<td>Completed robotic platform</td>
<td>7</td>
</tr>
<tr>
<td>2.2</td>
<td>Example of LabVIEW programming</td>
<td>8</td>
</tr>
<tr>
<td>2.3</td>
<td>The LED matrix from Seeedstudio, Inc</td>
<td>9</td>
</tr>
<tr>
<td>2.4</td>
<td>The circuit diagram of the LED tower circuit board</td>
<td>10</td>
</tr>
<tr>
<td>2.5</td>
<td>An image of the assembled LED tower circuit board</td>
<td>10</td>
</tr>
<tr>
<td>2.6</td>
<td>The Linksys WRT160n wireless router</td>
<td>12</td>
</tr>
<tr>
<td>2.7</td>
<td>Block diagram of the design of the system. The image data follows the red arrows</td>
<td>13</td>
</tr>
<tr>
<td>2.8</td>
<td>Axis Communications M1011-W Ethernet Camera</td>
<td>14</td>
</tr>
<tr>
<td>2.9</td>
<td>Bytes sent by the camera and their associated color bits (e.g. B4 is the most significant bit of the blue color component)</td>
<td>15</td>
</tr>
<tr>
<td>2.10</td>
<td>Image of the assembled cameraboard</td>
<td>16</td>
</tr>
<tr>
<td>2.11</td>
<td>Block diagram of the cameraboard</td>
<td>17</td>
</tr>
<tr>
<td>2.12</td>
<td>Schematic of the cameraboard</td>
<td>17</td>
</tr>
<tr>
<td>2.13</td>
<td>Using an Arduino Uno as a latch to test the output from the camera</td>
<td>19</td>
</tr>
<tr>
<td>2.14</td>
<td>Breadboard with breakout board for testing FIFO</td>
<td>20</td>
</tr>
<tr>
<td>2.15</td>
<td>Schematic of the redesigned cameraboard</td>
<td>23</td>
</tr>
<tr>
<td>2.16</td>
<td>Example ‘yellow noise’ picture obtained from CMUCam3</td>
<td>25</td>
</tr>
<tr>
<td>2.17</td>
<td>LogicSniffer graph with the lens unobstructed</td>
<td>26</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>2.18</td>
<td>LogicSniffer graph with the lens obstructed by a finger</td>
<td>27</td>
</tr>
<tr>
<td>3.1</td>
<td>Axes of unified coordinate system in relation to the robot (viewed from the front)</td>
<td>32</td>
</tr>
<tr>
<td>3.2</td>
<td>Model of camera and image plane</td>
<td>35</td>
</tr>
<tr>
<td>5.1</td>
<td>Results from first WDE experiment (routers on the floor)</td>
<td>50</td>
</tr>
<tr>
<td>5.2</td>
<td>Results from second WDE experiment (routers 23.5” off floor)</td>
<td>51</td>
</tr>
<tr>
<td>5.3</td>
<td>Image with active landmark off, using brightness correction</td>
<td>53</td>
</tr>
<tr>
<td>5.4</td>
<td>Image with active landmark on, using brightness correction</td>
<td>53</td>
</tr>
<tr>
<td>5.5</td>
<td>Binary image with brightness correction turned on</td>
<td>54</td>
</tr>
<tr>
<td>5.6</td>
<td>Standard repeater bridge mode</td>
<td>55</td>
</tr>
<tr>
<td>5.7</td>
<td>Original robotic communication testing</td>
<td>55</td>
</tr>
<tr>
<td>5.8</td>
<td>Network setup in experiment</td>
<td>55</td>
</tr>
<tr>
<td>5.9</td>
<td>Arduino and robot in client bridge setup</td>
<td>56</td>
</tr>
<tr>
<td>5.10</td>
<td>Arduino and computer in client bridge setup</td>
<td>56</td>
</tr>
<tr>
<td>5.11</td>
<td>Arduino and computer in a single router setup</td>
<td>56</td>
</tr>
<tr>
<td>5.12</td>
<td>Arduino and computer connected to same router with standard secondary router</td>
<td>57</td>
</tr>
<tr>
<td>5.13</td>
<td>Arduino and computer in wired network setup</td>
<td>57</td>
</tr>
<tr>
<td>5.14</td>
<td>Robots using a wired connection between the routers</td>
<td>58</td>
</tr>
<tr>
<td>5.15</td>
<td>Image in gymnasium showing reflection of active landmark off floor</td>
<td>59</td>
</tr>
<tr>
<td>5.16</td>
<td>Binary image showing reflection of landmark off of gymnasium floor</td>
<td>59</td>
</tr>
<tr>
<td>5.17</td>
<td>Image from Fig. 5.15 modified to remove reflection of active landmark</td>
<td>60</td>
</tr>
<tr>
<td>5.18</td>
<td>Resulting binary image using modified image</td>
<td>60</td>
</tr>
<tr>
<td>5.19</td>
<td>White dot showing the location of the center point of the image</td>
<td>62</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>5.20</td>
<td>Red dot showing the location of the primary axis on the image</td>
<td>63</td>
</tr>
<tr>
<td>5.21</td>
<td>The locations of the primary axis (red dot) and the location of the center point of the image given by the camera calibration algorithm (white dot)</td>
<td>63</td>
</tr>
<tr>
<td>5.22</td>
<td>Showing the black square used to determine the actual camera aspect ratio</td>
<td>64</td>
</tr>
<tr>
<td>5.23</td>
<td>The red dots show the locations of the first 10 landmarks, and the white box shows the range of the landmarks</td>
<td>65</td>
</tr>
<tr>
<td>5.24</td>
<td>Showing the primary axis (red) and the center point given from the camera calibration algorithm (white)</td>
<td>66</td>
</tr>
<tr>
<td>5.25</td>
<td>Plot of all 19 landmark locations (red dots) and the range in which they are found (white rectangle)</td>
<td>67</td>
</tr>
<tr>
<td>5.26</td>
<td>Relative error using data from original test</td>
<td>68</td>
</tr>
<tr>
<td>5.27</td>
<td>Diagram of landmark locations in second test</td>
<td>69</td>
</tr>
<tr>
<td>5.28</td>
<td>Relative error using data from second test</td>
<td>70</td>
</tr>
<tr>
<td>5.29</td>
<td>Camera Calibration Algorithm calculated center point (white) and the primary axis of the robot (red)</td>
<td>71</td>
</tr>
</tbody>
</table>
ABSTRACT


Mobile robotic platforms are becoming more and more popular, both in scientific research and in commercial settings. Robotic systems are useful for going places or performing tasks that are not suitable for humans to do. Robots are often able to precisely perform complicated or dangerous tasks with little or no human involvement. However, before a mobile robotic platform is able to be deployed, it must have a way of identifying where it is in relation to objects and obstacles around it. Often, this is being performed by using a visual system, such as a camera.

However, just wiring a camera onto a robot is not sufficient. With many of the tasks that are given to a robotic system to perform, a great deal of precision is required to satisfactorily complete these tasks. This precision requires that the robot be given accurate information by the camera.

Most cameras have minor imperfections, even though they are within the manufacturer’s tolerances. Some of these imperfections can cause the aspect of the image to be slightly distorted, causing a perfectly square object to appear to be slightly rectangular. This aspect can be corrected with an appropriate scaling factor, reducing or enlarging the size of the image in the horizontal or vertical direction.

Other imperfections in the camera manufacturing process can cause the image sensor to not be exactly aligned with the actual camera housing. This can cause the center point of the image to be slightly off of the center of the image sensor (e.g., the image could be shifted to the left by a few pixels, so the center of the image is actually at 317, 240 instead of 320,240).
This project will implement a camera calibration algorithm that was developed in [1] on mobile robotic platforms to autonomously determine the center point and correct scaling factors for the platform’s respective cameras.

These mobile robotic platforms were chosen to be the National Instruments Robotics Starter Kit DaNI. They were programmed using National Instruments graphical programming environment LabVIEW. A robotic platform is to determine the center point and scaling factor of its own camera, using other identical robotic platforms to perform this calibration. The robots can then trade responsibilities, and a different robot can perform the required calibration. This project will also attempt to use wireless communication between the robots, allowing cooperative behavior. This project will also attempt to use the wireless signal strength to determine the locations of the mobile robotic platforms relative to one another.
1. INTRODUCTION

One of the first and most important issues with mobile robotic platforms is self-localization. Robotic platforms need to be able to identify where they are in relation to everything else in the environment, such as walls, desks, other obstacles, and even other robotic platforms. Many robotic platforms are starting to perform this localization by using cameras via image processing. Two different ways of the camera mounting have been used for performing this localization.

The first method is to use cameras mounted in the environment, looking at the mobile robots. A central node then identifies the robotic platforms and communicates to the robotic platforms their respective positions in the world coordinate system. This method is very effective, but only in situations where the environment is well-known and has cameras placed in it at known locations. The entire environment must have camera coverage, usually with multiple cameras covering each position in the environment that a robot could possibly be. This requires a large number of cameras, and a great deal of processing power to be able to perform the image processing for each camera at the same time.

The second method is to use cameras mounted on the robotic platforms. These platforms would then need to identify their locations without the help of any sort of external vision system. There would need to be at least one camera per robot, though the environment would not need to remain constrained to a particular area. With the cameras mounted on the robot, the environment that has image coverage is anywhere that the robot is located. In the first method, the environment is fixed, but in this method, the environment is dynamic.
The first step to self-localization using any sort of image processing is to calibrate the camera that will be used to obtain the images. If the camera is incorrectly calibrated, the data retrieved by the platform is nearly useless.

In the static-camera model, each camera would need to be calibrated for the internal parameters of the camera. These parameters include the focal length, the pixel scaling in the horizontal and vertical direction, and the position of the image center in the image. In the mobile-camera model, each camera would need to be calibrated for these same internal parameters, but also for some external parameters, such as the location and orientation of the camera in the world coordinate system. These external parameters are usually known when the static-camera model is set up, and don’t change throughout the lifetime of the system, so they are already defined. However, in mobile robotic platforms, these parameters, even if they are known at the start of the system, quickly change.

The static-camera model works well for many instances in which the environment is suitable to add cameras. However, in many instances, it is not possible to set up these cameras in the active environment, so the primary focus of this research is on the mobile-camera model. In addition, in many instances which are not appropriate for the static-camera model, knowledge of the robotic platform in relation to the world coordinate system is not required, only a unified coordinate system shared among the mobile robotic platforms. The unified coordinate system can then be assumed to be a single robot’s coordinate system and communicating that system to any additional robotic platforms in the area.

One primary method of calibration uses landmarking. Landmarking usually involves mobile robots locating objects in the environment that can be easily identified. Another type of landmarking is called active landmarking. This project will attempt to use robot-mounted active landmarks and cameras, as well as wireless communication to dynamically create landmarks for other robotic platforms and perform camera calibration.

1.1 Active Landmarking

There are two primary ways of using landmarking in calibrating a camera. The first is to use a test image, usually in a laboratory setting. This test image is usually a
series of well-defined and easily identifiable shapes or objects. The image is placed in front of the camera in a particular position and orientation. This type of landmarking is known as passive landmarking. The other is known as active landmarking. Active landmarking is performed by changing the environment in which the camera already exists. This can be performed by using any sort of visual signal that can be activated and deactivated.

In this project, the second method is being used to calibrate the cameras attached to the robots. Each robot has a camera, and each robot has an active landmark. While a robot is performing the camera calibration, it is known as a calibrating robot. Any robot that is being used as an active landmark for a calibrating robot is known as a landmarking robot. These landmarking robots can move around, or a team of landmarking robots can be distributed throughout the environment, in order to provide multiple data points for the camera calibration of the calibrating robot. It should be noted that either before or after a robot has had its camera calibrated, it can be used as a landmarking robot for another calibrating robot. In practice, every robot should have its camera calibrated (i.e., take a turn as a calibrating robot), and every robot can be used as a landmarking robot (i.e., be used as a landmarking robot).

1.2 Robotic Platform

For the past few decades, autonomous mobile robots have transited from using large and expensive computing systems to inexpensive and small embedded computing systems which are carried on-board [2]. Such embedded robotic systems have been popularly used in teaching [3][4] and research [5][6]. The rapid advancement in embedded systems also opens great possibilities for building affordable, yet flexible, experimental robotic platforms [7] which allow researchers to conduct a variety of research in robotics and wireless communication. Such robotic networks should be able to communicate with each other through wireless links. It should also be equipped with cameras to aid in localization and navigation [1].

The first step to build such a network is to find a robotic base that has enough hardware power to perform in various research areas, and also provides a powerful
software development environment. After careful investigation and comparison, the National Instruments robotics starter kit (DaNI) was chosen. DaNI, shown in Fig. 1.1, is a platform designed for teaching, research, and prototyping [4]. This robotics kit is equipped with a sbRIO-9631 Single-Board RIO (Reconfigurable I/O Architecture), which combines a real-time processor, reconfigurable field programmable gate array (FPGA), and analog and digital I/O on a single board, which is programmed using NI LabVIEW software. This robotic platform has integrated batteries, motors, and four wheels. There are 110 digital I/O data pins and 32 analog I/O data pins. Each platform also has an Ethernet port and an RS-232 serial port.

Similar systems have been developed to include a wireless router and IP camera [8]. The IP camera has the ability to transfer the image through the internet. However, the camera wouldn’t allow the user to access the raw image data, but would only allow access through HTTP connections, which the robot only natively supported using a browser object. The browser object only would display a web browser on a computer screen, and not allow the robot access to the data that was contained within. Therefore, these types of systems can only be used in certain applications, and show strong restrictions in development work. In this project, the camera integrated to the robot base is able to provide the raw image to the robot for processing. To the best of my knowledge no other institution operates this robotic platform without human interaction.

The development process of the robotic platform/test-bed presented in this thesis has gone through many iterations with different cameras. Every camera that was used had at least one problem communicating with the robotic platform, usually being that the platform could not keep up with the incoming data, or that the platform would ignore the data being sent out. All of these iterations are described in detail in section 2.5.

These problems were further compounded by the programming environment that needed to be used in order to program the robot. LabVIEW is a graphical dataflow-type programming environment. It came pre-loaded with several functions that simplify the operation of the platform (e.g., movement, ultrasonic rangefinder/path planning). It also came pre-loaded with some communication protocols, such as TCP and UDP protocols, but didn’t have some others, such as I²C. In addition, many times in LabVIEW, one type
of object is returned by a function, but that object can only be viewed on the computer screen, and cannot be accessed in order to perform any sort of processing. (e.g. the web browser object only being able to display the image on the screen, but not return the image data in a meaningful way to the robot). These types of objects do not lend themselves to autonomous types of systems. In some cases, functions that are already “built-in” to LabVIEW needed to be rewritten in order to make them useful in an embedded robotic system. There were other problems with the programming environment, including an apparent lack of memory management capabilities, complicated user interfaces, and very little documentation that could be accessed and understood by a new user of LabVIEW.

Despite the hardware and software problems with the base robotic platform, this project provides a testbed for researchers in robotics to conduct experiments in localization and navigation through LabVIEW.

Fig. 1.1. The base robotic platform, DaNI.
1.3 Calibration Model

The image provided to the robot by the camera will be processed to find the pixel location of the active landmark. After many of these active landmarks are located, these pixel locations will then be processed according to the calibration method described in [1]. This calibration method first collects a list of 3D world coordinates and correlates them with a list of 2D image coordinates. These points are then analyzed, and, using a least-squares solution, the program obtains four values, being the horizontal and vertical center of the image (in pixels) and the horizontal and vertical scaling factor of the image (also in pixels). This model is explained in greater detail in Chapter 3.
2. ROBOTIC PLATFORM

There are five main components to the final design of the robot. These components are the base robotic platform, the active landmarking subsystem, the wireless communications subsystem, the power supply subsystem, and the camera subsystem. These five components are discussed in sections 2.1 through 2.5. The final robot assembly is shown in Fig. 2.1.

Fig. 2.1. Completed robotic platform.

2.1 Base Robotic Platform

As was mentioned earlier, the base robotic platform used was the National Instruments robotics starter kit, DaNI. This robot is programmed using National Instruments’ proprietary software, LabVIEW. This is a graphical data-flow type
programming environment, with data following lines (called wires) and entering into functions (called VIs) through terminals (called connections). A sample of some LabVIEW programming is shown in Fig. 2.2. LabVIEW has many different add-on modules, including the Robotics Toolkit. Many of the basic functions required to operate the robot (such as moving, using the included ultrasonic sensor, etc.) were already programmed into VIs in the Robotics Toolkit. This Robotics Toolkit shipped with the robot. However, programming in LabVIEW proved to be quite problematic, due to unfamiliarity with the programming environment and lack of documentation available for new users. Much of the available documentation was written for users who have had extensive experience with LabVIEW, and referenced many functions and terms which were never explained clearly.

![LabVIEW diagram](image)

Fig. 2.2. Example of LabVIEW programming.

Each robotic platform has a wired Ethernet port. Each robot is assigned a default IP address from the factory, being 169.254.62.215. In this project, one robot was assigned this IP address (from now on known as robot 215), and the other was assigned 169.254.62.205 (known as robot 205). This Ethernet port is available for programming the robots, as well as for TCP or UDP connections to and from the robot during run-time.
2.2 Active Landmarking Subsystem

To perform the active landmarking, the chosen method was to use an array of LEDs. The initial design was to use an 8x8 matrix of RGB LEDs available from Seeedstudio, Inc, which is shown in Fig. 2.3. This module provided the ability to control each of the LEDs individually, although that function was not required for this application. However, according to the datasheet, each LED was powered separately, each using approximately 50mA of current at 3.3V. The robot could supply the 3.3V power, but each output pin could only supply 3mA. So a board could be designed to power the LED matrix using a transistor switch, but the LED matrix had another drawback: it was in a single plane with all of the LEDs facing the same direction. This is fine as long as the robot with the active landmarking system was actually facing the calibrating robot, but could cause a problem if the landmarking robot was facing more than approximately 30 degrees away from the calibrating robot, as the calibrating robot would be outside of the viewing area of the LED panel.

Fig. 2.3. The LED matrix from Seeedstudio, Inc.

A new board was designed to be an omnidirectional landmark. This board has 16 LEDs facing outward from a circle, and has header pins located such that multiple boards can be stacked on top of each other. The board was powered by a +12V power line to
drive the LEDs. The LEDs were activated using a transistor switch circuit, using a +5V signal from the robotic platform to activate the switch. There is a hole in the center of the circuit board to allow the board to be mounted on the robot. The circuit diagram is shown in Fig. 2.4 and an image of the actual board is shown in Fig. 2.5.

Fig. 2.4. The circuit diagram of the LED tower circuit board.

Fig. 2.5. An image of the assembled LED tower circuit board.
2.3 Wireless Communication Subsystem

The mobile robots will need to be able to communicate with each other, telling other robots to turn on or off their active landmarks, so a wireless network is being used. This network is using Linksys WRT160n routers (Fig. 2.6), which use the IEEE 802.11n protocol. The routers are powered by +12V and GND lines.

Each mobile robot needs to have a wireless router connected to it. The routers needed to be configured to allow two robots (connected to different wireless routers) to be able to wirelessly communicate with each other, and that is not a normal function for a router. Normally, wireless routers work by having all other routers connected through wired connections and use wireless connections to connect to the networking clients. In this case, the role is reversed. The network clients (robots, cameras, etc.) are using wired connections, and the routers are using the wireless connections. There are two different network modes that will facilitate this network structure. The first is called a “bridge” mode. In this mode, any client connected to a router through a wired connection can communicate with any client connected to another router through a wired connection. The second mode is called a “repeater bridge” mode. This mode allows any client connected through a wired or wireless connection to a router to communicate to another client connected to the network through a wired or wireless connection. In this project, either of these solutions will work, though the “repeater bridge” mode was selected to allow wireless access for any development computer that may be needed.

The firmware of the router was not capable of supporting this mode, so an aftermarket firmware needed to be used on the routers. The chosen firmware was DD-WRT v24-sp2. This firmware was configured in a “repeater bridge” mode, with a single router being designated the “master” and all other routers being designated “repeaters.” There needs to be only a single router designated as a “master” in order for the network to function, but all of the routers will behave the same, irrespective of “master” or “repeater” status. The only other difference between the routers is the assigned IP address. The router that was mounted to robot 215 was assigned the IP address 169.254.62.15, and was designated as “master.” The router mounted to robot 205 was
169.254.62.5. This pattern of removing 200 from the last byte of the robot’s IP address to provide the router’s IP address can be continued.

The robots are connected to the wireless routers through the Ethernet port integrated into the sbRIO on the robotic platform.

Fig. 2.6. The Linksys WRT160n wireless router.

2.4 Power Supply Subsystem

The robotic platform has its own battery, but a power supply was needed to power the Active Landmarking Subsystem, Wireless Communication Subsystem, and Camera Subsystem. The Active Landmarking Subsystem and Wireless Communication Subsystem both required +12V and GND power lines. The initial Camera Subsystem design (which will be discussed in section 2.5.1) required +5V and GND lines.

The initial Power Supply Subsystem was designed using a +12V voltage regulator and a +5V voltage regulator. Two 8.4V, 3300mAh batteries were used in series to provide 16.8V (tested to have a maximum value of 18.8V) at 3300mAh. These batteries were connected to a switch, and then to the two voltage regulators.
It was determined that the Camera Subsystem needed to be redesigned (both the redesign and the need for the redesign will be covered in section 2.5.1 and 2.5.2). The redesigned Camera Subsystem required +3.3V, +2.8V, +1.5V, and GND lines.

The redesign of the Camera Subsystem required a redesign of the Power Supply Subsystem. The redesigned power supply used a voltage regulator for each of the required voltages, and these were integrated into the Camera Subsystem board. The same batteries and switch were used.

Later, the Camera Subsystem needed additional redesign (detailed in sections 2.5.10). The new design required +5V, +12V, and GND lines. Again, the same batteries and switch were used, along with +5V and +12V voltage regulators.

2.5 Camera Subsystem

The robotic platform did not come with a camera built-in, so a camera needed to be integrated into the robot.

The final design of the system is using an Axis Communications IP camera which communicates with an Arduino Mega 2560 board using a TCP/IP Ethernet connection. The Arduino then communicates with the robotic platform using a TCP/IP Ethernet connection. A block diagram of the design is shown in Fig. 2.7.

![Fig. 2.7. Block diagram of the design of the system. The image data follows the red arrows.](image-url)
The following sections (2.5.1 to 2.5.10) describe the different designs that have been attempted which do not work for the LabVIEW based robotic platform, DaNI. The camera being used eventually is actually the camera that’s chosen in the initial design, but in a different configuration. The final design was generated through a long process as the capabilities and limitations of DaNI were understood better.

### 2.5.1 Initial camera design

The initial camera chosen for use in the robotic platform was the Axis Communications M1011-W, shown in Fig. 2.8. This is an IP camera that connects via a standard TCP/IP (IPv4 or IPv6) connection to a host computer. This camera is designed to be a security camera connected to a security camera system. It is a CMOS camera capable of generating an RGB JPEG or BMP image at a resolution of up to 640x480 pixels. The M1011-W model was capable of connecting wirelessly to a host system via IEEE 802.11g/b wireless Ethernet. This camera is powered by a +5V signal.

![Axis Communications M1011-W Ethernet Camera](image)

Fig. 2.8. Axis Communications M1011-W Ethernet Camera.

This camera was chosen initially as it was believed that it would be simple to connect to the robotic platform. This camera was connected to the base robotic platform via an Ethernet cable (through a network hub). The camera was not designed to store an
image in its memory, but rather dynamically stream the image to a web browser or security system host computer. The firmware was not available for modification, so the standard API needed to be used to access the image. However, the standard API would only return an image to a browser object in LabVIEW, and the image could not be extracted from the browser object to perform any image processing tasks on it. This meant that this camera would not be suitable for the project.

### 2.5.2 Custom camera design

After it was determined that the Axis Communications camera was unable to be used, a new camera needed to be selected. It was decided to build a custom camera device, so a new CMOS image sensor was selected (a Toshiba TCM8230MD). The image sensor outputs one parallel byte at a time on signal lines D0-D7. In the selected RGB mode, the camera outputs two bytes for each pixel. The first byte contains 5 bits of data for the blue color component and the 3 LSB bits of the green color component. The second byte contains the 3 MSB bits of the green color component and the 5 bits for the red color component. These bytes are shown in Fig. 2.9. The image sensor operates at a minimum of 11.90MHz, and is recommended to operate at 20MHz. This means that every 50ps (at the recommended clock frequency), a new byte of data is available to be read from the data signal lines. This speed is too fast for the robotic platform to read, so a framegrabber was required. A framegrabber is a buffer device used to store an image until a device is ready for it.

![Bytes sent by the camera and their associated color bits](image)

The framegrabber chosen in this design was the Averlogic AL440B. This chip is a 4Mb FIFO, which reads 8 parallel bits into memory at a time, with maximum clock
frequency of 40MHz. The FIFO then stores the information until it is read out, also 8 parallel bits at a time.

Since the oscillator for the camera was selected to run at 20MHz, the maximum data rate out of the camera would be 20MHz, so the FIFO would be able to keep up with the data being sent to it.

The camera had two different voltage requirements. The digital logic required 1.5V, while the image sensor and I/O supplies required 2.8V. The FIFO required an input voltage of 3.3V.

A board was designed to integrate the camera, framegrabber, required voltage supplies, oscillator, output header, and required resistors and capacitors into a single unit (from here known as the cameraboard, shown in Fig. 2.10 with a block diagram shown in Fig. 2.11 and a schematic shown in Fig. 2.12). Due to limitations placed upon the design by the initial PCB manufacturer, a separate breakout board for the camera was designed to plug into a header on the integrated board. The oscillator fit into a standard 8-pin DIP socket, so it was decided to use a DIP socket and plug the oscillator into it. This was chosen to allow for greater modularity and the possibility of easier debugging.

Fig. 2.10. Image of the assembled cameraboard.
The camera and FIFO both used I²C. However, the robotic platform did not actually have any I²C bus built in, so a custom program needed to be developed to add this capability. This testing of this program is detailed in section 2.5.7.
2.5.3 Camera integration

The cameraboard was assembled, including the FIFO, the 8-pin DIP socket for the oscillator, and headers for the camera and connection to the robotic platform. The power supply lines to the board were attached, and turned on. The 1.5V voltage regulator sparked, and burnt itself out. The other regulators worked. In the design, the 1.5V regulator was hooked up to the battery voltage, but should have only been hooked up to the output of the 3.3V regulator. The regulator was changed from a through-hole part to a SMD part in the middle of the PCB design, but the maximum voltage was not accounted for. The wires on the burnt board running to the burnt voltage regulator were then cut. The wires running to the voltage regulator on the other board were also cut, and a jumper was soldered in place to provide the correct voltage to the regulator. The second board was then attached to the power supply lines, turned on, and there were no more obvious problems with the board design.

The voltage inputs to the camera were tested at the header, and the voltages were correct. The voltage inputs to the oscillator were tested at the DIP socket, and the voltage was correct. The camera module and oscillator were both plugged in.

The voltage at the output of the cameraboard going to the router and LED tower were checked, and both read at +12V, which is the required input for both. The power supply lines coming from the LED tower were attached to the cameraboard, and the robot could turn on and off the tower.

The power supply lines going to the router were attached to the cameraboard, and the router started up with no problems.

The cameraboard was then attached to the robot.

2.5.4 Cameraboard testing

The oscillator frequency was tested, and was determined to be 20MHz.

The robot was activated to read data from the cameraboard. The robot received all ‘0’ values for the red, green and blue color data for the pixel. Some different register settings were used, changing the size of the image, and various other properties, but the
cameraboard usually reported back all ‘0’ values. Occasionally, the program would report the pixel data (B, G, R) to be (0, 4, 16), (2, 16, 0), or (0, 2, 8), but each time this occurred, every pixel would be the same value. (0, 4, 16) corresponds to receiving 0x0101 from the cameraboard, (2, 16, 0) is 0x4040, and (0, 2, 8) is 0x0202. Since every pixel was reporting the same value, the FIFO was reporting the same 8 bits continuously.

2.5.5 Cameraboard debugging

The input to the robot was checked. The cable plugged into the header on the cameraboard was unplugged, and all of the data input pins to the robot were wired to be high. The robot received (31, 63, 31), which corresponded to a continuous input of 0xFF, so the input to the robot was working correctly.

The output of the camera was checked next. A breakout board for the camera module was created, so that individual signal lines could be checked. This breakout board, some LEDs, and an external switch were wired to an Arduino Uno, as shown in Fig. 2.13. The external switch was programmed to act as a data latch. Every time the switch was activated, the Arduino would read the data from four of the camera output pins and send that data to the LEDs. Every time the switch was activated, the data changed, so the camera appeared to be outputting different data.

---

Fig. 2.13. Using an Arduino Uno as a latch to test the output from the camera.
The only other major component (excluding the resistors, capacitors, and headers) was the FIFO. A breakout board for a FIFO was made.

The FIFO breakout board and camera plug-in module were wired into a breadboard. The robot and correct power supplies were wired to the breadboard, and the program was run again. The data returned by the program was similar to the original experiment.

2.5.6 FIFO debugging

An experiment to test the FIFO itself was required. The FIFO on the breakout board was wired to power (+3.3V) and ground on the breadboard. 8 LEDs were wired to the data output pins of the FIFO, and the input pins were wired to power and ground to produce an input of 0x0F.

![Breadboard with breakout board for testing FIFO.](image)

The control pins for the FIFO were wired the same way as on the camera board, wiring the PLRTY (Polarity Select) pin to VCC, so IE (Input Enable), OE (Output Enable), RE (Read Enable), RRST (Read Reset), WE (Write Enable), and WRST (Write Reset) should all be low-enabled. All of these pins were wired to low. The RST (FIFO reset signal) was set low and then high to perform a chip reset. RCLK (FIFO read clock signal) and WCLK (FIFO write clock signal) was pulsed to reset the read and write
pointers. WRST (FIFO write reset signal) and RRST (FIFO read reset signal) were then set high to move the chip from read and write reset modes to the normal read and write modes.

In normal mode, the FIFO should read in 8 bits whenever the WCLK is pulsed, and write out those 8 bits whenever the RCLK is pulsed. So WCLK was pulsed. Immediately, 0x0F appeared on the LEDs. This means that both the read pointer and write pointer are pointing at the first address location. The data input to the FIFO was changed to 0x8F and WCLK was pulsed again, to store a value in the second address location. The value of 0x8F immediately appeared on the LEDs. Which means that both the read and write pointers are still pointing at the same address location, though WCLK was pulsed and RCLK was not. This was continued, and the value continued to change immediately, for several WCLK pulses. Then, without explanation, the LEDs stopped changing. So RCLK was pulsed, and the LEDs still didn’t change. Until a chip reset was performed, the values that were output from the FIFO didn’t change.

This same test was performed multiple times, and the same thing usually happened. Occasionally, the FIFO would work correctly. A value was written to the FIFO by pulsing WCLK, and then RCLK could be pulsed to read it and display the value on the LEDs, until more than one value was written to the FIFO without reading a value out. If WCLK was pulsed more than once between RCLK pulses, the LEDs wouldn’t change, no matter how many times RCLK was pulsed.

The FIFO was reset, as well as the read and write pointers. Data was read from the FIFO (pulsing RCLK) without writing any information to it. Every time everything was reset, the same (or nearly the same) data would be read out of the FIFO. Table 2.1 is the list of bytes that was read from the FIFO, while having the input set to 0x76 (without pulsing WCLK). As soon as the FIFO was reset, the first value of 0xF7 would appear on the LEDs. The values in parentheses are values that were occasionally read, but not consistently. A few times, the list jumped from the first value of 0xF7 directly to the seventh value of 0x5D.
The FIFO was then reset, and WCLK was pulsed with the camera in the write reset condition (WRST low). The IRDY output was usually low, indicating that memory space was available for new input data, but would occasionally go high, indicating no free space was available, even though nowhere near the 512 byte capacity had been written into the FIFO. This indicated that there was a problem internal to the FIFO, either with the resetting function, the IRDY flag, or the write pointer. Any of these problems would indicate that the FIFO is unusable and not acting according to the datasheet. At this point, it is determined that the most likely problem with the FIFO is ESD damage when the FIFO was mounted to the board.

The board was redesigned to have the correct power input to the voltage regulators. It was previously recommended that on a redesign of the board, a couple of the capacitors should be moved closer to the chips, providing better noise filtering on the input power lines, and some current limiting resistors be added. After these capacitors and resistors were correctly placed, the boards were sent for manufacturing. The schematic of the new board is shown in Fig. 2.15.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0xF7</td>
<td>(0xE4)</td>
<td>0x3E</td>
<td>0x0D</td>
<td>0xE5</td>
</tr>
<tr>
<td>(0x13)</td>
<td>(0xA2)</td>
<td>0x19</td>
<td>0x59</td>
<td>0x36</td>
</tr>
<tr>
<td>0x97</td>
<td>0x89</td>
<td>0xD5</td>
<td>0xEF</td>
<td>0xF3</td>
</tr>
<tr>
<td>0x39</td>
<td>0x65</td>
<td>0x01</td>
<td>0xDB</td>
<td>0x8A</td>
</tr>
<tr>
<td>0x32</td>
<td>0x4D</td>
<td>0x68</td>
<td>0x59</td>
<td>0x1A</td>
</tr>
<tr>
<td>0x37</td>
<td>0x4A</td>
<td>0x62</td>
<td>0xA8</td>
<td>0x4E</td>
</tr>
<tr>
<td>0x5D</td>
<td>0x9F</td>
<td>0x6A</td>
<td>0xA9</td>
<td>0x03</td>
</tr>
<tr>
<td>0x75</td>
<td>0x37</td>
<td>0xEF</td>
<td>0xF9</td>
<td>0x6D</td>
</tr>
</tbody>
</table>
When the boards were received, the components were attached to the board, being particularly careful to avoid any ESD damage. The boards were then tested in a similar manner to the previous cameraboards. Despite the added resistors and capacitors, and being more careful about limiting possible ESD damage, the cameraboard behaved in a similar manner to the previous version of the cameraboard.

2.5.7 Camera testing

To make sure that the camera was sending correct data to the FIFO, a series of tests were performed on the camera. As was mentioned earlier, the camera had been

Fig. 2.15. Schematic of the redesigned cameraboard.
tested using the Arduino, and appeared to be working correctly, but further testing was required.

The first test was to find out whether the camera could accept changes from the I2C bus to the internal register. The camera was attached to appropriate power sources and oscillators. A BusPirate (available from Dangerous Prototypes) was used to send I2C commands from a computer to the camera module, and then read the registers back from the camera. The camera would accurately read the default register values when the camera was reset. A new value was written to one of the registers, and then read back. The new value was received by the BusPirate, so the register had been updated correctly. After every pulse on the reset signal line, the registers reset back to the default values. This indicates that the camera registers and I2C bus is working correctly.

The camera was then attached to the robot I2C lines, and the program started. The robot changed some of the camera registers. The robot I2C lines were disconnected from the camera. The BusPirate was then reattached to the camera I2C bus and the camera registers were read. The registers reflected the changes made by the camera, indicating that the robot’s I2C programming was working correctly.

A CMUCam3 was then used to check to see if the camera was sending out data correctly. The CMUCam3 uses the same Averlogic AL440B FIFO as a framegrabber, and an Omnivision CMOS camera. The CMUCam3 saves the image received to an SD card. The Omnivision camera is on a removable board, so the camera was removed, and the TCM8230MD camera was wired in place of it. After modifying the program running on the CMUCam3 to adapt to the new image size and number of bytes received for each pixel (3 for the Omnivision, 2 for the TCM8230MD), an image was obtained from the camera, but the image was apparently just yellow-colored noise (see Fig. 2.16). After many attempts to correctly retrieve an image from the CMUCam3 using the TCM8230MD, the data clock signal was removed from between the TCM8230MD and the CMUCam3. The image was still yellow-colored noise, indicating that the noise was stored in the AL440B FIFO, as opposed to being sent from the TCM8230MD module.
A device called a LogicSniffer (available from Dangerous Prototypes) was then used to test the output from the camera. The LogicSniffer is a USB-powered, USB-communicating, 16-channel digital oscilloscope. The LogicSniffer was attached to the camera, using one line for the input clock signal, another for the output data clock signal, another for the vertical synchronization pulse, another for the horizontal synchronization pulse, and four lines for four of the pixel data signals. The LogicSniffer was set to read those 8 lines at 200MHz for 24ksamples. The LogicSniffer indicated that the clock, data clock, and vertical synchronization pulse were all working correctly, but the horizontal synchronization pulse was not performing as the datasheet had indicated. The datasheet showed the HD (horizontal synchronization) signal going high when the data pixel data was being sent out. However, the LogicSniffer showed the HD signal following the data clock signal when the data pixel data was being sent out. It should be noted that the only datasheet that could be found for the Toshiba camera module was watermarked as “Tentative.”

The change in the HD signal was the cause of the problem with the CMUCam3, and the design could not be reconfigured to be corrected. This deviation from the
expected behavior, however, could not cause the problem with the function of the custom cameraboard.

The camera was again tested with the LogicSniffer to determine if the camera output data was accurate. Since the software for the LogicSniffer would only produce a visual representation of the data, not actually provide a data file, a test was developed to visually check the data coming in. The camera was set to turn off the Auto Luminance Correction. The LogicSniffer was activated to provide data from the four data lines, and a screenshot was taken of the result. A finger was then placed over the lens of the camera, and the test was repeated. A visual examination of the data showed that the more significant bits of each of the colors seemed to go low when the finger was placed over the lens. This was somewhat difficult to see, as one pixel’s data was spread across two bytes, and the MSB of each color didn’t line up from one byte to the other. The significance of each of the color bits was averaged to produce a significance of each data bit.

<table>
<thead>
<tr>
<th>Data bit</th>
<th>Color byte 1</th>
<th>Color byte 2</th>
<th>Data bit significance</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0</td>
<td>B0</td>
<td>G3</td>
<td>1.5</td>
</tr>
<tr>
<td>D1</td>
<td>B1</td>
<td>G4</td>
<td>2.5</td>
</tr>
<tr>
<td>D2</td>
<td>B2</td>
<td>G5</td>
<td>3.5</td>
</tr>
<tr>
<td>D3</td>
<td>B3</td>
<td>R0</td>
<td>1.5</td>
</tr>
<tr>
<td>D4</td>
<td>B4</td>
<td>R1</td>
<td>2.5</td>
</tr>
<tr>
<td>D5</td>
<td>G0</td>
<td>R2</td>
<td>1</td>
</tr>
<tr>
<td>D6</td>
<td>G1</td>
<td>R3</td>
<td>2</td>
</tr>
<tr>
<td>D7</td>
<td>G2</td>
<td>R4</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 2.2
Showing the significance of each data bit.

Fig. 2.17. LogicSniffer graph with the lens unobstructed.
Fig. 2.18. LogicSniffer graph with the lens obstructed by a finger.

As is seen in the previous two figures (Fig. 2.17 and Fig. 2.18), the data bits with higher significance values (specifically D2), is, on average, much lower when the lens was obstructed than when the lens is unobstructed. Even bits with slightly lower significance showed decreased average values. This indicates that the camera is working correctly and outputting correct data.

2.5.8 Temporary camera design

While the original cameraboard PCB was being made and the cameraboard was being assembled, a temporary camera was selected to continue testing the mobile robot. The chosen camera was a uCAM-TTL camera manufactured by 4D Systems. The camera was used “on loan” from another project. It is a serial camera which can provide either JPEG compressed images or raw image data. The images can be output in VGA, QVGA, 160x120, or 80x60 resolutions. The image sensor is an OmniVision OV7640 CMOS sensor. The camera is attached to an RS-232 Shifter bought from Sparkfun Electronics, and attached to the RS-232 port of the robotic platform. The programming on the robotic platform needed to be modified to allow the RS-232 port to be activated.

The camera was hooked up, but the software kept reporting errors every time the port was accessed. This is due to a programming error which was unable to be corrected due to unfamiliarity with LabVIEW.

The loaning project then determined that the camera did not work correctly, since it did not respond to the same program that they used with other identical cameras. The other cameras worked, but this camera did not respond to serial commands. Two new
cameras were ordered, but this time, the uCAM-232 model was ordered, so that the TTL/RS-232 level shifter was not required. However, in order to use these uCAM-232 modules, the serial port is required, but the programming problem with LabVIEW still is preventing the use of the serial port.

Later on, it was determined that the problem with the serial port was incorrect addressing of the serial port. Once the correct address (ASRL1::INSTR) was input to the program instead of the only available address on the default drop-down list (RIO0::INSTR), serial access was available. The uCAM-232 was then attached to the robot, and an image was accessed. However, the data that was received was much smaller than the image should have been. The camera was sending the image back, and the robot’s serial buffer was overflowing. Even after slowing the baud rate, the robot’s input was unable to keep up with the camera’s output. It was decided that the robot was too slow to allow that large of a file to be read through the serial port.

2.5.9 Revisiting the initial camera design

After working with the TCP connection between the robots, it was determined that a possibility existed for connecting to the Axis Communications IP camera using a TCP connection. After researching how HTTP connections are established, a TCP connection was created to the IP camera containing the HTTP request information for a bitmap image. The HTTP request is as follows:

GET /axis-cgi/bitmap/image.bmp HTTP/1.1
HOST:169.254.62.115

The connection was created and the HTTP header was returned, but no data followed. The HTTP header returned is as follows:

HTTP/1.0 200 OK
Cache-Control: no-cache
Pragma: no-cache
Expires: Thu, 01 Dec 1994 16:00:00 GMT
Connection: close
Content-Type: image/bmp
A different HTTP request was sent to a different page that did not contain a single image, but was instead an html document. The HTML code of the entire page was returned, so it was obvious that the header was correct. The HTTP request was again modified to point to a page that returns a JPEG image, and some of the image was returned. The returned data was of a much smaller size than the actual JPEG image, and every request was of a differing size. In the TCP connection VI, a comment mentions that the connection is terminated when a character of 0x04 is received. The character 0x04 is the ASCII End of Transmission character. Based on this information, it is assumed that DaNI stops receiving the JPEG image after the first 0x04 character received, and it was also assumed that there is an 0x04 character at the start of the bitmap image. To test this theory, a program was designed to send every value from 0x00 to 0xFF to the robot. The robot received every value, including 0x04, so this theory was incorrect.

To verify the TCP request, the same request was sent via a PC using software called PuTTY, and the response contained the entire image. This indicates that there is a problem with the Ethernet client on DaNI, as opposed to a problem with the TCP request or a problem with the Ethernet server on the IP camera, though exactly what the problem is with the Ethernet client is unknown.

2.5.10 New camera design

Due to the incompatibilities between DaNI and the IP camera, incompatibilities between DaNI and the uCAM-232 camera, and the problems with the Averlogic FIFO, a new camera, or at least a new interface to an existing camera, needed to be used. Two ideas were developed. The first of these ideas was to use the new CMUCam, CMUCam4. CMUCam4 is an OmniVision camera module that has been integrated into an Arduino shield. However, using this camera proved to be ineffective, as the CMUCam4 reads data from the image sensor as it is being sent out. The data is sent from the Arduino shield to the Arduino board through a serial port, which takes quite a bit of time. Obtaining an 80x60 pixel image takes approximately 12 seconds, while a 640x480 pixel image takes over one minute. Since the data is read from the image sensor when it is sent out, any motion of the camera will cause a drastic shift in the image at whatever
pixel is currently being read. Also, and more importantly, any shift in the environment will cause streaks and other artifacts in the image, which will cause problems with the image differencing algorithm.

A second option was to connect the uCAM-232 to the serial port on an Arduino and use an Arduino Ethernet shield to transfer the data back to DaNI. The implementation for the second option was developed, and the Arduino Mega 2560 R3 was chosen as the Arduino platform to use. The Ethernet shield for the Arduino also has an SD card slot, so the Arduino was programmed to accept one image from the uCAM-232 and store it on the SD card. It then sends an ACK signal back to the robotic platform, which will then instruct the landmarking platform to activate its active landmark. The platform then sends an ACK signal back to the Arduino, and it stores a second image from the uCAM-232. The Arduino then compares the two images performing the Binary Image Generation technique described below in section 3.2.1. However, the Arduino would not accept the entire image from the uCAM-232. Once again, the serial buffer would overflow before the Arduino was able to store the data to the SD card. The serial line was slowed down to a baud rate of 14400, but the Arduino still could not keep up with the incoming data.

There was one other configuration with the available hardware that could still be tried. An Arduino program was developed to send a TCP request to the Axis M1011-W IP camera that was first used. The Arduino received the entire bitmap image.
3. CAMERA CALIBRATION ALGORITHM

The camera calibration algorithm was developed by Dr. Yanfei Liu and Dr. Carlos Pomalaza-Ráez in [1].

The two categories of camera characteristics that need to be calibrated are internal and external parameters. External parameters would include the position and orientation of the camera relative to the world coordinate system, though these parameters can be ignored and the world coordinate system can be assumed to be identical to a unified coordinate system defined by the mobile robot. This unified coordinate system would be defined such that the camera mounted on the robot is at the origin of the coordinate system, with the camera looking in the same direction as the z-axis, and the top of the camera pointed toward the y-axis, as shown in Fig. 3.1. Internal parameters would be the focal length of the camera, the position of the image center in the image, the scaling of the horizontal and vertical pixels, and the skew of the image. In the case of CCD and CMOS cameras (such as the ones in this project), the skew of the image is zero, since the horizontal and vertical directions on the image are exactly 90 degrees apart. The focal length is defined to be the distance from the image plane to the focal point.
3.1 Active Landmarking

The project uses active landmarking, so a visual signal needs to be actuated in the environment. The chosen signal is an LED tower on a landmarking robot. The calibrating robotic platform is to take two images for each location of the landmarking robot. The calibrating robot instructs the landmarking robot to turn off its active landmarking LED tower and then takes the first image. The calibrating robot then instructs the landmarking robot to turn on its LED tower and takes a second image.

3.2 Blob Processing

The blob processing algorithm was developed by Dr. Yanfei Liu and Dr. Carlos Pomalaza-Ráez in [9]. The algorithm requires a single binary image, with each pixel either being true or false. In this case, a ‘true’ pixel is to indicate the location of the active landmark, and the ‘false’ pixel is for all remaining pixels.
3.2.1 Binary image generation

The two images that are generated in the active landmarking phase are compared. The pixel-wise difference of the images is taken, and then each pixel is compared to a threshold value. If the absolute value of the difference between the corresponding pixels of the two landmarking images is larger than this threshold value, that pixel is assumed to be a pixel on the image of the active landmark, and the binary image pixel is set to be ‘true.’ If the difference is smaller than the threshold value, the pixel is set to be ‘false.’

3.2.2 Slab and blob generation

The binary image is then processed to obtain the center of the landmarking blobs. Each row is scanned, and every ‘true’ pixel is either added to an existing slab or a new slab, depending on whether the previous pixel in the row was true. If the previous pixel is true, the current pixel is added to the slab containing the previous pixel. Otherwise, the previous pixel was false, and the current pixel is added to a new slab.

Once all of the slabs have been generated, the slabs are then joined together into blobs. If any slab shares a column with a slab in the previous row, it is added to that blob. If one slab shares a column with slabs in the previous row which are in different blobs, those blobs are combined into a single blob, and the current slab is added to the combined blob.

The largest blob (the one with the most ‘true’ pixels) is then determined, and the center point of that blob is found. This is assumed to be the center of the image of the active landmark.

3.3 Wireless Distance Estimation

In the calibration algorithm, the distance between the landmarking robot and each of the other points that have been previously landmarked, as well as the distance to the calibrating robot need to be known. The goal of this was to use the Received Signal Strength Indicator (RSSI) value from the router to determine the distance between the
robots. This would not require any additional hardware, as the wireless routers already attached to the robots would be capable of reporting the RSSI. An experiment was derived to determine a function to relate the RSSI value to the distance between two routers. Obviously many factors would influence this function, including the characteristics of the individual wireless channel. Not all factors can be accounted for, but the goal was to determine a few different approximation models based on the environment in which the camera was being calibrated. In [1], it is stated that “The consensus of most researchers is that it is very difficult to guarantee distance estimation errors of less than 10cm when using 802.11 (Wi-Fi) or 802.15.4 (ZigBee) devices in both indoor and outdoor environments,” so the experiment was performed to determine if this is the case with the wireless routers chosen in this design. The results of the experiment are given in section 5.1.

3.4 Mathematical Model

As was mentioned in section 1.3, many landmarking positions are required. Suppose that we choose \( N \) landmarking positions and perform the previous processing on them. The following figure, Fig. 3.2, shows a model of the camera in relation to the image plane and a landmark point \( A_i = \begin{bmatrix} A_{ix} & A_{iy} & A_{iz} \end{bmatrix}^T \) in the environment \((0 \leq i < N)\). The point \( a_i = \begin{bmatrix} a_{ix} & a_{iy} \end{bmatrix}^T \) is the projection of \( A_i \) onto the image plane, and \( f \) is the focal length. The center of the image is point \( p = \begin{bmatrix} p_x & p_y \end{bmatrix}^T \). The ray defined as passing from the camera center through point \( p \) is called the principle axis. \( A_{ic} \) is the projection of the point \( A_i \) onto the principle axis. The coordinates on the image plane start in the lower left corner of the image (shown as \( a_x \) and \( a_y \)).
\( \tilde{a}_i \) is the augmented vector of \( a_i \), so that \( \tilde{a}_i = [a_i^T \ 1]^T = [a_{ix} \ a_{iy} \ 1]^T \). Similarly, \( \tilde{A}_i \) is the augmented vector of \( A_i \), so that \( \tilde{A}_i = [A_i^T \ 1]^T = [A_{ix} \ A_{iy} \ A_{iz} \ 1]^T \). There is a relationship between point \( \tilde{a}_i \) and point \( \tilde{A}_i \), which is given in (3.1) below.

\[
A_{ix} \tilde{a}_i = K[R \ t] \tilde{A}_i
\]

where \( R \) is the rotation of the camera in the world coordinate system, \( t \) is the translation of the camera in the world coordinate system, and \( K \) is the standard CMOS camera matrix,

\[
K = \begin{bmatrix}
\alpha & \gamma & u_0 \\
0 & \beta & v_0 \\
0 & 0 & 1
\end{bmatrix}
\]

where \([u_0 \ v_0]\) are the coordinates of the center point \( p \) in the image plane, \( \gamma \) is the skew of the image, \( \alpha \) is the scaling factor of the horizontal axis, and \( \beta \) is the scaling factor of the vertical axes.

In this case, when the world coordinate system is assumed to be the robotic platform’s coordinate system, \( R = I \) and \( t = 0 \), so
\[ A_{ix} \tilde{a}_i = K \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tilde{A}_i \] (3.3)

\[ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tilde{A}_i = A_i, \text{ so equation (3.3) becomes} \]

\[ A_{ix} \tilde{a}_i = K A_i \] (3.4)

Since, when using a CMOS camera, there is no skew in the image, \( \lambda = 0 \), so

\[ K = \begin{bmatrix} \alpha & 0 & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \] (3.5)

\[ \alpha = m_x f, \quad \beta = m_y f, \quad u_0 = m_x p_x, \quad \text{and} \quad v_0 = m_y p_y, \] where \( m_x \) is the number of pixels per meter in the horizontal axis and \( m_y \) is the number of pixels per meter in the vertical axis. So the matrix \( K \) can be rewritten as

\[ K = \begin{bmatrix} m_x f & 0 & m_x p_x \\ 0 & m_y f & m_y p_y \\ 0 & 0 & 1 \end{bmatrix} \] (3.6)

To find the location of a point \( A_i \) from its respective point \( \tilde{a}_i \) on the image plane, equation (3.4) can be rewritten as

\[ A_i = A_{ix} K^{-1} \tilde{a}_i \] (3.7)

where
The distance vector \( \mathbf{D}_{ij} \) between two landmarks \( i \) and \( j \) (where \( 0 \leq i < j < N \) ) can be determined by the equation

\[
\mathbf{D}_{ij} = [d_{ijx}, d_{ijy}, d_{ijz}] = \mathbf{A}_i - \mathbf{A}_j
\]  

(3.9)

Define \( L_{D_{ij}}, L_{A_i}, \) and \( L_{A_j} \) to be the lengths of vectors \( \mathbf{D}_{ij}, \mathbf{A}_i, \) and \( \mathbf{A}_j \).

\[
L_{A_j}^2 = A_j^T A_j = (A_i^T + D_{ij}) (A_i + D_{ij})
\]

\[
= A_i^T A_i + 2 D_{ij}^T A_i + D_{ij} D_{ij}
\]  

(3.10)

\[
L_{A_j}^2 = L_{A_i}^2 + 2 D_{ij}^T A_i + L_{D_{ij}}^2
\]  

(3.11)

Rewriting equation (3.11) gives

\[
D_{ij}^T A_i = \frac{L_{A_i}^2 - L_{A_j}^2 - L_{D_{ij}}^2}{2}
\]  

(3.12)

Define \( \delta_{ij} \) to be

\[
\delta_{ij} = D_{ij}^T A_i = \frac{L_{A_i}^2 - L_{A_j}^2 - L_{D_{ij}}^2}{2}
\]  

(3.13)

Substituting equation (3.7) into equation (3.13) yields
Define \( M_1 = \frac{1}{\alpha}, M_2 = -\frac{u_0}{\alpha}, M_3 = \frac{1}{\beta}, \) and \( M_4 = -\frac{v_0}{\beta}. \) Equation (3.14) then becomes

\[
\delta_y = D_{ij}^T A_{iz} \begin{bmatrix} \frac{1}{\alpha} & 0 & -\frac{u_0}{\alpha} \\ 0 & 1 & -\frac{v_0}{\beta} \\ 0 & 0 & 1 \end{bmatrix} \tilde{a}_i
\] (3.14)

Define \( M_1 = \frac{1}{\alpha}, M_2 = -\frac{u_0}{\alpha}, M_3 = \frac{1}{\beta}, \) and \( M_4 = -\frac{v_0}{\beta}. \) Equation (3.14) then becomes

\[
\delta_y = A_{iz} \begin{bmatrix} d_{ijx} & d_{ijy} & d_{ijz} \end{bmatrix} \begin{bmatrix} M_1 & 0 & M_2 \\ 0 & M_3 & M_4 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{ix} \\ a_{iy} \\ 1 \end{bmatrix}
\] (3.15)

where \( d_{ijx}, d_{ijy}, \) and \( d_{ijz} \) are the \( x, y, \) and \( z \) components of \( D_{ij}. \)

Simplifying equation (3.15) yields

\[
\delta_y = A_{iz} \begin{bmatrix} d_{ijx} & d_{ijy} & d_{ijz} \end{bmatrix} \begin{bmatrix} a_{ix}M_1 + M_2 \\ a_{iy}M_3 + M_4 \\ 1 \end{bmatrix}
\] (3.16)

\[
= A_{iz} \begin{bmatrix} a_{ix}d_{ijx}M_1 + d_{ijx}M_2 + a_{iy}d_{ijy}M_3 + d_{ijy}M_4 + d_{ijz} \end{bmatrix}
\]

Rearranging equation (3.16) shows

\[
\frac{\delta_y}{A_{iz}} = d_{ijz} = A_{iz} \begin{bmatrix} a_{ix}d_{ijx} & a_{iy}d_{ijy} & d_{ijy} \end{bmatrix} \begin{bmatrix} M_1 \\ M_2 \\ M_3 \\ M_4 \end{bmatrix}
\] (3.17)

Define \( \lambda_{ij} \) to be

\[
\lambda_{ij} = \frac{\delta_y}{A_{iz}} - d_{ijz} = c_{ij}^T x
\] (3.18)

where \( c_{ij} = \begin{bmatrix} a_{ix}d_{ijx} & d_{ijx} & a_{iy}d_{ijy} & d_{ijy} \end{bmatrix}^T \) and \( x = \begin{bmatrix} M_1 & M_2 & M_3 & M_4 \end{bmatrix}^T. \)

As was mentioned earlier, for each calibration, a total of \( N \) landmarks are used. These landmarks can then be written as
$A = Cx$  \hspace{1cm} (3.19)

where $A = \begin{bmatrix} \lambda_{12} & \lambda_{13} & \cdots & \lambda_{1N} & \lambda_{23} & \cdots & \lambda_{(N-1)N} \end{bmatrix}^T$ and

$C = \begin{bmatrix} c_{12}^T & c_{13}^T & \cdots & c_{1N}^T & c_{23}^T & \cdots & c_{(N-1)N}^T \end{bmatrix}^T$.

The least-squares solution for $x$ is

$x = \left[ C^T C \right]^{-1} C^T A$  \hspace{1cm} (3.20)

Once $x$ has been estimated, the camera’s intrinsic values are found as follows:

$\alpha = \frac{1}{M_1}$  \hspace{1cm} (3.21)

$\beta = \frac{1}{M_3}$  \hspace{1cm} (3.22)

$u_0 = -\frac{M_2}{M_1}$  \hspace{1cm} (3.23)

$v_0 = -\frac{M_4}{M_3}$  \hspace{1cm} (3.24)

The horizontal center point of the image is $u_0$, and the vertical center point of the image is $v_0$. The horizontal scaling factor of the image is $\alpha$, and the vertical scaling factor is $\beta$. 
4. CALIBRATION ALGORITHM IMPLEMENTATION

Each robotic platform uses the same program, so that each robot will be calibrated using at least one other robot as a landmarking robot. The program starts out with initializing the robot, which is explained in greater detail in section 4.1. The robot then enters a loop, which iterates through all of the robots involved in the algorithm. For each iteration, the robots check to see if they are the ‘Master’, which is the calibrating robot. If it is the ‘Master’, the robot then starts the data collection algorithm, which consists of a loop of sections 4.2 through 4.5, followed by the calibration algorithm in section 4.6. If the robot is not the master, then it is a landmarking robot, and enters the Landmarking Subroutine, which is described in section 4.7.

4.1 Robot Initialization

The first part of the implementation involves initializing the robot with some important values.

The robots have each been assigned an IP address, but the program itself is unaware of the address. One of the important parts of this program is being able to quickly deploy this program to multiple robots, so the program itself should not need to be modified with the IP address. LabVIEW has a function which returns the IP address. This function is used, and the last byte of the IP address is stored into the memory of the robot as the global variable ‘Robot #’. The program uses this number as a basis to be able to identify the IP address of the router, camera, and Arduino. Each robot has each of these other devices attached to it, and it must be able to quickly identify the assigned IP address of the associated equipment. This is possible by using the Robot Number. The last byte of the Arduino will be 150 less than the Robot Number (ex. The robot with an IP address of 169.254.62.215 is Robot Number 215, and has an associated Arduino with the
IP address of 169.254.62.65). Similarly, the last byte of the router will be 200 less, and the last byte of the camera will be 100 less than the last byte of the IP address of the robot (following the same example, the router’s IP address will be 169.254.62.15 and the camera’s IP address will be 169.254.62.115). The robot number is also the identifier that the robot uses to identify itself in the Robot List.

The Robot List is a listing of the robot number for each of the available robots. This is used to identify which robots are available for use in the calibration algorithm.

Another value is the Camera Available list, which is a Boolean array of whether a given robot in the Robot List has a camera that needs to be calibrated. Each entry in the Camera Available list corresponds to one entry in the Robot List (using the same index).

There are also global Boolean variables ‘Calibrated’ and ‘End’. ‘Calibrated’ is a flag as to whether or not this camera has already been calibrated. This is useful in case a robot is entered multiple times into the Robot List. At the end of the calibration algorithm, the ‘Calibrated’ flag is set. If a robot that has ‘Calibrated’ set to true comes up on the Robot List, the calibration algorithm will be skipped. ‘End’ is set when all of the robots in the Robot List have been calibrated.

After all of these values are set, an Ethernet connection to the robot’s Arduino is created.

4.2 Obtaining the Binary Image

DaNI sends an image request command to the Arduino. The Arduino sends an HTTP request to the Axis IP camera, receives the image, and stores it on the SD card on the Ethernet shield. The Arduino then sends a ready command back to DaNI.

DaNI then sends a command to the landmarking robot to turn on its active landmark, and receives a response when that is completed.

DaNI sends another image request command to the Arduino, and the Arduino then sends the HTTP request and then receives and stores the image again.

The Arduino then compares the two images utilizing the Binary Image Generation technique described in section 3.2.1 to develop a binary image. This binary image has three bytes for every pixel, since it was developed using a bitmap image. Since this is not
required, the image is then compressed to only use one bit per pixel to decrease the transmission time of the binary image back to DaNI.

Once the Arduino has finished the compression, a signal is sent back to DaNI. DaNI then instructs the landmarking robot to turn off the active landmark and then requests the image from the Arduino. The Arduino sends the compressed image back to DaNI, which then creates a 2-dimensional integer array. In the integer array, a value of 0 means that the pixel did not change from one image to the next, while a value of 255 means the pixel did change. This integer array is then used for the rest of the calibration algorithm.

4.3 Slab Generation

The integer array is processed according the slab generation algorithm in section 3.2.2. Each slab contains three pieces of information: the row number, the starting X-value (x0), and the ending X-value (x1). The starting and ending X-values are both included in the slab, so the smallest possible slab would have \( x_0 = x_1 \), corresponding to a slab of only one pixel.

The binary image array is scanned across each row, starting with row 0, which is actually the bottom row of the image. Each pixel is checked to see if it is a 255 or a 0. If the pixel is a 0, the algorithm moves to the next pixel. If the pixel is a value of 255, the previous pixel is checked. If the current pixel is the first pixel in the row or the previous pixel was a 0, then a new slab is created, otherwise the pixel is added to the previous slab. A pixel is added to a previous slab by opening the previous slab and changing the ending X-value to be the current pixel. Once the entire image is scanned, the array of slabs is forwarded to the Blob Generation subroutine.

4.4 Blob Generation

The Blob Generation subroutine implements the Blob Processing algorithm described in section 3.2.2. Each blob has four data values: a list of slabs, the horizontal
center of the blob, the vertical center of the blob, and the weight of the blob (the number of pixels comprising the blob).

Each slab is checked to see if it should be added to a blob that has already been created. This check is done by a VI called “Blob Slab Link.vi”. This VI receives inputs of the current list of blobs, the list of all slabs, and the current slab. The VI checks to see if the row number of the current slab is one more than the row number of a slab in any blob. If the current slab is in the next row, the starting and ending X-values are checked. There are three cases in which the slab should be added to the blob. The first case is when the current slab’s $x_0$ value is between the previous slab’s $x_0$ and $x_1$ values. The second is when the current slab’s $x_1$ value is between the previous slab’s $x_0$ and $x_1$ values. The third case is when the previous slab’s $x_0$ value is between the current slab’s $x_0$ and $x_1$ values. The case when the previous slab’s $x_1$ value is between the current slab’s $x_0$ and $x_1$ values is then redundant, because either the previous slab’s $x_0$ value is between the current slab’s $x_0$ and $x_1$ (case 3) or the previous slab’s $x_0$ value is below the current slab’s $x_0$ value, in which case the current slab’s $x_0$ value is between the previous slab’s $x_0$ and $x_1$ values (case 1). If any of these three cases are true, the index of the current blob is then added to the output array (‘Links’) from this VI, otherwise, the blob is ignored. The next blob is then checked.

If there are no links between the current slab and any of the existing blobs (the size of ‘Links’ is zero), then the slab needs to be added to a new blob. A new array is instantiated with the current slab being the only value and set to be the list of slabs in the blob. The row of the slab is set to be the vertical center of the blob. The horizontal center of the blob is found by averaging the starting and ending X-values of the slab. The weight of the blob is determined by subtracting the starting X-value from the ending X-value and adding one (one must be added since the starting and ending pixels are both included in the slab).

If there is a link between the current slab and one blob (the size of ‘Links’ is one), then the slab needs to be added to that blob. The slab index is added to the array of slabs stored in the blob. Then a weighted average is taken to find the new center of the blob. The weight of the current slab is computed. This weight is multiplied by the horizontal
and vertical center of the current slab. The weight of the blob is likewise multiplied by the horizontal and vertical center of the blob. These weighted centers are added together, and divided by the total weight (the weight of the current slab and the weight of the blob). This total weight is then stored as the new weight of the blob.

If there are links between the current slab and more than one blob (the size of ‘Links’ is more than one), then multiple blobs need to be merged together, and the current slab needs to be added to the new blob. The blobs are combined with using a similar method to merging one blob and a slab. The list of slabs are combined, a weighted average is taken of the horizontal and vertical centers of the blobs, and the weights are combined for the total weight of the blob. The slab is then added to the combined blob by the same method as in the case of only one link.

The array of all of the blobs is then returned to the Image Processing VI. This VI then selects the largest blob by finding the blob with the largest weight. This is assumed to be the blob that represents the active landmark. The center point of the blob is then returned to the main program. The main program sends the x- and y-coordinates of the largest blob in the image to the Arduino to be stored on the SD card. The main program also stores these values into their respective arrays to be used in the Camera Calibration implementation discussed in section 4.6.

4.5 Robot Position Estimation

The position of the robots needs to be determined for the camera calibration algorithm to be used. However, as will be shown in section 5.1, the planned use of the RSSI value for Wireless Distance Estimation is not possible. The position estimation of the landmarking robot with respect to the calibrating robot will need to be further explored in future works. For experimental purposes, the relative positions that will be used will be pre-determined and programmed into the Arduino. This list of distances is returned by the Arduino one location at a time, each time through the loop. This results in one position being retrieved after every landmark location is determined on the image. These values are then stored in an array to be used in the Camera Calibration Implementation discussed in section 4.6.
Once the robot decides that enough data points have been collected, it will then exit the loop and begin the calibration algorithm. The definition of enough data points is not defined in this project, but left for future work.

4.6 Camera Calibration

The camera calibration subroutine follows the Camera Calibration Algorithm defined in chapter 3, specifically the Mathematical Model detailed in section 3.4. The previous loop provides three different arrays: an array of x-coordinates of landmarks on the image, an array of y-coordinates of landmarks on the image, and a two-dimensional array of relative positions of the landmarking robot with respect to the calibrating robot (one dimension being the x-, y-, and z-coordinates, and the other dimension being the landmark index).

The program then enters two nested loops, both iterating through the array of robot positions. The outer loop defines the start point \((i)\) and the inner loop defines the end point \((j)\) of a vector between two landmarks. Any iteration that the start point and the end point being the same landmark (any iteration with \(i = j\)) is skipped by the program.

In any case that \(i \neq j\), part of the camera calibration is performed. In the mathematical model, several equations for \(\delta_y\), \(\lambda_y\), and \(c_y\) were given. These equations are formed here. The equations used for this calculation are shown in Equations (4.1) through (4.9).
\[ d_{ijx} = A_{ix} - A_{jx} \] (4.1)
\[ d_{ijy} = A_{iy} - A_{jy} \] (4.2)
\[ d_{ijz} = A_{iz} - A_{jz} \] (4.3)
\[ L_i^2 = A_{ix}^2 + A_{iy}^2 + A_{iz}^2 \] (4.4)
\[ L_j^2 = A_{jx}^2 + A_{jy}^2 + A_{jz}^2 \] (4.5)
\[ L_{ij}^2 = (A_{ix} - A_{jx})^2 + (A_{iy} - A_{jy})^2 + (A_{iz} - A_{jz})^2 \] (4.6)
\[ \delta_j = \frac{L_j^2 - L_i^2 - L_{ij}^2}{2} \] (4.7)
\[ \lambda_{ij} = \frac{\delta_j}{A_{iz}} - d_{ijz} \] (4.8)
\[ c_{ij} = \begin{bmatrix} a_{ix} \ d_{ijx} \\ a_{iy} \ d_{ijy} \\ a_{iz} \ d_{ijz} \end{bmatrix} \] (4.9)

where \( A_{ix}, A_{iy}, A_{iz}, A_{jx}, A_{jy}, \) and \( A_{jz} \) are given by the robot position array, \( a_{ix} \) is given by the array of centers of the landmark on the image in the horizontal direction, and \( a_{iz} \) is given by the array of centers of the landmark on the image in the vertical direction.

In every iteration in which \( i \neq j \), the resulting \( \lambda_{ij} \) is added as a new element in the array \( A \), and every \( c_{ij} \) is added as a new column in the two-dimensional array \( C \).

After the loop has completed, \( x \) is estimated using the equation shown in (4.10), and the intrinsic values are calculated as shown in equations (4.11), (4.12), (4.13), and (4.14). These intrinsic values are then sent to the Arduino to be stored on the SD card to be read later. The global variable ‘Calibrated’ is then set to be true.
\[ x = \left( C^T C \right)^{-1} C^T A \]  \hspace{1cm} (4.10)

\[ \alpha = \frac{1}{M_1} \]  \hspace{1cm} (4.11)

\[ \beta = \frac{1}{M_3} \]  \hspace{1cm} (4.12)

\[ u_0 = -\frac{M_2}{M_1} \]  \hspace{1cm} (4.13)

\[ v_0 = -\frac{M_4}{M_3} \]  \hspace{1cm} (4.14)

where \( x = [M_1 \ M_2 \ M_3 \ M_4]^T \).

The calibrating robot then sends the command ‘DONE’ to every robot in the robot list. The main loop of each robot then moves to the next iteration, with the next robot in the Robot List being designated as the calibrating robot.

### 4.7 Landmarking Subroutine

In the landmarking subroutine, the robot is waiting for instructions from the calibrating (‘Master’) robot. The robot is able to receive five different commands: ON (0x0A), OFF (0x09), MOVE (0x0B), DONE (0x0C), and END (0x0D). The landmarking robot then responds with an ACK (0x0E) command.

#### 4.7.1 On

The ‘ON’ command is used to instruct the landmarking robot to turn on the active landmark for the calibrating robot to be able to take the appropriate image. When the ‘ON’ command is received, the landmarking robot sets the digital output for the control signal to the active landmark to be high, turning on the active landmark.
4.7.2 Off

The ‘OFF’ command is used to instruct the landmarking robot to turn off the active landmark. When the ‘OFF’ command is received, the landmarking robot sets the control signal to the active landmark to be low, turning off the active landmark.

4.7.3 Move

The ‘MOVE’ command is used when the landmarking robot is to be moved from one place to another. Where the landmarking robot moves to is up to the landmarking robot. The algorithm to determine where the landmarking robot should move to is not explored in this project.

4.7.4 Done

The ‘DONE’ command is sent by the calibrating robot when that robot has completed its camera calibration. This command is sent from the calibrating robot to every robot in the Robot List. Every robot then ends the current iteration of the main loop, moving to the next Robot Number in the Robot List.

4.7.5 End

The ‘END’ command is sent by the last calibrating robot in the Robot List when it has finished the calibration. This command is sent to every robot in the Robot List. This is a flag to be used in future projects to let the larger program know when the calibration subroutine has finished.
5. EXPERIMENTAL RESULTS

The first experiment that was performed was to determine the feasibility of using the RSSI value returned by the router to determine the distance between two landmarks. The results of this experiment are shown in section 5.1. The second experiment was to determine how well the camera calibration algorithm worked. These results are shown in section 5.2.

5.1 Wireless Distance Estimation Testing

One of the routers used on the robots was placed on the floor in a fixed location in a hallway outside of the Engineering Office in IPFW’s Engineering building. Another of the routers was moved one foot away, and the RSSI value read from the receiving router. The RSSI value was read and recorded 20 times, and then the router was moved another foot away. This data was then averaged and plotted. The resulting plot, with a $2^{nd}$ order polynomial curve, is shown in Fig. 5.1.
After seeing the results from the first experiment, a second experiment was conducted in the same hallway. The routers were placed in the same location, but placed on chairs that were 23.5 inches tall. The chairs were used to distance the wireless antennas away from any possible metal plating, conduit, or wiring present in the floor or ceiling of the floor below. The data was averaged and plotted again, and the resulting plot and curve is shown in Fig. 5.2.
As is clear from these graphs, the RSSI value obtained by the Linksys wireless routers is unable to be used to accurately determine the distance between the wireless routers. The rest of the project was performed assuming that the distance estimation could be obtained through some other means. The included ultrasonic range sensors were considered, but would not provide an accurate reading at the distances that were required (being capable of distancing objects no further than about 3m[10], while the robots would likely be positioned at a distance of 10m-30m). [1] mentions using Ultra-wide Bandwidth (UWB) communication to determine the distance between the robots. It is probable that this technology would provide much more accurate results for Wireless Distance Estimation. The distances between the robots were entered manually for the remaining testing.

Fig. 5.2. Results from second WDE experiment (routers 23.5” off floor).
5.2 Camera Calibration Testing

5.2.1 Experimental setup

To test the camera calibration algorithm described in section 4.6, a large, open area was required. A local gymnasium was used as the environment in which to test the implementation of the algorithm. Being approximately 17m by 29m, the gymnasium was large enough to obtain landmarks that could be representative of a realistic test in an unknown environment. The calibrating robot was placed at one end of the gymnasium, and placed on one riser of a set of metal bleachers. 19 locations around the gymnasium then were chosen. Many of these locations had cardboard boxes placed on them, which will allow for varying heights of the landmarking robot. The robot was placed at each of these locations and the $A_x$, $A_y$, and $A_z$ components were measured manually between the calibrating robot’s camera and the active landmark position due to the inaccurate RSSI estimations shown in section 5.1. These positions were recorded into a text file, which was downloaded to the SD card in the Arduino. These positions needed to be measured manually, since it was not possible to use the RSSI value as originally planned.

Due to the timing of the program, the robot’s battery and the batteries supplying power to the router, active landmark, Arduino, and camera would not last through the entire test. These batteries were bypassed by wiring a lab power supply in place of the batteries. However, a single power supply was not enough to run both robots and added components. Additional power supplies were gathered, and using one supply for each base robotic platform and one power supply for each set of added components, there was enough power supplied to be able to run the experiment.

Once these locations were recorded, the landmarking robot was placed in the first position, and the program was started. On the initial run, a large number of pixels on the binary image were marked as ‘true’, indicating a large difference between the first and second images taken. It was determined upon examination of the images that were differenced that the camera was performing Automatic White Balance and Automatic Brightness Correction functions. These images are shown in Fig. 5.3 through Fig. 5.5.
Fig. 5.3. Image with active landmark off, using brightness correction.

Fig. 5.4. Image with active landmark on, using brightness correction.
These functions were turned off, and the experiment was restarted. While the program was running, the landmarking robot’s LED tower would not light up. In testing the original program, however, there was no problem with the LED tower turning on and off. The wireless routers were then re-examined to determine the cause of the communication breakdown.

5.2.2 Wireless communications testing

When the routers were originally set up in a “repeater bridge” mode, one computer was attached to one router and another computer was attached to the second router (shown in Fig. 5.6). The computers were able to ping the associated router, the opposing router, and the other computer.
When the original robot communication was examined, the two robots were connected to a single router using standard Ethernet cables, as shown in Fig. 5.7. The communication between the robots was successful in this configuration.

During the experiment, one robot was connected to one router, and the other robot was connected to the other router, as is shown below in Fig. 5.8. In this configuration, the robots seemed to be unable to communicate with each other.

The Arduino code was modified to simulate the landmarking robot by accepting TCP packets from the calibrating robot. The Arduino was attached in place of the landmarking robot, as shown in Fig. 5.9. The Arduino was unable to receive any packets.
A computer was then used to send packets through both routers to the Arduino. This setup is shown in Fig. 5.10. The Arduino again did not receive any packets.

The Arduino was then attached to the same router that the computer was attached to, as shown in Fig. 5.11. The Arduino received packets without any problem.

The secondary router (the “repeater” router) was reconfigured to be a standard router (configured as an access point). The wireless broadcast was turned off. The two routers were then connected using an Ethernet cable. The Arduino and a computer were both hooked up to the “master” router, and packets were sent to the Arduino from the computer. Fig. 5.12 shows this setup. The packets arrived fine.
The Arduino was then attached to the second router, as shown in Fig. 5.13. Packets were again sent to the Arduino from the computer. These packets also arrived fine.

For the rest of the experiment, the LED tower on the landmarking robot was manually actuated to reflect the packets that the calibrating robot was sending out. It was actuated by plugging the signal lines into a +5V source, simulating the ‘high’ output from the robot or by unplugging the signal lines from the source, simulating the ‘low’ output from the robot.

After the experiment was completed, the wireless bridge was turned off, and the routers were connected via a wired connection. The robots were connected to their respective routers (as shown in Fig. 5.14), and the communication between the robots began working as it should. This series of tests shows that the communication between the robots is working fine, but that there is a problem with the communication between the routers. It is believed that there is a problem in the setup of the “repeater bridge” mode of the routers. This problem was unable to be corrected in the time frame of this
A suggestion for the correction of the wireless communication problem is described in section 6.3.

![Router to Router](image)

**Fig. 5.14. Robots using a wired connection between the routers.**

### 5.2.3 Camera calibration results

The images resulting from the experiment were promising. However, performing the experiment in the gymnasium had an unexpected downside. The gymnasium has a reflective floor, and the active landmark showed up twice, once in the actual position and once reflected in the floor (see Fig. 5.15), which led to an incorrect binary image, as shown in Fig. 5.16. Due to timing and location constraints, the experiment could not be repeated in another location.

Since the images had been saved on the SD card as the experiment was performed, the images could be modified to remove the reflection. This was performed by copying a block of pixels from the image with the active landmark off onto the image with the active landmark on to cover the position of the reflected landmark (see Fig. 5.17). A slight modification to the program on the Arduino allowed the images stored on the SD card to be used in place of images read from the camera. It should be noted that this does not cause any major difference in the actual implementation. Under normal conditions, the Arduino requests the image from the camera and saves the data to a file on the SD card. This file is then closed, and the connection to the camera is closed. The Arduino then opens the file on the SD card to perform the image differencing task. In this case, only the first step of obtaining the image from the camera is skipped, but the rest of the implementation is followed, including the image differencing. The resulting binary image using the modified landmarking image is shown in Fig. 5.18.
Fig. 5.15. Image in gymnasium showing reflection of active landmark off floor.

Fig. 5.16. Binary image showing reflection of landmark off of gymnasium floor.
Modifying the Arduino program to use the stored images provided two more advantages. The first advantage was the opportunity to reorder the locations and determine if the same results were obtained. The second was to allow the algorithm to be rerun with differing numbers of locations to determine how that would change the results.
of the algorithm. A list of all of the physical locations of the landmarks as well as the corresponding image coordinates is shown in Table 5.1.

Table 5.1
Landmark locations and corresponding image coordinates.

<table>
<thead>
<tr>
<th>Landmark #</th>
<th>Ax (m)</th>
<th>Ay (m)</th>
<th>Az (m)</th>
<th>ax (pixels)</th>
<th>ay (pixels)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.558</td>
<td>-0.138</td>
<td>7.075</td>
<td>466.8</td>
<td>293.734</td>
</tr>
<tr>
<td>2</td>
<td>-2.307</td>
<td>-0.124</td>
<td>9.485</td>
<td>104.09</td>
<td>300.582</td>
</tr>
<tr>
<td>3</td>
<td>-0.628</td>
<td>-0.227</td>
<td>9.477</td>
<td>246.505</td>
<td>289.815</td>
</tr>
<tr>
<td>4</td>
<td>3.953</td>
<td>-0.129</td>
<td>10.699</td>
<td>575.852</td>
<td>299.31</td>
</tr>
<tr>
<td>5</td>
<td>3.451</td>
<td>-0.077</td>
<td>12.201</td>
<td>514.571</td>
<td>303.417</td>
</tr>
<tr>
<td>6</td>
<td>1.909</td>
<td>-0.06</td>
<td>11.913</td>
<td>424.722</td>
<td>306.773</td>
</tr>
<tr>
<td>7</td>
<td>-0.667</td>
<td>0.148</td>
<td>13.627</td>
<td>258.068</td>
<td>319.796</td>
</tr>
<tr>
<td>8</td>
<td>-3.575</td>
<td>-0.065</td>
<td>13.168</td>
<td>83.236</td>
<td>305.512</td>
</tr>
<tr>
<td>9</td>
<td>-5.038</td>
<td>-0.155</td>
<td>13.48</td>
<td>15.385</td>
<td>299.008</td>
</tr>
<tr>
<td>10</td>
<td>-4.829</td>
<td>-0.155</td>
<td>16.138</td>
<td>64.914</td>
<td>301.075</td>
</tr>
<tr>
<td>11</td>
<td>-3.217</td>
<td>-0.155</td>
<td>15.603</td>
<td>130.704</td>
<td>301.722</td>
</tr>
<tr>
<td>12</td>
<td>-1.646</td>
<td>-0.065</td>
<td>17.136</td>
<td>216.565</td>
<td>308.283</td>
</tr>
<tr>
<td>13</td>
<td>0.749</td>
<td>-0.146</td>
<td>15.578</td>
<td>333.235</td>
<td>303.755</td>
</tr>
<tr>
<td>14</td>
<td>3.681</td>
<td>0.412</td>
<td>14.987</td>
<td>484.571</td>
<td>332.044</td>
</tr>
<tr>
<td>15</td>
<td>5.765</td>
<td>-0.159</td>
<td>15.863</td>
<td>570.538</td>
<td>300.859</td>
</tr>
<tr>
<td>16</td>
<td>-4.607</td>
<td>0.151</td>
<td>20.98</td>
<td>124.634</td>
<td>315.688</td>
</tr>
<tr>
<td>17</td>
<td>0.156</td>
<td>-0.348</td>
<td>25.35</td>
<td>299.546</td>
<td>300.67</td>
</tr>
<tr>
<td>18</td>
<td>0.752</td>
<td>-0.348</td>
<td>7.674</td>
<td>373.489</td>
<td>274.347</td>
</tr>
<tr>
<td>19</td>
<td>-1.419</td>
<td>-0.348</td>
<td>7.067</td>
<td>135.845</td>
<td>272.492</td>
</tr>
</tbody>
</table>

The modified images were then loaded onto the SD card, and the program was started, using the first 10 landmarks. These modified images provided binary images with only a single landmark in each image (see Fig. 5.18 as compared to Fig. 5.16). The resulting intrinsic camera values were:

\[ \alpha = 0.766133182 \]
\[ \beta = 0.589381043 \]
\[ u_0 = 0.296169604 \]
\[ v_0 = 0.0318319225 \]
The center point of the image is shown in Fig. 5.19. This point still appears to be off center. However, since the relative position of the landmarking robots were determined using the primary axis of the robot, it is entirely possible that the camera was not exactly aligned with the primary axis of the robot. It was determined that the primary axis of the robot was at point (303,309) on the image. This primary axis is shown by the red-colored dot in the image in Fig. 5.20. It can be seen that this point coincides very closely with the point that was determined through the camera calibration algorithm, when these two points are placed on the same image (see Fig. 5.21).

Fig. 5.19. White dot showing the location of the center point of the image.
As for determining the accuracy of $\alpha$ and $\beta$, the primary use of these values is to determine the aspect ratio of the image. This means that these values don’t have particular significance on their own, but rather as a ratio. To determine the actual $\alpha / \beta$
value for the camera, a black square on a plain piece of white paper was placed in front of the camera, as shown in Fig. 5.22. The image of this square was analyzed, by measuring the length of the sides of the square in the image. The ratio of the width of the square to the height of the square should be the same as the ratio of $\alpha$ to $\beta$. The ratio of the width to the height of the square is determined to be 1.003774. The ratio of $\alpha$ to $\beta$ from the camera calibration algorithm is 1.299895. As can be seen, there is a rather large difference between these values.

![Fig. 5.22. Showing the black square used to determine the actual camera aspect ratio.](image)

The locations of the center points of the first 10 landmarks were placed on an image (shown in Fig. 5.23), and it was determined that all of the points were in a very narrow area in the vertical direction. This narrow band could introduce error into the program.
The program was run again with all 19 landmarks. The intrinsic values from this experiment are as follows:

\[
\alpha = 766.112819 \\
\beta = 1306.568782 \\
u0 = 294.640918 \\
v0 = 321.270508
\]

As can be seen, \(u0\) and \(v0\) still are very close to the primary axis of the robot, as shown in Fig. 5.24. The figure shows the primary axis point in (color) and \(u0\) and \(v0\) as the white point.
The ratio between \( \alpha \) and \( \beta \) is now 0.586355. There is still a large difference between the ratio of \( \alpha \) and \( \beta \) and the ratio of the height to width of the black square that was tested earlier. The locations and image coordinates of the landmarks are shown in Table 5.1.

Again, the locations of the landmarks are placed into a single image, and shown in Fig. 5.25. Again, the landmarks are in a very narrow region in the vertical direction. It is possible that this is causing a problem with the algorithm, by either reducing the possible resolution in the vertical direction (introducing an error in the precision of the measurement) or causing a problem with the spherical distortion of the image. As can be seen from the images taken by the camera, there is spherical distortion in the image. If a very narrow region is used (in either the horizontal or vertical directions), it is possible that the spherical distortion causes errors in the algorithm.
Another test was run. In this test, the camera calibration section was the only section tested. The location data was entered into a two-dimensional array, and the image coordinate data was also entered into an array. The Camera Calibration 2 VI was put into a loop, each time using subarrays of the location and coordinate data. The first time through the loop, the VI used landmark sets 1 through 10. The second time, the VI used landmark sets 1 through 11, the third time 1 through 12, and so on, up to set 19. The data output data from this test is shown in Table 5.2. As can be seen from the table, $\alpha$, $u0$, and $v0$ hold nearly constant throughout all of the sets of data, but $\beta$ varies greatly. The relative error between this test data and the measured data is shown in Fig. 5.26.
Table 5.2
Intrinsic values for the given sets of data from the first test.

<table>
<thead>
<tr>
<th>Data Sets</th>
<th>$\alpha$</th>
<th>$\beta$</th>
<th>$u_0$</th>
<th>$v_0$</th>
<th>$\alpha / \beta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-10</td>
<td>766.133</td>
<td>589.379</td>
<td>296.17</td>
<td>318.319</td>
<td>1.299899</td>
</tr>
<tr>
<td>1-11</td>
<td>767.413</td>
<td>655.393</td>
<td>295.691</td>
<td>317.016</td>
<td>1.17092</td>
</tr>
<tr>
<td>1-12</td>
<td>767.616</td>
<td>630.509</td>
<td>295.479</td>
<td>317.043</td>
<td>1.217454</td>
</tr>
<tr>
<td>1-13</td>
<td>767.523</td>
<td>621.419</td>
<td>295.5</td>
<td>317.245</td>
<td>1.235114</td>
</tr>
<tr>
<td>1-14</td>
<td>767.662</td>
<td>832.909</td>
<td>295.459</td>
<td>312.083</td>
<td>0.921664</td>
</tr>
<tr>
<td>1-15</td>
<td>765.465</td>
<td>958.533</td>
<td>294.777</td>
<td>318.927</td>
<td>0.79858</td>
</tr>
<tr>
<td>1-16</td>
<td>766.009</td>
<td>1026.75</td>
<td>294.694</td>
<td>320.686</td>
<td>0.746052</td>
</tr>
<tr>
<td>1-17</td>
<td>765.847</td>
<td>1065.81</td>
<td>294.693</td>
<td>320.508</td>
<td>0.718559</td>
</tr>
<tr>
<td>1-18</td>
<td>765.841</td>
<td>1154.36</td>
<td>294.792</td>
<td>320.398</td>
<td>0.663433</td>
</tr>
<tr>
<td>1-19</td>
<td>766.113</td>
<td>1306.5</td>
<td>294.641</td>
<td>321.27</td>
<td>0.586386</td>
</tr>
</tbody>
</table>

Fig. 5.26. Relative error using data from original test.

To test the theory about having the data points spread throughout the image, another test was performed. In this test, images were again collected, but had the
landmarks spread throughout the image. 10 landmarks were used in this case. The locations of these landmarks are given in Table 5.3, and are shown on an image in Fig. 5.27.

Table 5.3
Landmark locations and corresponding image coordinates.

<table>
<thead>
<tr>
<th>Landmark #</th>
<th>Ax (m)</th>
<th>Ay (m)</th>
<th>Az (m)</th>
<th>ax (pixels)</th>
<th>ay (pixels)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-0.81605</td>
<td>0.994</td>
<td>4.728</td>
<td>183</td>
<td>405</td>
</tr>
<tr>
<td>2</td>
<td>-1.63305</td>
<td>1.316</td>
<td>4.728</td>
<td>63</td>
<td>445</td>
</tr>
<tr>
<td>3</td>
<td>-1.63305</td>
<td>-0.43</td>
<td>4.728</td>
<td>60</td>
<td>171</td>
</tr>
<tr>
<td>4</td>
<td>0.82305</td>
<td>0.59</td>
<td>4.728</td>
<td>453</td>
<td>341</td>
</tr>
<tr>
<td>5</td>
<td>1.63705</td>
<td>1.451</td>
<td>4.728</td>
<td>567</td>
<td>471</td>
</tr>
<tr>
<td>6</td>
<td>1.22205</td>
<td>-0.692</td>
<td>2.899</td>
<td>618</td>
<td>75</td>
</tr>
<tr>
<td>7</td>
<td>0.41605</td>
<td>-0.692</td>
<td>2.273</td>
<td>464</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>-1.01905</td>
<td>-0.692</td>
<td>3.054</td>
<td>70</td>
<td>72</td>
</tr>
<tr>
<td>9</td>
<td>-0.80905</td>
<td>-0.06</td>
<td>4.728</td>
<td>181</td>
<td>230</td>
</tr>
<tr>
<td>10</td>
<td>0.41105</td>
<td>-0.692</td>
<td>4.728</td>
<td>387</td>
<td>125</td>
</tr>
</tbody>
</table>

Fig. 5.27. Diagram of landmark locations in second test.
The data was put through the same test as the previous data, and the results can be seen in Table 5.4. The relative error between the test data and the measured data is shown in Fig. 5.28.

Table 5.4
Intrinsic values for the given sets of data from the second test.

<table>
<thead>
<tr>
<th>Data Sets</th>
<th>α</th>
<th>β</th>
<th>u₀</th>
<th>v₀</th>
<th>α/β</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-4</td>
<td>729.061</td>
<td>751.153</td>
<td>314.054</td>
<td>239.133</td>
<td>0.970589</td>
</tr>
<tr>
<td>1-5</td>
<td>754.797</td>
<td>750.275</td>
<td>321.03</td>
<td>240.16</td>
<td>1.006027</td>
</tr>
<tr>
<td>1-6</td>
<td>737.474</td>
<td>744.911</td>
<td>316.117</td>
<td>238.909</td>
<td>0.990016</td>
</tr>
<tr>
<td>1-7</td>
<td>733.908</td>
<td>723.484</td>
<td>313.636</td>
<td>246.55</td>
<td>1.014408</td>
</tr>
<tr>
<td>1-8</td>
<td>731.419</td>
<td>746.379</td>
<td>314.915</td>
<td>242.698</td>
<td>0.979957</td>
</tr>
<tr>
<td>1-9</td>
<td>731.643</td>
<td>746.878</td>
<td>315.032</td>
<td>242.556</td>
<td>0.979602</td>
</tr>
<tr>
<td>1-10</td>
<td>732.619</td>
<td>746.619</td>
<td>314.473</td>
<td>242.226</td>
<td>0.981249</td>
</tr>
</tbody>
</table>

Fig. 5.28. Relative error using data from second test.
As can be seen from this data, all of the intrinsic values seem to approach a constant value. In addition, the ratio of $\alpha$ to $\beta$ appears to approach 1, which is approximately what was obtained from the test using the black square. As can be easily seen from Fig. 5.29, the values of $u0$ and $v0$ are very close to the correct value.

Fig. 5.29. Camera Calibration Algorithm calculated center point (white) and the primary axis of the robot (red).

This data indicates that the camera calibration algorithm is working correctly when given landmarks with image coordinates across the entire image.
6. CONCLUSIONS

6.1 Blob Processing Algorithm

The Blob Processing Algorithm, as developed by Dr. Liu and Dr. Pomalaza-Ráez in [9], and as implemented in this project, works. The image differencing and thresholding performed on the Arduino yields a usable binary image. This binary image is then used to find the locations of slabs, and those slabs to find the largest blob.

6.2 Camera Calibration Algorithm

The Camera Calibration Algorithm does find a center point that very near to the primary axis. The Camera Calibration Algorithm also finds a ratio very near the scaling factor of the image when it has data points spread throughout the entire image. However, when the image points are clustered together in a small area of the image, there is a much larger error. The exact cause of this error should be investigated in future works.

6.3 Wireless Communication

At this point, the wireless communication between the robots was unsuccessful. A suggestion for possible future work is to implement this system using an ad-hoc wireless system. An ad-hoc system would allow greater flexibility in the implementation of this system by allowing more robots to be used, and not having a single ‘master’ router, but rather have a distributed, dynamic routing system for communication. There are many types of these systems which could be effectively used in this project [11][12], but it is left to future work for the actual implementation.
6.4 **Wireless Distance Estimation**

The Wireless Distance Estimation, as proposed in this project, was unsuccessful. The available technology was not accurate enough to be able to determine the actual location of the landmark to any usable precision. It is left to future work to be able to dynamically determine the distance between the landmark locations.

6.5 **Robot Position Estimation**

Similar to section 6.3, it is left to future work to determine the relative positions of the landmarks to the calibrating camera.

6.6 **Landmark Positioning**

In this project, the landmarks were placed in a random pattern. In future work, a method to locate where a landmark should be placed needs to be developed.

6.7 **Active Landmark Color**

It is possible that, on a red background, the red LEDs on the LED tower would not cause enough of a difference between the images to be able to be detected by the thresholding technique. This could be avoided by possibly having multiple different LED colors on the LED tower, or using multi-colored LEDs on the LED tower, and being able to dynamically change the color to provide contrast to the background.

6.8 **Image Differencing Algorithm**

It is possible to use a different algorithm to perform the differencing and thresholding, perhaps by using a particular color as being more important than others. For example, while using a set of red LEDs on the LED tower, perhaps only the red component of the images should be compared. This should also be explored in future works.
LIST OF REFERENCES
LIST OF REFERENCES


