The Report Committee for Matthew Wayne DeKoning
Certifies that this is the approved version of the following report:

Embedded Sensor Speed and Width Estimation

APPROVED BY
SUPERVISING COMMITTEE:

Supervisor:

Brian L. Evans

Jonathan Valvano

Embedded Sensor Speed and Width Estimation

by

Matthew Wayne DeKoning

Report

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

December 2017

Dedication

This work is dedicated to my wife, Anna DeKoning, who makes it all worthwhile.

Acknowledgements

Without the technical and financial support of Sandia National Laboratories this work as well as my continued education would not have been possible. Special thanks to Reinhard Stotzer and Sara Pecak for their constant support and help coordinating this project, as well as Jason Krein and Mark Koch for the concept that led to this report and their technical expertise, and Jeremy Giron for his technical expertise as well. The 3D printed cases for the sensors used in this project were both designed and fabricated by Jared Bare, whose work greatly improved how field tests were conducted. Finally, thanks to Raymond Sand and Lonnie Dahl whose help during field tests allowed me to gather all the data for this project.

At the University of Texas at Austin Dr. Brian L. Evan's advising ensured the success of this report. His deep knowledge, both technical and administrative, proved invaluable countless times throughout this work.

I thank Dr. Valvano for lending his technical expertise through the editing of this work.

Abstract

Embedded Sensor Speed and Width Estimation

Matthew Wayne DeKoning, M.S.E.

The University of Texas at Austin, 2017

Supervisor: Brian L. Evans

The goal of this report is to provide a novel system of estimating width and velocity of an object passing perpendicularly through a sensor field. By fixing four sensors on an axis at known angles the distance and velocity of an object can be estimated from the times of detection of each sensor using a system of equations. To develop a prototype of this system the scenario was modeled and simulated, a processor and sensors were selected, and algorithms for detection and estimation were developed. The goal of the report was to develop a self-contained sensor network and computing platform that would correctly estimate the speed of the object in miles per hour within 15% accuracy, and to estimate the width of the object within 20% accuracy. Due to prototype design errors, these requirements were not met, however useful algorithms and simulations were developed to lead towards successful future work.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1: The Goal of this Embedded System

Electronic advancements have allowed smaller, faster, and more connected computers and sensors to become commonplace. With these advancements, distributed sensor networks have become efficient means to monitor a real world environment. Unattended Ground Sensors (UGS) are embedded systems that typically consist of a processing component, sensor elements, and a network component which are placed into the field on a long term mission to monitor specific environmental features. These systems must work without human intervention and often must arrange themselves for proper data gathering after being deployed dynamically (for example dropped from a plane). Many military and some civilian applications exist for such systems with purposes ranging from monitoring the weather in a region to detecting and classifying intrusions. The type or types of sensors integrated into such systems can vary greatly. If a UGS is self-powered through batteries, solar panels, or other means, the need for conservation of power becomes the main limiting factor on the system's available mission time. Thus power efficiency is a key design consideration. This report details the simulation, prototyping, and design of a UGS for the purpose of detecting intrusions and estimating the width and speed of the intruding objects.

An array of thermal sensors are specified as the desired environmental monitoring due to the simplicity of computation when analyzing thermal signatures for intrusion. The UGS designed in this report detects and characterizes intrusions moving perpendicular to a horizontal axis, movements similar to those found in real world environments like roads and sidewalks. Previous works related to thermal sensing and characterization rely on novel sensor banks and lens configurations, but this project seeks to minimize the number of sensors used while using simple lenses and finding their optimal orientation. Given a

system with four sensors and known orientation, the geometry of the sensors can be exploited to estimate the speed and width of the intruding objects given only intrusion times from each individual sensor. This system provides very low power intrusion characterization along a road or sidewalk by utilizing a small array of low power sensors.

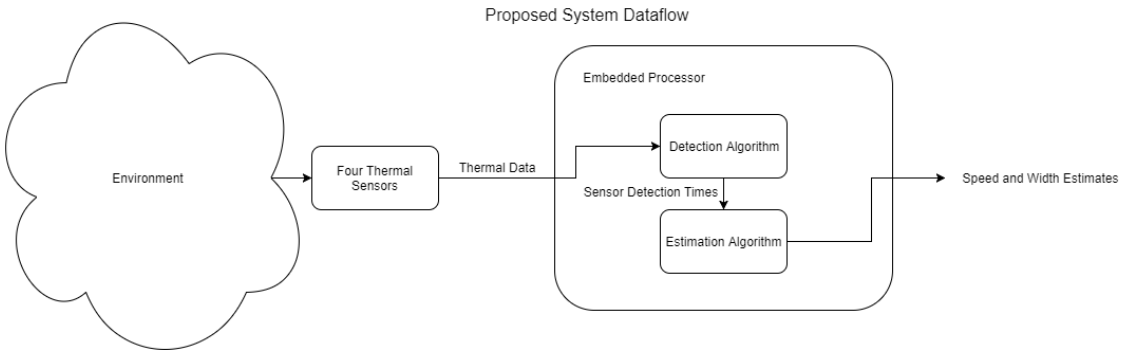

Figure 1.1: System Dataflow

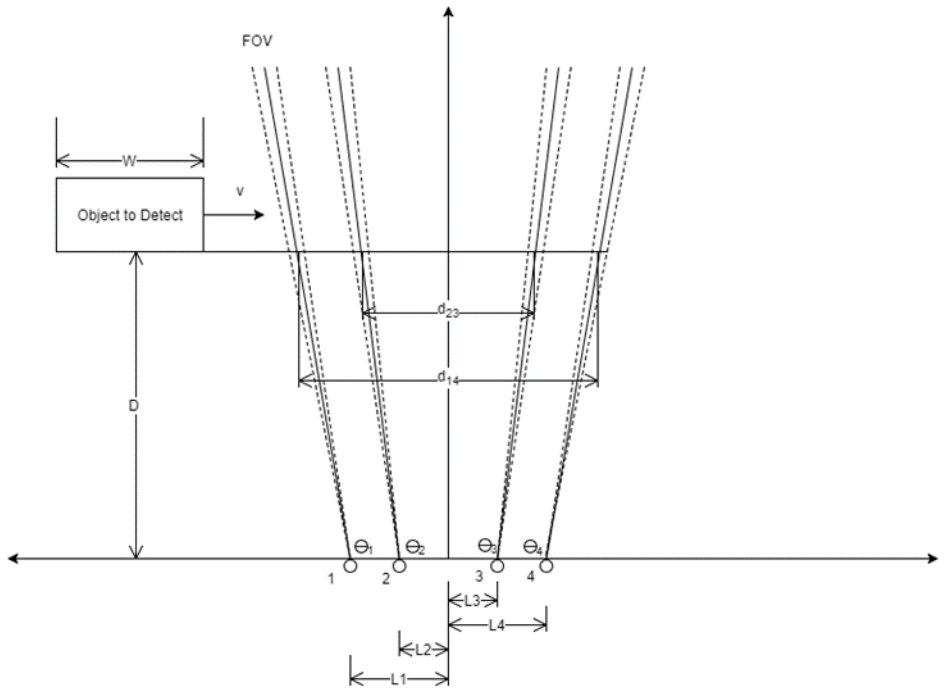

Figure 1.2: System Geometry

To optimally use the geometry of the system, the ideal sensor would behave like an infinitely small beam. This beam precisely monitors the environment at the angle the sensor is positioned at, recording times when the beam is broken by an intruder. The most realistic implementation of such a system involves a field of lasers as well as a set of sensors at the edge of the range of the system to monitor whether the beam is broken or not. This is neither power efficient, or discrete, and the system becomes quite large and difficult to properly place. Lasers consume a significant amount of power and placing the monitoring sensors at a reasonable range would require a high degree of precision. Pyroelectric infrared sensors (also known as passive infrared sensors or PIR) on the other hand monitor the environment with a much larger field of view than lasers, but consume very little power. Careful optical design can result in a small field of view and reasonable range for such a sensor. Commercially available products can monitor objects over one hundred fifty feet

away with a field of view less than four degrees. While a final system may ultimately use PIRs, the prototype presented here uses thermal camera technology due to budget and timing constraints. This choice ultimately pushes the amount of error in the system's estimate beyond the goal of this report.

To better understand tradeoffs in field of view, frames per second, and the optimal range of the system prototype a simulation of the scenario was developed and analyzed in Python 2.7. Once the characteristics of the system's sensors were decided upon, the optimal geometry and optimal set of test objects were also determined from the simulation. After simulation-aided prototype and test definition, algorithms to detect intrusions and estimate speed and width based on intrusion times were developed to complete the system. The detection algorithm must account for ambient thermal noise and temperature drift while providing accurate detection times. Any reduction of noise in the detection time measurements twice benefits the system, as the time deltas required for estimations are the product of two measurements, and therefore noise is added twofold. The estimation algorithm is written to solve the system of equations presented by the system's geometry and the time differences. Finally the functioning prototype gathered data and the accuracy of the simulation and design was assessed. Further design and algorithm implementations, as well as simulation updates are informed by the discrepancies between reality and simulation. The simulation informs the design of the embedded system, both of which are refined by the experimental findings.

1.2: Related Work

Since they became available in the late 1970's, PIR sensors have been used in general surveillance [5]. The development of the Fresnal lens allows an inexpensive and small device to monitor a very wide area for a short distance, making them ideal for

monitoring small but sensitive areas such as points of ingress or egress. Academia also explores ways to utilize these sensors in new and novel ways. In 2003 Gopinathan et al. contributed a lens and sensor network capable of characterizing the location of an object in a square region monitored by four PIR sensors [3]. This was accomplished by creating a unique mask for each of the four sensors and comparing the outputs of the sensors to determine the state of the object within the surveilled area. This enabled four sensors without fine location detection to function at a finer location granularity than normal. Their detection algorithm utilized four binary event signals triggered by the voltage waveform from the PIR sensor. These event signals were then analyzed and the state of the object in the field extrapolated from there. Tests on both a precisely moving robot and a human and successfully mapped their movement through the sensor field.

Similar work in this field includes the carefully laid out sensor fields proposed by Song et al. which would track an intrusion throughout a building by carefully positioning slightly overlapping PIR sensors throughout a region and observing the sequence of triggering [1]. This system efficiently tracked an intruder in a large environment made up of conjoined smaller spaces, such as a building or campus. A novel three sensor solution was proposed by a team from the University of Bologna to monitor a hallway and classify intrusions by object count and direction [11]. This method characterized the waveform generated by different numbers of people walking side by side in a hallway, arranged three sensors to gather the data at different angles, and used a separate processor to fuse the data gathered into an estimate of count and direction. Given the presence of three sensors monitoring the same region, the estimate exists when the majority of the sensors are in agreement. Another recent exploration of PIR sensor systems by the same team focused on the analog waveform coming from the differential PIR sensor [12]. This study correlated the amplitude and duration of intrusion signals to distance from the sensor and proposes a

mirrored two sensor approach for PIR sensors. In yet other work, a two column sensor module was developed for monitoring the intersections of two hallways [4]. In total eight sensors, each given a unique view of the hallway, comprise the two column sensor. The data captured by these sensors are then fed through several estimation algorithms including Bayesian tracking in order to more accurately track objects. Recent work at Sandia National Laboratories has fused off-the-shelf sensors with more complex Lab developed devices in order to avoid false positives and save power [2]. The off the shelf sensor are arrayed around the custom sensors, and their detections trigger the specialized sensors to further investigate an incident.

The work presented here draws considerable inspiration from some of the methods used before it, but also diverges in several key areas. Each of these IR sensing modules work to track an intrusion throughout a region as well as detect it. Many of them then use machine learning algorithms to then attempt to classify the intrusion through analyzing certain aspects of the data gathered. The solution proposed does not track an object, it assumes sequential detections will be the same object. Instead it merely detects and characterizes objects by speed and width. The threshold and event flag processing algorithm utilized by Gopinathan's team is emulated in this work.

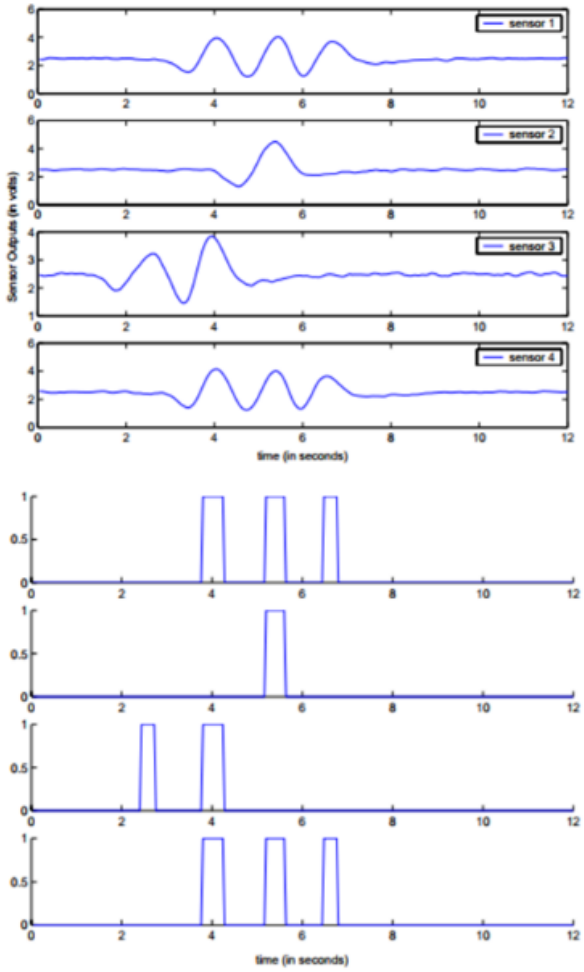

Figure 1.3: Gopinathan's Detection Thresholds [3]

Graphs like these can be seen throughout the test results section of this report, despite the fact the waveforms being analyzed are of very different nature. Many useful approaches to handling and estimating based on IR data were found in these projects, but this work aims to estimate specific characteristics of the intruding objects rather than tracking them or estimating direction or count.

	Sensor Arrangement	Lens	Goal
This Work	Four on an axis (parallel to test subject)	TBD (likely a very narrow long range FOV custom lens)	Speed and width estimation (object classification)
Gopinathan et al. 2003	Four in a grid (above test subject)	Custom "grid" lenses	Object tracking
Song et al. 2008	Custom (throughout a region)	Fresnal lenses	Object tracking
Zappi et al 2010	Two sensor mirrored clusters (throughout a hallway)	20° FOV limited Fresnal lens	Object tracking and classification
Hao et al. 2009	Eight per two column sensor module (multiple modules arranged at the end of halls)	Custom slices of FOV exposed for each stacked sensor	Object tracking and classification
Zappi et al 2007	Three sensors along a hallway	17° FOV limited Fresnal lens	Object classification (count and direction)

Table 1.2: Comparison to Previous Work

1.3: RELEVANT EQUATIONS

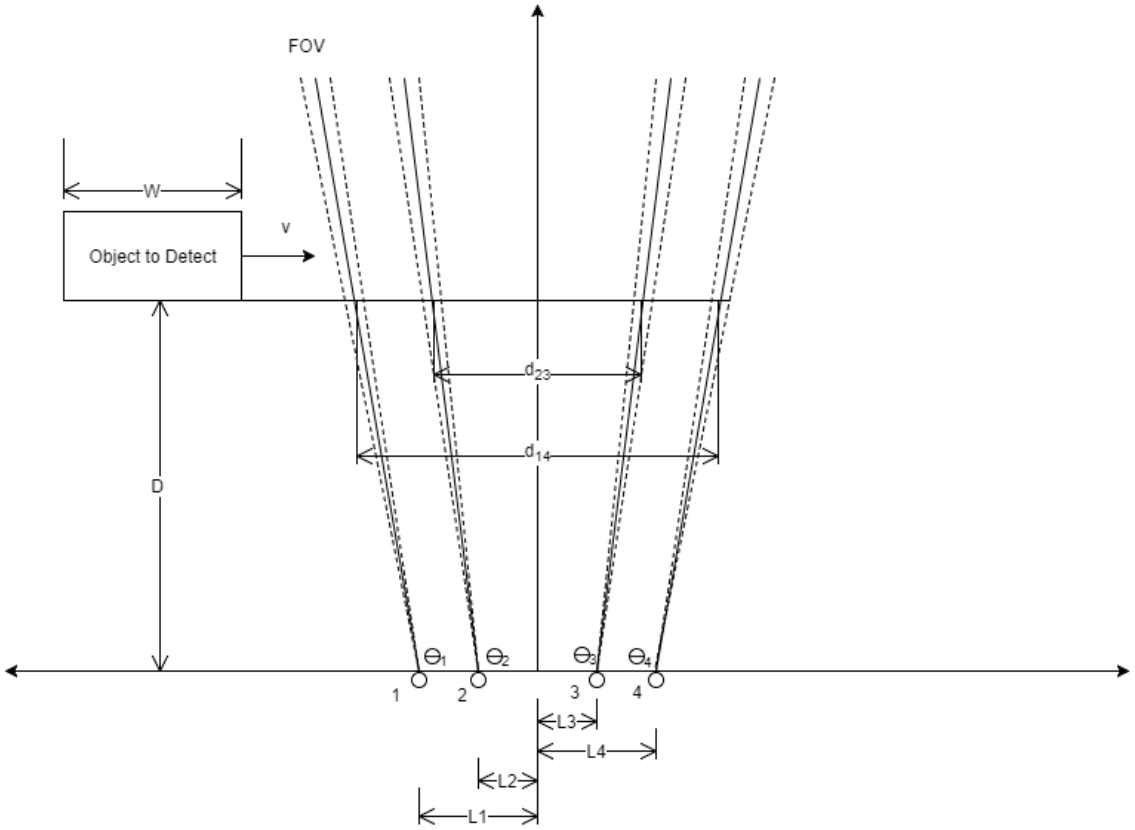

Figure 1.4: System Geometry Revisited

The goal of this embedded system is to produce an accurate output of sensor detection times, which are then used to calculate relevant time differences (Δt) to solve a geometric system of equations and obtain the speed and width estimates. The relevant Δt are the differences between detection times of sensors two and three (Δt_{23}) and the differences between detection times of sensors one and four (Δt_{14}). In the context of this system, all Δt are therefore considered to be known, and the average value of the times detected by one sensor during an intrusion will be called the sensor time (objects will trigger multiple detections times, the average of these is the sensors 'detection time' used in the Δt subtraction).

Distance from Time and Speed (1)

$$d_{14} = s * \Delta t_{14}$$
$$d_{23} = s * \Delta t_{23}$$

Distance Based on the Geometry of the System (2)

$$d_{14} = L1 + L4 + D * \tan\left(\theta_1 - \frac{\pi}{2}\right) + D * \tan\left(\theta_4 - \frac{\pi}{2}\right)$$
$$d_{23} = L2 + L3 + D * \tan\left(\theta_2 - \frac{\pi}{2}\right) + D * \tan\left(\theta_3 - \frac{\pi}{2}\right)$$

Combining (1) and (2)

$$L1 + L4 + D * \tan\left(\theta_1 - \frac{\pi}{2}\right) + D * \tan\left(\theta_4 - \frac{\pi}{2}\right) = v * \Delta t_{14}$$
$$L2 + L3 + D * \tan\left(\theta_2 - \frac{\pi}{2}\right) + D * \tan\left(\theta_3 - \frac{\pi}{2}\right) = v * \Delta t_{23}$$

Creating a Linear Equations with Respect to the Unknowns

$$\frac{L1 + L4}{\Delta t_{14}} + D * \frac{\left[\tan\left(\theta_1 - \frac{\pi}{2}\right) + \tan\left(\theta_4 - \frac{\pi}{2}\right)\right]}{\Delta t_{14}} = v$$
$$\frac{L2 + L3}{\Delta t_{23}} + D * \frac{\left[\tan\left(\theta_2 - \frac{\pi}{2}\right) + \tan\left(\theta_3 - \frac{\pi}{2}\right)\right]}{\Delta t_{23}} = v$$

Solving for D

$$\left(\frac{L1 + L4}{\Delta t_{14}} - \frac{L2 + L3}{\Delta t_{23}}\right) + D * \left(\frac{\left[\tan\left(\theta_1 - \frac{\pi}{2}\right) + \tan\left(\theta_4 - \frac{\pi}{2}\right)\right]}{\Delta t_{14}} - \frac{\left[\tan\left(\theta_2 - \frac{\pi}{2}\right) + \tan\left(\theta_3 - \frac{\pi}{2}\right)\right]}{\Delta t_{23}}\right) = 0$$

With these equations, knowing the dimensions and angles of the sensor array allows estimation of distance and velocity from the stream of detection times.

Estimating Width

$$W = \Delta t_{Sensor} * v$$

This equation can be used to estimate the width of the object based on the duration of time individual sensors detected its presence and the estimated velocity.

Embedded sensor networks provide almost unlimited potential applications. By utilizing cheap COTS sensors as well as processors, tracking and characterizing different phenomena can be accomplished. This report proposes a novel system of four IR sensors to monitor the speed and width of objects moving past it. Previous work has shown characterizing intrusions, as well as tracking intrusions is possible, even with the lack of precision afforded by PIR sensors. Equations have been derived to extract the speed and width estimates from the stream of detection times of the sensors. The next step is to define a system prototype.

Chapter 2: Prototype Design

2.1 Sensor Selection

For the prototype it was determined that the sensors should meet the following criteria: range of at least 75 feet, a small field of view, and a reasonable sample rate. The range of 75 feet allows the geometry of angles of the sensors to spread out enough for full scale (i.e. human or vehicle) tests. The small field of view allows each sensor to be treated close to a 'pixel' of detection, and the smaller the field of view, the more accurate the estimation can be. The sample rate of the device determines how fast the test objects can move, a full scale vehicle test would require a high refresh rate for a camera, but a low sampling rate for an IR sensor (30 – 60 Samples Per Second (SPS) performs well on most vehicle speeds according to simulation). Given these characteristics and a budget, several sensors presented themselves as options.

Four sensors were considered for this report. The first sensor considered was a simple passive infrared sensor, to be connected through an ADC or GPIO to the Raspberry Pi. The second was an off the shelf IR security sensor which would also interface to the Pi through a GPIO or ADC. The third option is a standard web camera with a USB connection to the processor. The final option is the FLIR Lepton infrared camera. This option interfaces to the Pi in one of two ways: SPI data and I2C command line or over USB like a webcam with a breakout board.

PIR Sensors

A standard PIR sensor has a range of up to 20 feet and a wide field of view. This can be narrowed by using lenses, the smallest of which produced a 10° FOV. The signal out of a PIR sensor would need to be sampled by an ADC to analyze the waveform for intrusions, as shown in Figure 2.1. Circuits also exist to simply raise a GPIO pin if an

intrusion is detected, however is not sufficient for this report as it only provides one bit of resolution. The power consumption of these sensors can be relatively low, and they are easily extended to an array of four with a multi-channel ADC or multiple GPIOs. Price is another strong point for this type of sensor as they are very cheap. Unfortunately for this prototype the PIR sensor's range and wide FOV make them nearly unusable. Ultimately in order to create a low power PIR solution a custom lens will need to be developed, however the lens design is beyond the scope of this and budget of this report.

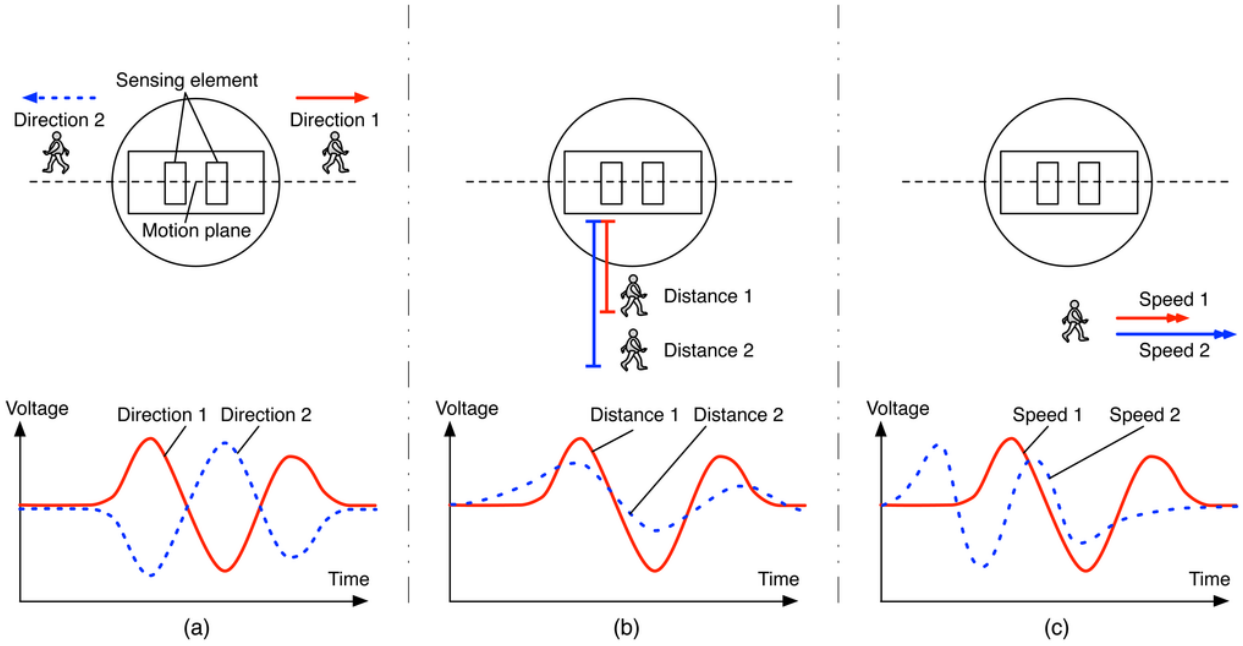

Figure 2.1: Standard PIR Detection Waveforms [7]

Takex PIR-50NE (COTS Sensor)

The Takex PIR-50NE is an off the shelf security solution with a PIR sensor at its core that solves the FOV and range problems of the standard PIR sensor. With a 165 foot range and 3.5° FOV this sensor has great range and narrow spread. The weaknesses of this sensor come from implementation complexity, power consumption, and cost. All TAKEX

sensors are produced ready to be integrated into existing security systems, meaning the device would need to be opened and modified to give an ADC direct access to the waveform from the PIR sensor at its core. In the out of the box state, the sensor sends a 20V 100 mA pulse for two seconds upon an intrusion detection, a sampling rate that is much too low and a power consumption that is much too high. With an individual cost of $520 this sensor also boasts the highest price. The TAKEX PIR-50NE contains a sensor and lens system that is nearly ideal for this report, however the price and unknown complexity of modifying the sensor hurts its viability.

Web Camera

The third option was using a basic web camera as the sensor. This option provides flexibility in FOV, great range, low cost, and is easy to implement. However, these cameras simply don't monitor the spectrum of interest for this report. A different image processing project could be completed using web cameras, but for this report an IR sensor is necessary.

FLIR Lepton 3

The final option was the FLIR Lepton 3 IR Cameras. This camera has a 160x120 resolution which allows for flexible FOV settings, a good range of 100 feet, medium cost, and low power consumption. The PureThermal 1 breakout board changes the interface from SPI video and I2C command to UVC USB video format. USB interfacing is found in nearly every off the shelf embedded processing solution, allowing quick and easy compatibility. On an embedded board running Linux the v4l2 (video for Linux 2) interface allows frames of data to be captured and processed in many languages (C and Python notably). The weakness of this option is its 9 FPS. This sample rate is very low; however this can be worked around by slowing test objects. Due to cost, range, and flexibility the

FLIR Lepton 3 Cameras with PureThermal 1 breakout board was chosen as the sensor for this system's prototype.

2.2: SYSTEM DESIGN

When considering the processor to use for this project four criteria were present: compatibility with the selected sensors, off the shelf availability, and my familiarity with the platform, and ease of deployment into the test environment. Off the shelf boards like the BeagleBone line of development boards, Arduino Uno, and Raspberry Pi were considered. Of these embedded processing solutions, only one fulfilled the two remaining criteria (easy interfacing to sensors and familiarity), the Raspberry Pi. While all platforms provide easy USB interfacing, my familiarity with the Raspberry Pi lead me to select the Raspberry Pi 3B board as the processing element of this project.

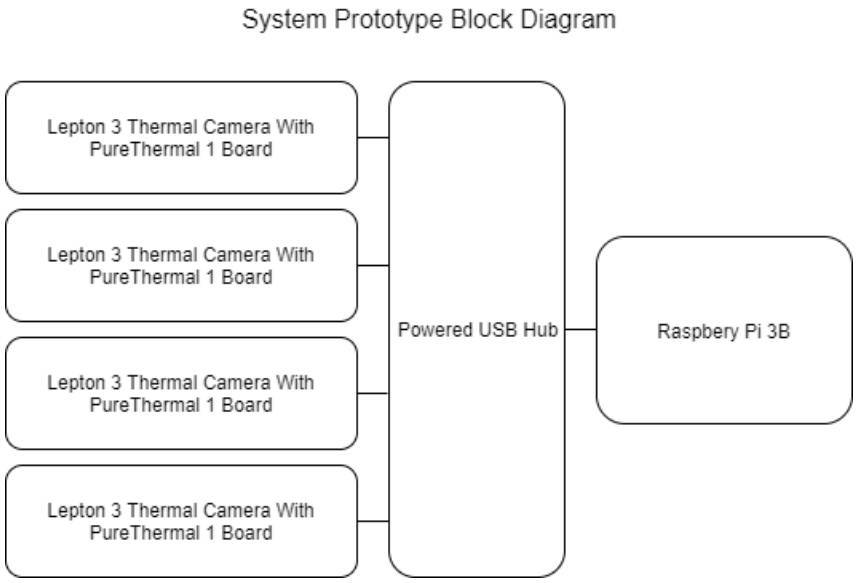

Figure 2.2: System Block Diagram

With the prototype architecture defined as a Raspberry Pi 3B with four Lepton 3 cameras, the remaining piece of hardware is a simple powered USB hub to allow mouse and keyboard connections as well as ensure proper power distribution to the cameras.

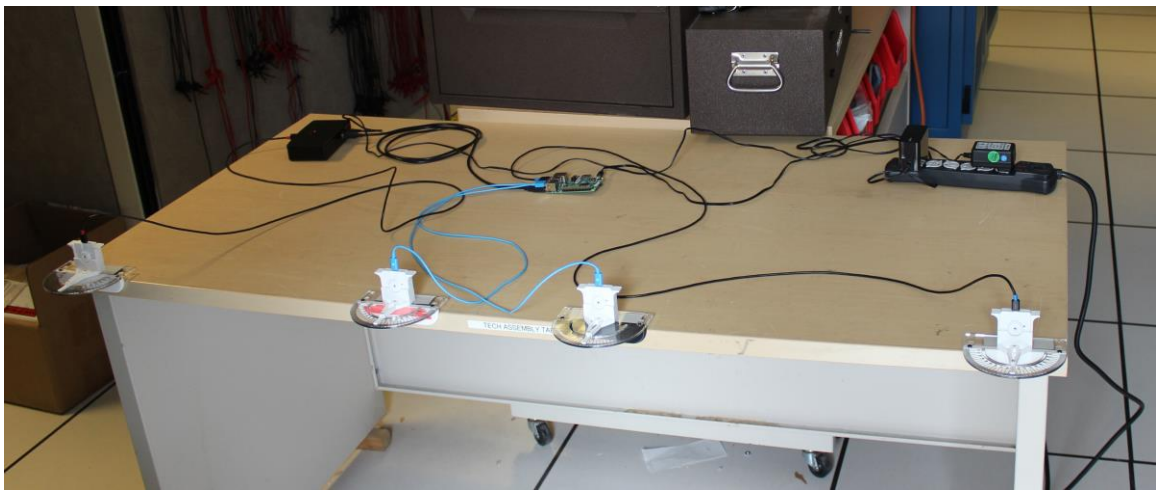

Figure 2.3: System Prototype

Firmware for compatibility between the third version of the Lepton camera and the PureThermal 1 is located on GitHub, as the previous versions had half the resolution, and therefore the older firmware is not compatible. After working with v4l2 libraries (in python and C), data from the cameras was successfully captured. A simple Python script (which was later converted to C to similarly validate data was good) generated a sketch of the thermal measurements by creating an array of Tkinter rectangles on a canvas with each rectangle's color scaled between black (cold) and white (hot) based on that pixels value. The algorithm first finds the minimum and maximum values in the image so that this color is scaled relative to these local maxima and not the global maxima of a twelve bit unsigned integer. The resulting image is consistent with the thermal profile of the surroundings. My face can be seen to be warm relative to the surroundings and lights and desks can be seen

in the background having slightly different heat signatures than their surroundings. Given successful data acquisition, the next step was to write algorithms for detection and estimation and run them on the prototype.

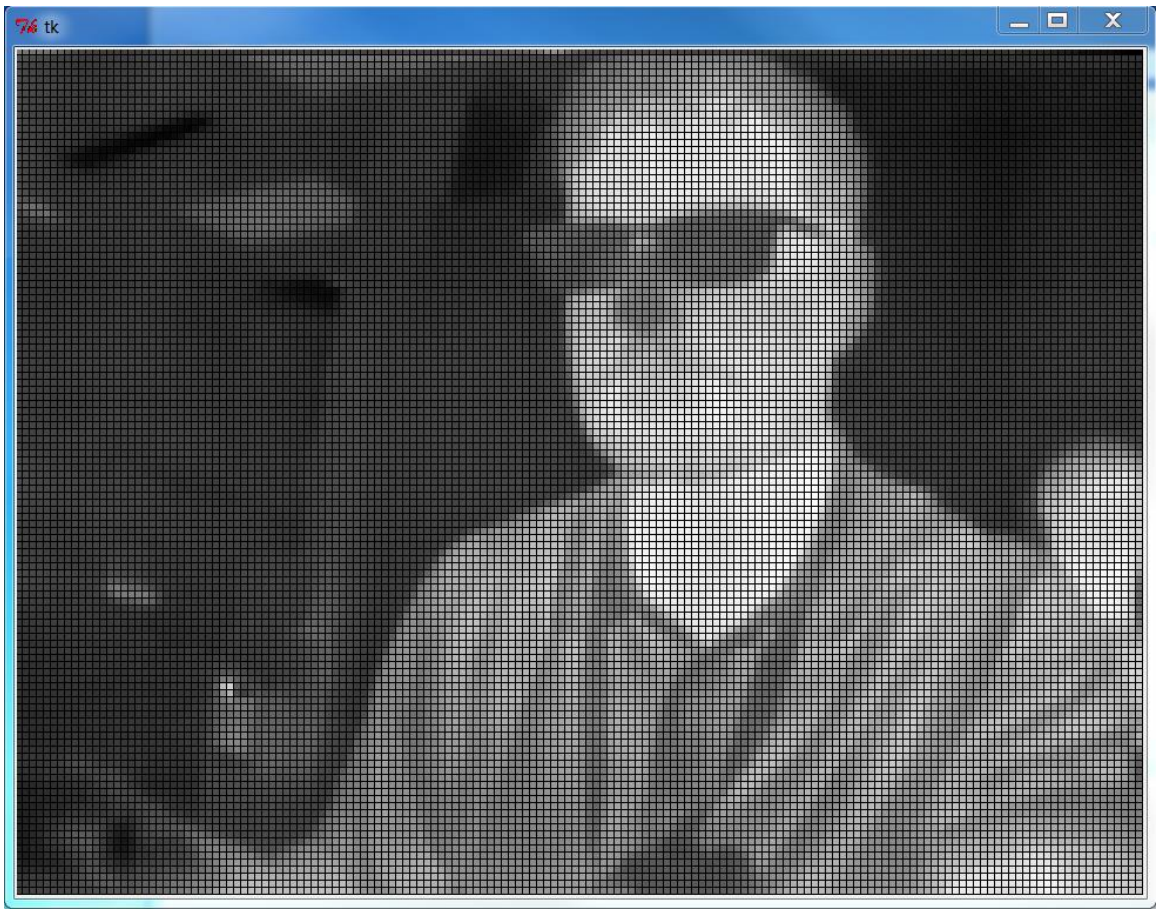

Figure 2.4: Successful Data Acquisition

2.3: CASES

GroupGets, the creator of the PureThermal1 breakout board for the Lepton Cameras, has posted the files for a 3D printable case for the camera and board. A base was created to mount these cases at a fixed angle to easily orient the sensors. This base consists of a protractor with a swiveling attachment on top of it for easy angle configuration, along

with mounting holes. Ultimately, these cases were positioned properly by attaching them to tripods with double sided sticky Velcro strips.

With the sensors, processor, and method for orienting the system selected, the next step in prototype development was developing algorithms and selecting test objects. However, while the prototype was being designed, a simulation of the system was being written to gain a deeper understanding of the geometry of the system, as well as the prototype's limitations.

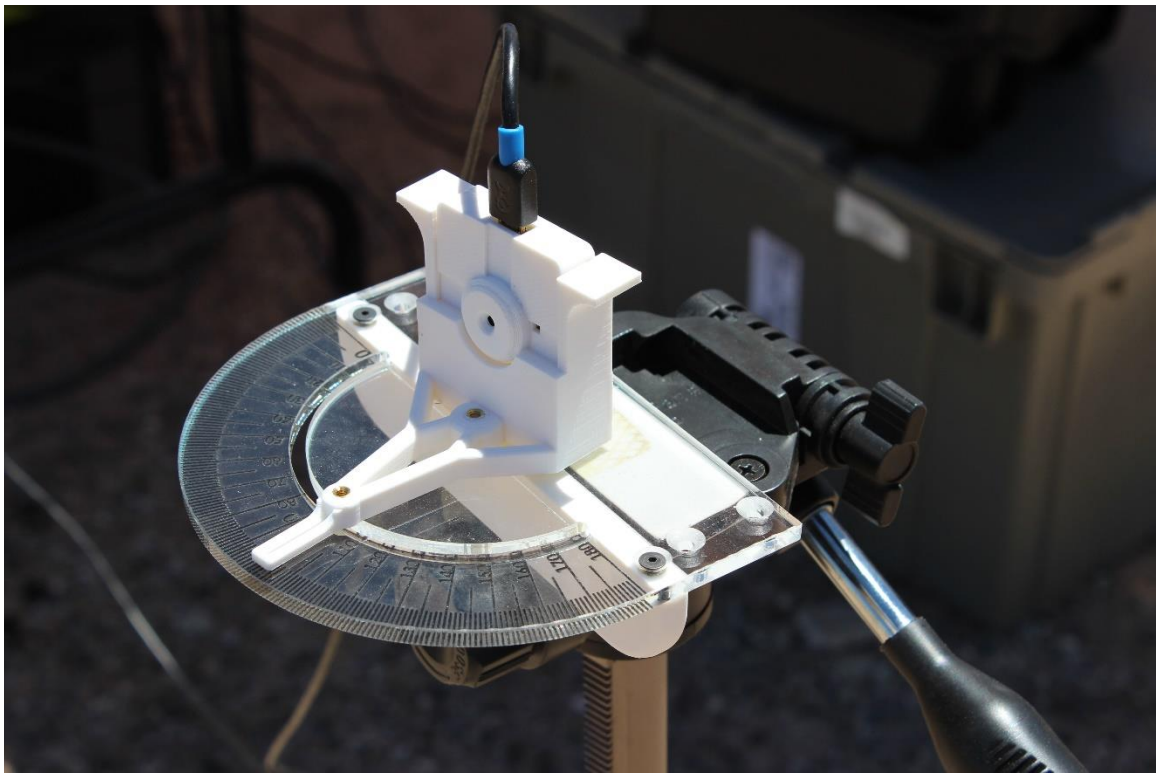

Figure 2.5: FLIR Lepton 3 Cases

Chapter 3: Simulation

3.1: Simulation Design

While the sensor-processor interface was being defined a simulation of the scenario was created to better understand the system as well as find optimal geometric parameters for the sensors and optimal testing conditions (width, speed, and distance of the test object. This simulation was written in Python 2.7. It functions by being given three sets of data: test object parameters, geometric parameters, and sensor parameters. Test object parameters are the distance, speed and width of the object passing through the system, the known truth that is being estimated. The geometric parameters define the distances apart and angles from the center the sensors are placed at. These are critical parameters for data that contributes to the estimation algorithm. Sensor parameters characterize the field of view of the sensor as well as the sampling rate. The test object data set is specified as a range of distances, speeds, and widths with a step size to iterate over. With the system configuration and test objects set, the program begins simulation of the scenarios specified by defining the sensor's beam width and x-values at the specified distance.

Next the test object is positioned beyond the most negative (left-most) sensor at the specified test distance (as shown by Figure 1.3). A loop then steps this object to the "right" across the sensor field by incrementing the front and back coordinates of the test object by the sampling rate multiplied by the speed of the object. This suggests the simulated system has a perfect sampling rate as well as uniform sampling times. As the test object moves across the sensor field, the range of x-values it occupies is tested against the range of x-values that each sensor monitors, if these two ranges coincide, the current time of the system is added to the a list of times for that respective sensor. After the test object completes its run through the sensor array, the average of each list of detection times are

found and the simulation uses these times to solve the relevant system of equations to return the estimated distance, velocity, and width estimates. For each of the test object parameters (a specific width, distance from the sensors, and speed) the simulation runs 10 tests at varying starting points within the distance the object can move within one frame (sample rate multiplied by speed) to determine a best, worst, and average estimate for this specific scenario. The simulation will output the percentage of the test set that are above 10% error when estimating velocity and the percentage of these tests above 20% error when estimating width.

A graphical user interface was developed to better show the test scenario specified in the simulation, as well as show the different snapshots the system would get of the object as it passed through the sensor field. First the geometric, sensor, and test objects of the system are specified.

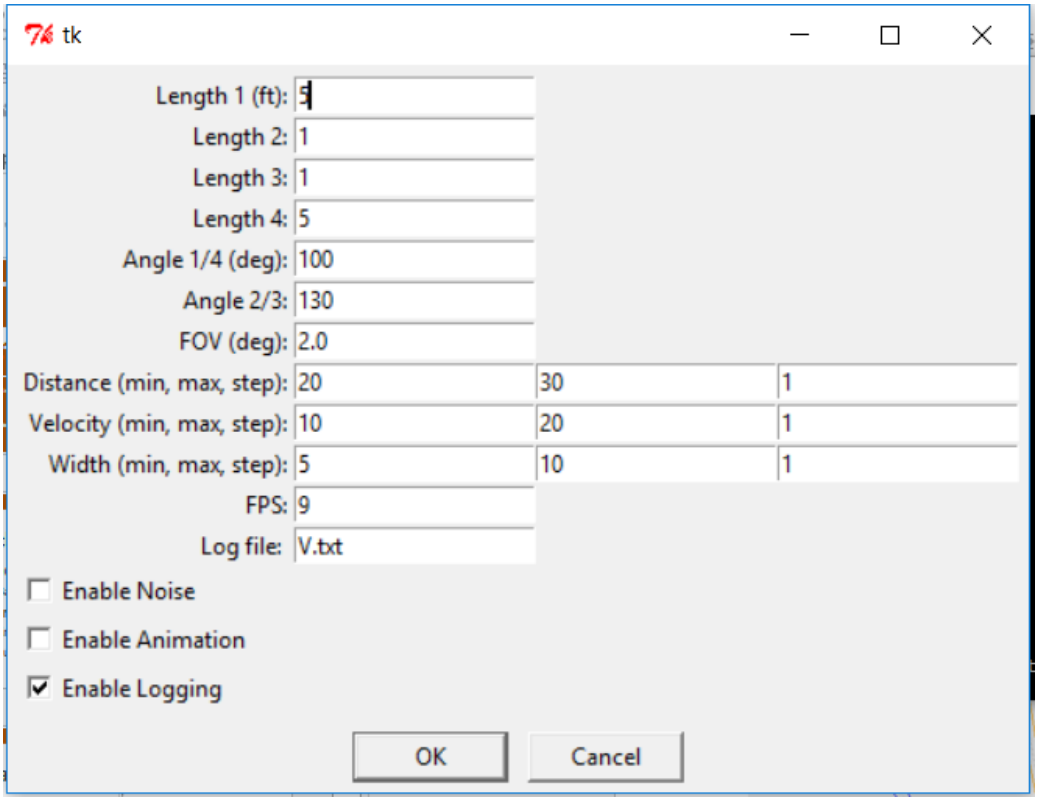

Figure 3.1: Simulation Parameter Selection

When the simulation begins, the geometry of the sensors are drawn on an x-axis, centered on a y-axis. This includes dotted lines to indicate the 'one pixel wide' perfect line of sight of the camera, as well as lines on either side of this to indicate the actual field of view of the camera. Figures 3.2, 3.3, and 3.4 show this GUI. As the simulation continues, an option can be checked to show the object's location at each subsequent sampling time. This illustrates how the test object steps through each sensor beam and allows better understanding of sensor misses and other anomalous behavior.

2.2: The Geometry of the System

The simulation was used to find an optimal geometric layout for the system. Many configurations of the sensors were considered. To determine which configuration was

preferred simulation results were considered alongside other qualities of the configuration such as the spread of the base (the sum of the horizontal axis lengths L1 – L4). To determine accuracy a wide range of test objects (characterized by distances, velocities, and widths) were used over several different geometric configurations to determine average performance.

Configuration One

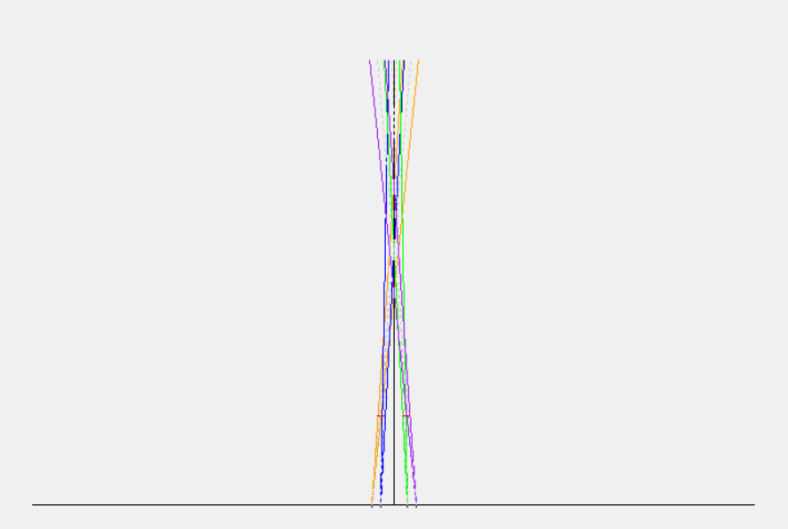

Figure 3.2: Sensor Configuration One

The first configuration considered had all the sensors at acute angles with respect to the x axis and the y axis (Figure 3.2). This proved neither accurate nor practical as the distances of the sensor from the base of the axis would need to be large to allow a decent range and spacing between sensors. Recording the same or very similar times for multiple sensors leads to inaccurate estimations.

The project proposal showed the sensors in this array, so for the first month of work this was the default configuration. This lead to several decisions that would later be to the

detriment of this report's prototype. For example the selection of the low frame per second sensors reduces the viability of commonly available test objects. If the configuration required acute angles toward the center, PIR sensors with basic lenses on them become unworkable. This is due to the fact that with off the shelf and cheap lenses, the field of view can be reduced to 10° and the range increased to approximately 20 feet, which is much shorter than the desired range. The spread of the sensors at the edge of the region of detection is 3.5 feet, suggesting for a good amount of the 20 foot range the signals would be colliding and aliasing together as the same detection time if it is assumed the sensors face inward. Having multiple sensors with the same average detection time results in division by zero in the system of equations to produce an estimate, and therefore should be avoided. Another issue with this configuration is the width of the base. Since overlapping sensors produce similar average times and decrease the ability of the system to estimate accurately or at all, this configuration favors sensors that are a spread out along the x-axis. This leads to several issues including difficulty in creating a reliable setup procedure or rig for the sensors given such a wide base, and ensuring connections for the prototype like USB cables are long enough, all contributing to the fact that a compact implementation is preferred.

When defining the system's prototype and selecting sensors, the design challenges of this configuration led to design choices that would greatly impact the success of the prototype, whereas later configuration options were not hampered by the cramped nature of this setup and would have resulted in a different prototype design. Not too long after selecting a purchasing the FLIR Lepton sensors, the geometry of the system was reconsidered, and more optimal geometries were found.

Configuration Two

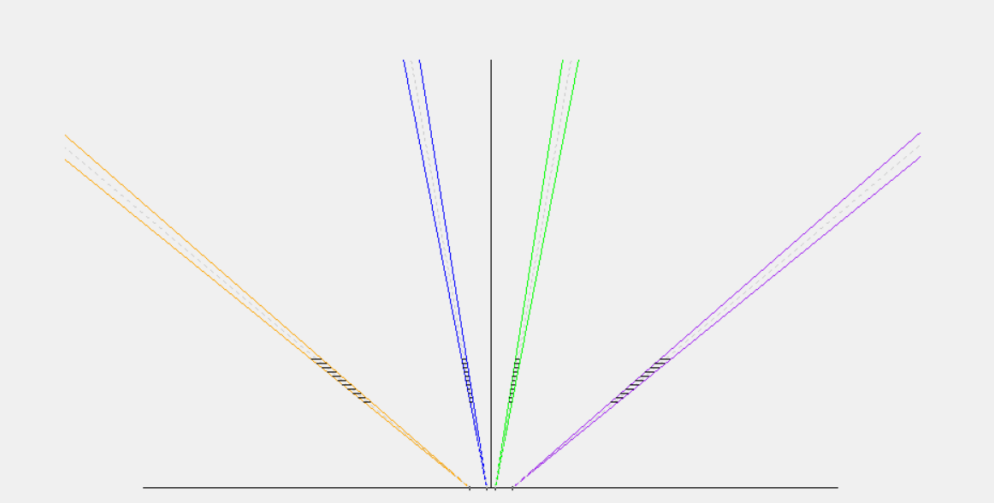

Figure 3.3: Sensor Configuration Two

The second configuration considered works considerable better for both accuracy and the horizontal size of the system, however, the two sensor beams in the middle are still closer than necessary, which can cause the sensor times to be grouped together, reducing the accuracy of the estimation.

Configuration Three

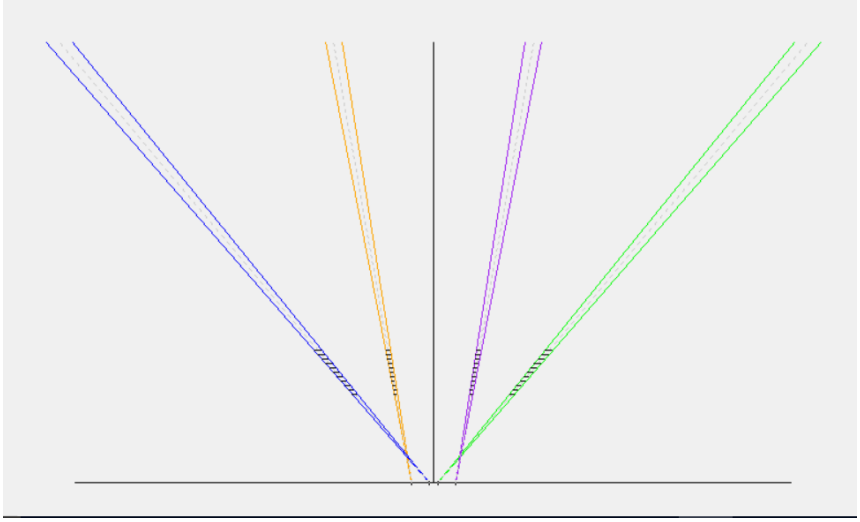

Figure 3.4: Sensor Configuration Three

The most accurate and space conscience configuration points the closest grouped cameras at the harshest angles, while allowing the further spaced cameras softer angles. This allows the system to be tightly spaced horizontally while still being accurate. It should be noted that with this configuration the object being observed and estimated should be beyond the points of convergence of the sensors, or aliasing can occur.

3.3: DETERMINING TEST OBJECTS THROUGH SIMULATION

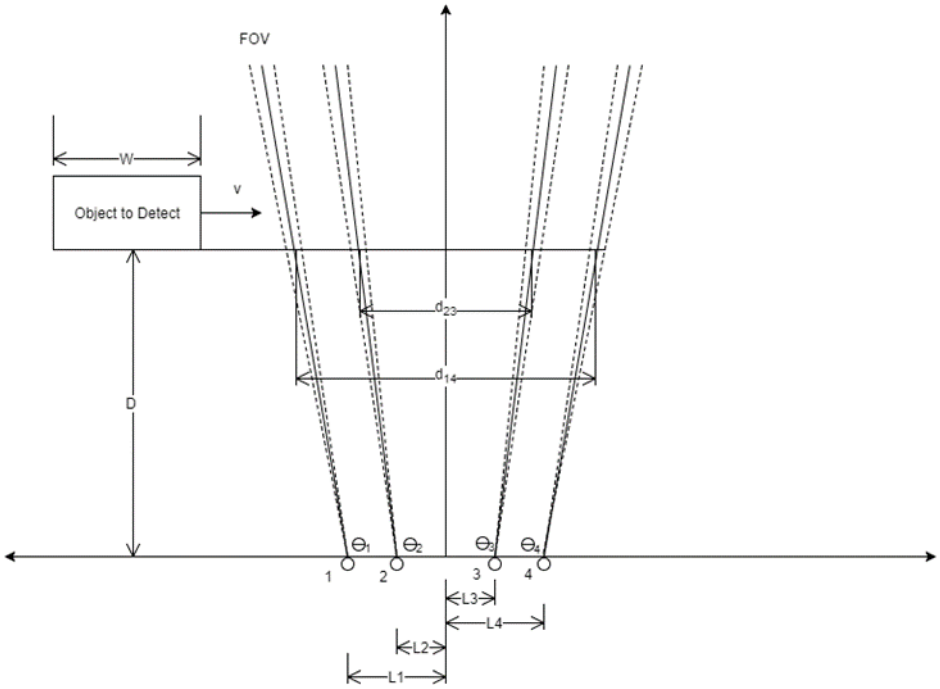

Figure 3.5: System Geometry

In order to define both the geometry of the system and the set of test objects, many scenarios were simulated. First relationships between the geometry of the system and the error were developed.

Optimal Angles

To determine the best angle for the two sets of sensors to use, the angles not under test are held at 90°, the base is set up for the outside sensors to be ten feet from center, the inside sensors are five feet from center, and the field of view and frames per second were set to 2° and 9 FPS respectively based on the FLIR Lepton 3 camera. The distance is set from 30 to 40 feet, the speed set from 5 to 15 mph, and the width is set to 5. All of these parameters are meant to showcase a favorable test setup, with the base being wider than desired in order to consider acute angles.

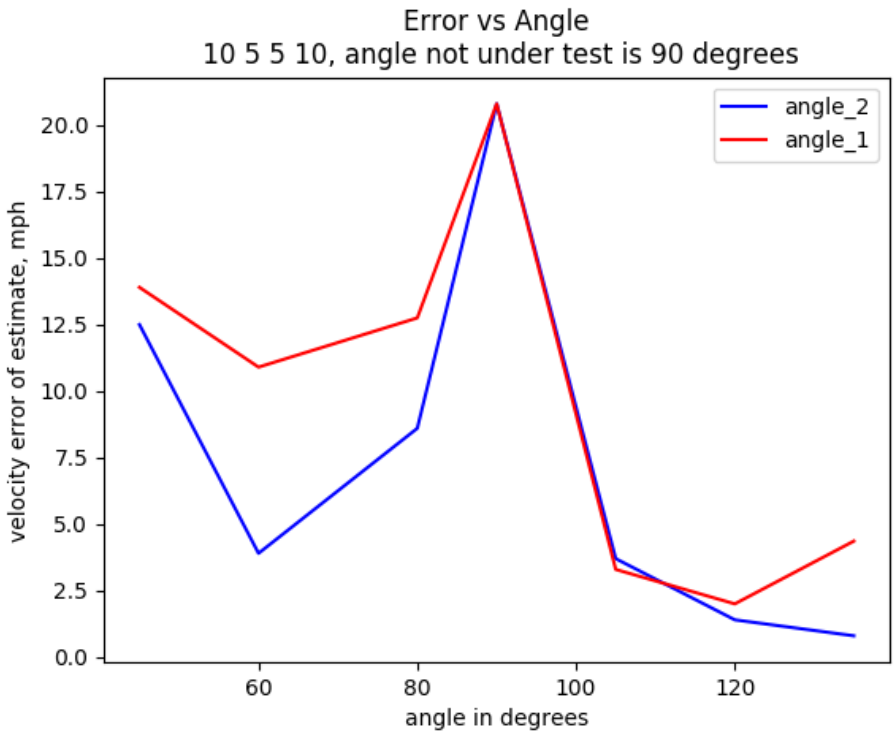

Figure 3.6: Angle of Cameras vs. Error in Velocity Estimate

The results suggest the wider the angle, the more accurate the estimation, which agrees with previous reasoning about conflicting regions of detection and decreased sensor accuracy. When sensors are pointed further away from each other, accuracy increases. It is interesting that a local maximum occurs when the angles are equal, suggesting that the estimation equation does not work well when the two geometries used are similar (in this case two rectangles). Given a minimum approximately one mph estimation error occurred when angles 2 and 3 were set to 130°, the test was repeated holding angles 2 and 3 at 130° to determine the optimal angles 1 and 4.

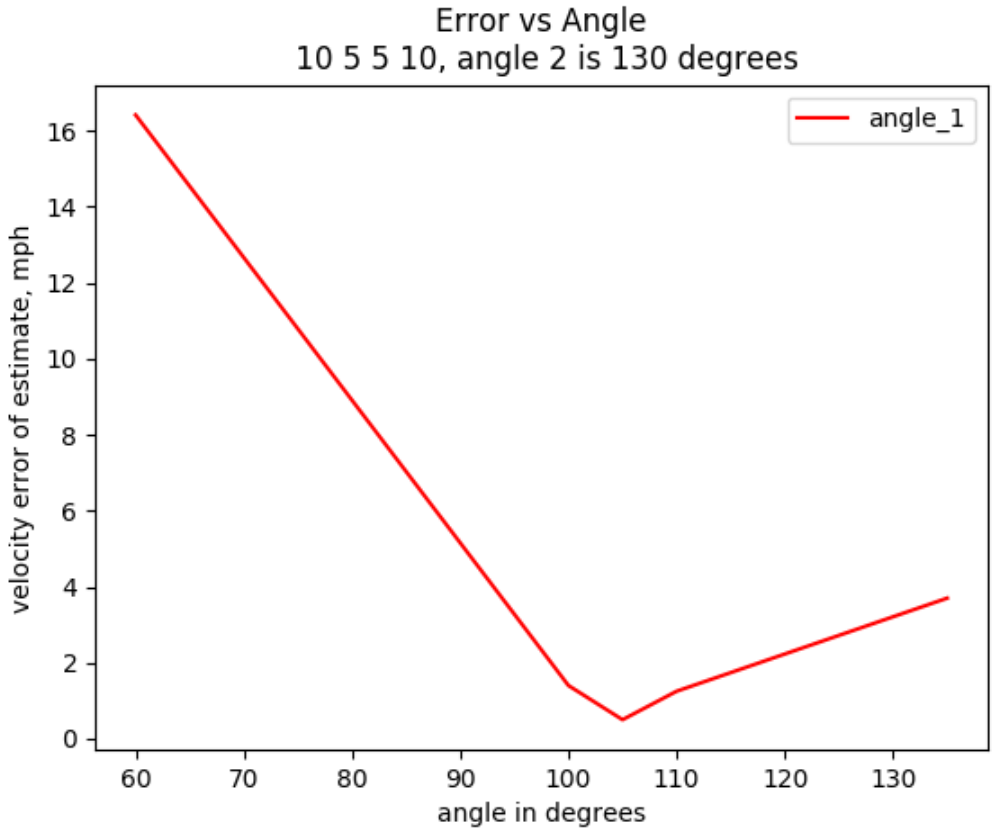

Figure 3.7: Angle One vs. Error

Here the velocity error was minimized beneath 1 mph per estimation when angles 1 and 4 were at 105°. Moving forward the optimal angles for a test configuration were considered to be 105° for angles 1 and 4, and 130° for angles 2 and 3.

Optimal L1 – L4 Distances

The next step in optimizing the geometry of the system is determining the best distances from the center of the system to position the sensors. Again, sensors 1 and 4 (distance 1) were considered mirrors of each other, as were sensors 2 and 3 (distance 2).

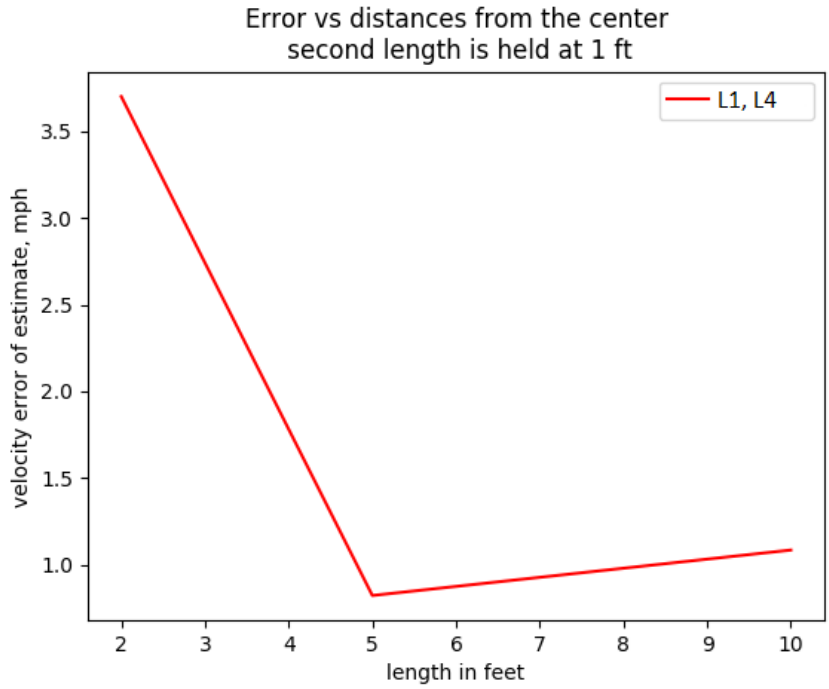

Figure 3.8: Outer Distances vs. Error

The first test set the angles to their “optimal values” while holding distance 2 at 1 foot in order to consider the smallest configuration possible (2, 1, 1, 2) while biasing the system towards a smaller setup. A minimum of viable sensor distances is found when the outside sensors are placed five feet from the center. Given this optimal location, the set of options for distance 2, the distance of the middle sensors from the center was now {1, 2, 3, 4} feet.

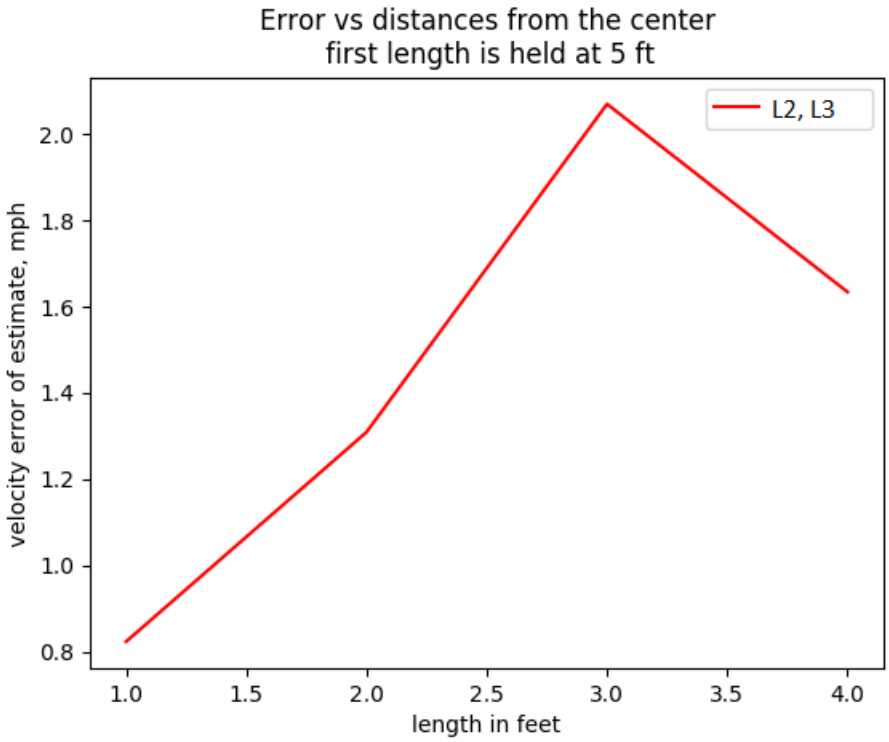

Figure 3.9: Inner Distances vs. Error

When testing this set, the minimum value is found to be where the inside sensors are furthest from the outside sensors. Given the inside sensors are at a hard angle of 130°, the range in results are much more tightly grouped than previous tests (all are within about 1.2 mph of error). The optimal geometry of the system is determined to be outside lengths of 5 feet, inside lengths of 1 foot, outside angles of 105° from center, and inside angles of 130° from center.

3.4: Determining Test Objects

The next step towards deploying and testing a prototype width and speed estimation device was to determine optimal test object parameters (characterized by a distance from the sensors, speed, and width) to test the system with. The first test determined a testable range of speeds that would result in a reasonably accurate output. For this test the distance

and width (10 feet) are held constant (multiple distances are tested) in order to isolate the effect speed has on the accuracy of the system. For this part of the analysis, Configuration 1 was considered the previously found optimal geometry of the system.

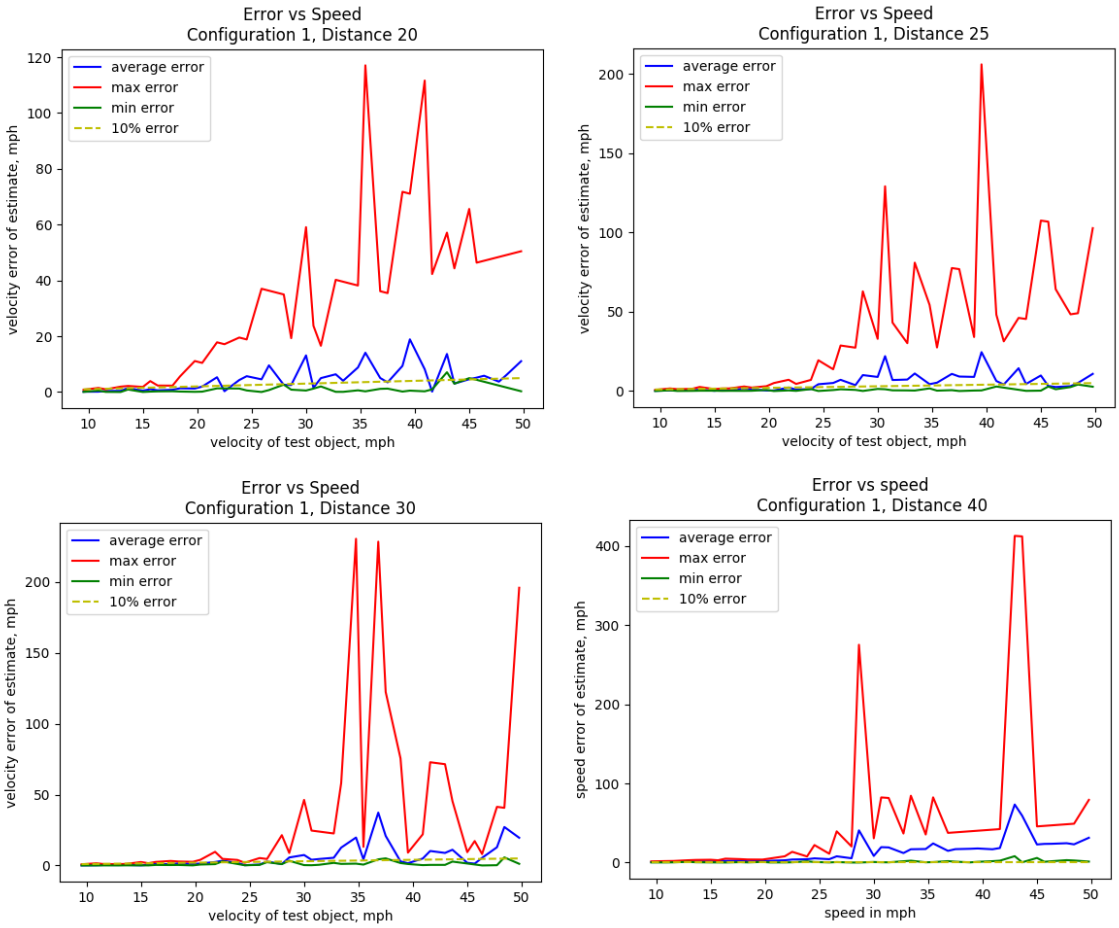

Figure 3.10: Determining Appropriate range of Speeds for Test Objects

From a distance of approximately 25 feet to 40 feet and a range of speeds up to nearly 20 mph the minimum and maximum error graphs bound the error by close to 10%. Thus test objects with a distance range of [25, 40] feet and a speed range of [0, 20] mph or some subset of these ranges would lead to an accurate test. It should be noted that the

effects of the low samples per second of the system were clearly shown in this test. The error bound grows exponentially when the speeds of the test objects rise above 20 mph.

The next test was to determine an optimal width for test objects given the previously determined speed and distance ranges. The same geometric configuration is again used, and constant distances and speeds are picked from within this subset to test a range of widths on. The range is selected to be from one foot to fifty feet in order to include possible test subjects ranging from a person to a semi.

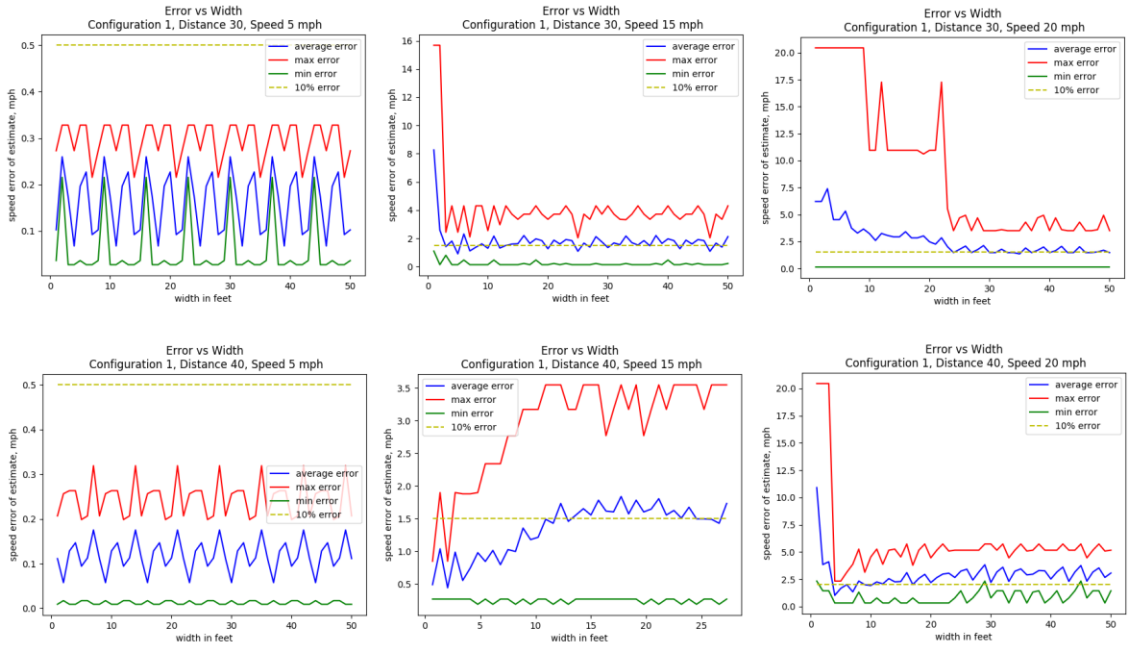

Figure 3.11: Error vs. Width given previous Distance and Speed ranges

It should be noted that an error of 16 for a 15 mph test indicates a sensor miss caused the estimation algorithm to fail (and return -1), or in general an error of the speed plus one indicates sensor misses led to failure. Thus when a high speed and a small width combine, the maximum error is often this number, as seen by the smaller widths on the 15 mph and 20 mph graphs. From these graphs and analysis it can be seen that distances from

30 to 40 feet perform well with a speed range of 5 to 15 mph and width of around 5 to 10 feet.

Thus test objects for this test can optimally be defined as objects with D = [25, 40] feet, S = [5, 15] mph, and W = [5, 10] feet. Running the simulation over these values with the optimal geometric configuration yields only 0.38% of tests above speed estimation error. Further, to characterize a test object as a golf cart, with a standard width of 8 feet, D = [25, 40] feet, and S = [5, 15] mph the simulation returns 0% of tests have a speed estimation error of above 15%. Given these conditions, optimal test object is a golf cart – capable of producing the range of speed and of the width desired. However, due available resources, a Chrysler Town and Country with a width of 17 feet is instead used (with the added benefit of built in speedometer).

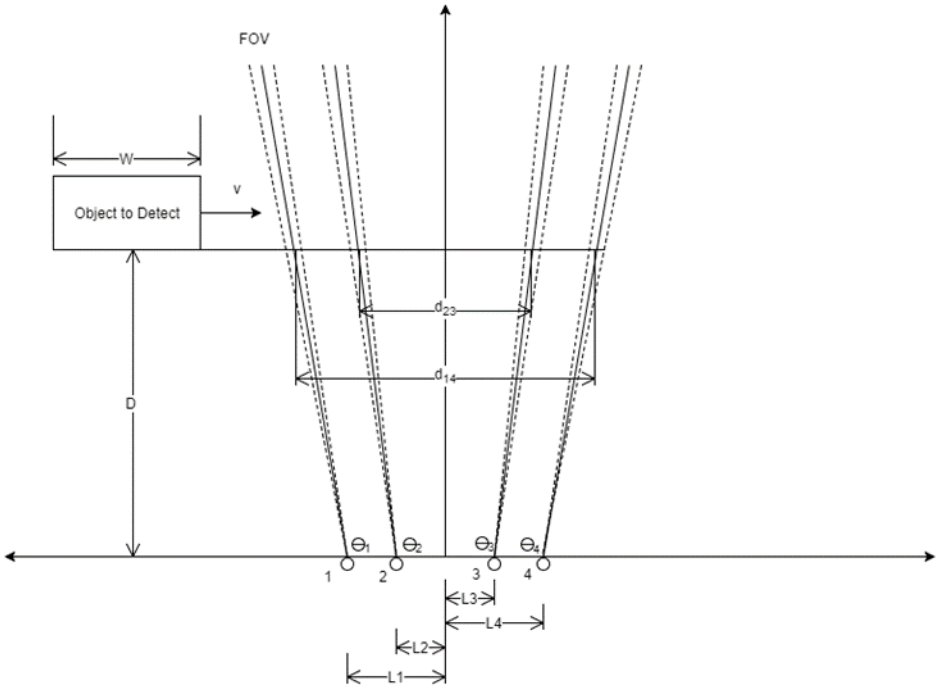

Figure 2.12: System Geometry for Reference

L1, L4	5 feet
L2, L3	1 foot
A1, A4	105°
A2, A3	130°
Distances for Test	[25, 40] feet
Speeds for Test	[5, 15] mph
Width for Test	8 feet
Expected Accuracy of Estimates	Within 15%

Table 2.1: Ideal Test Conditions, as found by simulation

Analysis of the system's geometry and parameters lead to a simulated model of the system's context. Analyzing this simulation lead to ideal geometric configurations, sensor characteristics, and test objects. The next step was to verify the simulation and its results with real world data. To accomplish this algorithms for intrusion detection and estimation were developed in parallel to the simulation's development. Programs to gather raw thermal data and data from the intrusion algorithm were written for the system prototype.

Chapter 4: Algorithms

4.1: Detection Algorithm

The goal of the detection algorithm is to allow for the instability of ambient thermal background noise while accurately recording intrusion times. To achieve this, a sample average is kept for a specific window of pixels from the Lepton cameras. To approximate a two degree field of view, a six by six square of pixels from the center of the camera was analyzed. Each of these pixels has a stored window of N samples with the mean, median, and standard deviation of this window. For the duration of this work, N is equal to 32 samples. The standard deviation is computed through Median Absolute Deviation.

When new data arrives each pixel will raise an alert given the data is beyond a certain statistical threshold. Samples are next added either to the N sample history of the pixel if they lie within a thermal anomaly threshold, or they are rejected and the sample history remains unchanged. The samples are stored in ascending value order, each sample with two values: a twelve bit thermal value from the camera, and a sample age between N (new, 32 in the current implementation) and zero (the oldest sample). The algorithm adds new data to the N samples by removing the oldest data and shifting the remaining samples into the lower elements of the array while also decrementing their age, leaving the top element empty. A binary search places the new data point in the proper place in this N-1 sorted list, and moves the values larger than the new data into the upper positions of the array. While the algorithm sorts the data, it tracks the index of the oldest piece of data (next to be removed) and stores this inside a pixel data structure which also includes the sample array and statistics. Finally the new statistics are calculated. The mean is obtained from a simple addition and shift in the current algorithm (the sum of the previous values are computed during the sorting loops, the shift by 5 divides by 32), the median is simply the

value in the middle index since the list is sorted, and the standard deviation is obtained through the MAD calculation [6]:

$$MAD = median(|X_i - median(X)|)$$

$$MAD = \sigma\sqrt{2}erf^{-1} \approx 0.67449\sigma$$

Figure 4.1: Median Absolute Deviation Equations

Several variables can characterize a specific implementation of this algorithm: the window size kept (N), the statistical threshold to determine intrusions (a multiple of standard deviation), and the threshold to determine thermal anomalies. After extensive testing in the same thermal environment outdoors, it was noted that between frames – or in approximately one ninth of a second – pixels could jump around ten to twenty in infrared value. The Lepton's accuracy of less than 0.050° C suggests that this jump would be at most 1° C between samples, which could be attributed to natural phenomenon such as natural heating or cooling during the day or sudden cloud cover. Given events like these do not reflect the changing ambient temperature as much as short transient events, they are to be excluded from the running average. Thus the threshold to consider a change in pixel value a thermal anomaly was set to 30 pixel value or 1.5° C to mitigate the effects of such temporary events on the samples and their statistics. The commonly used N equals 32 sample window counts for between 4 and 3.55 seconds, often providing a low standard deviation. Due to the low standard deviation and somewhat large thermal anomaly threshold, a large deviation coefficient is required to be detected as an intrusion. For example, two to four are commonly found to be a pixel's standard deviation given 32 samples, requiring five to six sigma thresholds in order to consider an alert any disturbance

greater than one degree Celsius. Despite this lack of statistical granularity, the combination of these two thresholds proves accurate in detecting intrusions in the field.

Given the detection algorithm works at a pixel level, a new threshold is introduced that determines frame detection. As samples are processed, a certain number of pixels must indicate change for the algorithm to determine an intrusion occurred during this frame and report the current time to the intrusion list for the particular camera. After experimentation a frame threshold of twelve changed pixels seemed reasonable in producing no false positives and only a few failures to detect.

4.3: Detection Algorithm Requirements

When designing an algorithm, it is important to consider the time and memory characteristics in order to better understand the architectural requirements a system running the algorithm. In order for the estimation algorithm to run, the microprocessor must hold four times the window size times the number of samples kept. In the experimental setup, the window was six by six pixels, with each pixel storing 32 previous two-byte values. This results in a requirement of 9.216 KB just to hold the sample data, and the other data the algorithm needs will push the memory requirement to at least 10 KB. Using the sorting methods described in section 4.2 takes O(N) time, as the list will at most be traversed once for each time it is searched (moving values into the lower N-1 values of the array and moving the larger value back into the upper part of the array). It should be noted that the O(N) time for the overall algorithm has a large coefficient, as the window size and number of cameras become multipliers for the number of times this algorithm is run per sample.

4.2: Estimation Algorithm

Throughout the tests documented in this work, only detection was attempted on-line during data gathering (raw data was also gathered) and the estimation algorithm was

purely used in off-line post processing. Thus simply transplanting the simple system of equation solving function from the simulation proved to be the quickest way to create an estimator. This function can be directly lifted from the estimation function written for the simulation. Determining how to group intrusions and extract speed and width estimations efficiently is included in the Chapter 9.2. Despite the lack of estimations, the detection algorithm was implemented in C and connected through the Raspberry Pi 3 to streams of IR data from the FLIR Lepton 3 cameras. The next step is to test the performance of these algorithms and ultimately compare estimates based on intrusion data from field tests to those of the simulation.

Chapter 5: Test One

5.1: Test One Setup

The first test site took place in an abandoned neighborhood on Kirtland Airforce base from 9 am to 12 pm on July 31, 2017.

Figure 5.1: Test Site

The sensor array was set up 25 feet from the straight road inside of the loop. The farthest two cameras were configured five feet from center focused at an angle of 140° from center and the middle two cameras were placed one foot from the center at an angle of 100° from center. This experimental setup was close to the optimal setup, and tested the edge of detectable speeds. It should be noted that the simulation had been tested to find the optimal geometry when this data gathering was conducted, but the optimal solution was found later.

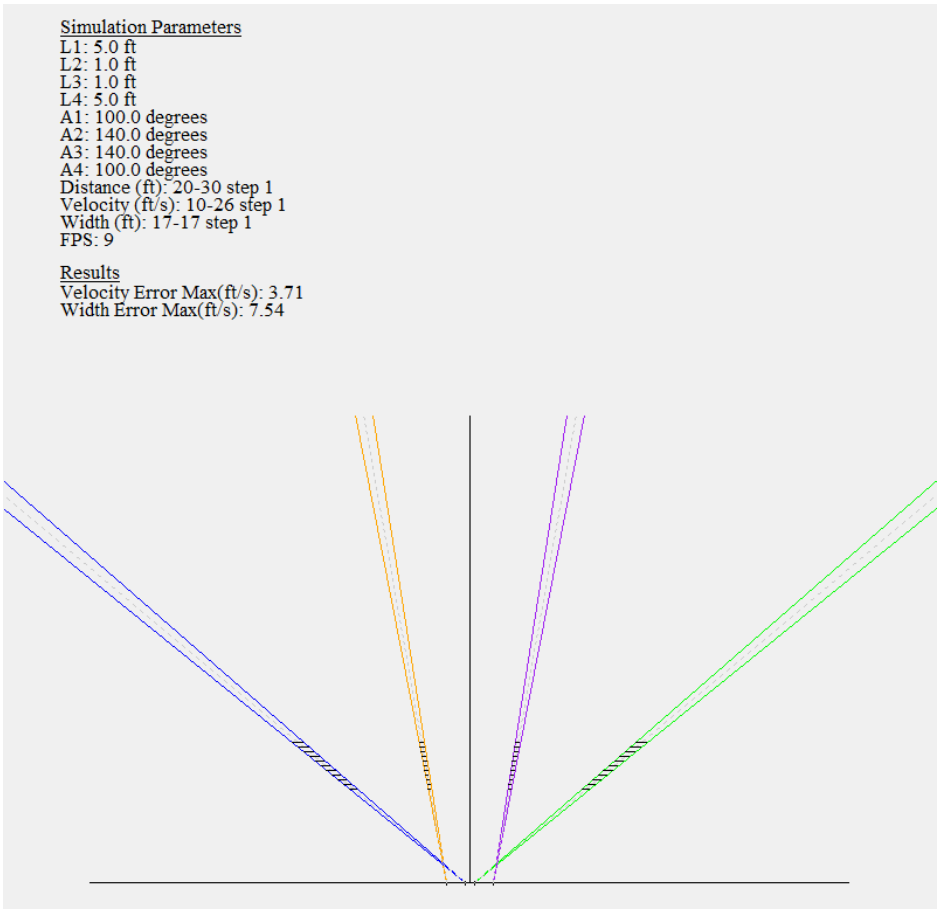

Figure 5.2: System Geometry for Test One

The temperature rose from approximately 70° Fahrenheit to slightly over 80° Fahrenheit. A fairly linear increase of 10° from 10 am to 12 pm.

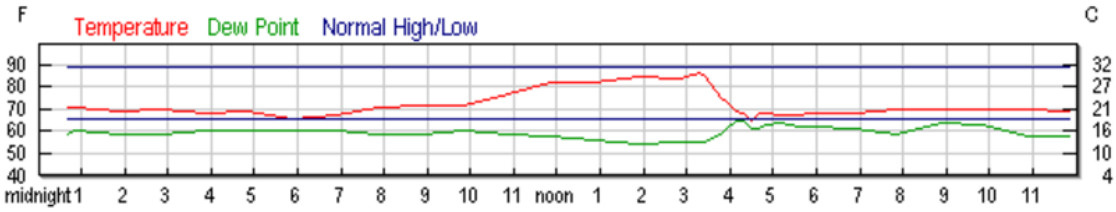

Figure 5.3: Temperature environment of Test 1 [10]

A dodge caravan van (17 feet wide) was driven at 15 – 25 miles per hour in both directions along the straight road as one test subject; a person walking in front of the cameras was used as the second test subject. The Raspberry Pi, cameras, monitor, and input devices were powered with a large battery. A powered USB hub compatible with the Raspberry Pi had not yet been found, so a script was created to trigger the data gathering program on a timer, allowing the keyboard to be unplugged and the fourth Lepton to be plugged in (a Raspberry Pi only has 4 USB ports, so without a hub those are monopolized by the cameras).

The detection algorithm tested has an N (sample history length) of 32 for each of the pixels in the 2° window (six by six center square) of the cameras and used them to find the mean, median, and standard deviation (extracted via the Median Absolute Deviation calculation) for each pixel. The threshold used to determine an intrusion was six sigma (giving a probability of 99.9999998026825 percent that an incoming temperature reading would lie within the acceptable bounds, given the thirty two previous samples have created a reasonable statistical model for the temperature drift the camera is experiencing). At this stage in the detection algorithm's development when an intrusion is detected one of two things can happen: the alerted pixel count for this camera is incremented and the value is added to the thirty two value window, or the alert flag is incremented and the value is not added. The latter method attempts to leave the running statistics undisturbed by the intrusion, but the effect of the rise or fall in temperature on the value it eventually returns/settles to is unknown. However, the implementation of discarding intrusions was flawed during the first test and a pixel's standard deviation could have become zero. Given any statistically unlikely data will not be added to the running average, this freezes the window and causes a constant alert. At this stage in development the thermal anomaly threshold was not implemented.

A variety of outputs for the tests were possible: camera number, time stamps, and pixel count changed (intrusion information), pure data – the value of each pixel analyzed paired with its index (camera, location within the six by six window) and statistics (mean, median, standard deviation), or pure data and intrusion information. The most useful proved to be both pure data and intrusion information. It should also be noted that due to the constant unplugging of USB devices, the cameras did not consistently hold one of the '/dev/vid#' names throughout the experiment, as they would change each time a camera was unplugged.

The algorithm did not perform well on the first test for a variety of reasons. The first and largest reason is the settling time of the cameras was not taken into account. Since the USB plugs were constantly being swapped out to allow editing and script starting with the keyboard or to run the algorithm with all four cameras, the cameras being swapped were not given time to adjust to the background temperature. The following graphs were created using Python and pyPlot.

5.2: Test One Problems

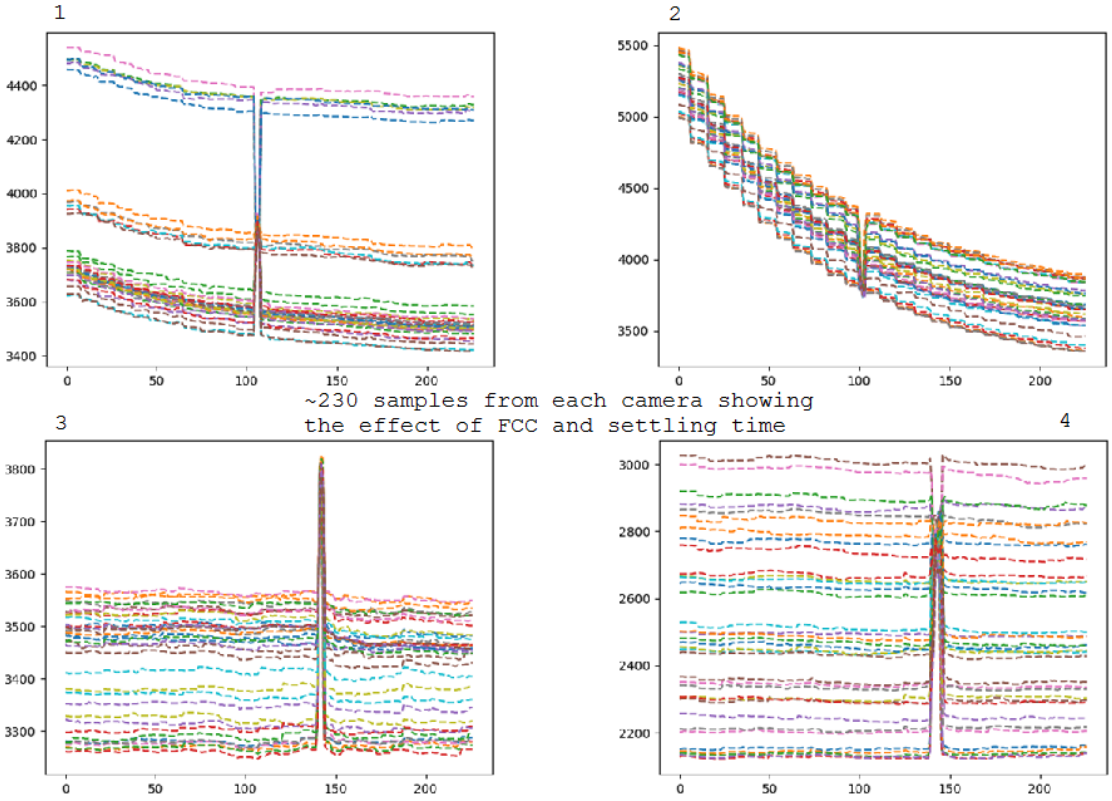

Figure 5.4: Power on Settling Issues

Sub-figures 3 and 4 in Figure 5.4 show cameras that have been allowed to adjust to the ambient temperature landscape they are viewing. Sub-figure 4 shows a good range of temperature scenery being monitored, yet the intrusion around sample 140 still clearly changes each pixel value. Sub-figures 1 and 2 show the cameras' response as they perform flat-field correction to adjust to the ambient surrounding temperature. Sub-figure 2 in particular shows the effect of the Far Field Correction (FFC), which is triggered every second or after nine frames of data.

These graphs were obtained during post-processing. In the field it was clear the algorithm was not behaving well, but pinning down the underlying problem was difficult.

Given time to analyze data, the settling problem was quickly diagnosed and follow up tests were conducted indoors to determine how long the cameras needed power to become stable. The cameras were all positioned at 90° and spaced out with a total span of six feet on a lab bench. With this setup it was determined the cameras need at least one minute to settle before reading their data to allow their FCC functions to stabilize their thermal readings. The standard deviation bug was also found and corrected during this post processing analysis. A powered USB hub that allowed all peripherals to remain plugged in was also located.

5.3: TEST ONE VALIDATIONS

The statistics in the detection algorithm (aside from steady state standard deviation) worked well.

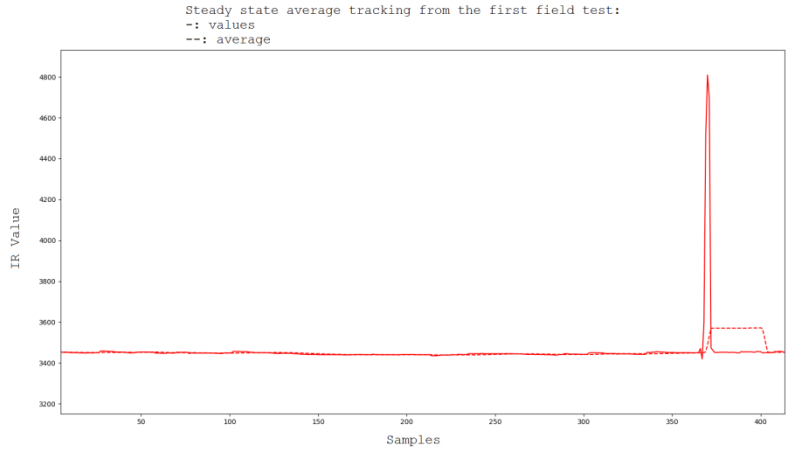

Figure 5.5: Average Tracking – Intrusion Values Added to Window

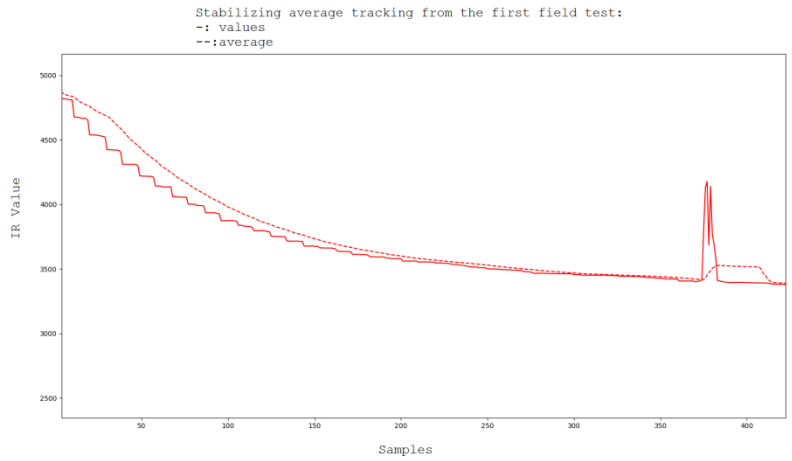

Figure 5.6: Average Tracking during Stabilization (Intrusions Added)

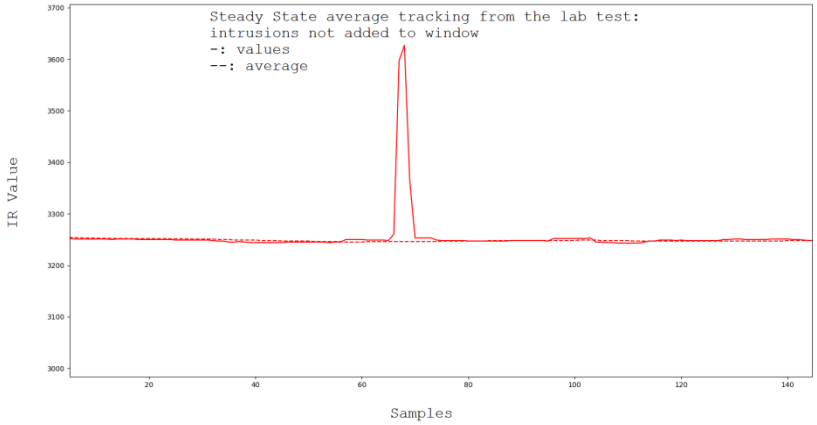

Figure 5.7: Average Tracking without Intrusions Added to sample window

5.4: TEST ONE RESULTS

The following graphs show data gathered and algorithm results versus simulated results run over the same data. Both the simulation and the Raspberry Pi algorithm use a statistical threshold of six sigma, but the field test had a changed pixel to detection threshold of 25 while the simulation had 30.

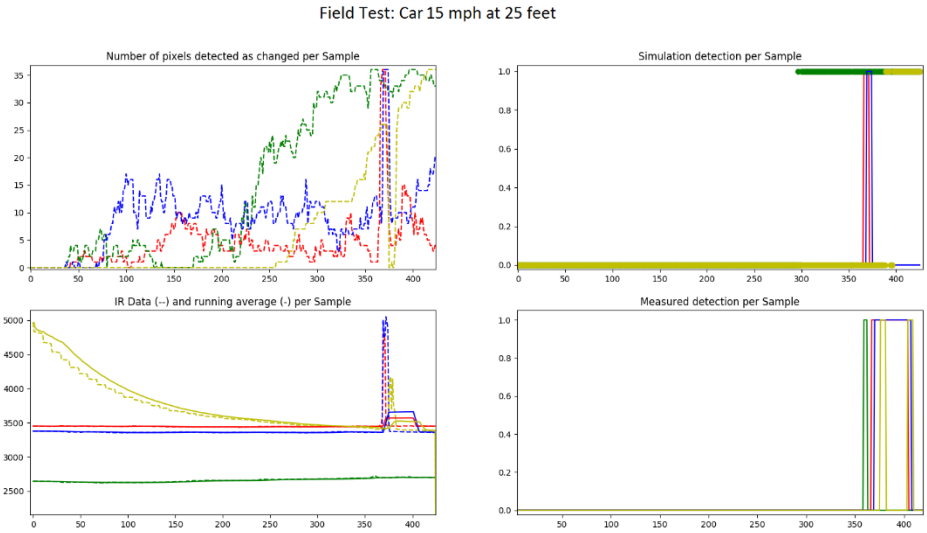

Figure 5.8: Overall System performance during test

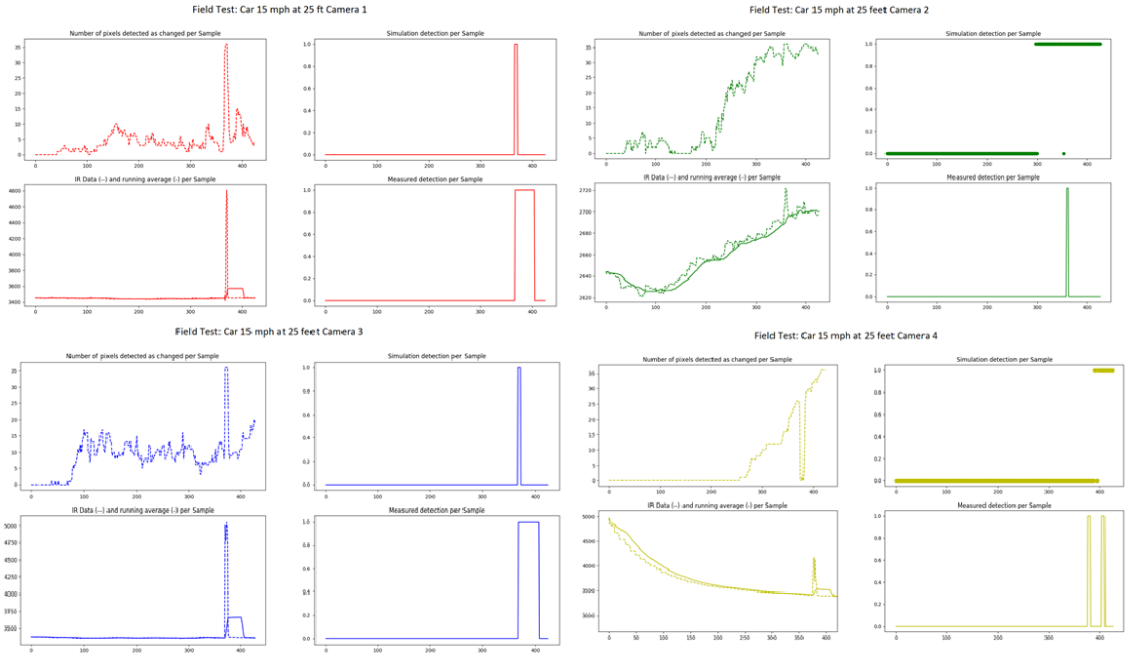

Figure 5.9: System Performance per Camera during test

5.5: Investigating Discontinuities

The data below was captured in an air conditioned (more thermally stable than outside) lab when trying to determine the cause of the discontinuities in the previous graphs.

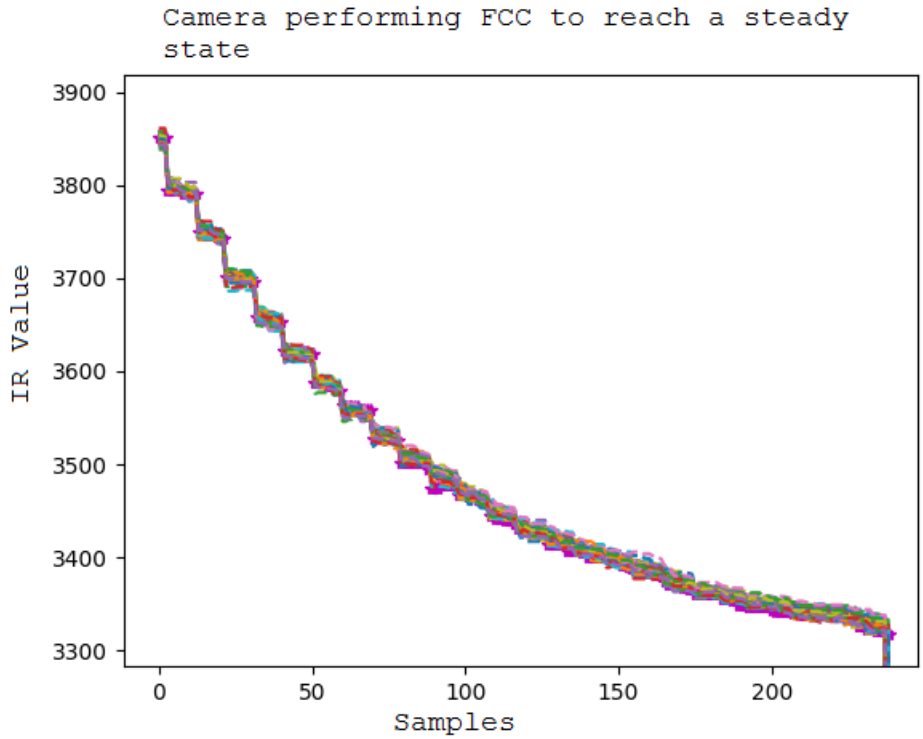

Figure 5.10: Settling Values, Single Camera

Given the powered USB hub allowed cameras to remain plugged in during all testing, and the rough estimate of 90 seconds for camera stability is acceptable, the cameras will be plugged in for multiple minutes before any future tests, so that problem is resolved, as shown by Figure 5.11.

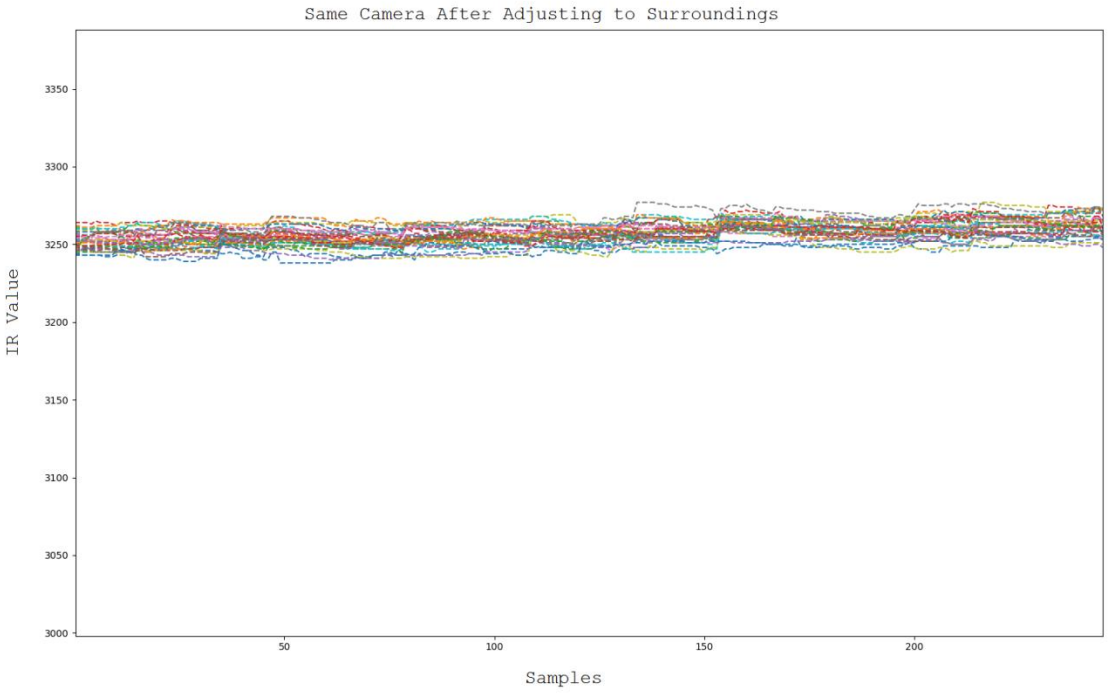

Figure 5.11: Camera Values after One Minute

5.6: Goals for Next Test

The following are graphs created from data in a lab after debugging the algorithm. The statistic threshold on the Raspberry Pi was changed to 27, aside from that the test remained the same.

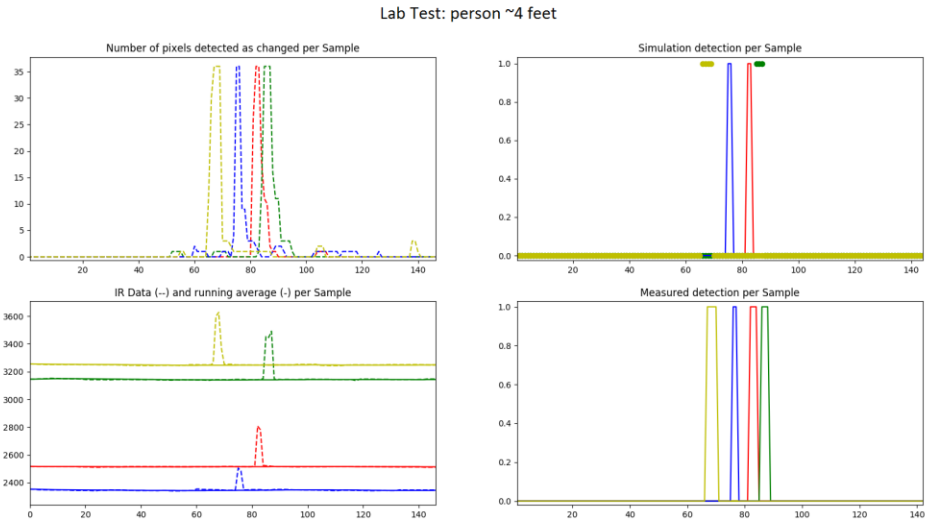

Figure 5.11: Successful Detection – All Cameras

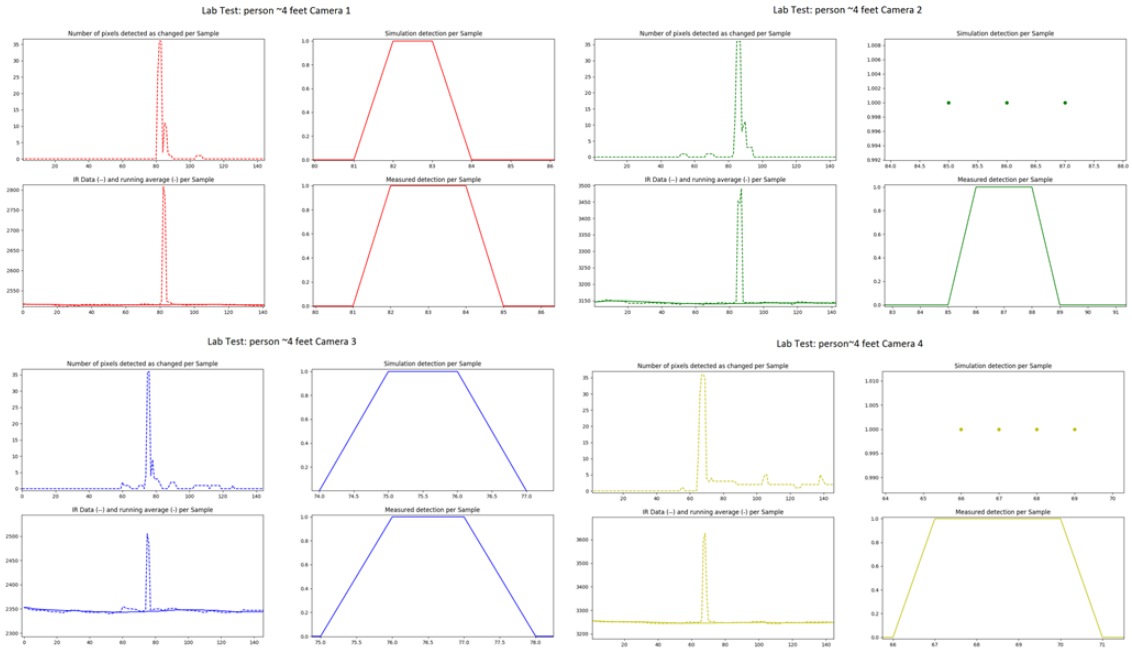

Figure 5.12: Each Successful Detection in Detail

This data shows nearly a one to one detection in simulation to detection in practice after adjusting the algorithm and sensor setup based on the knowledge gained from the first

test. False alarm changed pixels – those that are triggered by thermal noise or other interferers – in this scenario are kept beneath five per frame while residual pixels – those registering a changed due to a recent intrusion – are kept beneath sixteen. This suggests the changed pixel threshold could be lowered to around twenty and the results would remain the same. However this is in a much less noisy lab environment. The next step is to conduct basic outdoor tests, and to set up another large data gathering day.

Chapter 6: Test Two

6.1: Test Two Setup

Testing occurred outside of a lab building at Sandia National Laboratories. The sensors were all at 90° from the "x-axis" and spaced from the center as follows: 3', 0.5', 0.5', 3'.

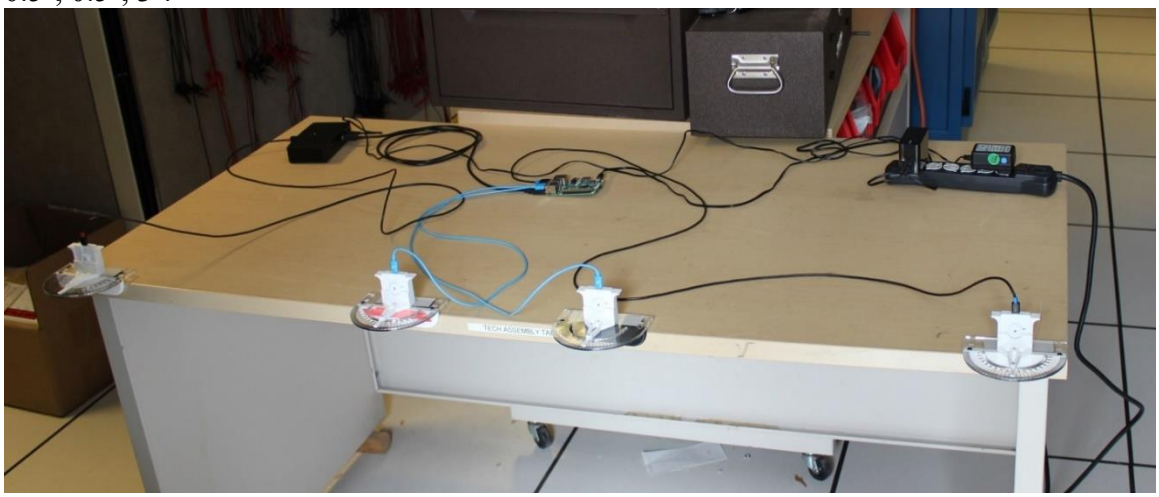

Figure 6.1: Test Setup 2

Testing was performed from 9:50 am to 11:15 pm on August 11, 2017 and the temperature ranged from approximately 70° to 83° Fahrenheit.

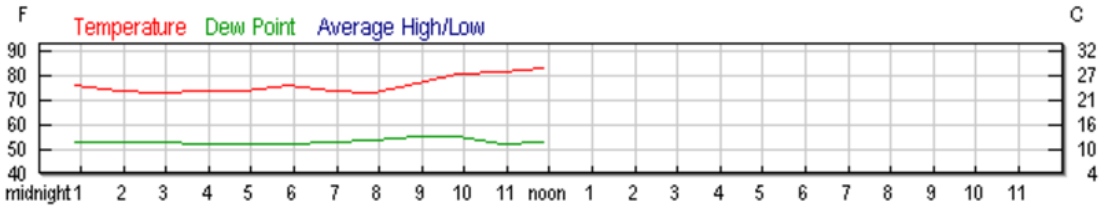

Figure 6.2: Temperature conditions during the second test [8]

The test subject was a person walking back and forth in front of the sensors. Data was gathered six times (9:50, 10:00, 10:10, 10:20, 10:30, 11:15) and the algorithm was adjusted to remove false positives and more to accurately calculate statistics.

All Tests	
Statistic Threshold	$6\sigma^2$
Thermal anomaly	30 pixel value
Pixel Window (N)	6x6 center pixels
Changed pixels to trigger detection	30 pixels
9:50 Test	
Gather first data	
10:00 Test	
Fixed algorithm – standard deviation value not properly divided. Pixel detection logic also fixed, previous changes inverted it.	
10:10, 10:20, 10:30, 11:15, and 11:45 Tests	
No testing values changed	

Table 6.1: Test Two Algorithm Updates and Thresholds

6.2: TEST TWO ANALYSIS

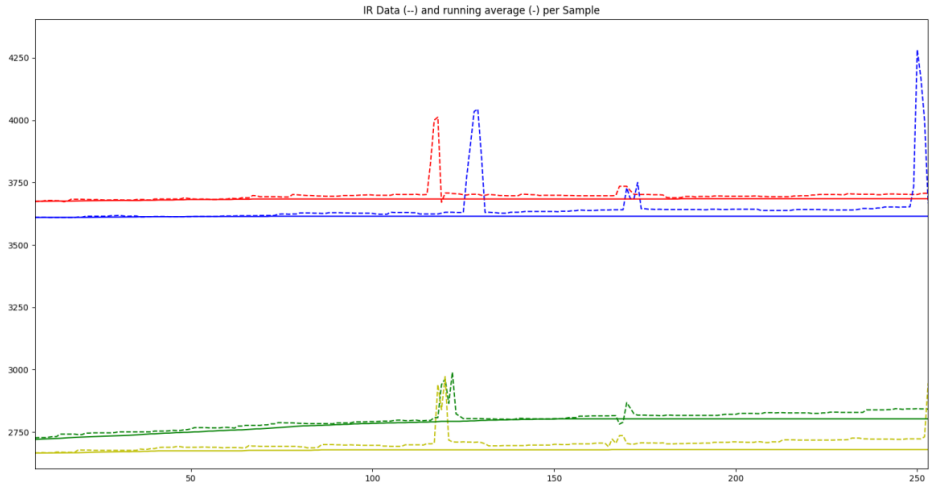

Figure 6.3: Pixel Mean Drift Separation

This data was gathered outside on the day before the second test. The IR data can be seen pulling away from the running average the algorithm should be tracking. This is caused by sudden shifts in a few fractions of a degree in temperature after a long period of stability. The standard deviation has been driven very low by at least a window's worth of closely clustered samples so that the small shift in temperature (15 to 20 sensor value) causes the pixel to alert and reject the values from the running average window.

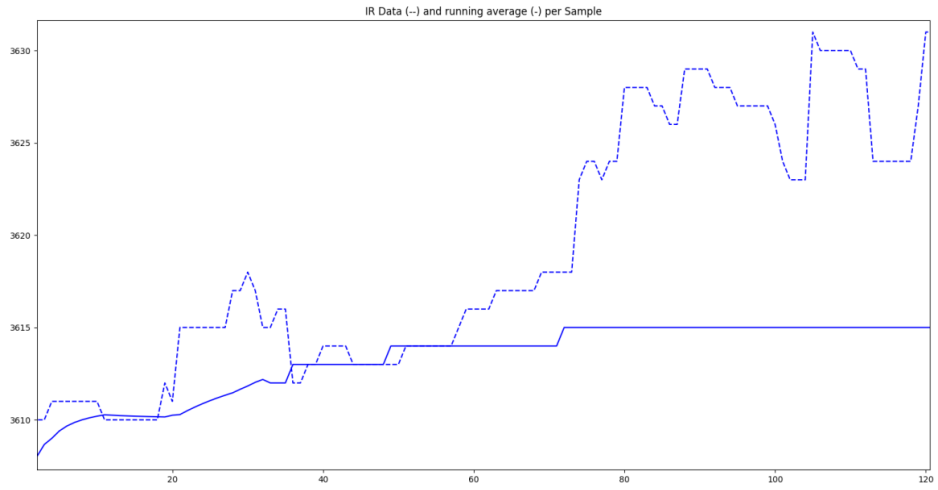

Figure 6.4: A closer look at the mean drift

Figure 6.4 further illustrates this problem. From sample zero to sixty the average is reasonably tracked and the variance of the data is reasonably stable. When the environment being monitored heats up slightly, around sample seventy, the standard deviation is low enough that the temperature is now running away from the mean, and every sample moving forward is rejected and triggers an alert.

To fix this problem, an additional test was added to the algorithm. The alert detection will work the same: if the pixel is outside of six standard deviations it will count toward the alert total, but a new threshold will be introduced to determine if the sample should be added to the running window or not. Based on previously gathered data shifts in temperature between samples should not exceed 20 to 30 sensor value. Thus a simple window of plus or minus thirty around the current mean should be sufficient to indicate whether or not the incoming value should be added to the window or not, and this is the change introduced to the algorithm: a threshold to reject thermal anomalies.

9:50

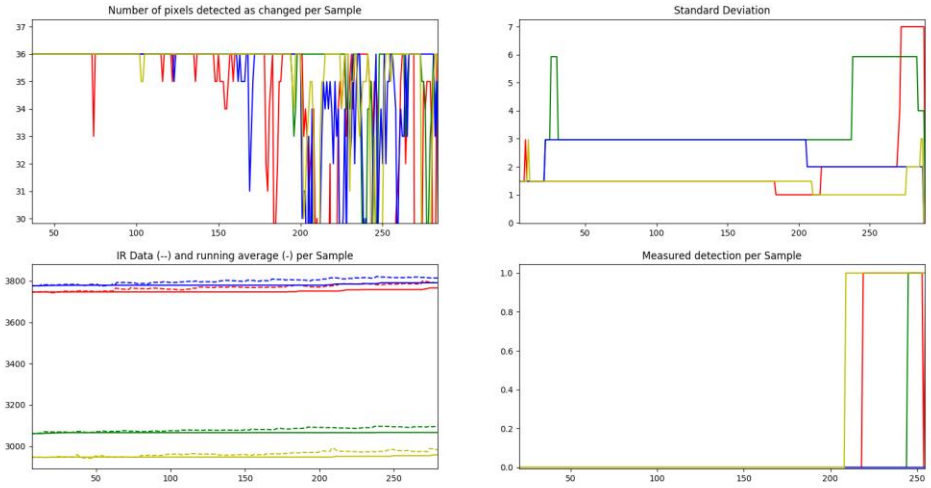

Figure 6.5: Inverted Pixel Detection Logic

The data shown above is poor due to inverting the logic of the added algorithm. The mean is left behind as the environment heats up. Data was gathered for approximately 30 seconds.

10:00 and 10:10 – Steady State tests

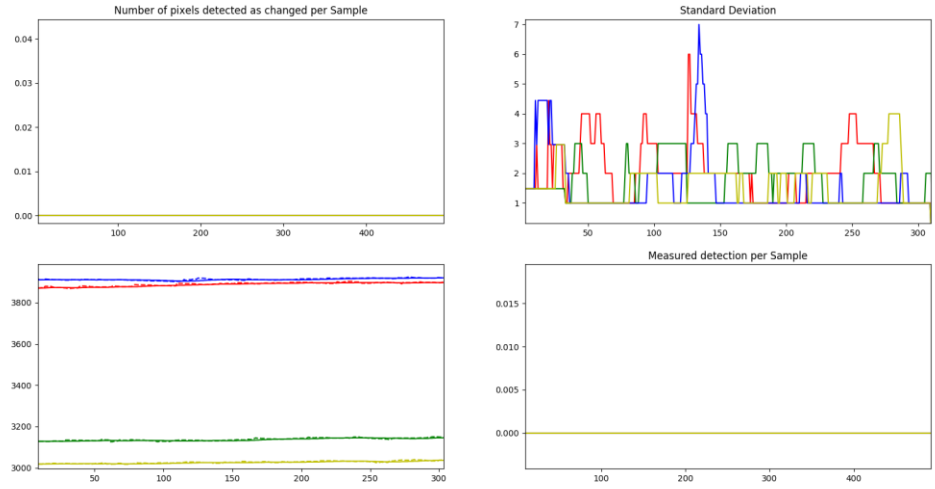

Figure 6.6: Steady State Test One

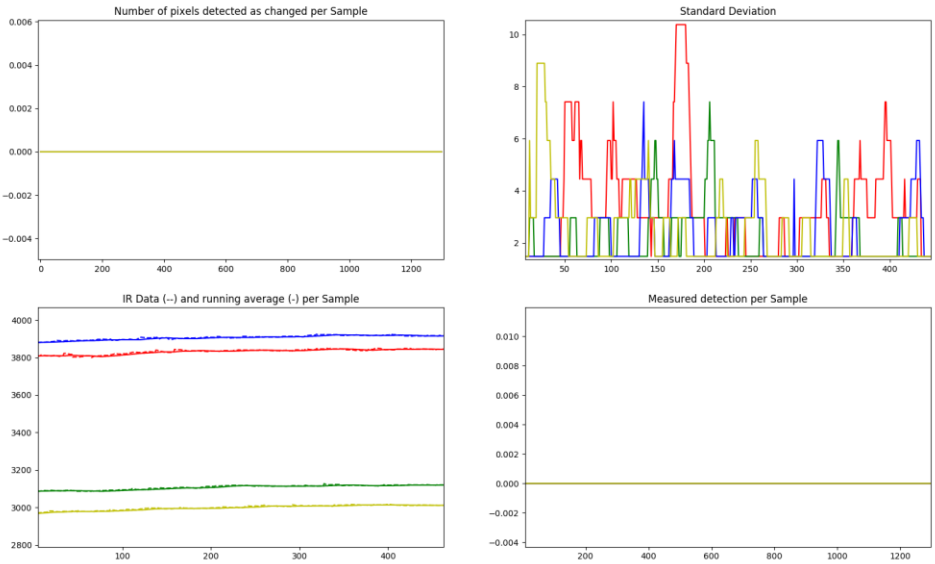

Figure 6.7: Steady State Test Two

No detections occur during the steady state tests which went on for 30 and 50 seconds respectively. This data shows the upward trend of the temperature not triggering false alerts while the mean tracks closely with the IR data. The addition of the thermal anomaly threshold is behaving as it is designed to.

10:20 – Intrusion detection

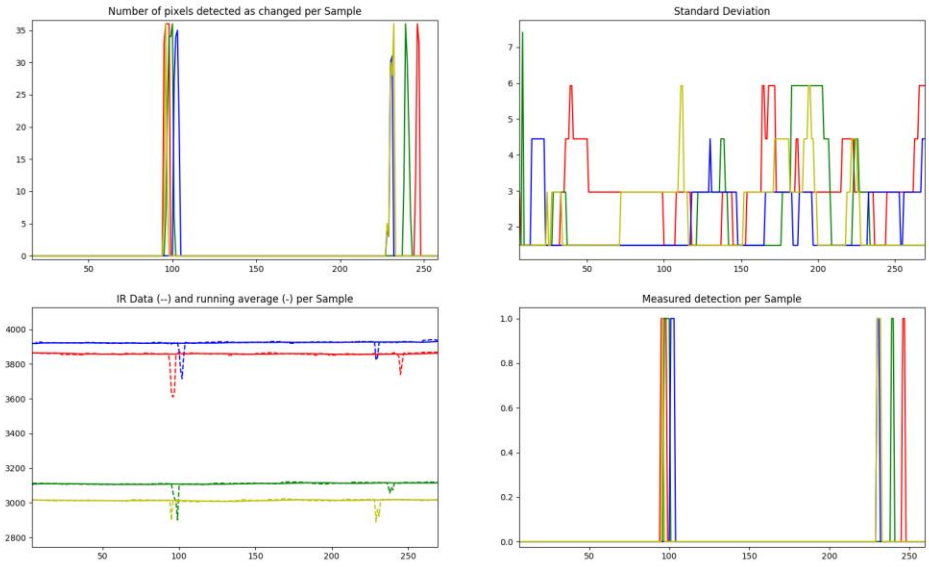

Figure 6.8: Double Intrusion Test

The test subject walked back and forth once in this 30 second test. The short nature of the test did not allow the temperature of the surroundings to shift much, however the algorithm successfully identified each intrusion with all cameras.

11:15: Longer Intrusion Test

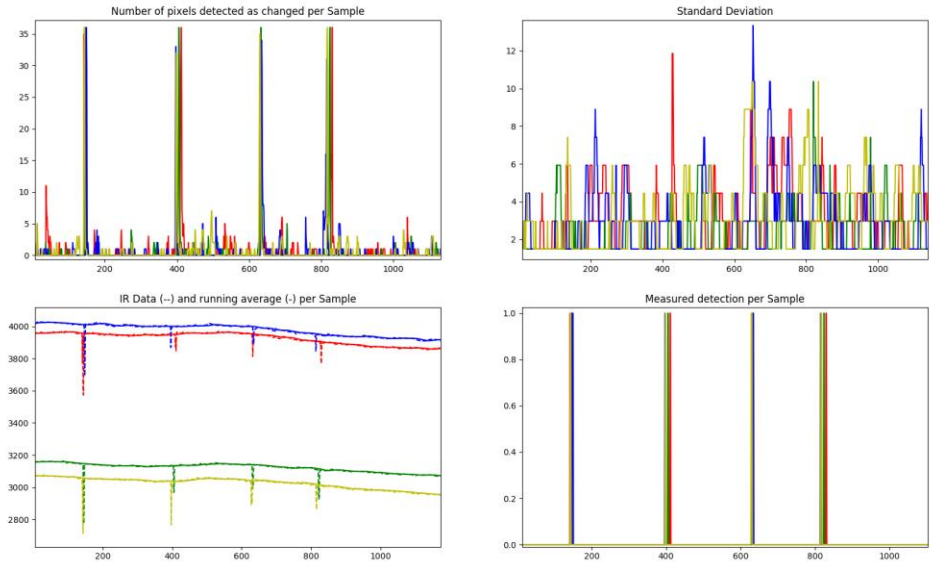

Figure 6.9: Two Minute Intrusion Test

This test lasted approximately two minutes and fifteen seconds. The test subject walked back and forth twice (4 total intrusions) at distances varying between 4 feet and 25 feet. The algorithm continues to behave admirably with at most 12 pixels signaling a false alarm while detections are very clear spikes.

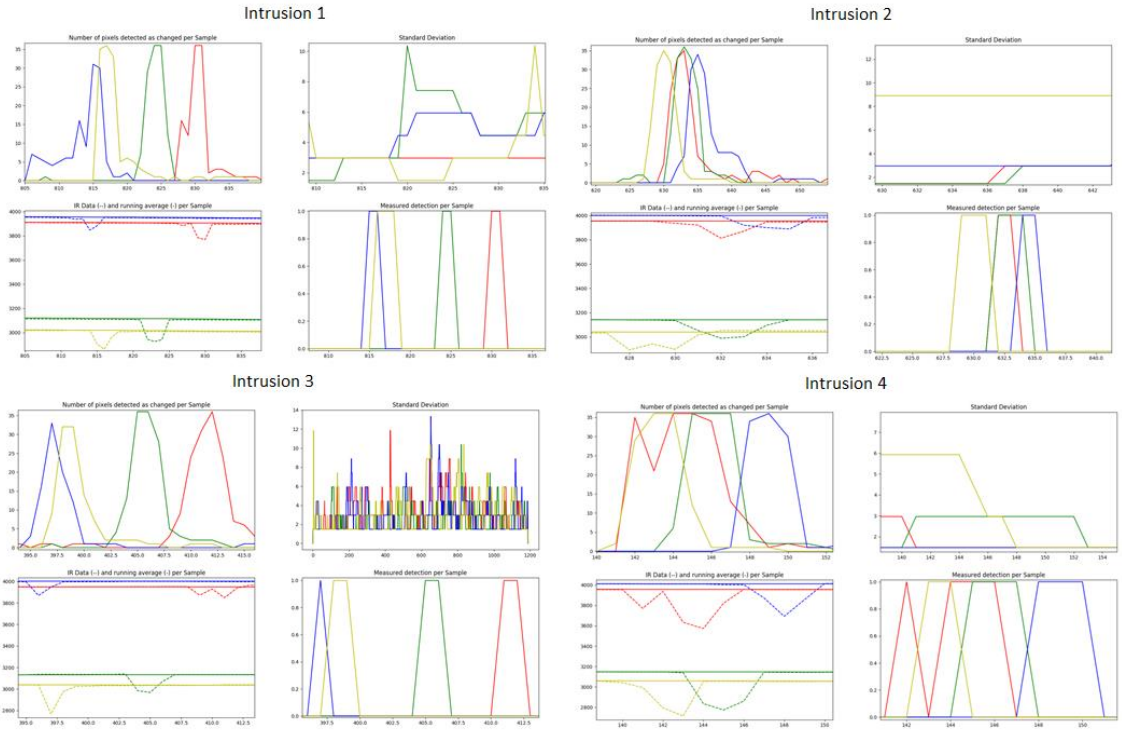

Figure 6.10: A precise look at the intrusions

11:40

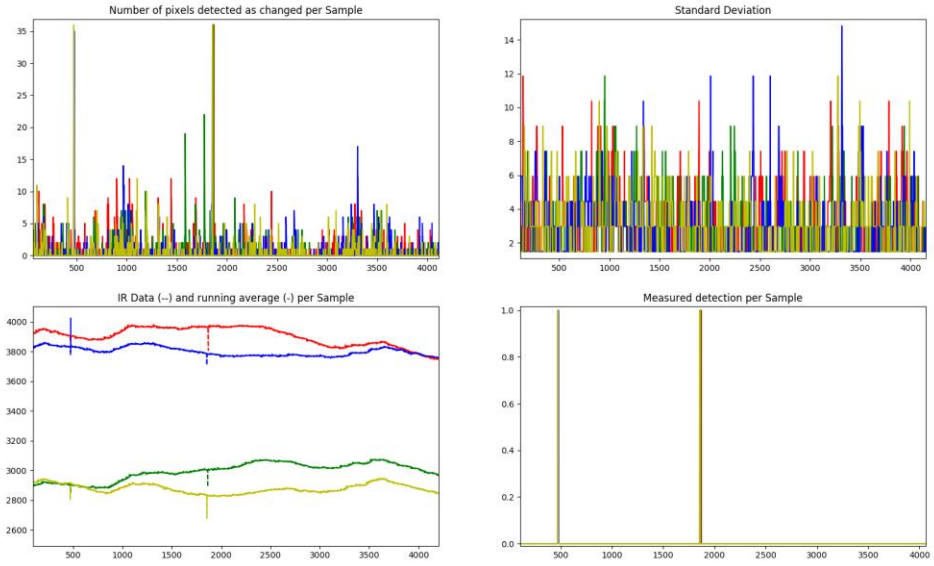

Figure 6.11: Long Term Detection Algorithm Stability

This test lasted 477 seconds (nearly 8 minutes) allowing the ambient temperature to shift. The largest range of this temperature is seen in the red graph and drifts between 3980 and 3736 in IR sensor value. Despite this drifting the algorithm performed very well, only identifying the two intrusions. This test was performed by leaving the cameras monitoring outside of a lab building, and the intrusions were scientists walking to their labs.

After an extended period of testing the algorithm in the outdoor, thermally fluctuating environment, the prototype is ready for another large scale data gathering experiment.

Chapter 7: Test Three

7.1: Test Three Setup

The second data gathering took place at the same location as the first. The testing time was between 11 am and 3 pm on August 16, 2017. From the following graph it can be seen that the temperature fluctuated between 75° Fahrenheit and 90° F.

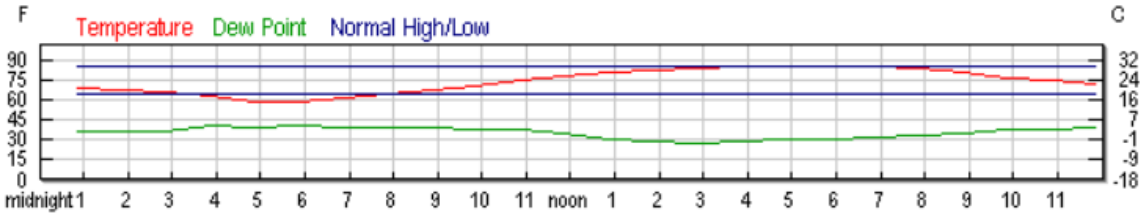

Figure 7.1: Test Three Temperature Conditions [9]

Configuration 1	
L1, L4	5 feet from center
L2, L3	1 foot from center
A1, A4	140° from center
A2, A3	110° from center
Configuration 2	
L1, L4	3 feet from center
L2, L3	1 foot from center
A1, A4	130° from center
A2, A3	110° from center

Table 7.1: Test Configurations

The idea was to compare two configurations that are close to the ideal found in exploration of the simulation, and determine if the lower horizontal profile of the second configuration could be viable. The test objects chosen were from the following sets: distances of 20, 25, or 30 feet, speeds of 15, 20, 25, 30, and 40 miles per hour, and the test vehicle is again the Town and Country van, a 17 foot wide vehicle. This allowed for two speeds that should be within target accuracy bounds (15 and 20 mph) and two speeds on the edge or outside the capability of the system. Data was gathered three times for most of the distance and speed configuration: one of the object passing through the sensors headed one direction, and two of it passing through in the opposite direction. Seventy eight individual tests were run.

7.1: Data Gathering Results

At first glance the data seems to exhibit the desired characteristics. The hard 140° configuration of the inner two cameras for the first configuration of this test did not perform well (possibly due to misalignment on the vertical axis), but some of the tests still generated valid data.

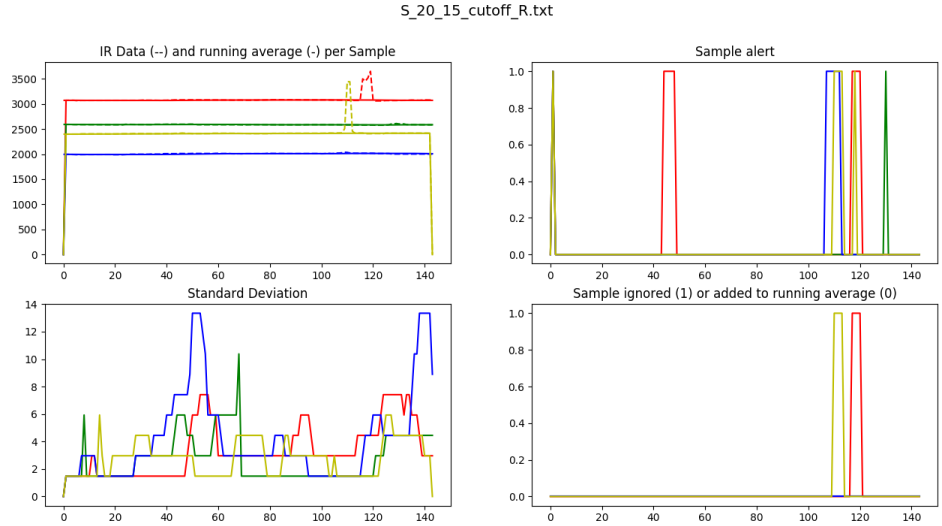

Figure 7.2: Test data with enough frame detections for an estimate

Figure 7.2 shows the weak intrusion signals from the cameras at 140° (blue and green) graphs. As the algorithm was running, with the threshold set to require 30 pixels changed to determine a detection, the weak observed shift in signal was not enough to generate a camera detection.

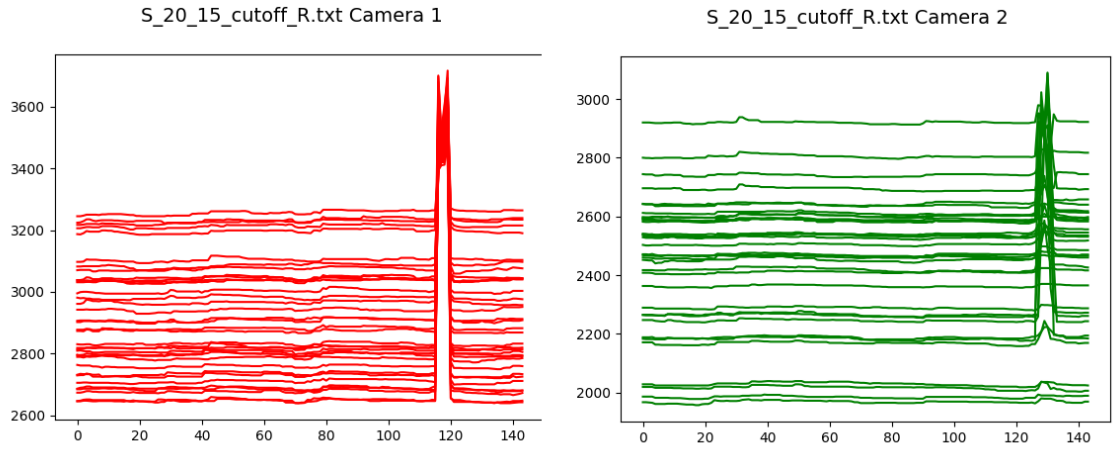

Figure 7.3: Strong detection compared to 140° weak detection

The side by side comparison of a camera that recorded an alert versus a camera that did not record an alert reveals the problem – only about half of the pixels are pulled up strong enough to warrant a pixel detection to trigger. The bottom lines in the graph show that for some, this rise was less than or equal to the effects of ambient temperature drift. However, despite some of the pixels not reacting to the intrusion, the event in the right graph can still clearly be identified visually, indicating the pixel changed threshold needs to be lowered. In post processing, this threshold was lowered to require 11 pixels changed to trigger an intrusion detection.

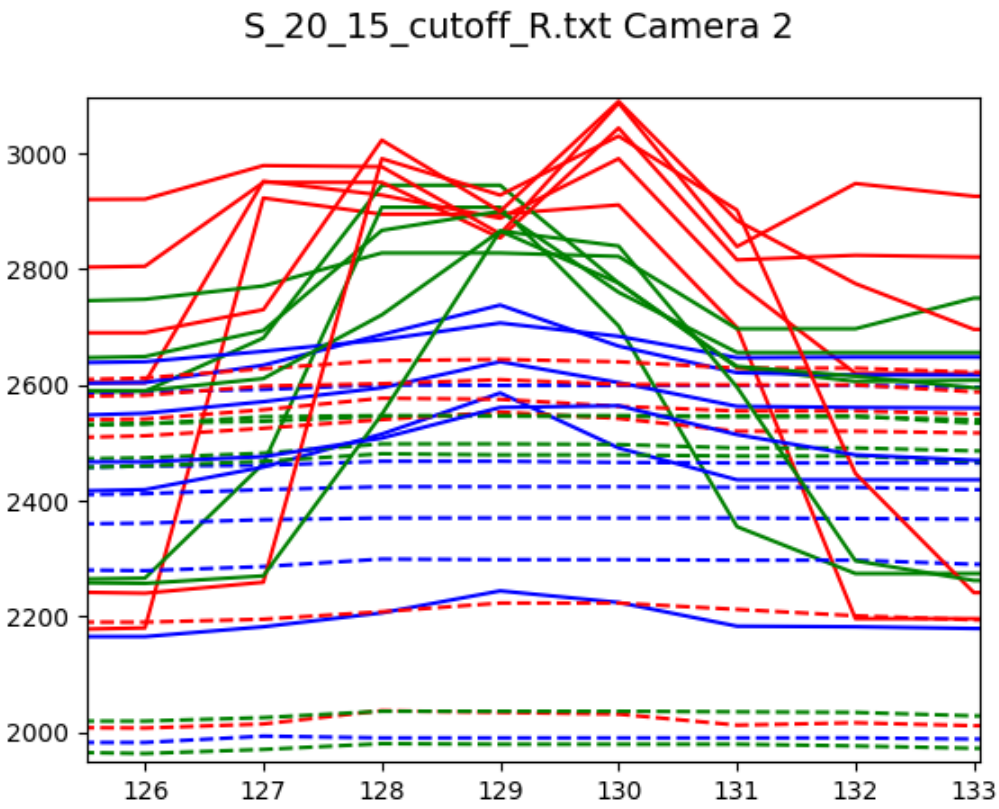

Figure 7.4: Weak Pixel Detection Close up

A closer analysis of the picture reveals that the bottom most rows of pixels reacted to the intrusion, whereas the top most rows did not react at all, or reacted weakly. This behavior was consistent throughout the tests for this geometric configuration. It appears that the interior cameras were perhaps angled upwards, as the vehicle only passes through the top half of the camera (the cameras are upside-down). It should be noted that the cameras were all verified to be facing perpendicular to the ground with a level before the simulation began, despite the evidence suggesting they were not. It is also worth noting that the second test configuration saw detection from all six rows and not just three, as shown in figure 7.5, which proves this is not an issue with the cameras. The last column

does not have great detection still, but the camera is at 130° from the center, so whatever angular reflection problem exists at 140° must come into effect here to a lesser degree as well. This suggests that there is a slight horizontal refraction problem being observed, as well as a vertical camera angle issue. Despite this hurdle, changing the post-processing simulation's pixel change threshold to 11 pixels renders most of the data collected still usable.

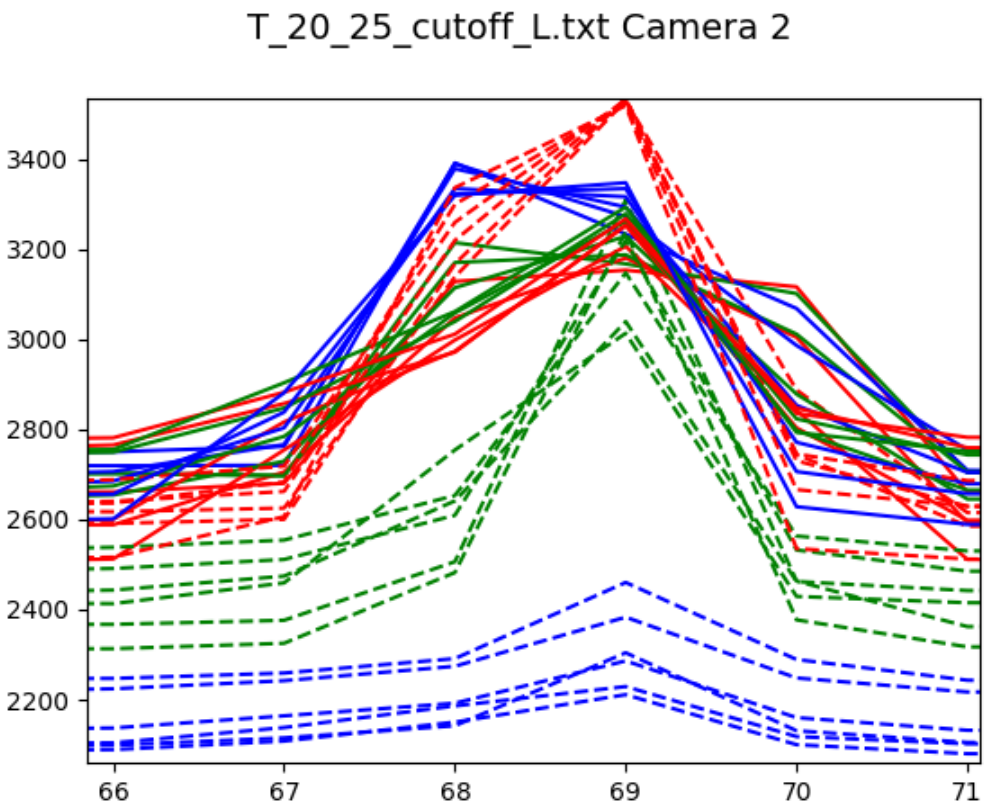

Figure 7.5: Second configuration, consistent detection

The next challenge presented by the collected data is a discontinuity between data sets from objects passing through the sensors to the right and those passing through the sensors to the left.

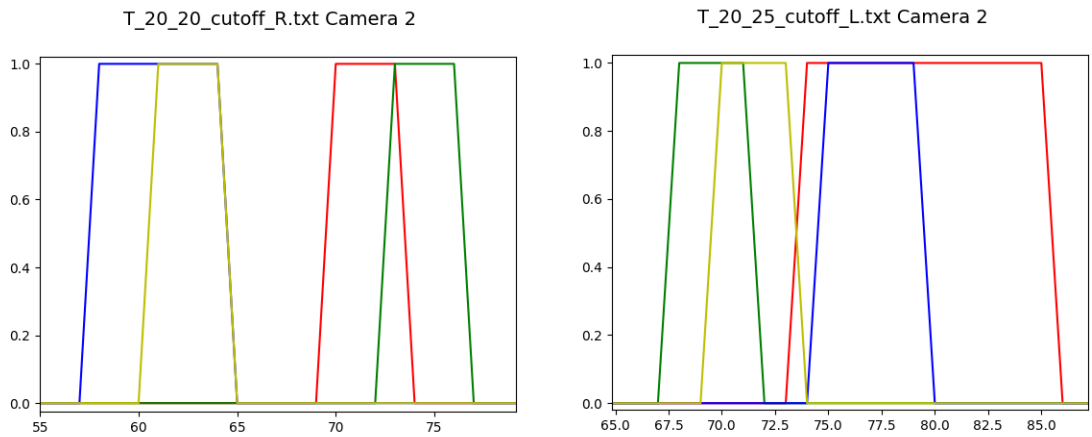

Figure 7.6: Inconsistent Detection Order

The middle two sensors should not be triggered in the same order when an object passes through it in opposite directions. Given the configuration of the cameras, and the static state of their device library assignment thanks to the powered USB hub and the knowledge learned from the first data gathering, the correct detection sequences are: {B, Y, R, G} or {G, R, Y, B}. From the graphs above, it is apparent that the left detection is out of order and incorrect. A factor that could contribute to this incorrect detection order would be the rolling shutter nature of the function that gathers data. Cameras are sequentially asked for frames: first the camera at '/dev/vid0' transfers one frame of data and it is analyzed, then the camera at '/dev/vid1' and so on. This effect causes sampling to take place asynchronously, and in the case of this test setup, cause inconsistent results. When the object moved left across the sensor array, it was moving in the opposite direction

that the cameras were sampling, exasperating the rolling shutter effect and causing some detections to occur out of order. While this cannot be the only cause of this inconsistency, the fact remains that all data gathered from an object moving left across the sensor field is rendered useless. This narrows the valid files from 78 to only 36 data files that contain sound data. After considering the previously addressed problem with Configuration 1, only 29 files detected intrusions on all four cameras and can possibly provide estimations.

During this test gathering the Raspberry Pi itself ran out of data. Simply redirecting standard output from the detection algorithm to a file (using ">" in Linux) does not alert users when the system's memory is exhausted, so half of the original data files were simply blank. A flash drive was used to free up system space, and the tests were repeated. After the previous problems were addressed, 29 good data files produced the following estimations.

7.2: TEST THREE ESTIMATIONS

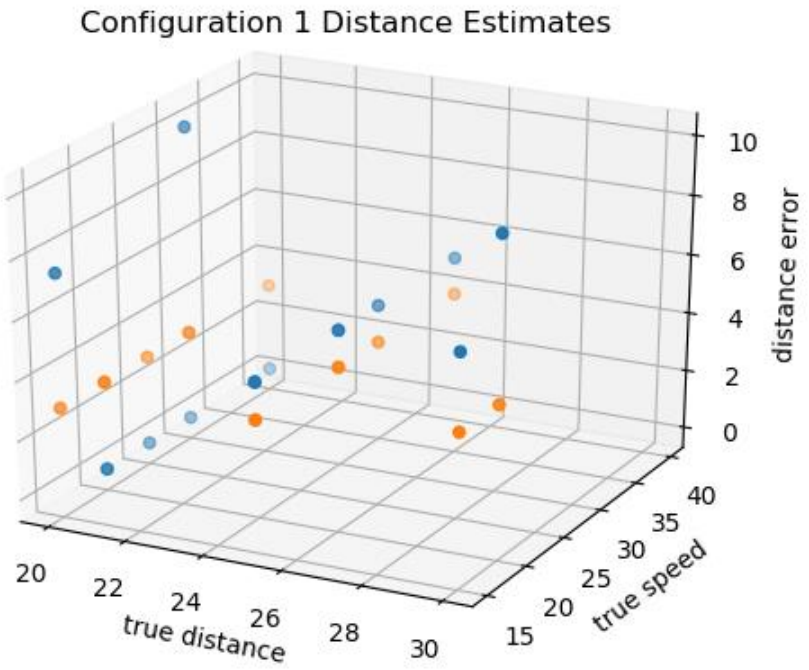

Figure 7.6: Distance Estimate Errors for Test Three Configuration One

It should be noted that the few estimates within acceptable distance error are due to a clipping feature of the estimation algorithm (clip the values at a known range given the position of the road, in this case [20, 30]) and not due to correct estimation. All other estimates can be seen to be well out of the target 15% error range that will yield accurate speed and width estimates. The orange dots in Figures 7.6, 7.7, 7.8, and 7.9 are the estimation goals and the blue dots are the actual estimates.

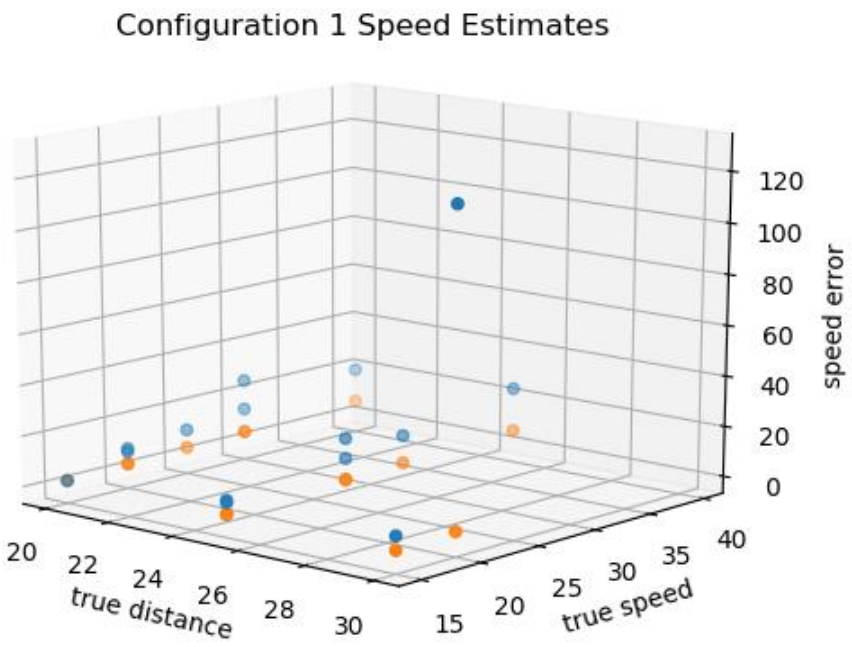

Figure 7.7: Test Three Speed Estimates for Configuration One

All speed estimates are sufficiently wrong as well, the only one that is close to the 15% target is the estimation at a distance of 20 and a speed of 15, which is the result of a clipped distance estimation, and therefore not a good representation of the estimation capabilities of the system. Most are within thirty to forty percent error, with a few outliers like the close to 100 mph error found when the distance was 30 and the speed was 20.

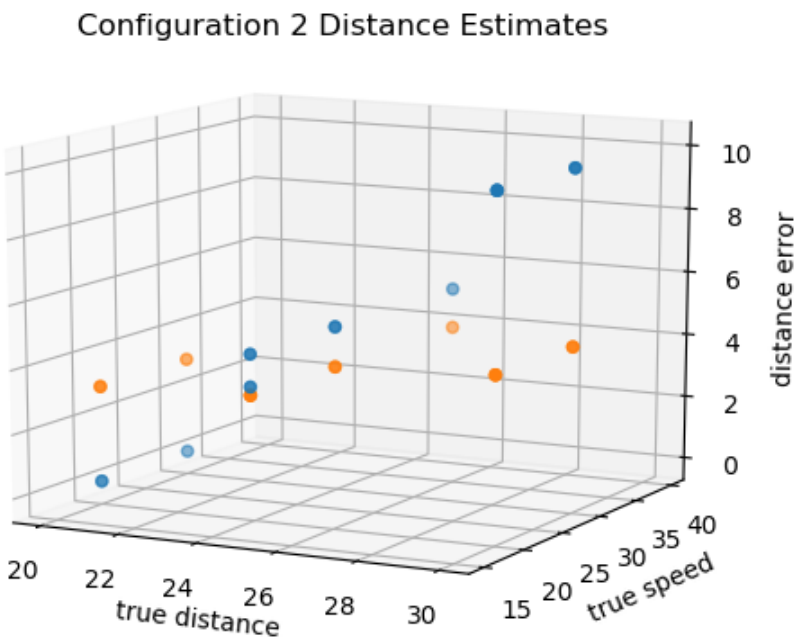

Figure 7.8: Test Three Distance Estimates Configuration Two

Configuration 2 did not fare any better than the first. Due to the clipping of estimates, it shows the same 10 foot bound on the maximum error, with some exact matches due to this aspect of the algorithm, but all actual estimates outside of the target accuracy.

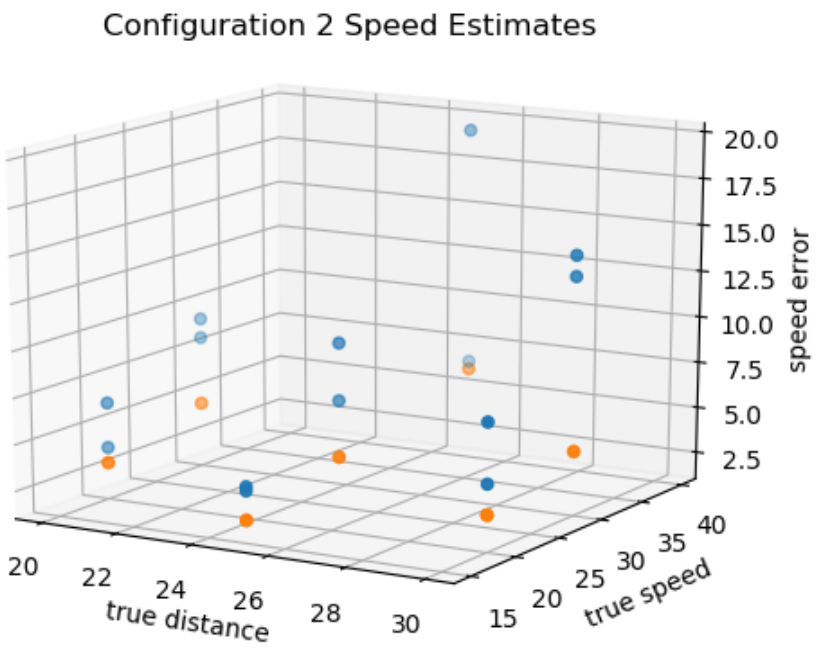

Figure 7.9: Test Three Speed Estimates Configuration Two

All speed estimates are again shown to be outside of the target 15% accuracy. Massive errors still exist. The highly inaccurate results of this test are due to an exploration of algorithms to correct these estimates.

Chapter 8: Post Processing

8.1: Linear Regression Setup

In an attempt to fit very noisy data to the known truth, a linear best fit model was attempted. The only truly measured data is the two time differences: the difference between intrusion detections on sensors two and three and the difference on sensors one and four. Thus the sequence of measured time differences was extracted and coefficients for these measurements to accurately estimate the truth are solved for.

$$\Delta T * C = R$$

In this equation the time differences are ΔT (Nx2), the coefficients to be solved for are C (2x1) and the true speeds are the R matrix (Nx2). Thus calculating the coefficients can be done as follows.

$$C = (\Delta T^T \Delta T) \Delta T^T R$$

Multiple C matrices are calculated using different methods. The first attempt simply used all 29 good files' data, making N 29. The resulting coefficients did not perform well enough to obtain the 15% error goal for the system.

The C vector found was:

[[18.29589238]

[0.83033984]]

Seven of the test configurations now result in estimates within the target error range, which is better than the previous two, however still only one fourth of the total test runs. For all figures in this section, the orange dots denote the 15% goal, while the blue dots denote actual points of data after the coefficients found in the linear regression were applied.

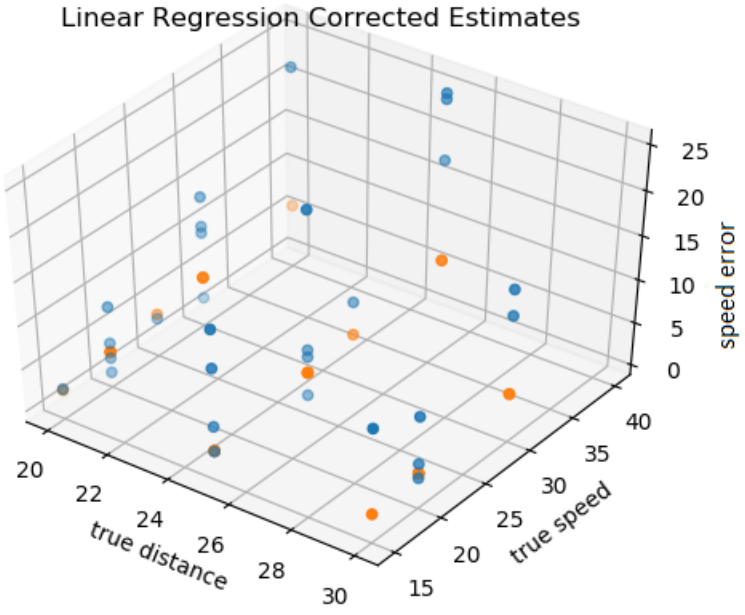

Figure 8.1: All Data Linear Regression Results

The second method utilized was to break the files into groups, calculate the coefficients based on the majority of the good files, and use the rest to test how accurate the system had been tuned to be. Three different categorizations were tried. The first calculation treated the test configuration one data as the training data and the test configuration two data as the test data. This should not work all that well, as it treats the estimations as independent of the geometry of the system, which they are not. The C vector found with this method is very similar to the first:

[[18.26675505]

[0.21802596]]

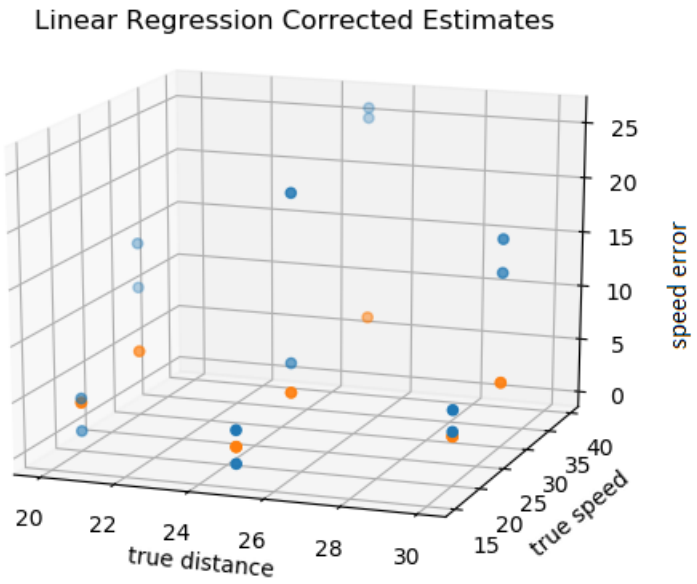

Figure 8.2: Linear regression for Configuration One

As shown by this graph, the twelve estimations of the second test configuration only result in two below the desired error threshold. This is better than the zero before scaling, but large errors in the other estimates remain. The next test sets used for the linear regression split each test configuration data into train and test data and calculated two coefficient matrices, one per configuration. This allows the geometry of the system to effect the estimations, as it does in the real world.

The first configuration test broke the test data into a set of nine training points and three test points. The test points were selected to have each distance (20, 25, 30) and speeds that had duplicates within the training data if possible. This resulted in the test set of {(20 feet, 20 mph), (25 feet, 25 mph), (30 feet, 20 mph)}.

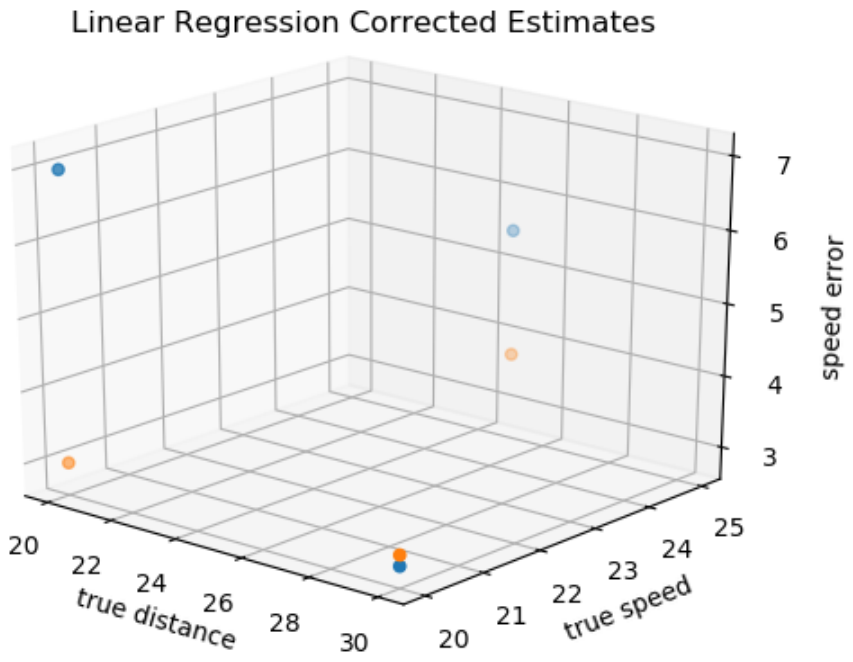

Figure 8.3: Linear Regression – Configuration One Tet Set

Here, one of the three test points has met the estimation goal of less than 15% error. The others remain a bit too large, but only one is outside of 20% error. Test configuration two is broken into a set of 10 training data and 2 test data points. The two test data points were again chosen to be test points that had another point of reference at the same configuration. They are {(20 feet, 20 mph), (20 feet, 30 mph)}. The resulting C vector is quite different than the first two found, demonstrating the difference in geometry of configuration two from the first configuration:

[[12.16577953]

[7.51219734]]

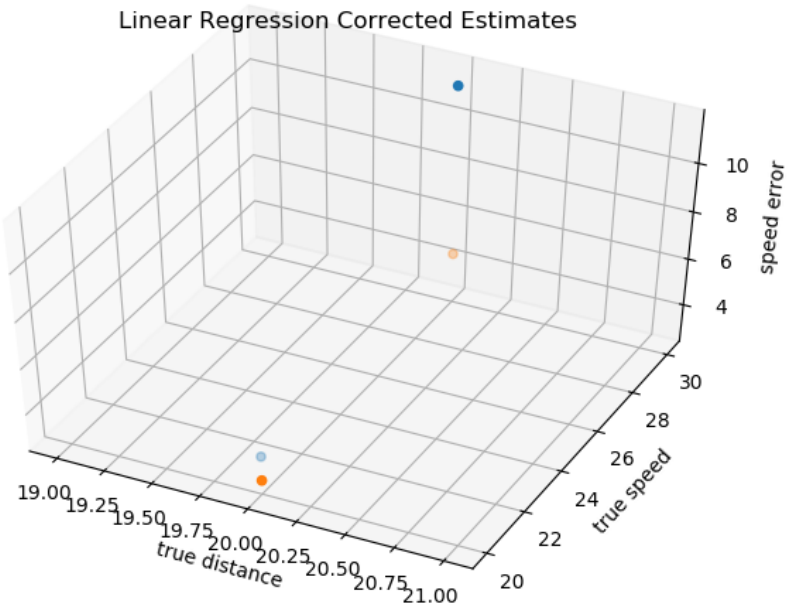

Figure 8.4: Linear Regression – Configuration Two Test Set

The results above show that neither scaled estimate managed to reach the 15% error goal. If truth for the two differences in sensor detection time existed, the linear regression model would work better scaling the existing values to that. However, no such data exists. Using linear regressions on all four intrusion times simply adds another time difference to the equation, as the three following time measurements must be considered relative to the first, as no true time vs distance data was gathered for the test object.

8.2 Linear Regression Results

Ultimately the goals of 15% error in speed estimations and 20% in width estimations were not met by this prototype, even with post processing techniques. The early

assumptions about the system geometry that led to the selection of the low FPS cameras, as well as choosing many test points slightly out of optimal range led to incredibly inaccurate test data. Unsynchronized cameras also greatly impacted system accuracy. This includes the rolling-shutter like effect of the sampling program also leading to inconsistencies in detection times and more error introduced into the system. Post processing tweaks to detection algorithm thresholds and attempts to use linear regressions to create accurate estimations improved accuracy slightly, but ultimately failed to meet the goals of the system. The unsynchronized and low sample rate prototype never met the objectives of the report.

8.3: Further Testing Issues

Along with the previously discussed flaws in the testing setup, the following issues also had a hand in the failure of the prototype to achieve the desired estimation accuracy.

Alternating Sampling Rate

In order to meet export guidelines, the FLIR Lepton 3 has a specification of less than 9 frames per second. This was found to be true experimentally, with more eight frame seconds recorded than nine frame seconds. These measurements were determined by counting samples recorded during the same second of Linux time. This also contributed to discrepancies between simulation and reality, although to a much smaller degree than the unsynchronized sampling. Given this issue, the ability of the program to produce accurate time readings for each sample was vital for the post processing to handle this fluctuating sampling rate.

Wrong Time Functions Recorded

Instead of the “/sys/time.h” file being included, which has time of day accuracy to the micro-second, the “time.h” and “clock.h” files were included. These files measure two

things: time since Unix epoch in seconds, and system clocks given to a process. Both of these were believed to combine to something useful: a second resolution global clock, and overall clock register, however this understanding was incorrect. The clock time was local (for the process) and not global, thus the effective granularity of the time measurements (discarding a good measurement of time to execute the frame to frame update algorithm) was one second, which is simply not good enough. Milliseconds would be acceptable but the time information provided from the data was three orders of magnitude too large.

Despite several attempts to train the estimation algorithm to transform the gathered data into accurate estimations, problems with the prototype's design ultimately resulted in failure of the system to perform to specification. Unsynchronized sampling proved a key difference between the simulated system and the prototype and greatly contributed to the system's inaccuracy. Linear regressions were used in an attempt to train the data, however, these methods were unsuccessful despite a variety of techniques and test cases. While the detection algorithm proved successful, the estimation algorithm failed due to prototype design failure.

Chapter 9: Conclusion and Future Work

9.1: Conclusion

To meet the design challenge of speed and width estimation given four points of data a simulation, prototype, and multiple algorithms were developed. The simulation modeled the system with perfect sensors and geometry. This program led to decisions about the optimal geometry, optimal test subjects, and limitations due to sampling rate. The prototype of the system was unfortunately defined during the simulation process and not after the analysis was complete. Knowledge of the system's ideal behavior was still being learned and understood. Thus the vital tradeoff of sampling rate versus range, which initially seemed to favor range, was misjudged and the low sample rate FLIR Lepton v3 cameras became the sensors for the prototype. In another miscalculation, care was not taken to synchronize sampling, as the simulation did. Unfortunately, due to the myriad of design issues, the comparison between simulated results and measured results never amounted to more than observing the measured results were much more inaccurate. With a more accurate prototype a deeper comparison could yield interesting results. Despite these shortcomings of the prototype, useful algorithms were developed to track thermal drifting statistically in order to ignore false positives, as well as estimation algorithms. When moving from simulation to prototype it is vital the conditions assumed in the simulation are replicated as closely as possible in the prototype. While the goal of 15% error was not met by the prototype developed in this paper, it provides progress towards a successful low power four beam speed and width estimator.

9.2: FUTURE WORK

Synchronize Sampling

To accurately match the simulation, the cameras or PIR sensors need to sample simultaneously. This could be accomplished on the Raspberry Pi 3B, which has a quad-core processor, by modifying the program to take advantage of these four cores, as well as finding the correct compiler. Given the final system will involve multiple ADCs monitoring voltage wave forms from PIR sensors, any further work should ensure these measurements are synchronized. The Lepton v3 cameras have a video sync GPIO that could accomplish this, and ADCs normally have synchronous sampling features. The "rolling shutter" of the main loop of the current program can lead to many oddities, including out of order detections. This greatly hindered the system's ability to accurately timestamp intrusions, which happens to be the only data with which the estimation algorithm works.

Real Time Operating System

Sensors with higher sampling rates and synchronized sampling will be useless if the system sampling them does not have guaranteed time accuracy. A Raspberry Pi initially seemed to fulfill the project requirements given my familiarity with the platform and the ease of interfacing it with the prototype's sensors. However, this Linux based board has no real-time guarantees. This system is only responsible for measuring time, and therefore a board running a real-time operating system is necessary to accurately measure and report the intrusion times. Any continued work on this system should transition away from the Raspberry Pi and towards a board capable of running operating systems such as RTLinux, VxWorks, or LynxOS to name a few.

PIR Sensor Implementation

While different biases and assumptions that were later changed led to the selection of the Lepton v3 cameras as sensors for the prototype, much could be gained by sticking with PIR sensors. For example, the low frame rate on the Lepton cameras greatly decreased the range of speeds possible to detect. A PIR sensor sampled by and ADC would not have these limitations, as the sampling rate could be selected as one much higher. Thus a scaled down geometry of the system to allow for the smaller range of detection afforded by an off the shelf PIR sensors would more accurately prototype this system.

This prototype will need a new detection algorithm as well. The voltage wave form coming from the PIR sensor exhibits some instabilities due to ambient temperature changes, but unlike the thermal cameras, this sensor relies on a differential measurement and is therefore more stable. Instead of reading what can be extrapolated to temperature values, this differential waveform takes different shapes when intrusions occur from different directions. Thus the sensor would be monitored for this sort of behavior, while featuring some of the statistics based approaches found in the current algorithm to limit false positives due to ambient temperature changes. The poor sensor selection for the implemented prototype leaves a good deal of exploration to be done with the PIR sensors.

Given the width estimates are dependent on the speed estimates, it can be inferred that these estimates would be equally incorrect if not more, as they also have a factor of the incredibly noisy time estimates.

Appendix A

All code for this report can be found at:

https://github.com/MatthewDeKoning/Graduate-Project

Appendix B

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Glossary

ALL ABBREVIATIONS

ADC: Analog to Digital Converter

COTS: Commercially available Off The Shelf

FFC: Far Field Correction

FLIR: Forward Looking Infrared Radiometer (used as the name of a company)

FOV: Field Of View

FPS: Frames Per Second

GPIO: General Purpose Input/Output pin

I2C: Inter-IC bus

MPH: Miles Per Hour

PIR: Pyroelectric InfraRed sensor

SPI: Serial Peripheral Interface

SPS: Samples Per Second

USB: Universal Serial Bus

UVC: USB Video Class

V4L2: Video For Linux (version 2)

References

[1] B. Song, H. Choi, and H. S. Lee, "Surveillance tracking system using passive infrared motion sensors in wireless sensor network," in Proc. Int. Conf. Inf. Netw. (ICOIN 2008), Jan. 2008, pp. 1–5.

[2] Burroughs, C. (2006, December 4). Sandia researchers develop better sensor detection system. Retrieved November 09, 2017, from https://share-ng.sandia.gov/news/resources/releases/2006/sensor.html

[3] Gopinathan, U., Brady, D. J., & Pitsianis, N. P. (2003). Coded apertures for efficient pyroelectric motion tracking. Optics express, 11(18), 2142-2152.

[4] Hao, Q., Hu, F., & Xiao, Y. (2009). Multiple human tracking and identification with wireless distributed pyroelectric sensor systems. IEEE Systems Journal, 3(4), 428-439.

[5] Keller, H. J. 30 Years of Passive Infrared Motion Detectors-a Technology Review. In Proc. OPTO/IRS2 Conf, Chicago

[6] Leys, C., Ley, C., Klein, O., Bernard, P., & Licata, L. (2013). Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. Journal of Experimental Social Psychology, 49(4), 764-766.

[7] PIR motion detector - a sensor for Arduino and Raspberry Pi (1st part). (n.d.). Retrieved November 09, 2017, from http://www.meccanismocomplesso.org/en/pir-motion-detector/

[8] Weather History for KABQ - August 11, 2017. (n.d.). Retrieved November 09, 2017, from https://www.wunderground.com/history/airport/KABQ/2017/8/11/DailyHistory.html?req_city=&req_state=&req_statename=&reqdb.zip=&reqdb.magic=&reqdb.wmo=

[9] Weather History for KABQ - August 16, 2017. (n.d.). Retrieved November 09, 2017, from https://www.wunderground.com/history/airport/KABQ/2017/8/16/DailyHistory.html?req_city=&req_state=&req_statename=&reqdb.zip=&reqdb.magic=&reqdb.wmo

[10] Weather History for KABQ - July 31, 2017. (n.d.). Retrieved November 09, 2017, from https://www.wunderground.com/history/airport/KABQ/2017/7/31/DailyHistory.html?req_city=&req_state=&req_statename=&reqdb.zip=&reqdb.magic=&reqdb.wmo=

[11] Zappi, P., Farella, E., & Benini, L. (2010). Tracking motion direction and distance with pyroelectric IR sensors. IEEE Sensors Journal, 10(9), 1486-1494.

[12] Zappi, P., Farella, E., & Benini, L. (2007, September). Enhancing the spatial resolution of presence detection in a PIR based wireless surveillance network. In Advanced Video and Signal Based Surveillance, 2007. AVSS 2007. IEEE Conference on (pp. 295-300). IEEE.