Computer Vision Tracking of sUAS from a Pan/Tilt Platform

Jeremy P. Ogorzalek

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Aerospace Engineering

Jonathan T. Black, Chair
Scott L. England
Mark L. Psiaki

May 13, 2019
Blacksburg, Virginia

Keywords: sUAS, Computer Vision, Target Tracking, PTU

Computer Vision Tracking of sUAS from a Pan/Tilt Platform

Jeremy P. Ogorzalek

(ABSTRACT)

The ability to quickly, accurately, and autonomously identify and track objects in digital images in real-time has been an area of investigation for quite some time. Research in this area falls under the broader category of computer vision. Only in recent decades, with advances in computing power and commercial optical hardware, has this capability become a possibility. There are many different methods of identifying and tracking objects of interest, and best practices are still being developed, varying based on application. This thesis examines background subtraction methods as they apply to the tracking of small unmanned aerial systems (sUAS). A system combining commercial off-the-shelf (COTS) cameras and a pan-tilt unit (PTU), along with custom developed code, is developed for the purpose of continuously pointing at and tracking the motion of a sUAS in flight. Mixtures of Gaussians Background Modeling (MOGBM) is used to track the motion of the sUAS in frame and determine when to command the PTU. When the camera is moving, background subtraction methods are unusable, so additional methods are explored for filling this performance gap. The stereo vision capabilities of the system, enabled by the use of two cameras simultaneously, allow for estimation of the three-dimensional position and trajectory of the sUAS. This system can be used as a supplement or replacement to traditional tracking methods such as GPS and RADAR as part of a larger unmanned aerial systems traffic control (UTC) infrastructure.

Computer Vision Tracking of sUAS from a Pan/Tilt Platform

Jeremy P. Ogorzalek

(GENERAL AUDIENCE ABSTRACT)

The ability to quickly, accurately, and automatically identify and track targets in digital images has been of interest for some time now. Research in this area falls under the broader category of computer vision. Only in recent decades, with advances in computing power and commercial optical hardware, has this ability become a possibility. There are many different methods of identifying and tracking targets of interest, and best practices are still being developed, varying based on application. This thesis examines background subtraction methods as they apply to the tracking of small unmanned aerial systems (sUAS), commonly refered to as drones. A system combining cameras and a moving platform, along with custom developed code, is developed for the purpose of continuously pointing at and tracking the motion of an sUAS in flight. The system is able to map out the three-dimensional position of a flying sUAS over time.

Dedication

This thesis is dedicated firstly to my parents, who have given me everything I've ever needed to succeed at anything. I hope to follow in their footsteps, and pay that forward some day. Secondly, this thesis is dedicated to my significant other, whoever that may be at the time of this reading. They mean the world to me.

Acknowledgments

I would like to acknowledge my advisor, Dr. Jonathan Black, for giving me the opportunity to study aerospace engineering at Virginia Tech. I do not take such an experience for granted, and consider myself very lucky. I also thank him for his advice and guidance throughout the past two years. I am grateful for my other two committee members, Dr. Psiaki and Dr. England. I have learned much from both of them. I would also like to acknowledge the rest of the faculty, staff, and students from the Space@VT lab and the Hume Center. Interacting with so many intelligent, hard-working, and professional people has been incredibly enjoyable and motivating.

Contents

List of Figures

List of Tables

List of Abbreviations

CCD Charged-Coupled Device, the type of imaging technology used in this work.

COTS Commercial Off-The-Shelf.

ECEF Earth-Centered Earth-Inertial, a reference frame centered on the Earth, with X-axis aligned with latitude=0 and longitude=0, and Z-axis aligned with the polar axis North.

ENU East-North-Up, a local reference frame centered on a particular spot on the surface of the Earth.

FN False Negative, an error that occurs when an algorithm thinks an object of interest is not present when one is.

FP False Positive, an error that occurs when an algorithm thinks an object of interest is present when one isn't.

FPS Frames Per Second, a measurement of the speed of an algorithm.

GPS Global Positioning System, a system of satellites that provide precise geolocation.

LLA Latitude, Longitude, Altitude, measured in degrees, degrees, and meters.

MOGBM Mixtures of Gaussians Background Modeling, a family of background modeling and subtraction techniques used for motion detection.

PTU Pan-Tilt Unit.

RMSE Root Mean Square Error, a measure of distance error.

sUAS Small Unmanned Aerial Systems.

UTC Unmanned Aerial Systems Traffic Control.

Chapter 1

Introduction and Motivation

1.1 Introduction to Target Tracking

Tracking objects and modeling their behavior has long been of broad interest to the scientific and engineering community. Radar was developed during World War II expressly for this purpose, and offered an immense tactical advantage to those who possessed it over those who did not. Global navigation satellite systems, of which the Global Positioning System (GPS) is the most widely known, is an extension of the principles of radar allowing for precise positioning across the Earth. Both these methods have their limitations. RADAR systems require a signal reflection from the object. As such, the object needs to be of suitable size and shape to return a useful signal. Additionally, any other objects in the area can reflect signals, making it difficult to isolate the intended object. Tracking and object via GPS requires the object of interest to have a receiver and to be transmitting the relevant state information, making it unsuitable for non-transmitting or non-cooperative objects. An alternative and/or supplement to these techniques is tracking based on electro-optical sensing, otherwise known as digital imaging. Digital images of an object can be captured and analyzed, extracting information about the object's three-dimensional properties and behavior from the two dimensional images. The process of extracting useful information from digital images is known broadly as computer vision. All that is required to image an object is direct line-of-sight. Clear air and sufficient lighting are required to image an object

in the visible spectrum, but are not necessary in some other wavelengths. Images can be taken using light from across the electro-magnetic spectrum. Digital images function even in cluttered environments where the object of interest is one of many objects present, and do not require the object of interest to be cooperative or transmitting in any sense.

Today, with the all-around advancements in autonomy and AI that are entering our society, there are a plethora of desirable objects to track. There is a large focus in the automobile industry on bringing self-driving cars to market. Of paramount importance to both consumers and developers of self-driving cars is the safety of the drivers of these cars, as well as the drivers and pedestrians around them. In many cases, object tracking computer vision techniques have been implemented, alongside other sensing phenomenologies, to aid in the decision-making process for these vehicles. Research efforts in autonomous automobiles have tested and implemented a wide variety of computer vision techniques, and conclusions of best practices can vary widely depending on use-case and environment [7], [8], [31].

1.2 sUAS Proliferation

Small Unmanned Aerial Systems (sUAS) are becoming an increasingly prevalent component of our society. There are many ongoing efforts to integrate them safely and efficiently into our existing infrastructures [39]. sUAS are already being used to inspect bridges and buildings for damage, decreasing risk for humans. On the entertainment side, there exists a rapidly growing first-person-view drone racing community as well as one for aerial photography. In retail, large companies hope to revolutionize how goods are delivered to consumers by using sUAS.

The same level of emphasis placed on the safety of autonomous automobiles must be placed on sUAS going forward. With increasing frequency, sUAS are operating in environments

alongside humans, and pose much of the same risk as driverless cars [33]. One important element of safety will be knowing when an sUAS enters an environment and where it is within that environment. Many of the same computer vision techniques being used in the automobile industry can be applied, with some modification, to the problem of detecting and tracking sUAS for safer flight.

Many companies are working towards delivery of consumer products via sUAS. Operating on a large scale, logistics would require numerous sUAS departing and returning to central distribution locations. The potential for collisions would be ever-present. A computer vision system with a pan/tilt unit tracking the individual sUAS, could, in concert with on-board sensors, provide valuable information to human supervisors, schedulers, and operators, as well as to the sUAS themselves to be used in both on-board and off-board processing and decision making. In effect, the system could act as a kind of autonomous air traffic control for the 'airport' that is the distribution center.

ADS-B, used as a component of air traffic management on most manned aircraft, is assumed to become an integral part of sUAS safety. As [29] points out, however, there are issues of both practicality and reliability. One of the issues is the high cost of an ADS-B system. At a few thousand dollars, it is not much compared to the cost of a commercial airliner, but it is for a typical sUAS. Another issue raised is the congestion of the wavelengths that ADS-B systems transmit on, which can result in severe degradation or entire loss of transmitted information. Finally, there is always the possibility for jamming or spoofing of all types of wireless communication, including GPS signal. An electro-optical ground-based system for monitoring sUAS traffic, such as the one being investigated in this work, would be immune to these issues, and would increase the robustness and safety of any air traffic management system involving sUAS.

1.3 Applicability to Other Domains

Computer vision techniques are applicable to a wide variety of domains outside of autonomous vehicles and commercial drone management. Fields as diverse as medical pathology [14] and agriculture [1] are inventing novel uses for the technology. Real-time object detection and tracking is a broad field in and of itself that includes use cases such as orbital debris cataloging [30], fishing regulation enforcement [52], and interpreting hand gesture commands [38]. The military and defense uses for autonomously detecting, tracking, and aiming at moving objects are numerous and easily imagined. The techniques discussed in this work are especially helpful when dealing with non-cooperative objects which are not sharing information such as GPS data.

1.4 Thesis Contributions

This thesis details the development of a deployable, computer vision-enabled, sUAS tracking system. Several different background subtraction algorithms are tested to determine their suitability for tracking sUAS. These algorithms have been compared previously in other domains, such as automobile traffic monitoring, but here they are compared with a custom test video data set focused on flying sUAS. Once a background subtraction algorithm is selected, the various parameters of said algorithm, along with parameters controlling the pre- and post-processing of the videos, are optimized.

Many computer vision applications include a Kalman filter for improved state estimates and predictive capabilities, but only include up to acceleration in their state vector. This thesis develops and tests Kalman filters that take into consideration not only velocity and acceleration, but also jerk and snap, the third and fourth order time derivatives of position.

Many previous research efforts use background subtraction methods only with a static, non-moving camera. This thesis briefly described a novel method that could be used to deal with camera motion while using background subtraction, and compares it to an existing solution.

Two cameras are put on the PTU, enabling three-dimensional tracking of an object via stereo vision. The system's three-dimensional real-world tracking ability is tested against truth data generated by a GPS receiver on-board the sUAS. Stereo vision itself is not new, but its use with measurements taken of a flying sUAS via background subtraction methods is novel.

Such a system could be used as part of a UTM in conjunction with existing detection and tracking technologies. In addition, the system can be used as part of future research efforts in the lab. The system has excellent mobility, as all required components fit on and can be transported on a single cart.

1.5 Document Preview

This document is laid out in the following manner. Section 2 describes the fundamental theory behind much of the work, as well as reviewing relevant research available in the literature. Section 3 describes the hardware components being used. Section 4 details the development of the two-dimensional computer vision tracking algorithm, and briefly evaluates a novel technique for dealing with camera motion. Section 5 extends the tracking capabilities of the system to three dimensions through the simultaneous use of two cameras, and compares results against truth data provided by GPS. Section 6 concludes the thesis with general discussion and future work.

Chapter 2

Theory and Literature Review

2.1 Computer Vision Theory

2.1.1 Computer Vision Basics

Computer vision, or machine vision, as it is sometimes called, is, at its most basic level, the analysis of pixel intensity values from digital imagery for the purpose of extracting useful information from the images. Often, the information that is to be extracted would be trivial for a human to perceive. For example, identifying an animal in an image, or determining the size and shape of a building, are tasks that can be completed by small children. For a computer to do the same thing often requires advanced algorithms and clever strategies.

Digital images are composed of rows and columns of pixels, each assigned one or several values corresponding to the intensity of the light the pixel replicates. Pixels in black and white images, also known as greyscale images, are assigned a single value, whereas pixels in color images typically are assigned three values. The three values of a pixel in a colored image can correspond to levels of red, green, and blue, or they can follow a variety of other color coordinate systems that have been created. Images with multiple intensity values assigned to each pixel are called multi-channel images. Pixel intensity values are typically represented in an 8-bit system, meaning there are 256 possible values ranging from 0 to 255. In a black and white image, a pixel value of 0 is pure black, and 255 is pure white, with values in

between corresponding to varying levels of grey.

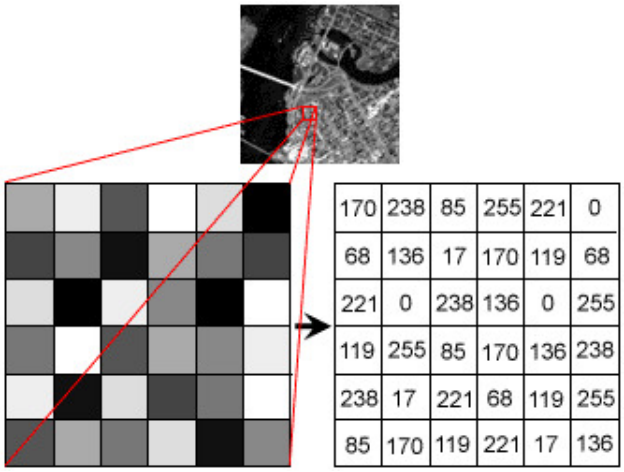


Figure 2.1: How computers 'see' the world. An image represented by an array of pixel intensity values ranging from 0 to 255. [57]

In computer vision, a single image can be analyzed, in which case pixel values can vary based on their row (x) or column (y) position in the image. Alternatively, a series of digital images, a video, can be analyzed, which provides another dimension over which pixel values can change. This third dimension is typically time, as images in a video are typically ordered chronologically.

2.1.2 Basic Computer Vision Techniques

As mentioned before, computer vision is the field of extracting useful information from the pixel values in digital images. These pixel intensity values, typically represented with large arrays of integers, are the only information any computer vision algorithm is able to use. Many useful ways to manipulate these pixel values have been developed over the years. Some of the basic techniques used in this work are explained in this section.

Digital images are susceptible to noise in the form of random pixel values distributed across

the array. The source of this noise is often electrical jitter somewhere in the circuitry of the imaging hardware being used, or corruption of the data. Because computer vision techniques rely on images representing the real world accurately, this noise is undesirable. A popular method of de-noising an image is the process of blurring. Blurring involves performing a convolution between the image and a kernel. In this context, a kernel is a relatively small matrix whose elements are chosen such that when convolved with the image, a particular result is achieved. The process of convolution involves element by element multiplication between the kernel and every sub-array, with dimensions equal to those of the kernel, that can be formed from the digital image. The result of that multiplication is typically divided by the number of elements in the kernel, such that the result is a type of average.

A blurring kernel is designed such that each pixel value in an image is replaced with an average of the pixel values surrounding it. The simplest blurring kernel is an array of all ones. More popular is a Gaussian blurring kernel, whose values roughly follow a Gaussian distribution, with the center value being the highest. An example illustrating this concept and its effect is shown in Figure 2.2.

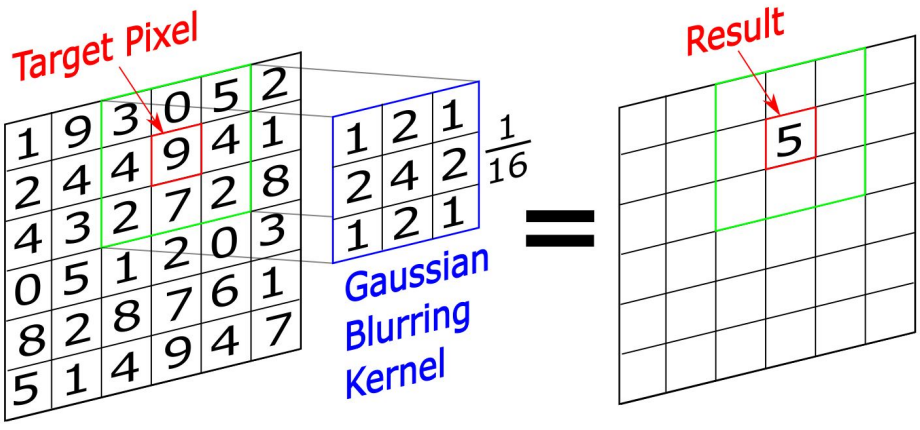


Figure 2.2: Operating on a target image with a Gaussian blurring kernel.

Another set of common computer vision techniques are known as morphological operations,

named so because they change the shapes of objects in images they operate on. Erosion and dilation are two of the most basic morphological operations. These techniques are applied to black-and-white, also known as binary, images. These images typically have mostly black, with objects of interest, whatever they may be, in white. A morphological operation uses a kernel, but in a different way than blurring.

In the erosion operation, as the kernel passes over the image, it determines if any of the pixels it covers are black. If one or more elements under the kernel are black, then the central pixel will be turned black. Only if all pixels under the kernel are white will the central pixel remain white. Dilation is the opposite, where is any pixels under the kernel are white, the central pixel will become white.

(a) Original. (b) Erosion. (c) Dilation.

Figure 2.3: Basic morphological operations on binary image. [59]

Erosion and dilation are often performed sequentially. Erosion can help rid the image of small areas of noise, but also shrinks the size of the object of interest in the image. Dilation is then performed, which enlarges the object, but does not bring back any noise that was removed during the erosion process. Erosion followed by dilation is known as opening. Dilation followed by erosion is known as closing, and is used to fill small holes in the object of interest, or to join together separate areas of white [59].

(a) Effect of Opening. (b) Effect of Closing.

Figure 2.4: Sequential morphological operations on binary image. [59]

An important concept in computer vision is that of features. Features in an image are areas or regions, typically very small in relation to the whole image, that are unique enough to determine where in the image they came from. In practice, the best features in an image tend to be areas with strong pixel intensity gradients in two directions, ie. corners. Figure 2.5 makes it easy to see why corners are considered good features. Templates A and B contain only patterns, and can be localized to a region of the image, but not a specific point. Templates C and D contain edges, and can be localized to a line segment, but, again, not to a specific point on the line. Templates E and F contain corners, and it is easy to localize them to a specific point in the image. Corners are easy for humans to match between images, and in this case, the same is true for computers.

Given a template of an object, that object can be found in a new image via feature matching. Feature matching works in two steps. The first is the discovery and description of features in two separate images. The second is matching similar features between the two sets. Features are matched based on the similarity of their descriptors, which are characterizations of the pixels composing and surrounding the features. A number of ways to find, describe, and match features have been developed.

Figure 2.5: Visualization of why corners are considered good features to find and track in an image. Only templates E and F are easy to localize in the larger image. [59]

2.1.3 Computer Vision and Detection/Tracking

Computer vision, as a recognized field of serious study, began in the 1970's, although efforts to have computers analyze and describe digital images can be found in the 1960's [51]. The first computer vision algorithms developed performed tasks such as identifying lines or edges based on pixel value gradients. Throughout the next few decades, more complex models and methods were introduced, including those relying on probability and machine learning.

The detection and tracking of objects of interest in images has always been a goal in the field of computer vision. Here, detection refers to identifying and localizing a feature of interest in an image. Tracking refers to maintaining localization knowledge of that feature of interest across multiple images, typically ordered chronologically. Applications for such a capability include remote security monitoring, augmented reality sports broadcasting, and many more.

To track objects in real-time in high resolution video requires appreciable computational power. With this power now widely accessible, especially to researchers, there have been major strides in the advancement of detection and tracking methods. There are a great number of object detection and tracking techniques that have been developed over the past decades, and best practices are still being sought. [58] groups detection methods into four categories: point detectors, segmentation, supervised classifiers, and background modeling.

Point detector algorithms locate individual pixels or groups of pixels that are easily distinguishable from the surrounding image features. These features are typically corners where two distinct lines meet, as these features are easiest to identify and correlate between images, as explained in Section 2.1.2. Popular algorithms for feature detection are the Harris Corner Detector [17], and the Shi-Tomasi Good Features to Track method [47]. Many different algorithms for describing and matching features between images have been developed, including the patented methods SIFT [32] and SURF [2], along with the free and open-sourced ORB [45].

Segmentation algorithms aim to divide the image into separate regions based on some defined criteria. In the field of object detection and tracking, the image is typically divided into background and foreground. Background refers to anything that is not the object of interest, where foreground is the object. Popular segmentation methods include the Watershed Transformation [9] and Continuously Adaptive Meanshift [5].

Supervised classifiers encompass the machine learning techniques in which an algorithm 'learns' what it is looking for. The classifier can be trained offline beforehand, or can learn online with some input from the user. The classifier will then attempt to find the object of interest in each subsequent frame.

Background modeling and subtraction is the group of methods that attempt to estimate the

composition of the background, and compare information in new frames to that background model. Any pixels in new frames that do not fit the background model can be classified as foreground. In this way, background modeling and subtraction can be considered a form of image segmentation. One major difference between background modeling and the segmentation techniques described by [58] is that background modeling techniques all leverage some amount of history in their algorithms.

Background modeling techniques are the main detection and tracking method used in this work. Their evolution and function are described more fully in Section 2.1.4. Background modeling and subtraction methods have the advantage of not requiring any information prior to operation. Classification methods, on the other hand, require prior knowledge about the appearance of the object to be located. Background modeling and subtraction methods are therefore much more general, and can be used to detect any moving object.

2.1.4 Mixture of Gaussians Background Modeling and Subtraction

As mentioned before, most detection and tracking algorithms attempt to find an object of interest in an image, often referred to as the foreground, and follow its motion over time. Background modeling and subtraction algorithms are a family of techniques that all involve trying to resolve an image into its background and foreground components by building a model of the background scene, and comparing each subsequent image to that model. By comparing subsequent frames to this background model, the algorithm can determine what is not part of the background, and therefore must be part of the foreground.

One of the simplest methods of background subtraction in video is sequential frame differencing. In this method, first discussed in [21], the background model is simply the previous

frame, or a frame that was supplied beforehand and is known to contain only background. Each new frame's pixel intensity array is literally subtracted, element by element, from the previous frame's. Pixel intensity values that did not change between the two frames will cancel out, resulting in a zero. Regions in which values did change will result in a non-zero values. If the absolute value of subtracted pixel intensities is above a predetermined threshold T, the pixel is classified as foreground, as seen in Equation 2.1. This method, visualized in Figure 2.6, is rigid, and requires an unchanging background to perform well.

$$Pixel \quad (x,y) \quad is \quad foreground \quad if \quad |I_{x,y}(t-1) - I_{x,y}(t)| > T \tag{2.1}$$

(a) Background Image. (b) New Image. (c) Differencing Result.

Figure 2.6: Literal differencing, in which the new image intensity array is subtracted from the background image intensity array.

Instead of each background pixel being represented by a single intensity value, [56] suggested that each background pixel be represented by a changing Gaussian probability density function. In this model, each pixel of the background has it's intensity modeled by a normal Gaussian probability density function, seen in Equation 2.2.

$$P(I_{x,y}) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(I_{x,y}-\mu)^2/2\sigma^2} \tag{2.2}$$

Here, μ is the mean and σ^2 is the variance of the intensity values expected at pixel (x, y).

μ for each pixel's function is a type of weighted average of the past intensity values of that pixel. The rate at which μ changes is controlled by the parameter α, often referred to as the learning rate of the model. Each new μ and σ^2 for the component representing pixel (x, y) at time t, are calculated with Equations 2.3 and 2.4.

$$\mu_{x,y}(t) = \alpha * I_{x,y}(t) + (1 - \alpha) * \mu_{x,y}(t - 1) \tag{2.3}$$

$$\sigma^2_{x,y}(t) = \alpha * (I_{x,y}(t) - \mu_{x,y}(t - 1))^2 + (1 - \alpha) * \sigma^2_{x,y}(t - 1) \tag{2.4}$$

In this way, only the mean at the previous step $\mu_{x,y}(t-1)$ and the newly measured intensity value $I_{x,y}(t)$ are required to compute the updated mean $\mu_{x,y}(t)$. No history of past measurements are required to be stored in memory. In subsequent frames, the new pixel values are compared to their respective background model Gaussian probability distributions, and some form of hypothesis testing is performed. If it is deemed likely that the new pixel value belongs to the distribution, based on some predefined confidence interval, then the new pixel value is classified as background. Otherwise, the new pixel value is classified as foreground. This strategy affords the benefit of statistical probability to background/foreground segmentation as well as adaptation over time, allowing for a changing background scene.

α's value controls how quickly $\mu_{x,y}$ and $\sigma^2_{x,y}$ converge to newly observed data values. If α was set to 1, then $\mu_{x,y}(t) = I_{x,y}(t - 1)$ at every step, likewise for $\sigma^2_{x,y}(t)$, and previous measurements have no influence. Conversely, if α was set to 0, $\mu_{x,y}$ and $\sigma^2_{x,y}$ would never update.

As [25] points out, this formulation updates the background model with every new pixel, even if that pixel is deemed to contain foreground. [25] suggests to reformulate the background model Gaussian mean and variance update with Equations 2.5 and 2.6.

$$\mu_{x,y}(t) = (1 - M) * (\alpha * I_{x,y}(t) + (1 - \alpha) * \mu_{x,y}(t - 1)) + M * \mu_{x,y}(t - 1) \quad (2.5)$$

$$\sigma^2_{x,y}(t) = (1 - M) * (\alpha * (I_{x,y}(t) - \mu_{x,y}(t - 1))^2 + (1 - \alpha) * \sigma^2_{x,y}(t - 1)) + M * \sigma^2_{x,y}(t - 1) \quad (2.6)$$

Here, the M coefficient is equal 1 if the new pixel is deemed foreground, and 0 otherwise. In this formulation, if the new pixel is foreground, and therefore $M = 1$, the new function mean and variance are simply equal to their previous values, and the newly measured pixel intensity value has no impact on the model. The background model is only updated by new measurements that are part of the background, which is intuitively beneficial.

This method remains unfit for a background that may take on several distinct values at a given pixel. Take for instance, the case of a tree branch blowing in the wind. A background pixel's intensities may alternate between the light blue of the sky and the dark brown of a tree branch rapidly. Changes in illumination in a scene can cause similar issues. In these cases, a single Gaussian distribution can not accurately model the background behavior. Mixture of Gaussians background modeling (MOGBM) aims to address this issue.

First suggested in [50], and improved in [22], MOGBM method models each background pixel with a mixture (a summation) of K separate Gaussian probability density functions, each with an associated weight factor. Each new pixel value is compared to the resulting distribution, and judged to either be part of that distribution or not, again using some form of hypothesis test. For example, in [50], a pixel is determined to be part of the background scene if it's value falls within 2.5 standard deviations of one of the component distributions' means. If the pixel is classified as foreground, a new Gaussian distribution is created with mean equal to the pixel's intensity value. A weighted mixture of Gaussians probability density function, consisting of K components takes the of Equation 2.7.

$$P(I_{x,y}) = \sum_{k=1}^{K} \omega_k * \eta(I_{x,y}, \mu_k, S_k) \tag{2.7}$$

Each $\eta(I_{x,y}, \mu_k, S_k)$ component is an individual Gaussian probability density function of the form shown in Equation 2.2. ω_k is the weight associated with the k'th component. The weight ω_k quantifies how much of the measured data belongs to the k'th component of the probability distribution. In total, these weights must sum to one, and must remain non-negative.

In MOGBM, each component k is able to describe a separate background object. For instance, $k = 1$ may describe the sky, and $k = 2$ may describe a tree branch. [50] artificially sets the number of components, K, to be between three to five. As new pixel intensity values are measured, they are classified as either background or foreground. If classified as background, the weight associated with the Gaussian component that best describes the new pixel intensity value is increased. All components that do not describe a new pixel's intensity value have their weights decreased. The weight adjustment formula is Equation 2.8, where M_k is the same as in Equations 2.5 and 2.6.

$$\omega_k(t) = (1 - \alpha) * \omega_k(t - 1) + \alpha * M_k \tag{2.8}$$

If new pixel is classified as foreground, a new normal Gaussian is created, centered around that foreground pixel's intensity value, but is not immediately considered part of the background model. In this way, the algorithm may store a great number of separate normal Gaussians, but only the K most likely to describe the background are used to compose the mixture of Gaussians background model. Figure 2.7 shows a mixture of Gaussians model with $K = 3$. Even though five components are stored, only the three with the greatest

associated weights ω_k, shown shaded in blue, are considered part of the background model. The figure shows how two different new pixel intensity measurements, represented by the green and red dashed lines, would likely be classified.

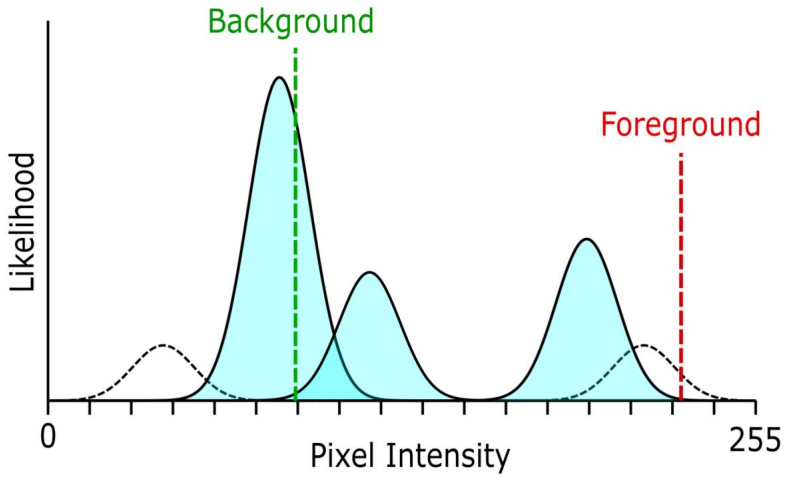


Figure 2.7: Classifying newly measured pixels as either foreground or background using a mixture of Gaussians background model with K=3.

[60] improved the MOGBM theory further by allowing for a changing number of distributions K to model each pixel. Instead of defining this number beforehand, the algorithm adapts while running, finding the optimal number of distribution components. In theory, this allows a background pixel to be described by a single normal Gaussian distribution, as all pixels are in [56], which is optimal when that background pixel's intensity remains a constant value, as with a static background scene. In [60]'s formulation, the weights of individual Gaussian components are updated with equation 2.9.

$$\omega_k(t) = (1 - \alpha) * \omega_k(t - 1) + \alpha * M_k - \alpha * c_T \tag{2.9}$$

This equation is the same as 2.8, but with the addition of the $-\alpha * c_T$ term. This term allows weight ω_k to become negative, at which point its associated component is removed

from the background model composition. In this way, the number of components that make up the mixture of Gaussians probability density function background model can change over time. $c_T = c/T$, the ratio of previous measurements that are well defined by the component distribution under consideration to all T previous measurements. ω_k becomes negative when less than c out of the previous T measurements are described well by component k, at which point component k is removed from the background model.

The algorithm described in [60] is implemented in OpenCV as MOG2. It is widely regarded as the best performing background modeling and subtraction method available in OpenCV.

2.1.5 OpenCV

OpenCV is an open-source library of code containing a large number of computer vision algorithms. It is usable with the C++, Java, and Python programming languages, and is supported on Windows, Mac OS, Linux, and other operating systems. OpenCV has been used across many disciplines, and is used frequently in computer vision research, as its functions are simple to used and highly optimized. All code used this this thesis was written using Python version 3.7.1 and is compatible with the 3.4.5 version of OpenCV.

2.2 General Theory

2.2.1 Pinhole Camera Model

The pinhole camera model, seen in Figure 2.8, is the starting point for most basic analysis involving cameras. The pinhole camera model is an idealized, yet useful, model of the mapping between point locations in a 3D space and, having been photographed, their 2D

location in the resulting image. In other words, the model is used to transform 3D world coordinates to 2D image coordinates. Known geometrical distortions that commonly occur with the use of camera lenses are ignored. The camera is modeled as having a single, infinitesimal aperture through which light may pass and project onto the image plane. In this manner, only a single ray of light may fall on any one point of the image plane. The distance between the image plane and the aperture is defined as the focal length of the camera, f.

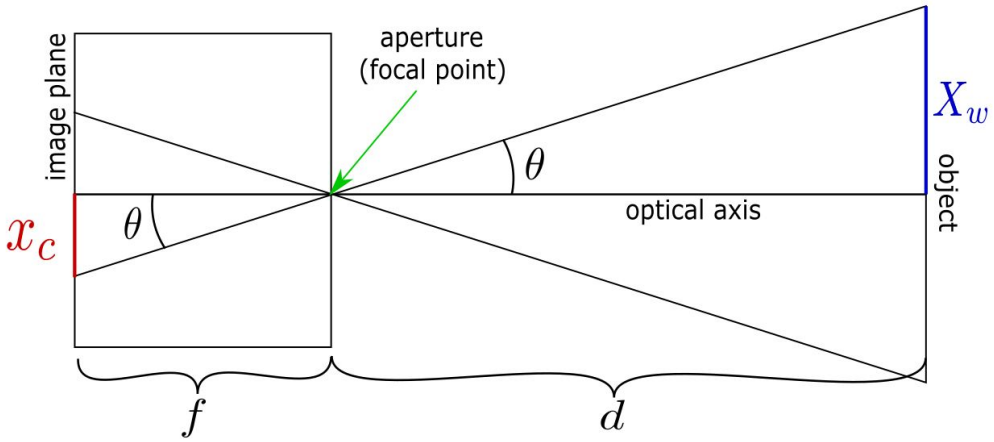


Figure 2.8: The pinhole camera model.

All 3D points can be mapped to a position on the 2D image plane with the use of the geometry of two similar right triangles. The line segments denoting the ray of light traveling from the 3D point to the aperture, and from the aperture to the focal plane, make up the hypotenuses of the two similar right triangles. For cameras where the image plane is behind the focal point, the mapping is given by Equation 2.10.

$$\begin{bmatrix} x_c \\ y_c \end{bmatrix} = -\left(\frac{f}{Z_w}\right) * \begin{bmatrix} X_w \\ Y_w \end{bmatrix} \tag{2.10}$$

Where x_c and y_c are positions in the image plane, X_w, Y_w, Z_w are 3D positions in the world, and f is the focal distance of the idealized camera. The information about the distance to the object is lost. The negative is used because the image is flipped across both the X and Y axes. The same geometry applies, with a sign change, to scenarios in which the image plane is brought in front of the focal point of the camera, as it is in modern charged coupled device (CCD) cameras. In this scenario, the image is not flipped, and the negative sign is not required.

In a CCD camera, the image plane is made up of rectangular light sensors, each of which correlate to a specific pixel in the resulting image. With knowledge of the number and size of the pixel sensors in a CCD, along with the focal length of the camera/lens system, the angular field of view of a particular camera can be determined through simple trigonometry as seen in Figure 2.9. With n_p being the number of pixels in a certain direction, and l_p being the length of each pixel in that direction, the equation for field of view is Equation 2.11. The horizontal and vertical fields of view are solved for in the same manner, but the n_p and l_p variable values may differ in the two directions.

$$\boldsymbol{\theta_{FOV}} = 2 * \arctan\left(\frac{n_p * l_p}{2f}\right) \tag{2.11}$$

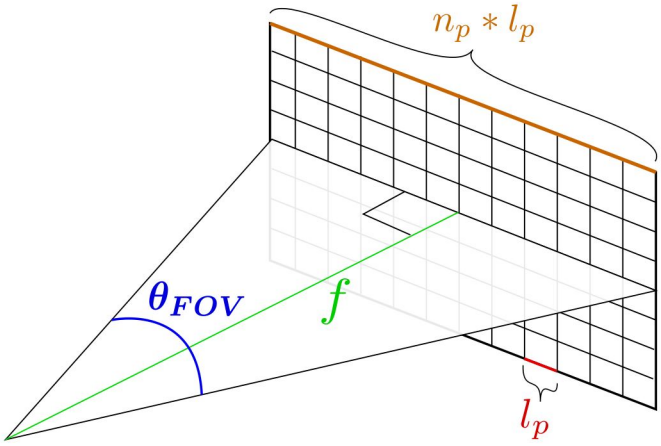


Figure 2.9: Angular field of view geometry using the pinhole camera model.

With this same theory, one can determine the angle between two points in world coordinates based purely on their pixel location in the image plane. In this work, the main concern is the angles between a object of interest and the optical axis of the camera system, which is a line perpendicular to and intersecting the center of the image. These angles are the angular offsets of the object. One of these angular rotations must be defined as occuring before the other. In this work, the horizontal angle offset, ψ, will be defined as occuring before the vertical angle offset, ϕ. Using the geometry of the problem as seen in Figure 2.10, we can calculate these angles using Equations 2.12 and 2.13.

$$\theta = \arctan\left(\frac{x_c}{f}\right) \tag{2.12}$$

$$\phi = \arctan\left(\frac{y_c}{\sqrt{x_c^2 + f^2}}\right) \tag{2.13}$$

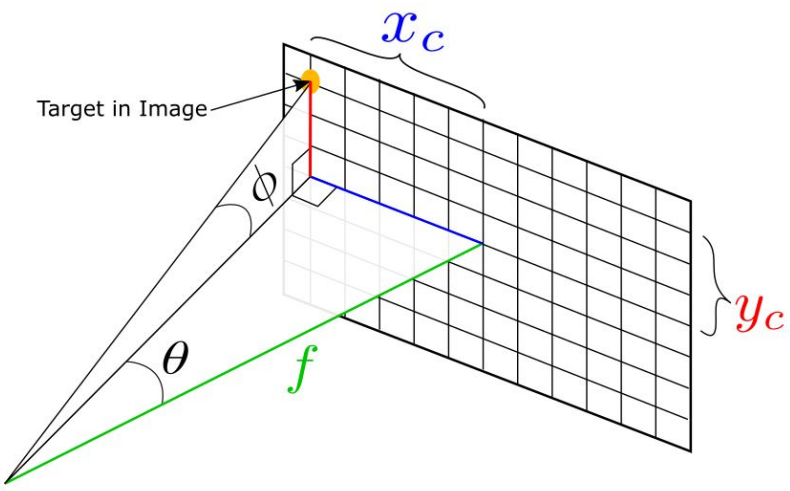


Figure 2.10: Geometry of view angles to a target point in an image using the pinhole camera model.

All distances used in these equations must be in the same units, either pixels or meters. It is important to note that pixel sensors may not be square, and therefore may have a different l_p depending on which axis they are being measured across.

2.2.2 Camera Calibration

The pinhole camera model is not an exact representation of any modern camera, as there are distortions induced by the hardware being used. A common type of distortion, typically induced by photographic lenses, is radial distortion. This type of distortion is due to the radial symmetry of the lens being used, and is much more apparent with the use of short focal length, wide field-of-view cameras. Radial distortion causes straight lines to appear curved, more so near the edge of the image.

Another factor to consider when using off-the-shelf cameras, is that practicality typically limits the ability of camera components to match their specifications exactly. Like any manufacturing process, there are error tolerances that must be accepted in the assembly of

CCD cameras and lenses. Two specifications that often do not match the real hardware are the focal length and the optical center of the imaging plane. These physical characteristics are referred to as the camera's intrinsics, and are captured in a standard matrix form as follows:

$$\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.14}$$

The optical center of the camera, (c_x, c_y) represents the pixel location where the image plane is intersected by the optical axis of the camera/lens system which is, ideally, the horizontal and vertical center of the image. This location is often offset by some amount due to imperfection in the manufacturing process. The two focal length measurements, f_x and f_y, represent the focal length of the camera/lens system in units of pixels. These values will differ from each other when the pixels are not geometrically square. If the pixels are square, $f_x = f_y$. Focal length is typically given in the documentation for a particular camera/lens combination, but can vary, just like the optical center location, due to assembly imperfections.

A camera system's intrinsics and distortions can be measured and corrected for through the process of camera calibration. OpenCV provides several functions allowing for camera calibration. The process involves taking a number of images of a flat checkerboard-patterned object. A usable pattern can be printed out and taped to a hard flat backing. The camera calibration algorithms search for the corners of the checkerboard pattern, and, knowing that they form straight lines in reality, can develop the corrective coefficients required to fix the radial distortion in the images. In addition, the algorithm generates an estimate of the focal lengths and optical center coordinates, both in pixel units, that make up the elements of the

camera's intrinsic matrix [59].

2.2.3 Stereo Vision

As stated previously, the distance-to-object information, or range, is lost when mapping from world to image-plane coordinates from a single image. Therefore, from a single x and y-pixel coordinate pair, only a set of spherical angular offsets that define a line of bearing passing from the center of the camera to the object can be determined. With no further knowledge, the object can exist at any distance along that line. The distance to the object can be recovered only by combining information from two or more images, taken from different perspectives. These two images can come from the same camera at different times and orientations, as long as the object does not move. Alternatively, the images can come from two different cameras at the same time.

Using two cameras to take images at the same time is known as stereo vision. The human vision system, with its two eyes, is a stereo vision system, and is what allows a person to estimate depth to an object. With two sets of x and y-pixel coordinates from different camera perspectives, we can define two unit vectors $\hat{n_1}$ and $\hat{n_2}$, both pointing towards the same object, but with separate origin points, O_1 and O_2. The origin points are the focal points of each camera. The two unit vectors can be determined with the use of Equations 2.12 and 2.13. The geometry of the scenario, assuming the cameras are aligned such that their image planes share a common plane, is shown in Figure 2.12.

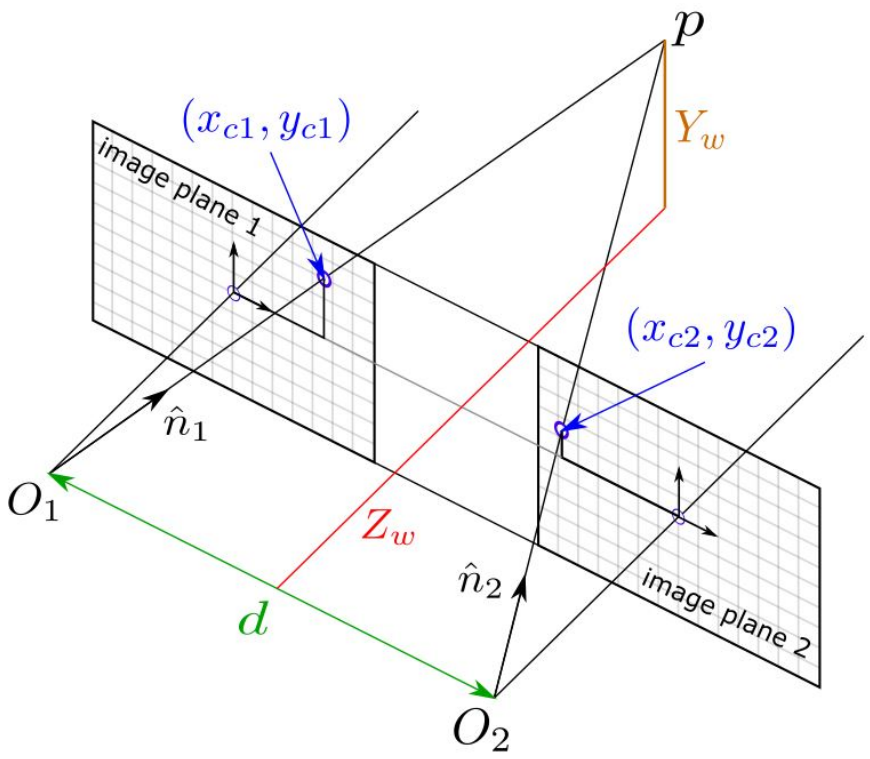
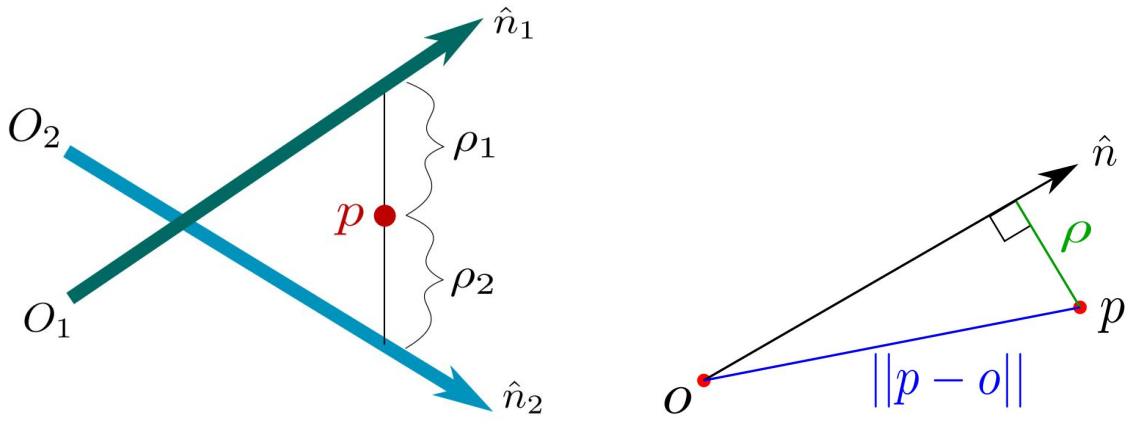


Figure 2.11: Dual view geometry

Figure 2.12: Geometry of stereo vision using cameras with co-planar image planes.

In a perfectly aligned and measured system, these two vectors will intersect precisely at the object. In reality, it is unlikely that these two vectors will intersect. The discretization of the image into a limited number of pixels, the camera calibration coefficients, and other sources of error cause the information to be imperfect. These non-intersecting vectors represent an over-constrained system, where there is no true answer. See Figure 2.13a. The best that can be done is to formulate a solution that most-closely satisfies all of the constraints. One method is to find the least-squares solution, in which the sum of the square of the errors is minimized.



(a) Point between two non-intersecting lines (b) Distance between point and line

Figure 2.13: Geometry of stereo vision point localization.

The minimum distance ρ from any point p to a line defined by an origin and a unit vector, o and $\hat{n}$, can be solved for knowing that the line of minimum distance will intersect $\hat{n}$ at a right angle, as seen in Figure 2.13b. The minimum Euclidean distance is solved for using Equation 2.15.

$$\rho = \sqrt{(\|p-o\|)^2 - ((p-o)^\intercal \cdot (\hat{n}))^2} = \sqrt{(p-o)^\intercal \cdot (p-o) - ((p-o)^\intercal \cdot (\hat{n}))^2} \tag{2.15}$$

With N lines, a least-squares solution can be formulated using the sum of the square of the minimum distances from the point to each line.

$$\begin{aligned} \sum_{i=1}^{N} \rho_i^2 &= \sum_{i=1}^{N} [(\|p-o_i\|)^2 - ((p-o_i)^\intercal \cdot (\hat{n_i}))^2] \\ &= \sum_{i=1}^{N} [(p-a_i)^\intercal \cdot (p-o_i) - ((p-o_i)^\intercal \cdot (\hat{n_i}))^2] \end{aligned} \tag{2.16}$$

The goal is to find the location of p which minimizes this sum of squared distances. As with many classical calculus optimization problems, we differentiate with respect to the dependent

variable. Differentiating with respect to point location p, and setting equal to zero results in Equation 2.17.

$$0 = \sum_{i=1}^{N} [2 * (p - o_i) - 2 * ((p - o_i)^\intercal \cdot \hat{n_i}) \cdot \hat{n_i}] \tag{2.17}$$

This equation can be manipulated into a more convenient form.

$$\sum_{i=1}^{N} (p - o_i) = \sum_{i=1}^{N} [(\hat{n_i} \cdot \hat{n_i}^\intercal) \cdot (p - o_i)] \tag{2.18}$$

$$\sum_{i=1}^{N} (p - o_i) = \sum_{i=1}^{N} [(\hat{n_i} \cdot \hat{n_i}^\intercal \cdot p) - (\hat{n_i} \cdot \hat{n_i}^\intercal \cdot o_i)] \tag{2.19}$$

$$\sum_{i=1}^{N} p - \sum_{i=1}^{N} o_i = \sum_{i=1}^{N} (\hat{n_i} \cdot \hat{n_i}^\intercal \cdot p) - \sum_{i=1}^{N} (\hat{n_i} \cdot \hat{n_i}^\intercal \cdot o_i) \tag{2.20}$$

All of the p terms are moved to the left side, and the o_i terms are moved to the right.

$$\sum_{i=1}^{N} (\hat{n_i} \cdot \hat{n_i}^\intercal \cdot p) - \sum_{i=1}^{N} p = \sum_{i=1}^{N} (\hat{n_i} \cdot \hat{n_i}^\intercal \cdot o_i) - \sum_{i=1}^{N} o_i \tag{2.21}$$

$$\left(\sum_{i=1}^{N} (\hat{n_i} \cdot \hat{n_i}^\intercal - I) \right) \cdot p = \left(\sum_{i=1}^{N} [(\hat{n_i} \cdot \hat{n_i}^\intercal - I) \cdot o_i] \right) \tag{2.22}$$

Equation 2.22 is of the familiar form $Ax = B$, with

$$A = (\sum_{i=1}^{N} (\hat{n_i} \cdot \hat{n_i}^\intercal - I)), \qquad B = (\sum_{i=1}^{N} (\hat{n_i} \cdot \hat{n_i}^\intercal - I) \cdot o_i), \qquad x = p$$

which we can solve by inverting A. We are left with Equation 2.23, the solution to p, a three dimensional vector representing the point closest to all lines in a least-squares sense. When expanded fully, Equation 2.23 becomes Equation 2.24.

$$p = \left(\sum_{i=1}^{N} (\hat{n}_i \cdot \hat{n}_i{}^{\intercal} - I)\right)^{-1} \left(\sum_{i=1}^{N} [(\hat{n}_i \cdot \hat{n}_i{}^{\intercal} - I) \cdot o_i]\right) \tag{2.23}$$

$$\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \left[\sum_{i=1}^{N} \begin{bmatrix} n_{ix}^2 - 1 & n_{ix}n_{iy} & n_{ix}n_{iz} \\ n_{ix}n_{iy} & n_{iy}^2 - 1 & n_{iy}n_{iz} \\ n_{ix}n_{iz} & n_{iy}n_{iz} & n_{iz}^2 - 1 \end{bmatrix}\right]^{-1} \left[\sum_{i=1}^{N} \begin{bmatrix} (n_{ix}^2 - 1)o_{ix} + (n_{ix}n_{iy})o_{iy} + (n_{ix}n_{iz})o_{iz} \\ (n_{ix}n_{iy})o_{ix} + (n_{iy}^2 - 1)o_{iy} + (n_{iy}n_{iz})o_{iz} \\ (n_{ix}n_{iz})o_{ix} + (n_{iy}n_{iz})o_{iy} + (n_{iz}^2 - 1)o_{iz} \end{bmatrix}\right] \tag{2.24}$$

2.2.4 Kalman Filters

Kalman Filters are a family of recursive state estimation algorithms used when it is infeasible to perfectly measure the state of a system. These filters provide the optimal state estimate, in a least-squares sense, when the measurement and process noises can be modeled as zero-mean and Gaussian-distributed. First described in [23], the standard Kalman Filter discrete-time system model takes the form of Equations 2.25 and 2.26.

$$\bar{x}_k = F * \bar{x}_{k-1} + B * \bar{u}_{k-1} + \bar{w}_{k-1} \tag{2.25}$$

$$\bar{z}_k = H * \bar{x}_k + \bar{v}_k \tag{2.26}$$

$\bar{x}$ represents the state of the dynamic system, typically a vector of elements. The subscripts $k-1$ and k represent the previous and the current time steps, respectively. F is the state

transition matrix, describing the dynamics of the system in the absence of any input or process noise, $\bar{w}$. In other words, simply multiplying the state transition matrix by the state vector at time k gives the state at time $k+1$ if there is no controller input or noise. B is the input matrix, relating the input, $\bar{u}$ to the state. $\bar{z}$ is the measurement, which may contain all, some, or none of the state variables. The measurement is related to the state through the measurement equation matrix, H, perfectly in the absence of measurement noise, $\bar{v}$. The two noise components, $\bar{w}$ and $\bar{v}$, are typically assumed to be normally distributed random variables that are independent of each other. $\bar{w}$ and $\bar{v}$ have covariances Q and R, respectively.

$$w \sim N(0, Q), \qquad v \sim N(0, R), \qquad Cov(w, v) = 0 \tag{2.27}$$

Kalman filters works recursively in a two-step process, the two steps commonly referred to as the prediction, or propagation, step and the correction step. In the prediction step, the filter estimates the state at the next time step using information about the previous state and control inputs. The measurement taken at the next time step will usually suggest a different state than that predicted. The measurement and the prediction, along with the certainty of both, are taken into account in the correction step. The two steps can be seen in Figure 2.14.

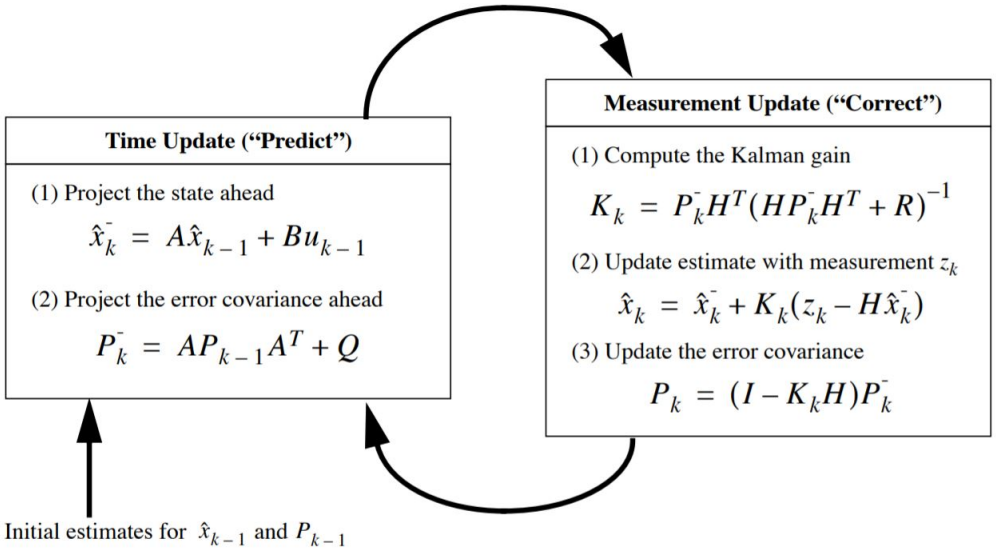


Figure 2.14: Equations used in the recursive two step process of the Kalman filter [54]

At any time step that a measurement is unavailable, such as when the object is occluded by another object, the filter's predictions alone can provide an estimation of the object state. This capability can be seen in Figure 2.15, where a rolling ball being tracked is occluded by a bin. While occluded, the computer vision algorithm cannot make any direct measurements, shown in green. The Kalman filter, using information from previous time steps, can continue to make predictions, shown in red, until the object is no longer occluded, and measurements are available again.

In tracking objects and estimating their motion in an image, the measurement is typically a two dimensional vector whose elements are x_c and y_c, representing the x and y pixel coordinates of the object, as determined by whichever computer vision algorithm is being used. It is common to model the dynamics as linear, and to take into account only position and velocity, even though the true dynamics of the system are likely to be different. In a two dimensional Kalman filter of this sort, the state will be a four element vector, containing both x_c and y_c, as well as the object's velocity in both directions, v_x and v_y. In a Kalman filter

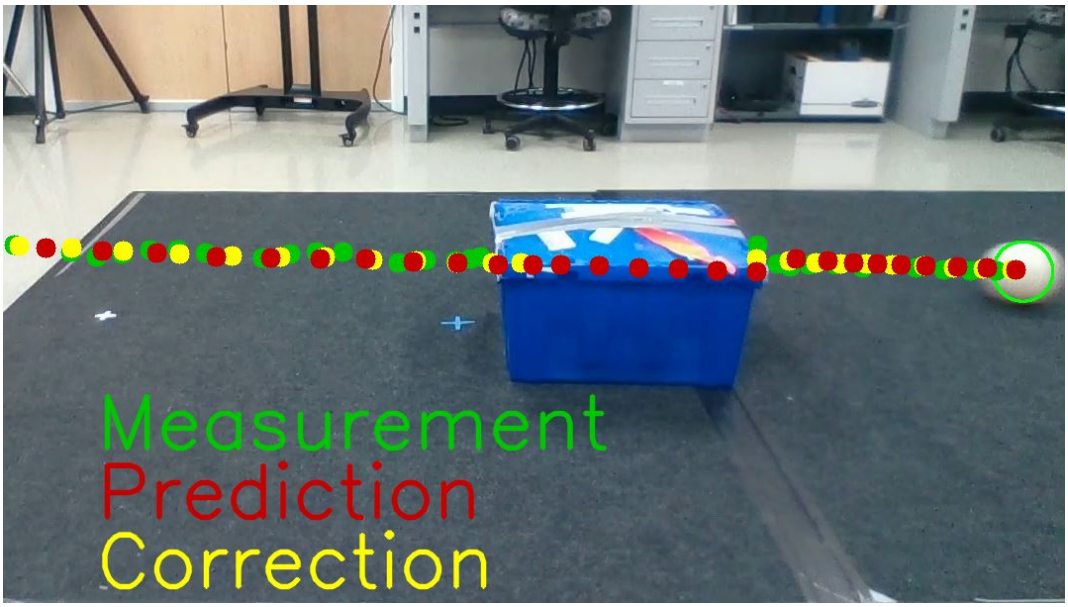


Figure 2.15: Predicting the position of an occluded object using a Kalman filter.

keeping track of acceleration, a_x and a_y are added to the state vector. Assuming a simplified linear dynamics model is obviously erroneous, but the benefits of the assumption outweigh the detriments in many cases. The assumption of linearity greatly reduces the complexity of the estimation problem. If this were not the case, the dynamics equations would have to be linearized around the state at each time step in order to solve for the state at the next time step. Assuming linearity eliminates this otherwise necessary calculation, improving computation time. With these assumptions, we can also assume that there are no control inputs, and therefore can ignore the $B * \bar{u}_{k-1}$ term in Equation 2.25. In a measurement-rich system such as a computer vision application where many measurements are taken per second, these assumptions often do not lead to significant error, as the Kalman filter is 'corrected' by the measurement frequently.

The vectors and matrices used in the two dimensional Kalman filter tracking velocity are shown here. The formulation found here is common throughout the computer vision literature [11], [20], [53], [46], [41].

$$\bar{x} = \begin{bmatrix} x_c \\ y_c \\ v_x \\ v_y \end{bmatrix} \quad F = \begin{bmatrix} 1 & 0 & t & 0 \\ 0 & 1 & 0 & t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad Q = \begin{bmatrix} \sigma_w^2 & 0 & 0 & 0 \\ 0 & \sigma_w^2 & 0 & 0 \\ 0 & 0 & \sigma_w^2 & 0 \\ 0 & 0 & 0 & \sigma_w^2 \end{bmatrix}$$

$$\bar{z} = \begin{bmatrix} x_c \\ y_c \end{bmatrix} \quad H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \quad R = \begin{bmatrix} \sigma_v^2 & 0 \\ 0 & \sigma_v^2 \end{bmatrix}$$

In the most basic implementation of the Kalman filter, these vectors and matrices are constant, with the exception of the state vector. The state transition matrix can be simplified further in computer vision applications if the time step used is one frame. The time t in the state transition matrix is then equal to one. Vectors and matrices of higher order Kalman filter formulations can be seen in Appendix A.

2.3 Relevant Work

Many surveys on computer vision object detection and tracking are available in the literature. Most of these surveys aim to categorize methods based on various properties, while discussing the pros and cons of each. Background modeling and subtraction methods are typically seen as fairly accurate while requiring very low computational resources compared to other methods [40], and, as such, are great candidates for real-time applications.

[58] states that "... most state-of-the-art tracking methods for fixed cameras ... use background subtraction methods..." due to the ability to deal with dynamic background scenes and the computational efficiency of the algorithms. [26], which looks specifically at computer vision detection methods in use with pan/tilt/zoom cameras, agrees that in a static

camera application, background subtraction algorithms perform very well, with MOGBM method being one of the best. Both surveys discuss the poor performance of background modeling and subtraction when used with a continuously moving camera, with [58] claiming that background subtraction methods can not work at all in these circumstances. In these type of applications, it seems best practice to work with a template matching algorithm that does not require any history of the previous frames.

Several of the surveys, including [43], note other limitations of background subtraction methods which occur when the object is very large in the frame and/or is slow moving. In this case, some of the object's pixels may remain the same color/intensity long enough to convince the algorithm that they are background. The result is that parts of the object of interest are identified as background. sUAS are typically small and fast-moving in video, and so this phenomenon is not considered a performance risk for the work in this thesis.

There are methods that undeniably perform better than mixture of Gaussians background modeling and subtraction in certain scenarios, but these algorithms require either a training period or some input from the user, and are better suited for situations where there is a very specific object to track. This work aims to detect and track generic sUAS, which may take many forms. As previously stated, many sources claim background subtraction methods are not suitable for use with a moving camera, such as one on a PTU. The work described here explores two solutions in Section 4.3.

Some researchers have developed ways to adapt background modeling and subtraction methods to situations with a moving camera, like on a PTU. [6] and [18] both attempt to apply background subtraction methods to a moving camera video by first aligning each new frame with the established background model, and then performing the subtraction. The frames are aligned based on the estimated camera motion as well as feature matching. The algorithms developed are computationally intensive, achieving real-time speed only in a GPU

implementation in [6], and only at 10 Hz on very low-resolution images in [18]. [42] notes that motion compensation is difficult and rarely accurate. This work proposes to use not just temporal, but spatial Gaussian distributions to model each pixel. This method allows only for small camera movements, and results in a lack of precision across the frame.

Several sources have used techniques similar to those above to develop and maintain a background model mosaic when the camera is moving. These works aim to maintain a background model of parts of the scene even after the camera has moved on to another view. [24] and [3] both keep track of the movement of the camera to help stitch together moving background scenes.

[16] attempts to augment pure tracking methods such as optical flow and mean shift with traditional background subtraction methods on a pan-tilt camera. The study suggests that the addition of background subtraction often lead to worse results than the tracking methods alone.

Most of the work combining computer vision and sUAS has revolved around cameras on the sUAS themselves. [10] uses CAMshift to detect and track simple objects moving in video taken by a flying sUAS. [55] investigated a leader-follower scenario with two quadcoptors. The leader is covered in several IR lights. The follower has a light-weight PTU holding an IR sensor, by which the follower can determine the relative position and pose of the leader. [34], [12], [37] all use computer vision on-board an sUAS, allowing it to gather information such as its own attitude and altitude.

Recently, more work has gone into detecting and tracking sUAS from ground-based sources. [15] recommends using various computer vision techniques as part of a system to detect and track unauthorized sUAS entering an airspace. They recommend both ground based and aerial cameras operating in different spectrums and with different fields of view.

[48], [49] is actually putting this advice into practice. It combines acoustic arrays and digital cameras to manage traffic of UAS, especially those that are non-compliant, and not offering GPS or ADS-B information. These sensors can work together with or take the place of RADAR and ADS-B. 6-camera hemispherical viewing sensors detect motion based on background modeling and subtraction, then task pan/tilt cameras to track the object using histogram of gradients matching techniques. These researchers note that RADAR is much less effective when the object has such a small cross-section, like an sUAS does. The goal of their project is fully autonomous UTM (unmanned traffic management). Results were compared to ADS-B and showed good agreement.

[11] developed the theory for estimating the movement of a point in an image taken from a camera on a PTU that would result from a combined pan and tilt operation. The theory can be used to improve continuity of tracking of an sUAS in flight when using optical flow and a camera that undergoes pan and tilt operations. The method was tested against data from an indoor motion capture system, and proved to be accurate. [27] explores the possibility of guiding an sUAS to a safe landing by sending it position information gained from the stereo-vision capabilities of two cameras on a single PTU. In [28] the same authors extend their work by using cameras on separate PTUs which allows for more separation between the cameras, and therefore greater depth perception. In both cases, infrared cameras and the Meanshift detection algorithm are used.

[44] use motion-stabilized series of images from videos to assist in matching a histogram-of-gradients based on the object. The method out-performs some optical flow and background subtraction methods, but requires training of both the motion stabilizing regressor and the object classifier. One of the videos used in their test set was taken for use in this work.

The work described in this thesis attempts to put all of this knowledge to practical use in the development and testing of a computer-vision enabled object tracking system. Several

well-understood techniques are implemented, verified, and extended in several instances. Additionally, some of the practical implementation issues, such as determining the correct angle to pan the PTU when the camera is offset from the pan axis, are discussed and solutions are developed.

Chapter 3

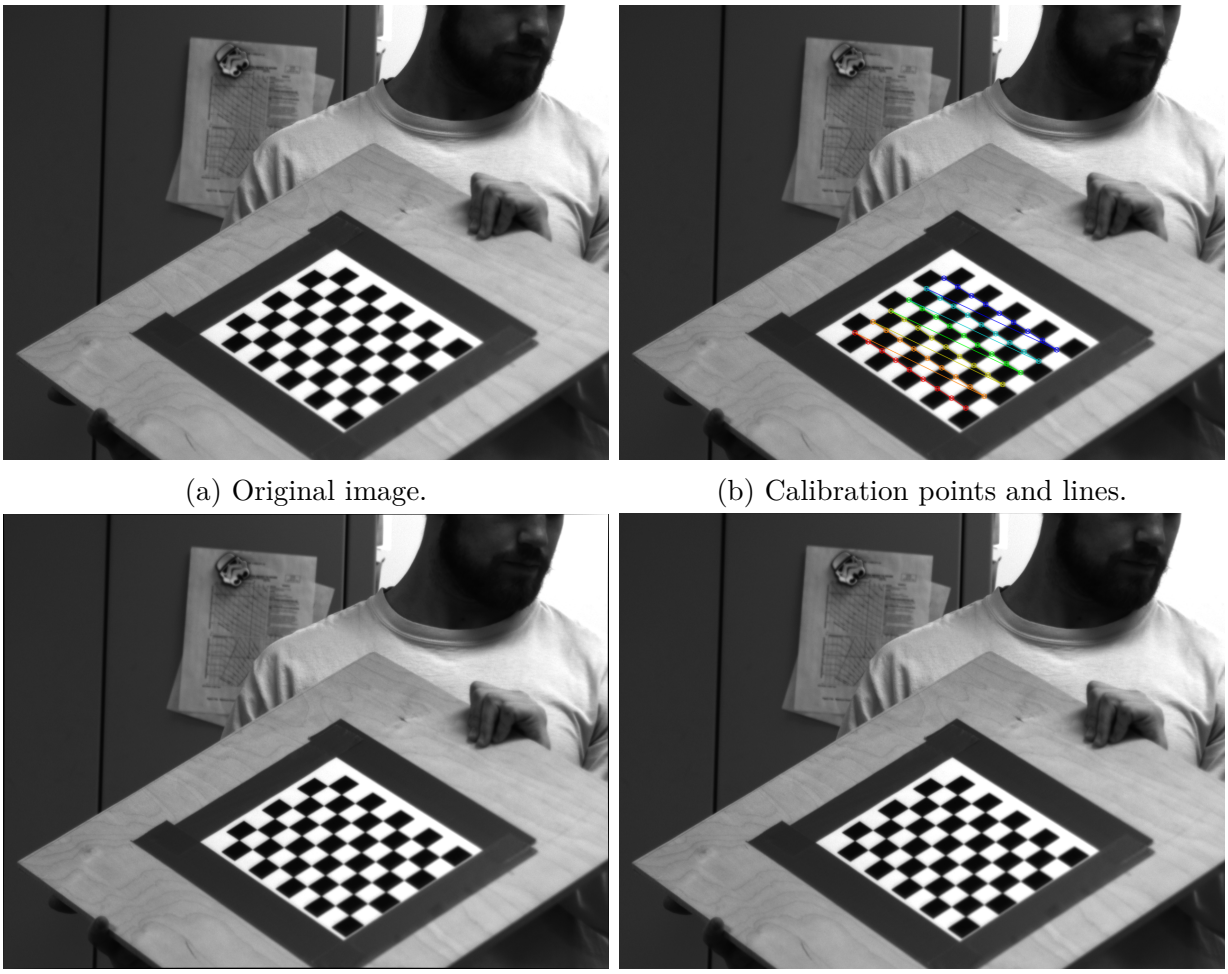
System Hardware

3.1 Cameras and Lenses

The two cameras used in this study are both Imaging Source DMK 41BU02.H USB 2.0 monochrome industrial cameras, referred to from here on out as Camera 1 and Camera 2. These cameras use a half-inch CCD sensor with a pixel array of 1280 (horizontal) by 960 (vertical) pixels. Each pixel is 4.65 micrometers square. With each of the cameras, a Nikkon AF-S DX NIKKOR 35mm f/1.8G lens is used, which has a documented focal length of 35 mm. The documented focal length in pixels, f_{pix} of this camera and lens combination can be calculated by dividing the focal length in meters by the length of each pixel as shown in Equation 3.1.

$$f_{pix} = \frac{35e^{-3}m}{4.65e^{-6}\frac{m}{pix}} = 7526.88pixels \tag{3.1}$$

If perfectly assembled, the optical center of the cameras would be be $x_c = 640, y_c = 480$. As discussed previously, however, the optical center and the focal length of the camera/lens system in rarely exactly as specified, but can be determined through the process of camera calibration. Both camera/lens combinations were calibrated using OpenCV's included calibration procedures. Images from this process are shown in Figure 3.1.



(a) Original image.

(b) Calibration points and lines.

(c) Remapped image with distortions corrected.

(d) Remapped image cropped.

Figure 3.1: Images from the camera calibration process. 3.1c shows the image with distortions corrected for. Not all pixels in the corrected image have a pixel mapped to them from the original image, hence the missing pixels around the edges.

Calibration was performed ten times for each camera/lens combination, using an average of 50 images each time. The resulting camera intrinsic matrix values were averaged, and are shown in Table 3.1. Optical center values varied greatly from calibration test to test, and were deemed untrustworthy. For this study, optical center values of $x_c = 640, y_c = 480$ are used for both cameras. Focal length values resulting from the calibrations were much more consistent, and were deemed trustworthy. Camera calibration provides two measures of focal

distance, f_x and f_y, since pixel size can vary in the x and y direction. In this case, all pixels are square, and therefore, $f_x = f_y$. For each camera, we average the provided f_x and f_y to attain a single f. Camera 1's focal length in pixels is then 7088.5 and camera 2's focal length in pixels is 7189.8.

Table 3.1: Averaged camera calibration results.

Parameter	Camera/Lens 1	Cameras/Lens 2
c_x	538.7	633.7
c_y	448.3	454.5
f_x	7089.4	7167.7
f_y	7087.6	7211.9

Radial distortions are present, but are very small. To correct for these distortions, the algorithm maps pixels from the original image into their new positions in the corrected image, essentially 'squeezing' the image. In Figure 3.1c, the pixels colored black, most visible around the upper right corner of the image, did not have values mapped to them from the original image. The mapping is not one-to-one due to the discretization of the image into pixels. The strips of the image left empty are only several pixels in width. Therefore, the radial distortions of this system are judged to be too small to have an appreciable effect, and so will be ignored for this work.

3.2 Pan Tilt Unit

The Pan/Tilt Unit (PTU) used in this work is a Directed Perception (now FLIR) PTU-D300. The specifications for this model can be seen in Table 3.2.

Table 3.2: Pan Tilt Unit PTU-D300 Specifications.

Property	Pan	Tilt
Maximum Speed	50°/s	50°/s
Position Resolution	0.00625°	0.00625°
Minimum Position	-168°	-90°
Maximum Position	168°	30°

The pan-tilt unit is commanded and queried via serial communication. Strings of ASCII characters are converted to their binary equivalent and sent to the pan-tilt unit through a USB COM port. The available commands and queries, as well as the correct syntax for communicating with the pan-tilt unit, can be found in the Command Reference Manual provided by FLIR [19]. For ease of use in this and future work, a Python function class was created that contains all of the relevant commands and queries. The function class can be imported like any other Python class into a custom script.

3.3 Computer

All work has been done on a Samsung personal laptop with an Intel i7-8550U CPU @ 1.80Ghz. All code has been developed in the Python programming language and relies heavily on the OpenCV library. This library is widely used in the computer vision community, as it contains many of the most-used algorithms and is well-optimized. It is important to note that algorithm frame rates discussed in this work are only meaningful in comparison to other frame rates in this work, as any use of a different computer will produce different results.

3.4 Machined Parts

Several parts for mounting the cameras to the PTU were designed and machined. These parts serve two functions. First, they raise the cameras up such that the optical center of the cameras are in line with the tilt axis of the PTU, therefore eliminating the error caused by the offset that would otherwise exist. Second, they separate the cameras further from each other than possible via direct mounting to the PTU, increasing depth perception capabilities with stereo vision. Mechanical drawings of the machined parts are shown in Appendix D.

3.5 Tracking System Setup Setup

A photo of the complete tracking system, including the PTU, the cameras/lenses, and laptop can be seen in Figure 3.2. Camera 1 is mounted on the left arm of the PTU, while Camera 2 is mounted on the right. Not pictured is the DC power supply that powers the PTU. The PTU is mounted to a thick wooden board, with handles attached for easier lifting and carrying. The full system can be transported on the cart shown in Figure 3.2, but cannot be operated on it. This is due to the lack of stiffness of the cart. When the PTU is commanded to move, the cart is not rigid enough to stay stationary, and continues to shake after the PTU is finished moving. During this shaking, the scene in the cameras' views are moving, rendering background modeling and subtraction methods unusable. For operation, the wooden board is removed from the cart, and placed on either a very stiff table or the ground. In such a configuration, the weight of the PTU and the wooden board are enough to keep the system in place during PTU movement.

Figure 3.2: Full tracking system setup showing laptop, PTU, cameras, lenses, and cart.

Ideally, the two cameras would take images simultaneously. This is impossible, however, without the use of parallel processing. A simpler method is to take an image with the second camera as quickly as possible after taking it with the first camera. In order to test the viability of this method, the cameras were tasked to take images, one after the other, as fast as possible. Time, as judged by the CPU clock, was recorded immediately before the first camera take an image, and immediately after the second camera takes an image. Both images are guaranteed to be taken within the two recorded times. Over 1939 pairs of images taken, the average difference between these two recorded times was 2.564 milliseconds, with a standard deviation of 0.677 milliseconds. For this work, this time was considered small enough to ignore, and to assume that the cameras take images simultaneously.

3.6 sUAS Truth Data

The sUAS used to generate truth data and validate the tracking system is custom built and based on the DJI Flame Wheel F550 model frame. The sUAS is equipped with a 3DR uBlox GPS kit that provides geodetic latitude, longitude, and altitude. This GPS location information can be continuously transmitted over a wireless network, and read into a Python script on demand.

Given two sets of geodetic LLA's, relative Cartesian distances in a local East-North-Up (ENU) frame, centered at the one of the two sets of LLA's, which will be called the origin, can be derived. First, both sets of LLA's are converted to Earth-Centered Earth-Fixed (ECEF) position vectors. The conversions from geodetic LLA to ECEF are shown in Equation 3.2, [4], where R_E is the equatorial radius of the Earth, and e_E is a measure of the eccentricity of the Earth. $R_E = 6,378,137$ meters and $e_E = 0.08181979099211309$. The ECEF vector pointing from the origin to the other location, the target location, can then be calculated as the difference between the ECEF position vectors of the target and the origin, as seen in Equation 3.4.

$$\bar{r}_{ECEF} = \begin{bmatrix} X_{ECEF} \\ Y_{ECEF} \\ Z_{ECEF} \end{bmatrix} = \begin{bmatrix} (N_e + h_{alt})\cos(\phi_{lat})\cos(\lambda_{long}) \\ (N_e + h_{alt})\cos(\phi_{lat})\sin(\lambda_{long}) \\ ((1 - e_E^2)N_E + h_{alt})\sin(\phi_{lat}) \end{bmatrix} \tag{3.2}$$

$$N_E = \frac{R_e}{\sqrt{1 - (e_E^2)\sin(\phi_{lat})^2)}} \tag{3.3}$$

$$\bar{r}_{ECEF}^{Pointing} = \bar{r}_{ECEF}^{Target} - \bar{r}_{ECEF}^{Origin} \tag{3.4}$$

What is left is to express this pointing vector, from the origin to the target, in the local ENU frame based on the position of the origin. This is done via multiplication with a rotation matrix as shown in Equation 3.5. The ϕ_{lat} and λ_{long} used are those of the origin's location.

$$\bar{r}_{ENU} = \begin{bmatrix} -\sin(\phi_{long}) & \cos(\lambda_{long}) & 0 \\ -\sin(\phi_{lat})\cos(\lambda_{long}) & -\sin(\phi_{lat})\sin(\lambda_{long}) & \cos(\phi_{lat}) \\ \cos(\phi_{lat})\cos(\lambda_{long}) & \cos(\phi_{lat})\sin(\lambda_{long}) & \sin(\phi_{lat}) \end{bmatrix} \bar{r}_{ECEF} \tag{3.5}$$

GPS precision of the sUAS's unit was tested by placing the sUAS in three measured locations and reading the transmitted location data for some interval of time. The first location was considered as the origin point for this experiment. The sUAS was placed on the ground here, and data was recorded for 50 seconds at 20 Hz, for a total of 1000 measurement points. The sUAS was then moved to the second location, which was a point measured to be 10 meters North of the first location. Data was again recorded for 50 seconds at 20 Hz. The third location was 10 meters West of the second location, at which the same number of data points was recorded at the same sampling rate. This same process of sampling at three locations was repeated a second time. The process was repeated a third time, but with a change in the sampling rate. During the third round of testing, 1000 data points were recorded at 4 Hz over the course of 250 seconds.

The average latitude, longitude, and altitude of the data sampled during the first round of testing at the first location was used as the origin of the ENU coordinate frame into which the rest of the recorded LLA data was transformed. Figures 3.3 and 3.4 show the sampled data.

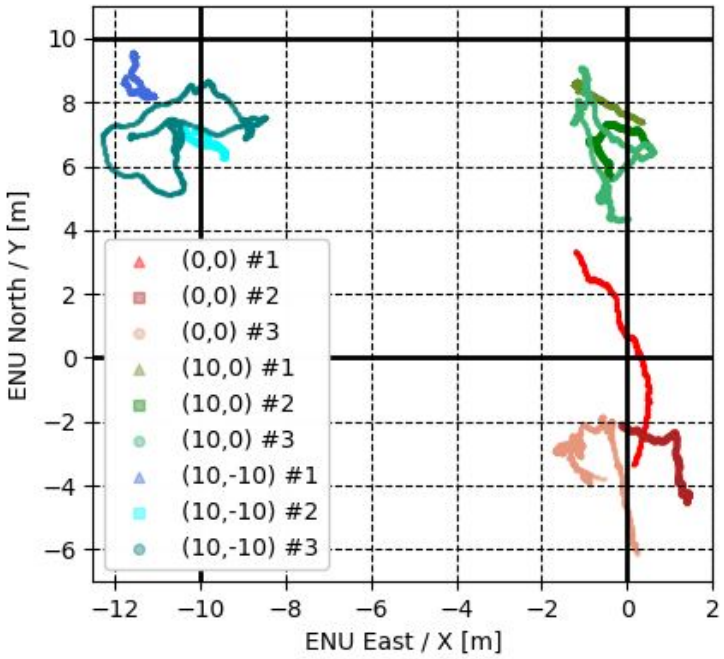


Figure 3.3: X and Y data from testing of GPS unit on sUAS.

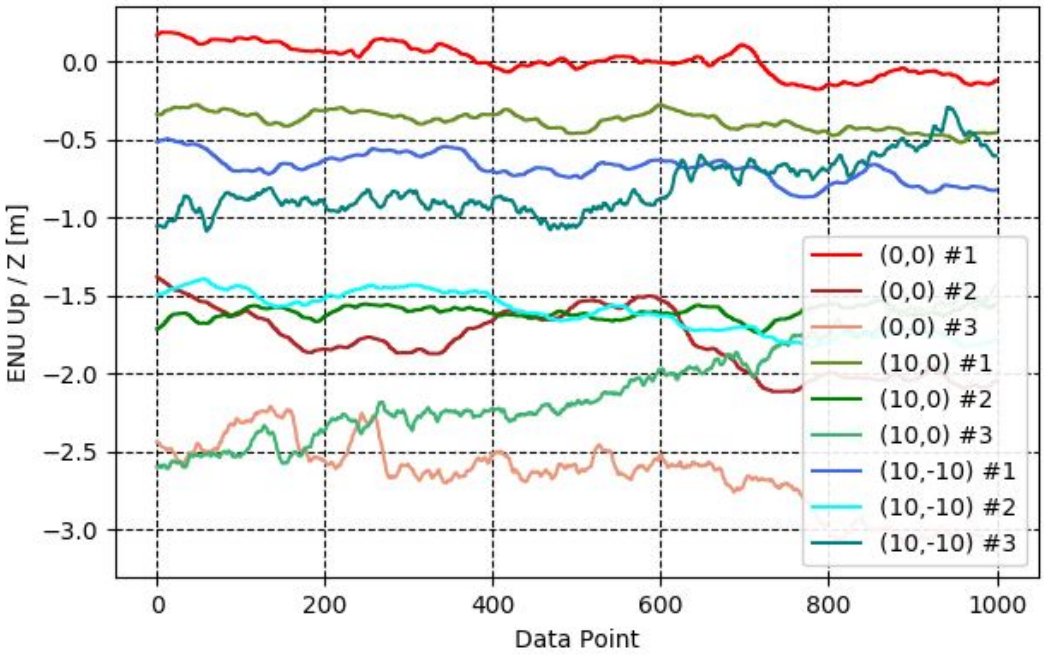


Figure 3.4: Z data from testing of GPS unit on sUAS.

The sampled data was analyzed, and mean errors, along with standard deviations, were calculated, and are shown in Table 3.3. The data was analyzed separately for each location and test, then with all tests combined, and finally as a whole. The errors were calculated as the difference between the ENU location the sUAS was placed at and the location as given by the GPS unit. To calculate the standard deviations for all tests at all locations combined, the data sets were first normalized around the 'true' locations. As such, they are effectively the standard deviations of the errors.

Table 3.3: GPS precision test results.

Location	Test	Mean Error [m]			Standard Deviation [m]		
		X	Y	Z	X	Y	Z
$(0, 0)$	1	0.395	1.617	0.085	0.483	1.934	0.101
$(0, 0)$	2	0.993	3.285	1.795	0.450	0.829	0.201
$(0, 0)$	3	0.937	2.945	2.657	0.471	0.850	0.230
$(0, 10)$	1	0.729	1.830	0.381	0.491	0.394	0.057
$(0, 10)$	2	0.489	3.223	1.608	0.387	0.468	0.040
$(0, 10)$	3	0.633	3.448	2.090	0.492	1.390	0.317
$(-10, 10)$	1	1.492	1.246	0.684	0.187	0.462	0.089
$(-10, 10)$	2	0.270	3.192	1.612	0.304	0.276	0.130
$(-10, 10)$	3	0.862	2.875	0.803	0.979	0.846	0.172
$(0, 0)$	All	0.782	2.576	1.511	0.912	2.008	1.126
$(0, 10)$	All	0.617	2.834	1.360	0.476	1.132	0.744
$(-10, 10)$	All	0.875	2.438	1.033	0.915	1.031	0.434
All	All	0.758	2.616	1.301	0.795	1.456	0.818

Chapter 4

Tracking Algorithm

4.1 Tracking in a Static Frame

4.1.1 Static Camera Test Video Data Set

For the purpose of initial vision-algorithm testing and parameter tuning, a set of five test videos was assembled. Four of these videos were custom-created for this study, while the fifth was taken from the test set of videos used in [44]. These five videos were created and selected for their applicability to this study. Four of the five videos contain a copter-style sUAS as the object of interest, while the fifth contains a bouncing ball. The bouncing ball video was included to ensure the algorithm developed did not lose generality. This work intended to develop algorithms capable of tracking any moving object, although the envisioned application is for sUAS. When tracking sUAS, it will be important to identify any other flying objects, such as birds, that may enter the environment. The videos included in the test set are of varying pixel resolution and length in frames.

Table 4.1 below describes each of the videos in the test set. Some of the test videos include moving objects other than the object of interest. Their movements are much less in magnitude than that of the intended objects', but provide an interesting challenge nonetheless. The videos comprising the test set aim to cover the range of scenarios and conditions that the tracking system is expected to work in.

Table 4.1: Test video data set for 2D tracker testing.

Video	Object	Resolution	Frames	Description
1	Hexacoptor	1280x720	303	Object moves side to side, towards and away from camera
2	Quadcoptor	640x480	303	Object moves erratically, leaving and re-entering frame
3	Quadcoptor	1280x960	175	Object begins above skyline, dips below it, then rises above it
4	Quadcoptor	1280x960	250	Object moves side to side, is very small in frame
5	Ball	1280x720	290	Object bounces, then rolls

Code was developed that allowed for the selecting of the object location frame-by-frame in each video. At each frame, the pixel coordinates of the center of mass of the object, as perceived by the user, was manually selected if it was present in the frame. These frame-by-frame pixel coordinates, along with a binary flag indicating the object's presence or absence, make up the ground truth for each test video. Accuracy of subsequently developed tracking algorithms are compared to these ground-truth sets and scored in two ways.

The first measure of accuracy for the developed tracking algorithms was a simple root mean square error (RMSE) in pixel location. At each frame where the algorithm determines the location of the object and ground-truth object location exists, the error can be determined with a simple as shown in Equation 4.1.

$$RMSE = \sqrt{(x_t - x)^2 + (y_t - y)^2} \tag{4.1}$$

Here, x_t and y_t are the ground-truth x and y pixel coordinates of the object, and x and y are the x and y pixel coordinates the algorithm estimates the object to be at. Each algorithm is given a final RMSE value by averaging the errors for each frame in a given video over that entire video, and then averaging those results for each video.

The second measure of accuracy for each algorithm is the false negative (FN) and false positive (FP) count. A false negative occurs when the algorithm fails to find an object of interest in an image when the ground-truth indicates one exists. A false positive is the opposite, and occurs when the algorithm determines the object exists in an frame, when it in fact does not. It was expected that false positives would occur with some regularity when testing, due to the moving objects that were not the intended object in some of the videos, as mentioned before.

An example of measured test data points compared to the associated truth data is shown in Figure 4.1.

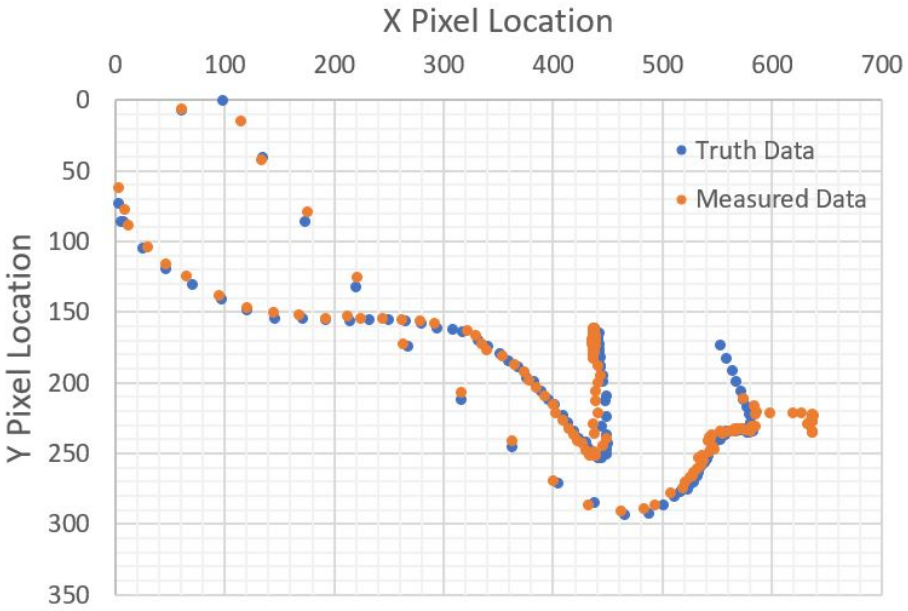


Figure 4.1: Example X and Y pixel location data of object in video, truth vs. measured.

4.1.2 Initial Algorithm Selection/Verification

Although many sources indicate that the algorithm described in [60] is the best that openCV has to offer with regards to background subtraction methods, this needed to be verified. Five separate background modeling and subtraction algorithms implemented in openCV were compared. These five algorithms are named in OpenCV as KNN, CNT, GMG, MOG, and MOG2. They are each briefly described here. It should be noted that all five of the algorithms tested simply classify pixels as foreground or background, and that the location of an object is determined as the centroid of the foreground pixels remaining after post-processing.

- KNN (K-Nearest Neighbors): Algorithm described in [61], similar to [60], but with K-Nearest Neighbors method used for kernel density estimation.
- CNT (CouNT): A method that relies on the 'stability' of a pixel in order to distinguish it as background or foreground, created seemingly as a personal project, and available in [36].
- GMG: (Godbehere, Matsukawa, Goldberg) Another take on the mixture of Gaussians background modeling using a bank of Kalman filters and matching algorithms to distinguish separate objects, developed in [13].
- MOG (Mixtures of Gaussians): The improved algorithm of [50] as suggested by [22], where the update equations for the background model are reformulated for improved learning.
- MOG2 (Mixtures of Gaussians 2): Algorithm described in [60], where the algorithm decides for itself how many components to include in each Gaussian mixture model, and considered the best of what OpenCV has to offer for background subtraction.

These algorithms were tested and compared for accuracy using the performance metrics described above. In addition, the speed of the algorithms were recorded in frames-per-second. Each of the five algorithms have between two and three settable parameters that define the behavior of the algorithm. These were varied to ensure a fair comparison between the algorithms. Additionally, the algorithms were tested with three separate combinations of pre and post processing parameters.

Out of all of the runs of a particular algorithm, the five runs with the lowest RMSE were selected. Second, the five runs with the lowest sum of FP and FN were selected. In the event that there was a tie between runs for either of these criteria, the run with the highest FPS was selected. The results of these selected runs were averaged, and the averaged MSE, FP, FN, and FPS are displayed in Table 4.2.

Table 4.2: Comparison of OpenCV background subtraction methods. Methods are ranked by their RMSE and their FP+FN count. For both metrics, lower is better.

Lowest RMSE					Lowest FP+FN				
Algorithm	RMSE	FP	FN	FPS	Algorithm	RMSE	FP	FN	FPS
MOG	6.79	0.0	267.4	25.22	KNN	17.00	24.8	0.0	14.66
MOG2	9.21	18.2	59.2	17.98	MOG2	10.43	22.0	4.0	17.18
KNN	12.17	3.0	234.8	14.36	CNT	33.94	31.4	16.0	11.31
GMG	17.26	7.0	462.4	11.83	GMG	88.22	26.6	82.4	9.48
CNT	25.44	9.0	249.6	17.31	MOG	23.90	0.0	83.4	14.71

In the lowest RMSE comparison, shown in the left half of the table, only MOG performed better than MOG2, but with a very large FN count. This would indicate that the algorithm is being overly 'cautious' in its determination of the presence of an object of interest. In the

lowest FP+FN count comparison, KNN slightly outperformed MOG2, but at the expense of RMSE. In the lowest FP+FN comparison, maintained a low RMSE, comparable to its score in the lowest RMSE comparison. All other algorithms suffered a large increase in RMSE to achieve their lowest FP+FN scores. These results confirmed the superior performance of MOG2 over the other background subtraction algorithms in OpenCV.

4.1.3 MOG2 Optimization

The MOG2 algorithm in OpenCV has three parameters that can be set to change the behavior. These parameters are described here.

- Shadow Detection: Determines whether or not the background subtractor will include an objects shadow in its output. If set to 'True', the subtractor will output shadow pixels as grey. This study is not interested in shadow detection, and so this parameter will be set to 'False.'
- Learning Rate: Determines how fast the background model is updated when given new information. The default value is -1, which allows the algorithm to automatically select the learning rate .
- Variance Threshold: Sets the threshold distance a new pixel intensity must be away from an existing Gaussian distribution component in order to be considered part of a new distribution. Larger threshold values allows for more variation in lighting without identifying objects.

In the overall vision algorithm developed that includes the MOG2 background subtractor, there are five parameters that define pre- and post-processing of the incoming frames. These parameters are listed and described here.

- Gaussian Blur Kernel Size: The size of the matrix kernel that will be used to blur the new image before it is passed to the background subtractor.

- Morphological Transform Kernel Size: The size of the matrix kernel that will be used to perform morphological operations on the output from the background subtractor.

- Open Iterations: How many iterations of the 'Open' morphological operation that should be performed.

- Close Iterations: How many iterations of the 'Close' morphological operation that should be performed.

- Standard Deviation Exclusion Bound: Defines the number of standard deviations away from the centroid of the blob remaining that pixels beyond will be removed.

The effects of each of these pre- and post-processing steps, in addition to the actual background subtraction step, can be visualized in Figure 4.2. Each of these pre- and post-processing steps, with the exception of the 'standard deviation filter,' is commonplace in the computer vision object detection field. The unique order of the steps, however, was developed specifically for this work. For a first round of testing, three levels to test were chosen for each of the parameters varied, centered around the default value when possible. The values chosen for testing are shown in Table 4.3.

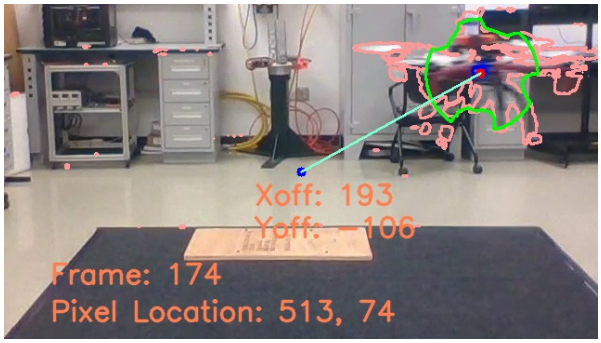
(a) Original frame

(b) After Gaussian blurring

(c) Background subtractor output

(d) After morphological operations

(e) Standard deviation filter



(f) Results on original frame

Figure 4.2: Visualization of the the 2D tracker algorithm steps. The determined pixel location of the target is the centroid of the white pixels in 4.2e.

Table 4.3: Parameters and selected levels for first round of MOG2 testing.

Parameter	Level 1	Level 2	Level 3
Learning Rate	-1	.01	.1
Variance Threshold	8	16	24
Gaussian Kernel Size	5x5	7x7	9x9
Morphological Kernel Size	3x3	5x5	7x7
Open Iterations	1	2	3
Close Iterations	1	2	3
Standard Deviation Threshold	1	2	None

A full factorial test with a total of 2,187 runs was completed. Here, full factorial means that every combination of parameter levels is tested. Results for MSE, FP, and FN varied widely based on the combination of parameter levels. The results were analyzed in Minitab software, which provides a useful suite of statistical analysis and visualization tools [35]. Main effects plots resulting from the analysis done in Minitab are shown in Figure 4.3. A main effects plot shows the effect of a parameter on a result, averaged across all levels of other parameters.

As an indicator of which scoring metrics the optimization should focus on improving, the minimum and maximum scoring metrics across all test runs was found, and can be seen in Table 4.4. The maximum false positives count was 32, which can be considered low, with not much improvement possible. The maximum RMSE and FN, however, at 131.29 and 1093, respectively, were much higher. Compared to their respective minimum values of 7.95 and 0, it is clear these can be improved. Therefore, it was determined that the main results which needed to be optimized for were MSE and FN. Looking at the main effects plots in

Figure 4.3, it can be seen that these two results correlate similarly to the selected parameter values for almost all of the parameters. It seems that the parameters with the largest effects on MSE and FN are the Learning Rate, the Morphological Operations Kernel Size, and the number of Open Iterations.

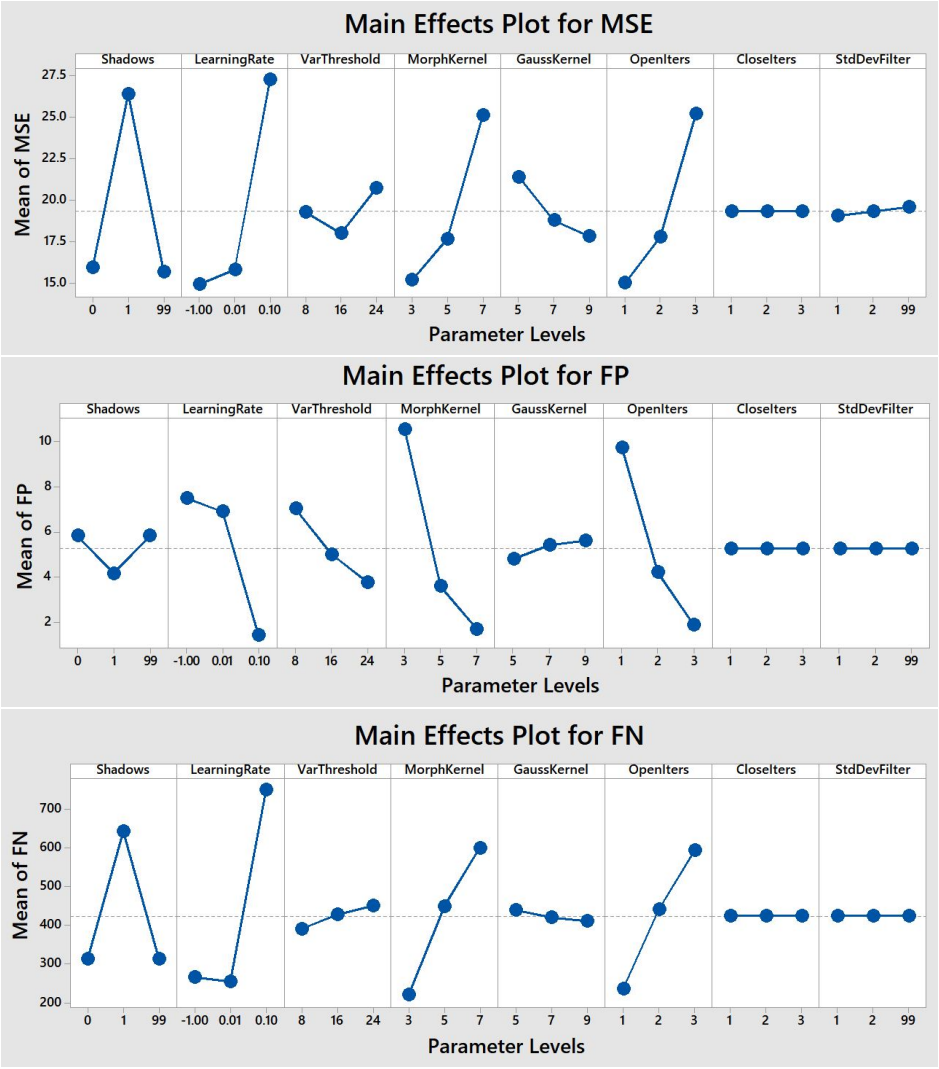


Figure 4.3: Main effects analysis plots from Minitab for 2D tracker parameters. A main effects plot shows the effect of a parameter on a result, averaged across all levels of other parameters.

Table 4.4: Minimum and maximum RMSE, FP, and FN results from first round of MOG2 testing.

	RMSE	FP	FN
Min	7.95	1	0
Max	131.29	32	1093

Another full factorial study was performed, but with fewer parameters varied, based on the results of the first study. From the first study, it was clear that the learning rate should be set to -1, the morphological kernel size should be set to 3x3, and the Gaussian blurring kernel size should be set to 9x9. The other parameters did not have clear best performing values. In the second study, the other parameters were varied as seen in Table 4.5. The final selected parameter levels are shown in Table 4.6.

Table 4.5: Parameters and selected levels for second round of MOG2 testing.

Parameter	Level 1	Level 2	Level 3
Learning Rate	-1	N/A	N/A
Variance Threshold	12	16	18
Gaussian Kernel Size	9x9	N/A	N/A
Morphological Kernel Size	3x3	N/A	N/A
Open Iterations	1	2	N/A
Close Iterations	1	2	3
Standard Deviation Threshold	.5	1	1.5

Table 4.6: Optimized parameter levels for MOG2.

Parameter	Optimum Values
Learning Rate	-1 (Automatic)
Variance Threshold	16
Gaussian Blur Kernel Size	9x9
Morphological Kernel Size	3x3
Open Operations	1
Close Operations	3
Standard Deviation Filter	1

These finalized parameter levels result in a RMSE of 8.32 pixels, a FP count of 19, and a FN count of 4. With the total number of frames being 1,321, these FP and FN counts equate to a .01438 and .00302 FP and FN ratio, respectively. It should be noted that it may be possible to implement on-line adaptation of parameter levels that could result in better performance across a range of tracking scenarios. Distance to target, environmental conditions, and other variables could influence the optimum parameter levels both for pre- and post-processing, and for the MOG2 background subtractor itself. This work aimed to have one solution for all tracking scenarios, and, as such, the parameter levels determined above are kept constant throughout the rest of the research.

4.1.4 2D Kalman Filter Implementation

No matter the amount of parameter tuning and image processing, there will always be some inaccuracy in the tracking of objects using computer vision, as proved by the non-zero RMSE of even the best performing combination of parameters. The Kalman filter, as described in

Section 2.2.4, can help correct for some of the inaccuracy inherent in the system.

It was noticed during initial testing that across almost all combinations of σ_w^2 and σ_v^2 tested, the acceleration model Kalman filter performed better, if only slightly, than the variant considering only velocity, in terms of RMSE. It seemed worth investigating how higher order Kalman filters would perform. If accounting for acceleration, the time derivative of velocity, provided a better assumption for the model, maybe higher order derivative would be superior. Formulations of the Kalman filter considering jerk, the time derivative of acceleration, and snap, the time derivative of jerk, were developed, implemented, and tested. The components of these formulations, along with those considering acceleration, are shown in Appendix A. These Kalman filters were implemented alongside the MOG2 algorithm that was being tested in the previous section. Every frame, the Kalman filter predicts where it expects the center of the object to be, based on the state of the last frame. If an object is detected by the vision algorithm, the measurement is passed to the Kalman filter, and the correction step takes place. The output of the correction step, which is a state vector, takes into account both the Kalman filter's prediction and the vision algorithm's measurement, and hopefully provides a more accurate answer than either of the two alone.

So, in testing, the x and y pixel positions of the object provided by the correction step, or the prediction step if there was no measurement, is compared to the ground truth data of the test video set. In using the Kalman filter for this work, there were only two parameters for the user to vary. These are σ_w^2 and σ_v^2, the values that comprise the diagonal elements of the process and measurement noise covariance matrices, respectively. Values of .01, .1, 1, and 10 for both σ_w^2 and σ_v^2 were tested, for a total of 16 runs.

Table 4.7: Initial Kalman Filter Results: RMSE [pixels] for velocity, acceleration, jerk, and snap formulations, resulting from different combinations of σ_w^2 and σ_v^2.

σ_w^2	σ_v^2	Vel	Acc	Jerk	Snap
0.01	0.01	7.72	7.61	8.43	7.71
0.01	0.10	8.65	8.20	8.07	7.85
0.01	1.00	11.67	9.27	11.13	10.19
0.01	10.00	20.05	11.90	11.76	9.37
0.10	0.01	7.46	7.43	7.47	7.34
0.10	0.10	7.72	7.61	7.66	7.43
0.10	1.00	8.65	8.20	8.02	7.68
0.10	10.00	11.67	9.27	13.28	11.04
1.00	0.01	7.46	7.44	7.52	7.40
1.00	0.10	7.46	7.43	8.15	7.55
1.00	1.00	7.72	7.61	7.66	7.42
1.00	10.00	8.65	8.20	11.41	7.72
10.00	0.01	7.46	7.43	7.37	7.26
10.00	0.10	7.46	7.44	7.46	7.31
10.00	1.00	7.46	7.43	7.51	7.72
10.00	10.00	7.72	7.61	8.61	8.31

These trials showed that no combination of σ_w^2 and σ_v^2, with any of the Kalman filter types, provided a massive increase in accuracy, as measured by RMSE, over that of the MOG2 background subtraction algorithm alone. The Kalman filter is not being used to simply increase the accuracy of the current frame position estimation, however. When the object has moved some distance away from the center of the image, the PTU will be commanded

to move. The PTU will take some amount of time to move, and during that time, the object may still be moving. The goal, therefore, is to command the PTU to aim at where the object will be when the PTU stops moving, not when it starts. In order to do this, we predict the object's position several frames in the future. This is the more important function of the two dimensional Kalman filter in this work. Additionally, the Kalman filter's predictions are helpful if the object is ever occluded and measurements are not available.

Code was developed to test the predictive capabilities of the Kalman filters. The code is similar to the initial Kalman filter testing code, in that it goes through each test video frame by frame, applying the MOG2 algorithm to take measurements, and completing the prediction and correction steps of the Kalman filter. Now, however, at each frame i, the Kalman filter is asked to predict the object's position for the next five frames, $i+1$ to $i+5$, without any measurement input. These predictions are the Kalman filter's estimation of the location of the object in each of the next five frames, $(x_p^i(1), y_p^i(1))$ through $(x_p^i(5), y_p^i(5))$. These five predicted locations can then be compared to the truth data for the next five frames, $(x_t^{i+1}(0), y_t^{i+1}(0))$ through $(x_t^{i+5}(0), y_t^{i+5}(0))$, and a RMSE can be calculated for each, as shown in Equation 4.2. In addition, the truth data locations for the five subsequent frames will be compared to the corrected measurement from the frame the Kalman filter will start predicting from, $(x_c^i(0), y_c^i(0))$. The prediction process can be visualized in Figure 4.4

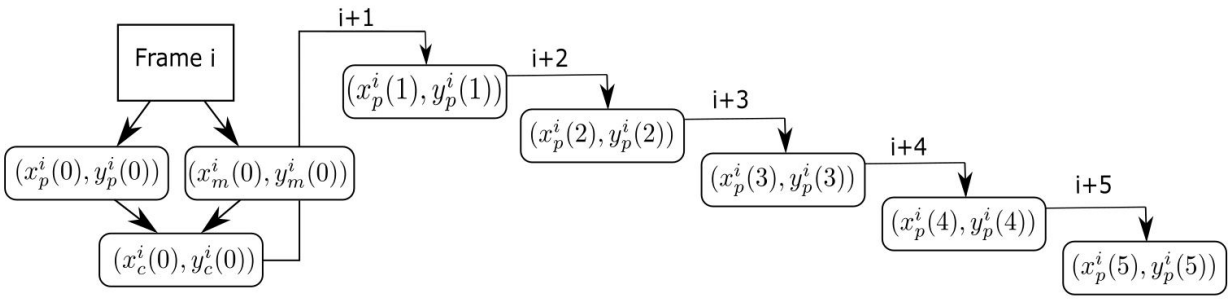


Figure 4.4: Visualization of Kalman filter prediction generation process in the absence of measurements.

$$for \qquad n = 1:5$$

$$RMSE_p^i(n) = \sqrt{(x_t^{i+n}(0) - x_p^i(n))^2 + (y_t^{i+n}(0) - y_p^i(n))^2} \tag{4.2}$$

$$RMSE_c^i(n) = \sqrt{(x_t^{i+n}(0) - x_c^i(0))^2 + (y_t^{i+n}(0) - y_c^i(0))^2}$$

RMSE's for a given Kalman filter configuration across all videos will be averaged for each of the number of predicted steps. Therefore, each Kalman filter configuration tested will result in eleven total averaged RMSE scores. An example results tables from one round of testing can be seen in Table 4.8

Table 4.8: Kalman filter prediction testing with $\sigma_w^2 = \sigma_v^2 = 1$, RMSE results. The 'Pred.' column shows the RMSE of the predicted positions up to five frames in the future. The 'No Pred.' shows the RMSE that occurs when using the corrected position from a frame as the predictions for the next five frames.

	Velocity		Accel.		Jerk		Snap	
Frame	Pred.	No Pred.	Pred.	No Pred.	Pred.	No Pred.	Pred.	No Pred.
i=0	N/A	7.72	N/A	7.61	N/A	7.57	N/A	7.42
i=1	10.87	12.55	13.32	12.38	18.36	12.31	23.79	12.24
i=2	15.09	18.75	22.68	18.55	38.78	18.46	59.96	18.48
i=3	19.49	25.17	34.91	24.97	69.86	24.89	126.54	24.94
i=4	23.79	31.41	49.27	31.21	113.27	31.12	233.81	31.19
i=5	28.51	37.39	66.35	37.20	173.05	37.12	394.08	37.26

We can see from this table that when using the velocity Kalman filter, it is more accurate to use the predited position of the object, all the way through five frames ahead, then to use the corrected position of the filter from the initial frame. The same is not true for any

of the other Kalman filter formulations, however. Using the acceleration Kalman filter, or one of higher order, the predicted positions, even for one frame in the future, have a greater RMSE then the corrected position from the initial, zero-th frame. In these cases, it is more accurate to use the corrected position from the initial frame as our predicted position of the object in the next five frames then it is to use the Kalman filter's predicted positions. Other combinations of σ_w^2 and σ_v^2 values give similar results. It is thought that the errors associated with incorrectly estimating any of the time derivatives are propagated through and increased with every prediction. The higher order Kalman filter formulations are causing the errors to increase at exponentially greater rates.

4.2 Commanding the PTU

4.2.1 Prediction Steps Determination

The number of steps to predict ahead can be determined using the bounding box size, the current frame rate of the algorithm, and the time to slew calculations discussed in Section3.2. The bounding box is the region of the camera's view in which we want to keep the sUAS. A typical bounding box size will be half of the size of the camera's view. With our cameras having a 1280x960 pixel view, the bounding box will be 640x480 pixels. With this bounding box centered in the frame, the furthest the PTU will have to pan or tilt will be to half of the bounding box's width, which is 320 pixels. Using the equations from Section 2.2.1, and the focal lengths determined using camera calibration, it can be determined that this would require a panning operation of 2.58 degrees.

With the PTU's base speed and operational/object speed both set to the maximum of 50 deg/sec, this slew operation will take .0517 seconds. Assuming a frame rate of 15 fps, this

slew time translates to .775 frames, less than one frame. This result suggests that when the object is leaving the bounding box in frame, we should have to predict no more than one frame ahead when deciding where to command the PTU.

4.2.2 Pan Angle Error Due to Camera Offset and Correction

Using the pixel coordinates of the object in the image, we are able to determine the angular position of the object relative to the optical axis of the camera. However, unless the optical center of the camera is aligned with the pan and tilt axes of the PTU, we can not use these angles directly to command the PTU. The machined parts discussed earlier align the camera's optical center with the tilt axis, but the cameras are placed .325247 meters offset from the pan axis. The geometry of this setup, and the angular error η that would be induced from directly using the angles to command the PTU, can be visualized in Figure 4.5. All PTU movements in this work are based on measurements taken from camera 1, positioned to the left of the center of the PTU.

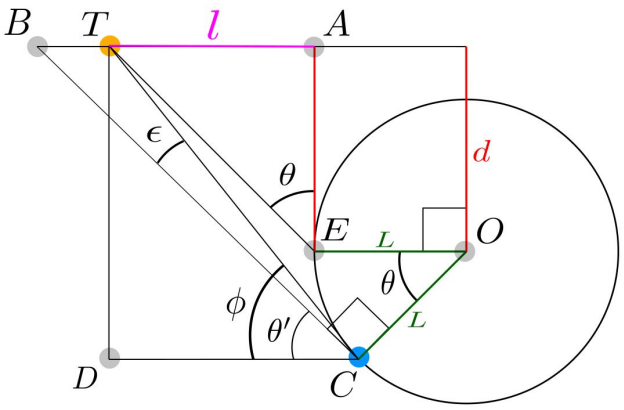


Figure 4.5: Geometry of error due to camera's offset from PTU's pan axis.

The point T is the object we wish to aim at, point O is the pan axis of the PTU, and point C is the location of the camera after panning by angle θ, which is the angle that the

object appears to be from the initial camera position, point E. θ' is the co-angle of θ. The resulting pointing error depends both on the distance to the object and the total pan angle. The angular error, ϵ, and horizontal pixel coordinate error, e_{pix} can be predicted using the following equations.

$$TA = l = d * \tan|\theta| \qquad DC = l + (L - L\cos|\theta|) \qquad TD = d + L\sin|\theta|$$

$$\phi = \arctan(\frac{TD}{DC}) \qquad \theta' = \frac{\pi}{2} - \theta \qquad \epsilon = |\phi - \theta'|$$

$$e_{pix} = f_{pix} * \tan\epsilon \tag{4.3}$$

The equations above are valid when the object is 'to the left' in the cameras view. The equations change slightly when the object is 'to the right', as the camera will move closer to the object in both axes. The two equations that change are shown here.

$$DC = l - (L - L\cos|\theta|) \qquad TD = d - L\sin|\theta|$$

The error that would occur for various distances to object and pan angles are calculated using the above equations and shown in Figure 4.6. In this figure, negative pan angles indicate the PTU panning 'to the left,' or counter-clockwise. After either counter-clockwise or clockwise panning, the camera pointing error will be 'to the left' of the target. These errors were considered too small to be of concern.

For a larger offset, larger pan angle, or closer object of interest, these errors could increase to a significant level. The correct pan angle can be solved for via the geometry of the scenario, as viewed in Figure 4.7, where L is the camera offset from the pan axis of the PTU. Therefore,

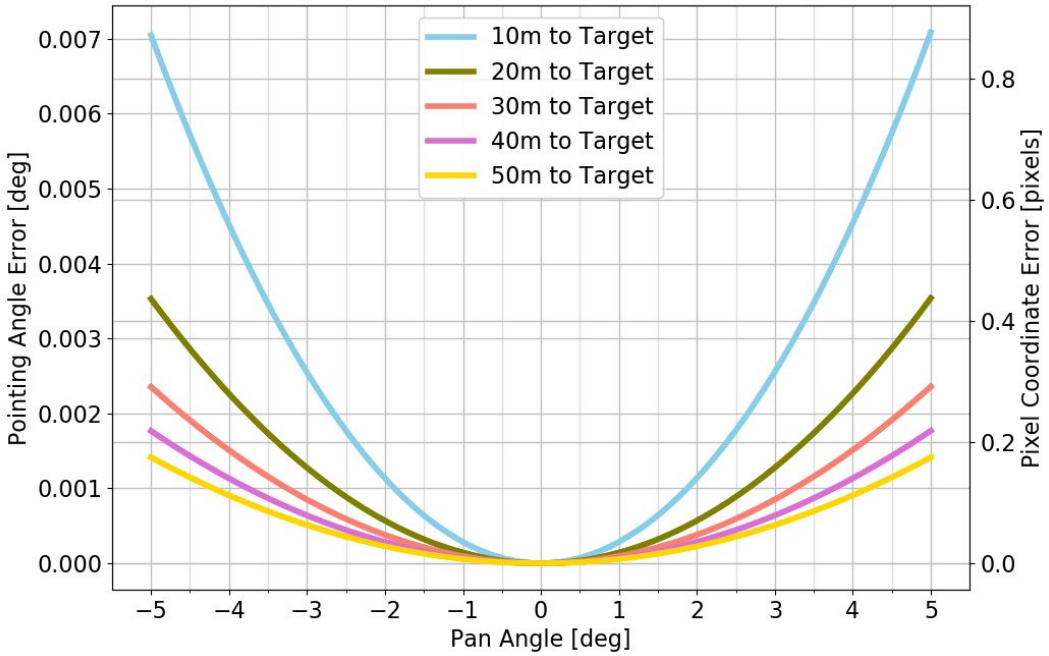


Figure 4.6: Pointing error after panning due to camera offset.

the camera is constrained to moving on a circle of radius L with center at point O, the pan axis of the PTU. The angle ϵ is the angular pointing error that results when the PTU pans by angle θ, which is the angle offset that the camera initially sees the object, represented by the yellow dot.

The pan angle required to correctly point the camera at the object is ψ, which can be found by first solving for angles ϕ and ξ, and subtracting the one from the other. Again, there are slight differences when the object is 'to the right' of the camera, and the PTU must pan clockwise.

$$\xi = \arctan(\frac{TD}{DO}) \qquad TD = AE = d \qquad DO = d * \tan(\theta) + L = l + L$$

$$\phi = \arccos(\frac{CO}{TO}) \qquad CO = L \qquad TO = \sqrt{TD^2 + DO^2} = \sqrt{d^2 + (l+L)^2}$$

$$\psi = \phi - \xi$$

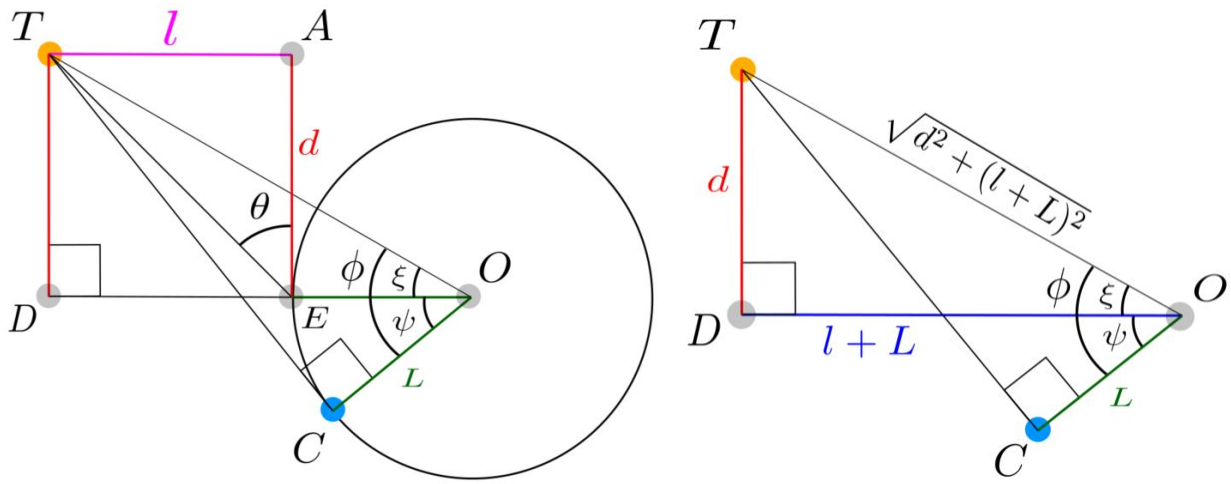


Figure 4.7: Geometry of camera offset and solving for correct pan angle.

4.3 Tracking in a Moving Frame

The algorithm discussed in previous sections is able to detect and track a moving target in a static scene. The background modeling and subtraction method used relies on a mostly constant background scene. When the target moves close to the edge of the camera's view, however, the PTU is commanded to move. While the PTU is moving, the entire scene in the camera's view is changing. If the background subtraction algorithm is run continuously through this movement, it will classify everything in frame as a moving target.

One solution to this problem is to end the background subtraction algorithm during the movement of the PTU, and to reinitialize the background model once the PTU has stopped moving. This can be accomplished by continuously querying the PTU's speed, and re-initializing the background subtractor model once both current pan and current tilt speeds are zero. Once reinitialized, it takes 4 to 5 frames for the algorithm to model the background scene sufficiently and to start detecting moving objects. As long as the target does not move out of view of the camera in those 4 to 5 frames, this method works well. If the target is

moving very quickly relative to the camera's frame rate, those 4 to 5 frames may be long enough for the target to leave the view.

Two methods of maintaining track of the target during and immediately after PTU movement are explored. The two methods are template feature matching and the optical flow background estimation developed in [11].

4.3.1 Template Feature Matching

The template feature matching method explored here involves taking features from a known image, or template, of the target and attempting to match them to features in a new image. If enough of the features from the template are successfully matched to features in a new image, it can be assumed that those features in the new image belong to the target. In this work, the template will come from the image immediately preceding PTU movement, and its features will be searched for in the following frames, until the PTU has stopped moving, and background modeling and subtraction methods can successfully detect the target again.

A template can be created by defining a rectangular region of interest (ROI) that encompasses the target. When the target moves close to the edge of the camera's field of view, and before the PTU is commanded to move, the targets location and approximate size in-image is known from the background modeling and subtraction algorithm. A rectangular box is drawn centered on the target, with width and height defined by the size of the target's blob, as seen in Figure 4.2e, plus a buffer in each direction. In this study, the buffer size is one of the parameters varied. The section of the image inside this rectangle is removed and saved as a separate image to be used as the matching template.

Feature detection and description, as described in Section 2.1.2, is performed on this template image. Feature detection and description is then performed on the proceeding frames. Each

new frame's features are matched to the features from the template. Because the template is composed mostly of the target, any feature in the new frame that matches a feature in the template is likely to be part of the target. The mean location of these matched features in the new frame can be used as the estimate of the target's location.

Figure 4.8: Rectangular ROI around target defining feature matching template.

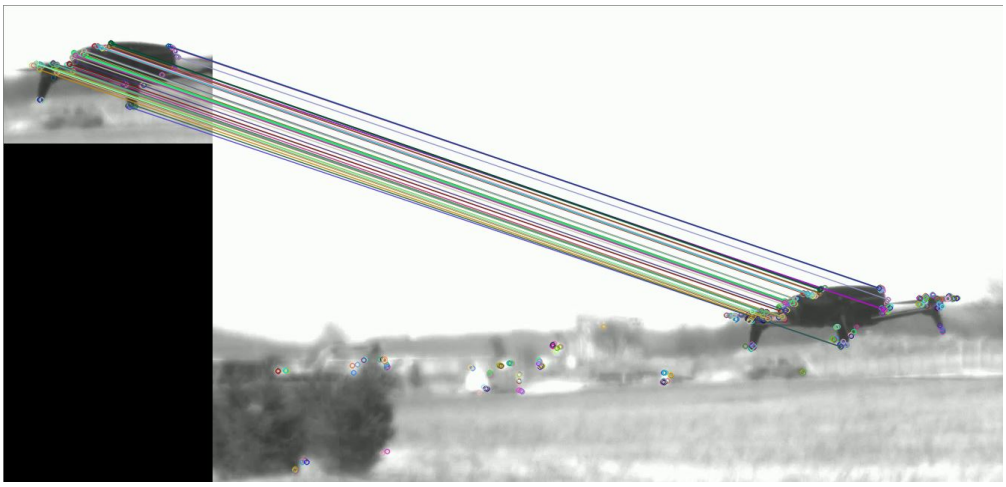


Figure 4.9: Matching features from the template to the new frame.

OpenCV includes a number of feature finding, describing, and matching algorithms, several

of which are patented. ORB (Oriented FAST and Rotated BRIEF), developed in [45], is a free-to-use alternative, and is what is used in the Template Feature Matching method described above. Claimed to be somewhat scale and rotation invariant, ORB finds features using the FAST (Features from Accelerated Segment Test) method. FAST analyzes a circle of pixels surrounding a potential feature point. If that circle of pixels contains a continuous chain of pixels above or below a certain intensity threshold, the center point is determined to be a corner, and therefore a feature worthy of description and matching. This method is shown in Figure 4.10. ORB uses the BRIEF (Binary Robust Independent Elementary Features) method of feature descriptions, which assigns binary string descriptors to a feature based on the relative intensity of pairs of surrounding pixels. These string descriptors are matched between images via brute force search using the Hamming distance as a measure of similarity.

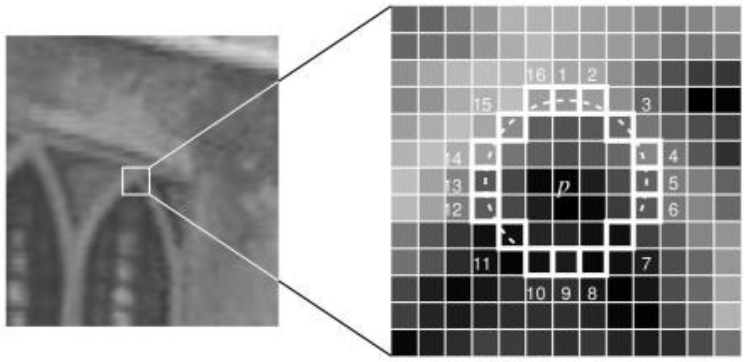


Figure 4.10: FAST method of corner detection looks for a continuous arc of pixels with relatively high or low intensity relative to the rest of the circle of pixels [59].

4.3.2 Optical Flow Background Estimation

The theory and process behind optical flow background estimation were developed in [11], and rely on Equations 4.4 and 4.5. These equations take the position of a point in an image, and predict that point's position in the next image, given the pan and tilt rotation angles of

the camera between the two images. The distance and direction of movement of a point in an image are known as the point's optical flow.

$$\hat{x}_{k+1} = f * \left(\frac{x_k - f * \tan \psi_k}{x_k * \tan \psi_k * \cos \theta_k - y_k * \frac{\sin \theta_k}{\cos psi_k} + f * \cos \theta_k} \right) \tag{4.4}$$

$$\hat{y}_{k+1} = f * \left(\frac{x_k * \sin \psi_k * \tan \theta_k + y_k - f * \cos \psi_k * \tan \theta_k}{x_k * \sin \psi_k - y_k * \tan \theta_k + f * \cos \psi_k} \right) \tag{4.5}$$

When the PTU moves, points in the image belonging to the background should behave according to these formula, and therefore fall within a specified bounds of their predicted location in the new image. Points belonging to a moving foreground object will not match their predicted locations, falling outside the specified pixel bounding radius, and can be identified as foreground. This process is visualized in Figure 4.11, where the points belonging to the sUAS are drawn red, meaning they have been classified as foreground.

In each image, good features to track (corners) are found using the Shi-Tomasi algorithm included with OpenCV [59]. These features are found again in the next sequential image, which occurs after the pan/tilt movement, and their pixel distance movement from one image to the next are measured. At each new frame, after calculating the optical flow from the last frame, new points are found. This is to counteract the loss of points that occurs naturally through the rotation of the target, or even simple changes in illumination that cause points to appear differently to the algorithm.

The implementation used in this work relies on the 'Good Features to Track' function to find points to track, as well as the 'Calculate Lucas-Kanade Optical Flow' function to calculate the movement of the points between frames. Both functions are included with OpenCV. The Good Features to Track method of feature finding is described in [47], and searches for

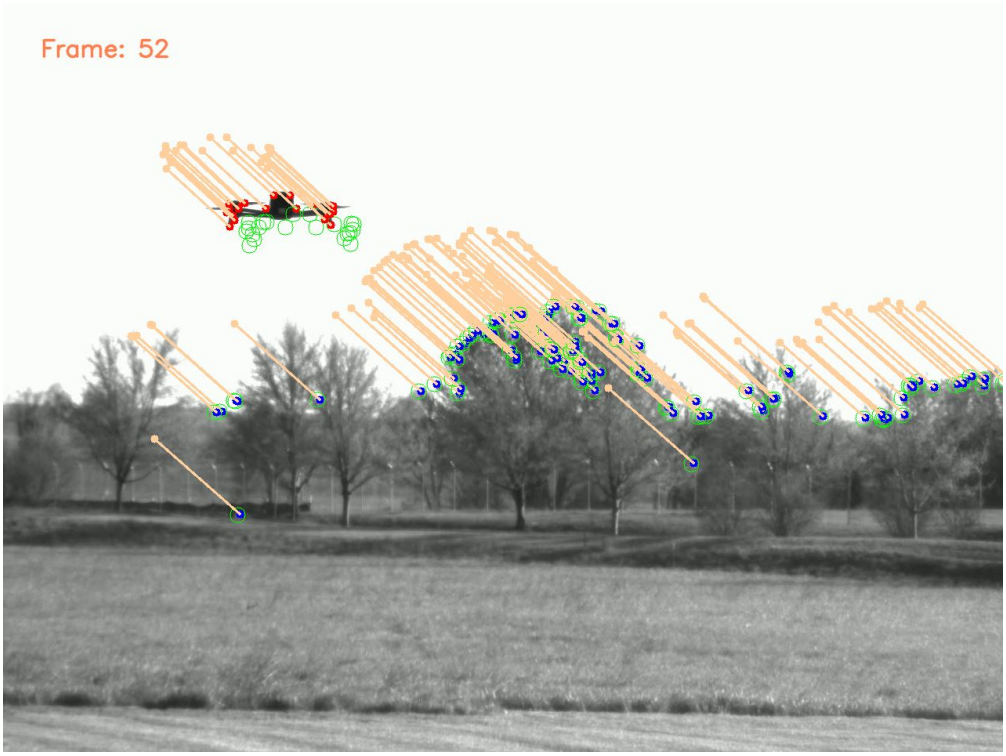


Figure 4.11: Using optical flow background estimation to identify a moving target.

strong intensity gradients in two directions within a small region of the image. Optical flow looks for bulk movement of pixels maintaining constant intensity distributions in order to match features from one image to the next. More on these functions, as well as the theory of optical flow background estimation, can be found in the original source, [11].

4.3.3 Moving Camera Test Video Data Set

Test videos of a flying sUAS taken from a camera mounted on the moving PTU were taken. The videos each contain one full movement, made up of both a pan and tilt rotation, that encompasses several frames. As the video frames were recorded, the PTU's pan and tilt angles were queried. This data was logged, alongside the corresponding video frame number, so that full knowledge of the camera's rotations between frames was known. This information

is required to calculate the estimated optical flow background movement, but is not needed for the template frame matching.

Four video segments were selected to test the two methods described above. Each video contains five or six frames where the camera has rotated from the previous frame. These frames, along with the five subsequent before the background subtraction algorithm begins to work again, are the frames that were used to test and compare the accuracy of the two methods. Truth data for these test videos was generated in the manner described in Section 4.1.1.

The background subtractor parameters will be kept consistent across these tests, and so there will be no difference between the two methods during the frames that only the background subtractor is acting on.

4.3.4 Comparison of Methods

Each method has a variety of settable parameters that were varied for this testing. Descriptions of these parameters are available in Appendix B . For both methods, these parameters were altered one at a time to understand their effect on performance. The most impactful parameters were then varied in small full-factorial tests. Table 4.9 shows the best performance achieved for both of the methods in terms of RMSE, FPS, and FN. It seems as though the feature matching method is more accurate and reliable, whereas the optical flow background estimation method is faster. Both methods are fast enough to be implemented in real-time alongside the background subtractor. Across all combinations of parameters tested, the feature matching method averaged 38.39 FPS, and the optical flow background estimation method averaged 42.78 FPS.

Table 4.9: Results of feature matching and optical flow background estimation.

Method	RMSE	Avg. FPS	Avg. FN
Feature Matching	16.30	34.42	0.00
Optical Flow Background Estimation	28.66	57.50	0.75

Of note are the knowledge requirements for each of the tested methods. Template feature matching requires knowledge of the targets location and size in the final frame before PTU movement in order to form the template. Optical flow background estimation does not require this information, but instead requires precise knowledge of the angular movements of the camera between each frame. The performance of either will have to be weighed against the ability to meet these requirements in any specific application.

As the videos used for this testing were fairly homogeneous, it is recognized that further testing is required in order to understand both methods' viability across a range of scenarios. It should be noted that neither of these methods were used while experimentally testing the system as described in Section 5.

Chapter 5

Stereo Vision Implementation

5.1 Stereo Vision Setup

Another camera is added to the opposite side of the PTU to enable stereo vision ranging capability, as seen in Figure 3.2. Position of the target relative to the orientation of the PTU can be calculated using the equations discussed in 2.2.3. To maintain a consistent coordinate frame as the PTU move, the measured points can simply be rotated around the center of the PTU by the current pan and tilt angles.

The stereo vision geometry and equations discussed in 2.2.3 assume that the two cameras' image planes are co-planar, and therefore their optical axes are parallel. Additionally, the two cameras must be oriented similarly (the two cameras should agree on what 'up' is). This can be ensured using the process described here.

A target point is selected that is a distance Z_w away from the cameras. This target must be far enough away from the cameras that it is within the FOV of both cameras. The distance between the two cameras, d is known, and simple geometry will give the horizontal angle offsets, θ_1 and θ_2, to the target from both cameras. These angles can be converted to a horizontal pixel location, x_c, in each camera's image. This geometry is shown in Figure 5.1. Because the cameras are mounted at the same height, the target point should appear at the same vertical pixel location.

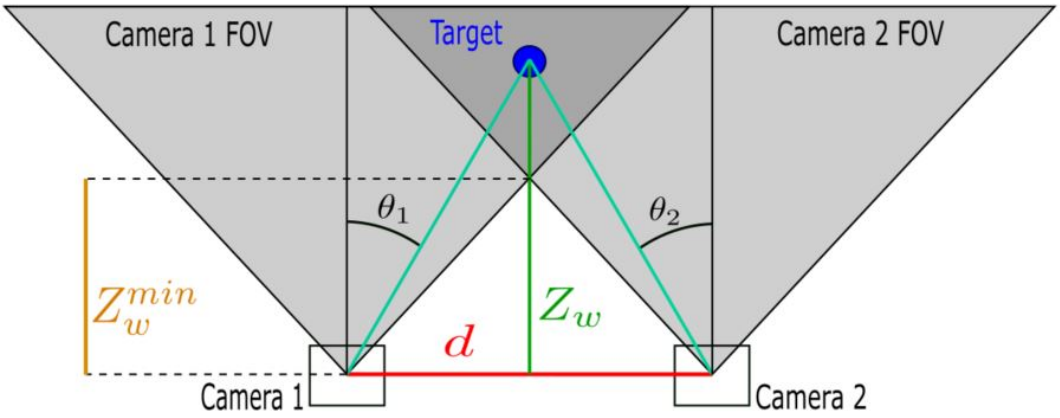
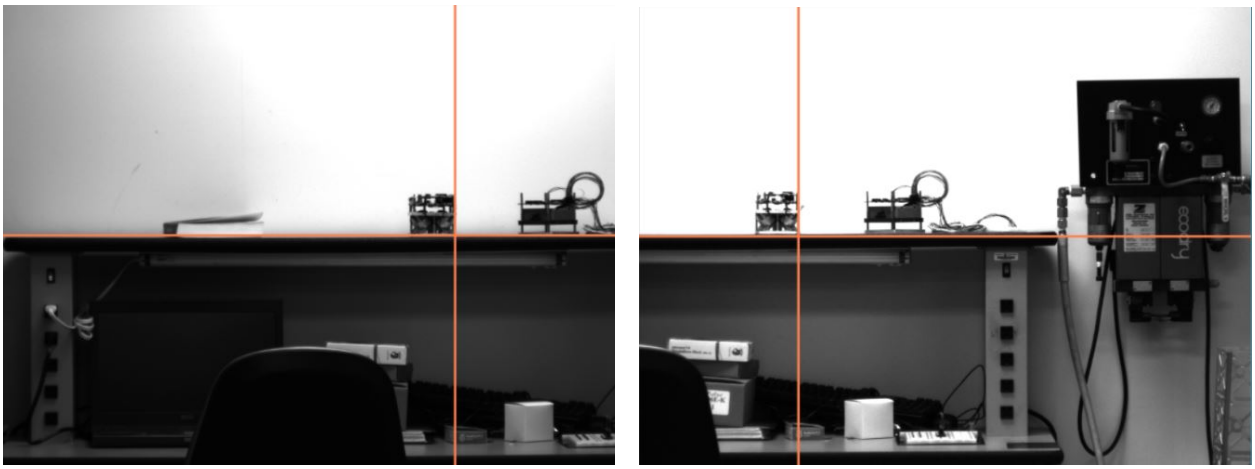


Figure 5.1: Geometry of camera alignment and field of view overlap.

By streaming images with lines overlaid at the horizontal and vertical pixel locations the equations above calculate the target point to be, the alignment of the cameras can be checked. Images from this process are shown in Figure 5.2. These lines, particularly the horizontal line, can also be used to check the rotation of the cameras. The overlaid horizontal lines should be parallel to known horizontal lines in the image. In these images, the overlaid lines intersect at the same target location for both cameras. The horizontal lines are, within reason, parallel to the horizontal surfaces in the image. This process was done at two different distances.

Almost any stereo vision system will have a blind spot distance: a distance before which, no point in the real world can be seen by both cameras at the same time. This distance is shown in Figure 5.1 as Z_w^{min}, and can be calculated using the distance between the two cameras, as well as their respective angular fields of view. The boundaries of the fields of view and the line connecting the cameras form a triangle for which one side length and two angles are known, allowing for the other dimensions to be solved for. In the system being tested, $Z_w^{min} = 3.824$ meters.



(a) Left Camera's View (Camera 1) (b) Right Camera's View (Camera 2)

Figure 5.2: Images from dual camera alignment verification.

5.2 Experimental Data

The system was tested multiple times via tracking of a user-operated sUAS flown outside. Three dimensional measurements taken by the system, along with their time stamp, were logged. In addition, latitude, longitude, and altitude (LLA) data of the sUAS itself was provided by the on-board GPS receiver at the same frame-rate as the tracking algorithm, and was logged. These GPS coordinate points, along with the known location of the PTU, were used to generate truth data. This was done using the same process of conversion between two sets of LLA's and Cartesian ENU coordinates that is described in Section 3.6.

An example of experimentally measured data plotted against truth data generated via GPS can be seen in Figure 5.3. The errors in each Cartesian direction at each time step are shown in Figure 5.4. Of note are the large, jagged jumps in the errors. Also, it should be noted that the GPS altitude reading proved to be inconsistent across experiments. Throughout testing, ground level was read as being a an altitude from 603 m to as high as 617 m. This is much worse than what was expected after performing the GPS testing discussed in Section 3.6. In order to correct for this, in processing of the test data, the PTU's altitude used in

the transformation from LLA to ENU coordinates was adjusted up or down as needed.

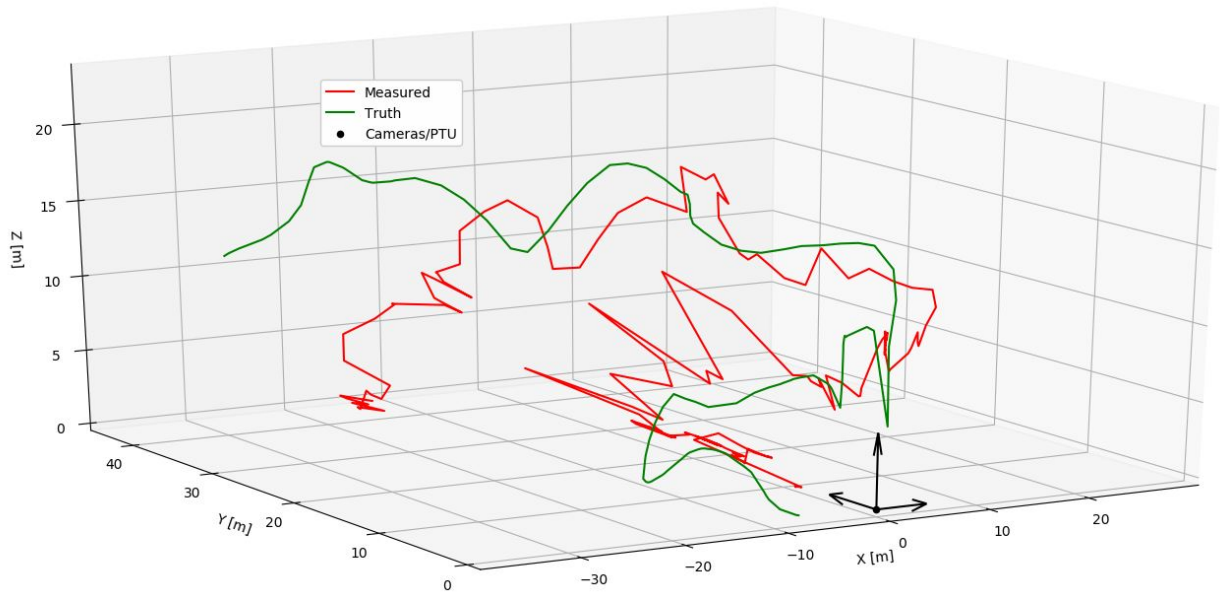


Figure 5.3: Raw measurements (red) plotted against truth (green), data set 1.

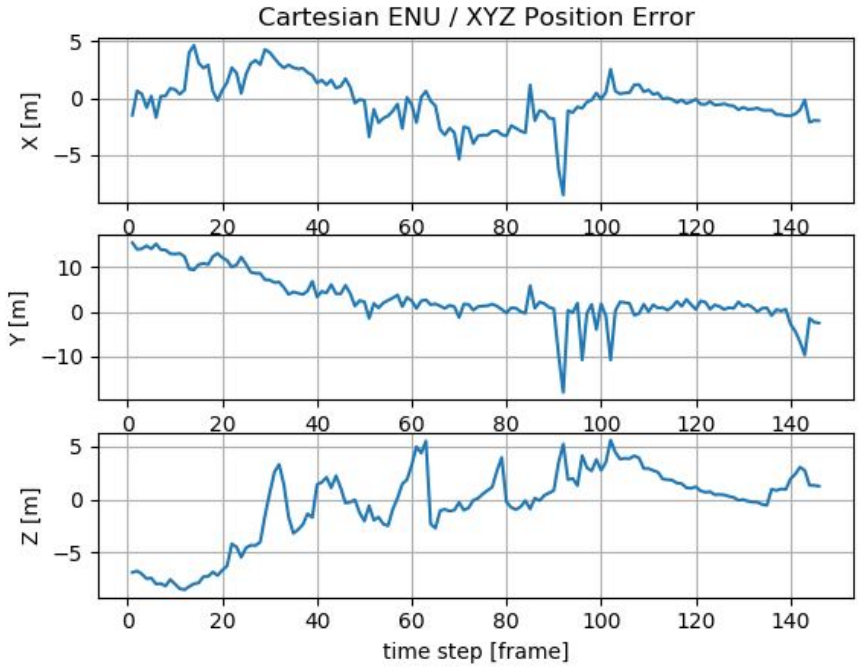


Figure 5.4: Error (truth - measured) for each coordinate axis at each time step, data set 1.

The experimentally measured data was smoothed using a median filter with a window size of 9 data points. What this means is that at every point, the X, Y, and Z coordinates of the data point are replaced by the median coordinate values of the 9 surrounding data points, including the original point itself. This filtering preserves the overall form of the path of the sUAS as measured by the system, but eliminates large jumps. The filtered measurements are shown plotted against the truth in Figure 5.5, and errors for each coordinate axis are shown in Figure 5.6

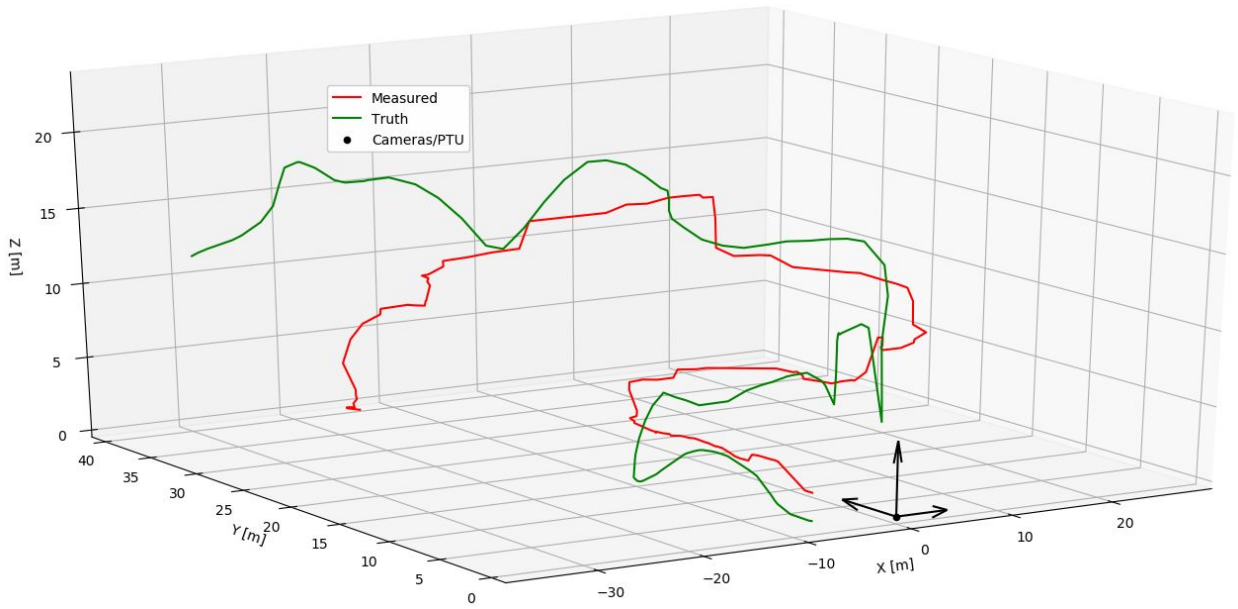


Figure 5.5: Filtered measurements (red) plotted against truth (green), data set 1.

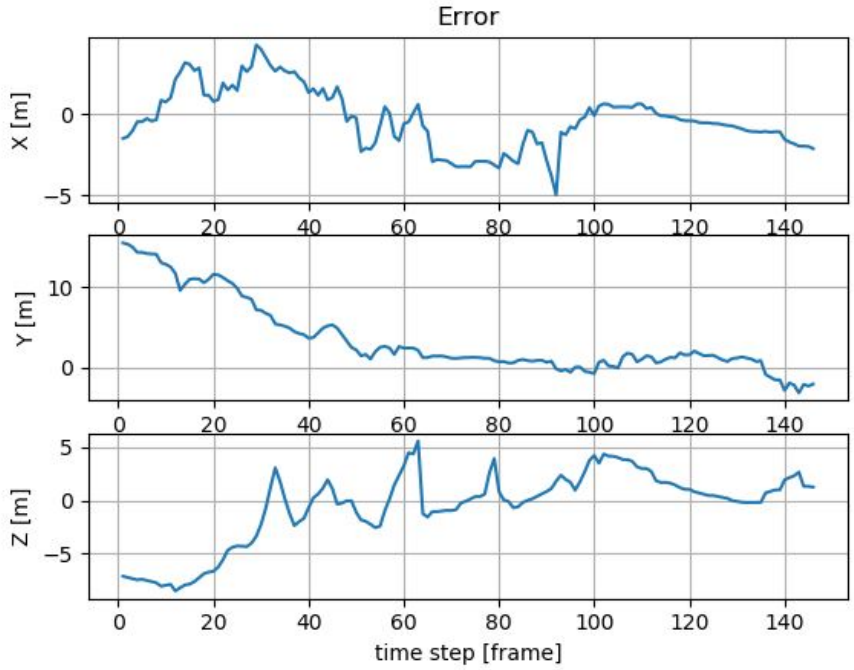


Figure 5.6: Error (truth - measured and filtered) for each coordinate axis at each time step, data set 1.

Figures 5.7, 5.8, 5.9, and 5.10 show the same results for a second set of data.

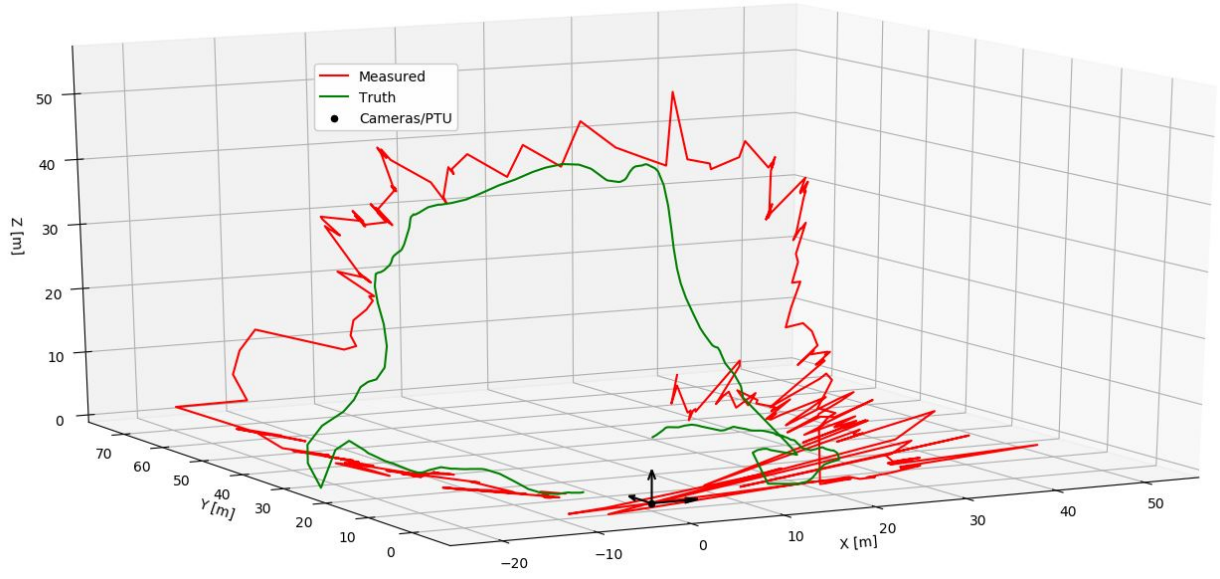


Figure 5.7: Raw measurements (red) plotted against truth (green), data set 2.

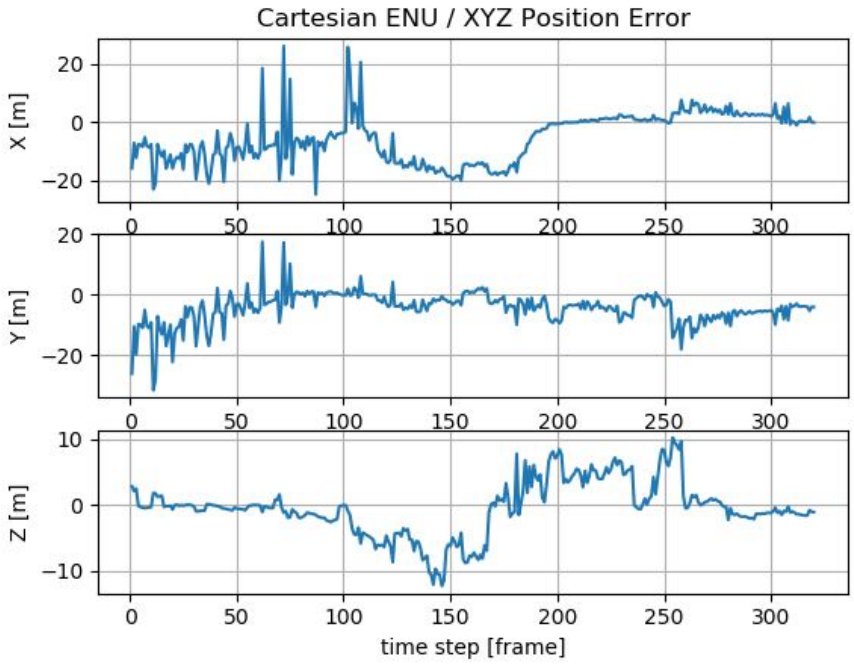


Figure 5.8: Error (truth - measured) for each coordinate axis at each time step, data set 2.

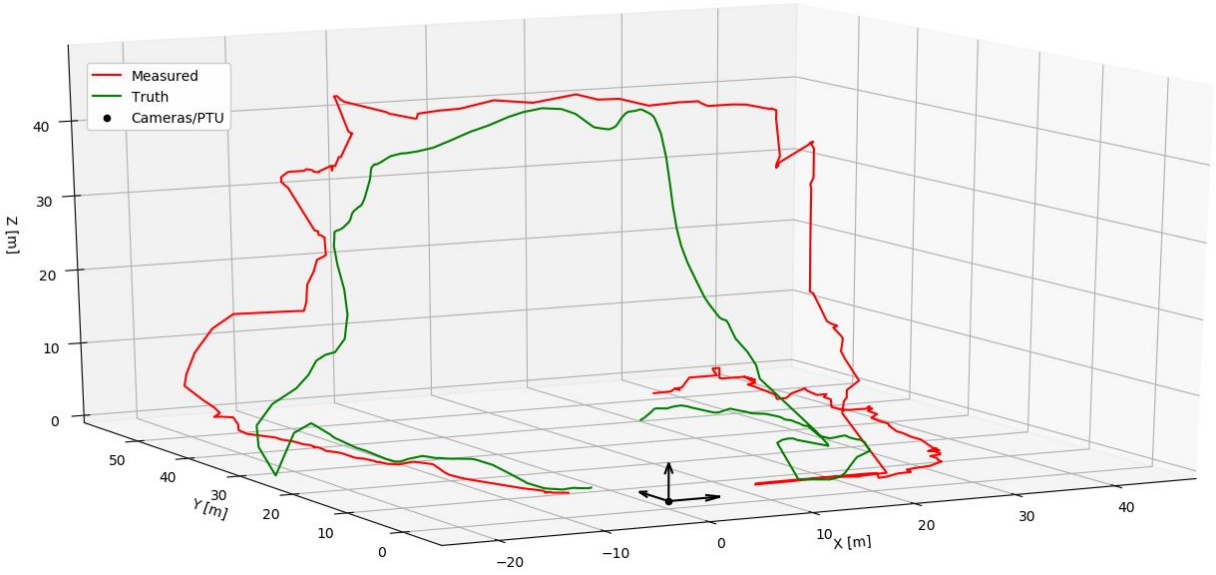


Figure 5.9: Filtered measurements (red) plotted against truth (green), data set 2.

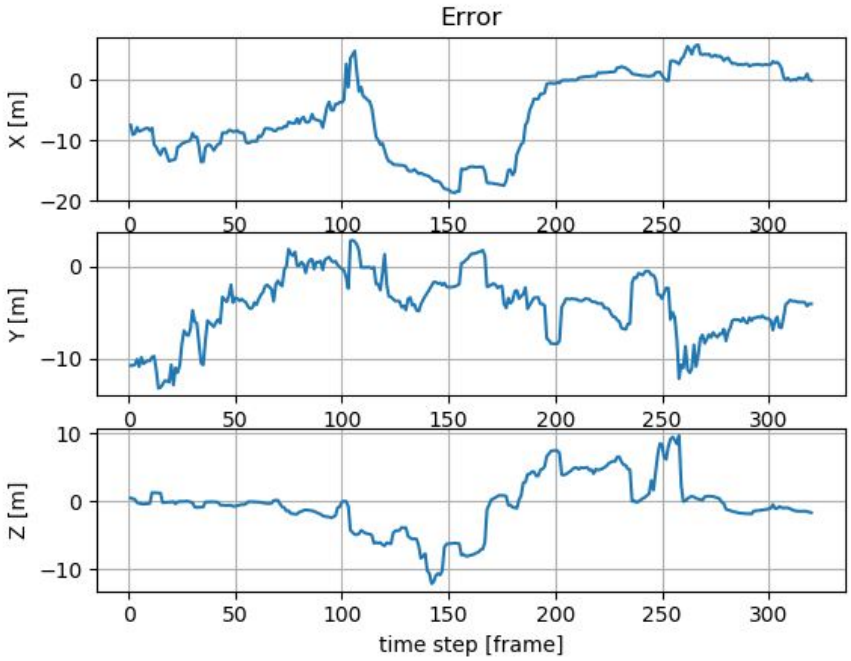


Figure 5.10: Error (truth - measured and filtered) for each coordinate axis at each time step, data set 2.

More data can be found in Appendix C.

5.3 Analysis and Discussion of Results

The three-dimensional position determination capabilities of the tracking system leave much to be desired. The general form of the flight path of the sUAS being tracking is preserved, but large errors occur in all directions at times. Is seems as though these errors grow larger at increasing range. This trend can be seen in Figure 5.11, which is the same plot from Figure 5.5, but viewed from the side along the X axis. It can be seen that as the sUAS moves further away from the PTU (at the origin), the error in the measured position grows.

The same phenomena can be seen in Figure 5.12, which is the same plot from Figure 5.9, but viewed from the top down.

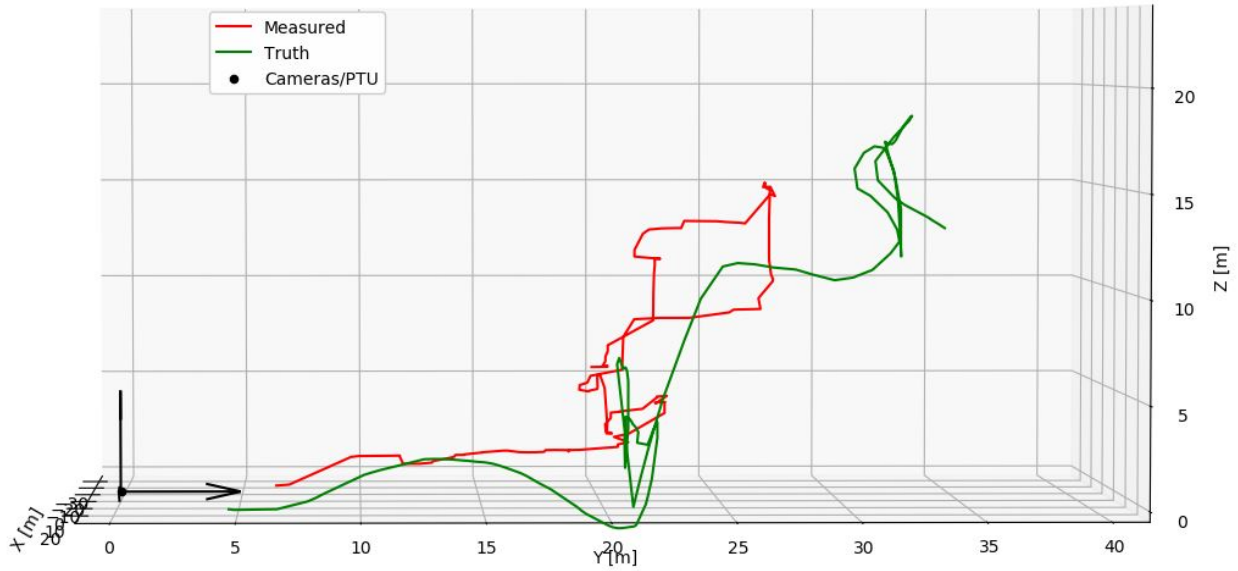


Figure 5.11: Filtered measurements (red) plotted against truth (green), data set 1, seen from the side.

To verify this observed trend, the relationship between range to the sUAS and magnitude of position error was developed. The truth data from the GPS was converted from Cartesian to spherical coordinates, thereby giving the range from the PTU to the sUAS at every step. The magnitude of the errors between the truth data and the filtered measured data were plotted against these ranges. Magnitude of the errors here are defined as the RMSE between the measured point and the truth point, considering all three Cartesian coordinate directions. This plot is shown in Figure 5.13. The linear trend line fit to this data has a positive slope, confirming the correlation between measurement error and range to the sUAS. The same linear correlation does not exist between error and the other two components of the spherical coordinates, azimuth and elevation, as can be seen by the nearly horizontal trend line of Figures 5.14 and 5.15. Here, zero degrees azimuth is aligned with the zero pan position of the PTU, which is along the Y axis of our established Cartesian coordinate system.

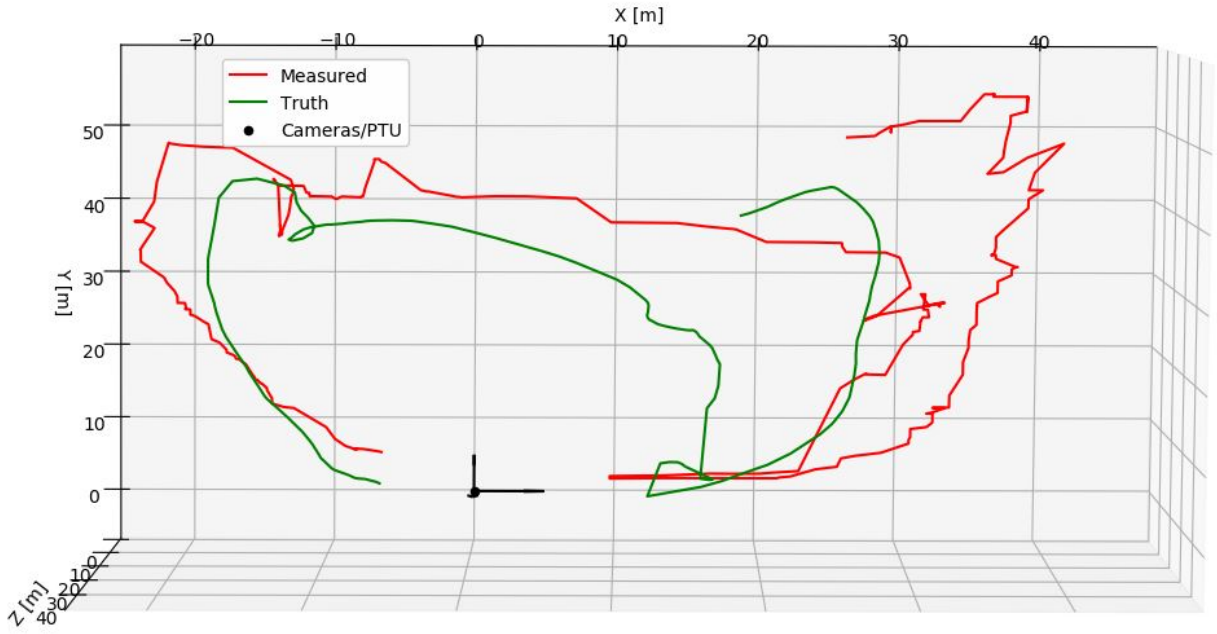


Figure 5.12: Filtered measurements (red) plotted against truth (green), data set 2, seen from above.

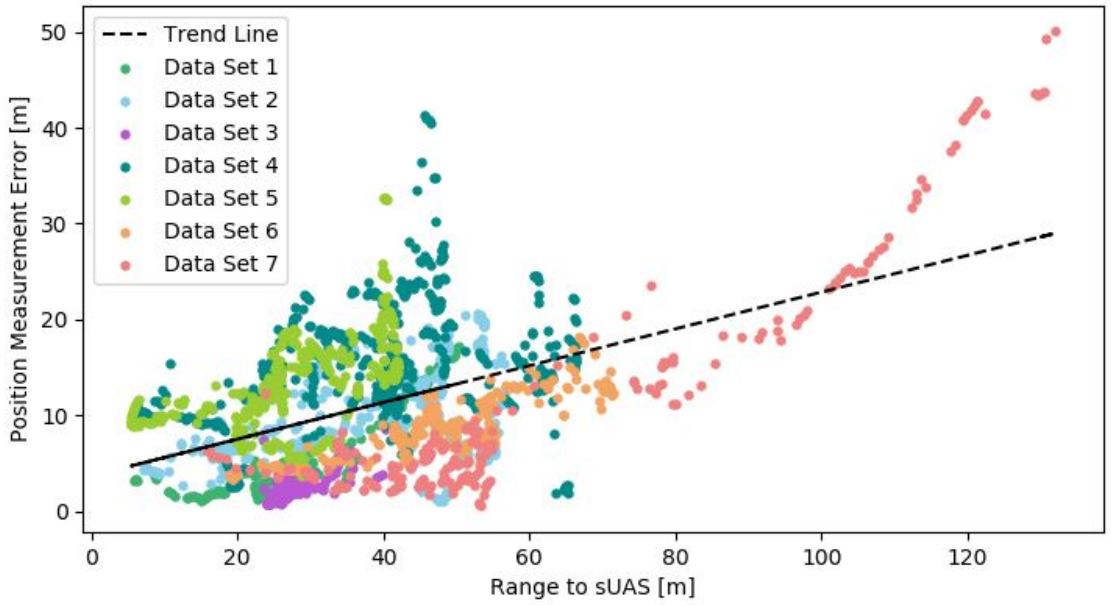


Figure 5.13: Magnitude of the positional errors of the measurements plotted against range from the PTU to the sUAS. Trend line shows positive linear correlation.

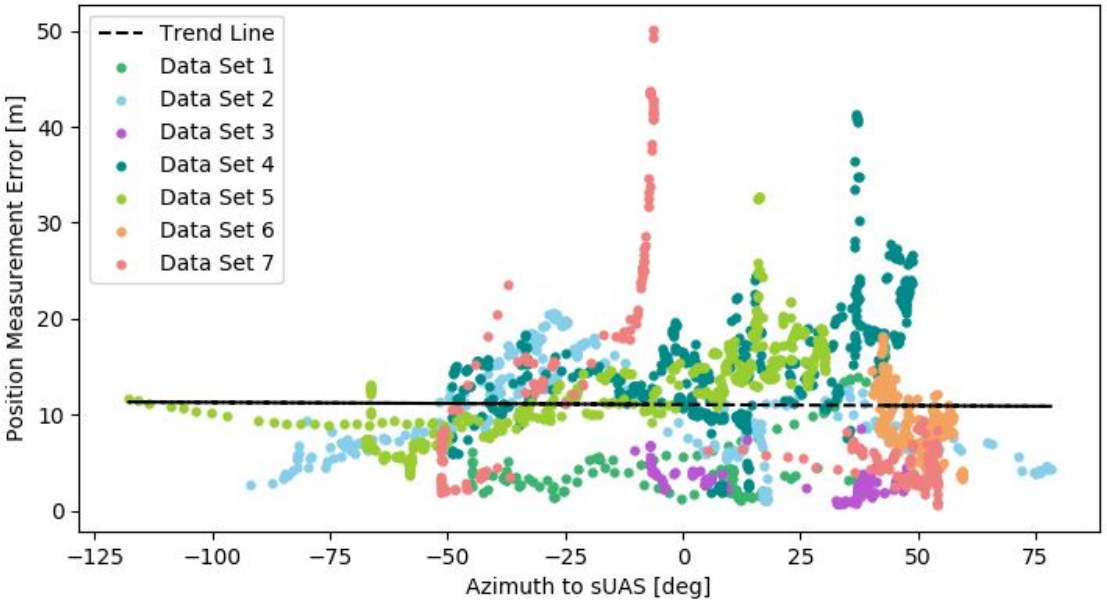


Figure 5.14: Magnitude of the positional errors of the measurements plotted against azimuth from the PTU to the sUAS. Trend line shows no linear correlation.

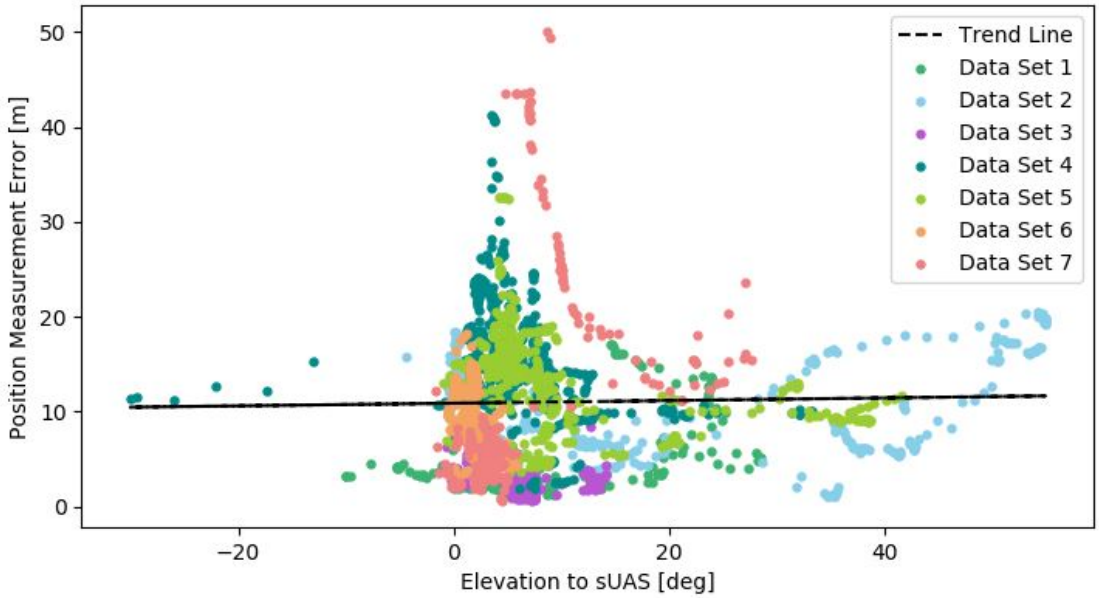


Figure 5.15: Magnitude of the positional errors of the measurements plotted against elevation from the PTU to the sUAS. Trend line shows no linear correlation.

Although no linear correlation exists between error and azimuth or elevation, the plots of

Figures 5.14 and 5.15 do not appear completely random. The data could suggest a different sort of relationship, such as a Gaussian distribution, with peaks near the origins. It is believed that this appearance of a relationship is misleading, and resulted from the fact that the longest range data taken occurred near the zero azimuth and elevation points. This is supported by the plots of Figures 5.16 and 5.17, which show range plotted against azimuth and elevation, respectively.

To receive GPS data, the tracking system must request a data point from the server, to which the sUAS is broadcasting. The server then sends the latest recorded LLA data to the tracking system. All of this communication occurs over a Wi-Fi network. It was thought possible that latency resulting in misalignment of the truth and measured data may be responsible for some of the error observed. The truth and measured data were plotted together against time. Figures 5.18 and 5.19 show these plots for data set 1 and data set 2, respectively. A small horizontal offset between the truth and the measured data can be seen at times, but not one large enough to be a significant contribution to error.

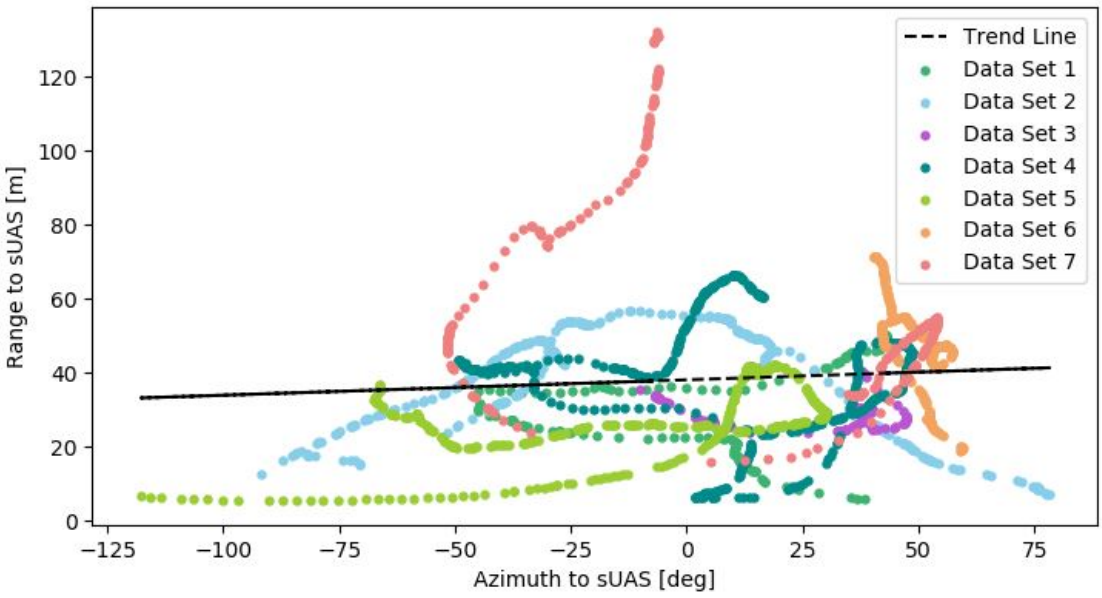


Figure 5.16: Range to sUAS plotted against azimuth to sUAS. Distribution appears similar to Figure 5.14.

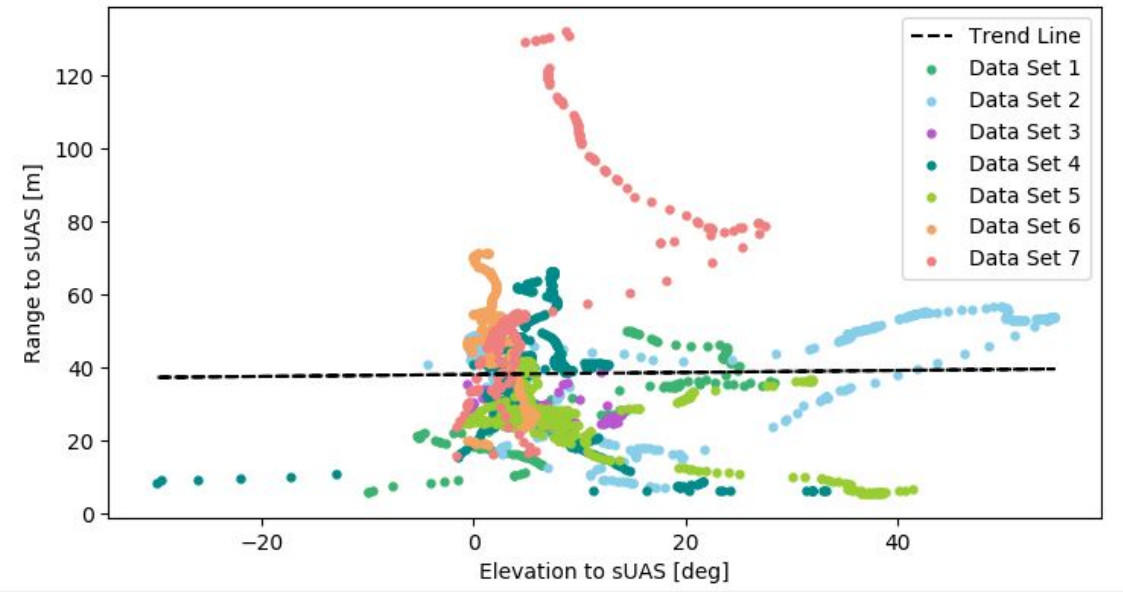


Figure 5.17: Range to sUAS plotted against elevation to sUAS. Distribution appears similar to Figure 5.15.

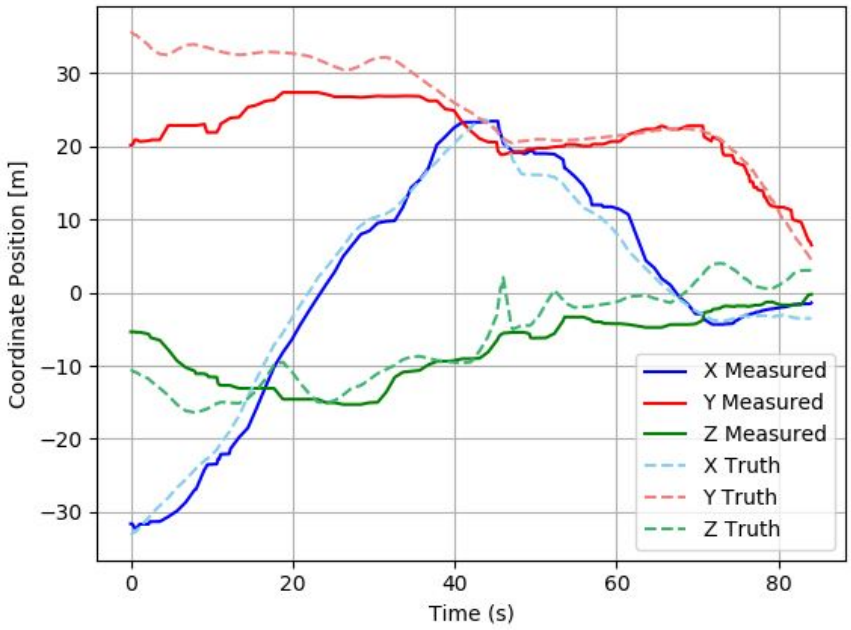


Figure 5.18: Truth and measured data, individual Cartesian components, plotted against time, data set 1.

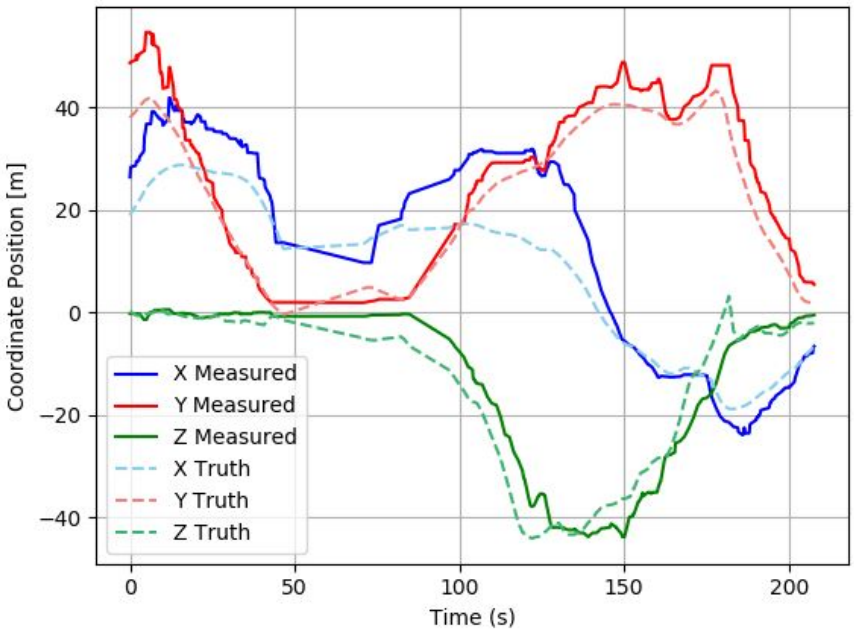


Figure 5.19: Truth and measured data, individual Cartesian components, plotted against time, data set 2.

The alignment process described in Section 5.1 ensured the rotational alignment of the cameras, but is not precise enough to verify the translational alignment. The cameras were assumed to be separated horizontally along the tilt axis of the PTU by .650494 m, but assumed to be aligned along the other two axes. The accuracy of this assumption rests on the precision of the machining of the mounting hole dimensions for both the PTU and the custom-designed brackets. A sensitivity analysis was performed, aiming to determine how errors in the relative translational positions of the cameras would have affected the results.

Along with the sUAS's position in 3D world coordinates, the sUAS's position in camera coordinates, and PTU rotation angles, at each time step were saved during testing. These camera coordinates can be used to recalculate world coordinates using different values for the camera's relative positions. The assumed relative offset of the cameras, (.650494, 0, 0) m, was varied by as much as 0.5 cm in each direction, and the real-world coordinates were

recalculated using the recorded camera coordinate measurements. The newly calculated real-world coordinates were compared to the originally calculated real-world coordinates, and the RMSE between the two was calculated.

Across all data sets, a change in relative camera position of 0.5 cm in each direction produced the greatest RMSE, which was 0.47402 m. The same RMSE was produced by using a change in relative camera position of -0.5 cm in each direction. This amount of error is significant, but it alone could not account for the error seen by the system while testing. Furthermore, is is highly unlikely the relative position of the cameras is off by 0.5 cm in any one, let alone all three, directions.

Considering the camera calibration process provided less-than-ideal results for the optical centers of both cameras, it was thought that the results for the focal lengths may have been unreliable. The same process used above to recalculate the real-world measurements can be performed with new focal lengths instead of different relative camera positions. In this way, new focal lengths can be found which minimize the RMSE between the recalculated world coordinates and the GPS truth data across all test sets.

Two multi-variable optimization schemes, the Nelder-Mead method and the Powell method, were used to find the combination of focal lengths, f_1 anf f_2, which minimize the RMSE of the recalculated data. Both of these methods are direct-search algorithms, and do not require derivatives of the functions, as they do not operate using gradients. For both methods, an initial guess of each variable to be optimized must be supplied. The initial guesses used, and the resulting focal lengths converged to by each optimization method are shown in Table 5.1. Across the wide range of initial guess values, both methods tend to converge to a similar answer, which is $f_1 = 5877$, $f_2 = 6094$.

Table 5.1: Focal length optimization for minimizing RMSE of experimental data.

	Nelder-Mead	Powell
f_1, f_2 Initial Guess	f_1, f_2	f_1, f_2
1000, 1000	5877, 6094	861, 1003
5000, 5000	5877, 6094	5877, 6094
6000, 6000	5877, 6094	5773, 6002
6800, 6800	5877, 6094	5879, 6095
7088, 7188	5877, 6094	5877, 6094
7500, 7500	5877, 6094	5877, 6094
8000, 8000	5877, 6094	5877, 6094
9000, 9000	5877, 6094	5877, 6094
20000, 20000	5877, 6094	5877, 6094

The optimized focal length values were used to recompute sUAS world coordinates from the saved camera and PTU coordinates. Plots for data sets 1 and 2 showing the newly computed world coordinates, originally recorded world coordinates, and GPS truth data are shown in Figures 5.20 and 5.21. Plots showing the reduced error in each Cartesian coordinate axis resulting from using the optimized focal lengths are shown in Figures 5.22 and 5.23. It is clear from Figures 5.20 and 5.21, that although the new focal length values may be optimized for the data sets as a whole, they actually increase the error for data set 1. The same is true for data set 7. If the focal lengths used were incorrect, and the optimized values were true, it would be expected that errors would decrease for all individual data sets. Therefore, although the original focal lengths used may not exactly match the truth, they are not the cause for the majority of the error in the data. The effect of the new focal lengths is

essentially a scaling of the data, an effect that could be achieved by misalignment of the two cameras. A more likely cause of the error observed is misalignment of the cameras that changed throughout the course of testing. A more rigorous camera alignment validation process, and further testing, would be required to confirm this.

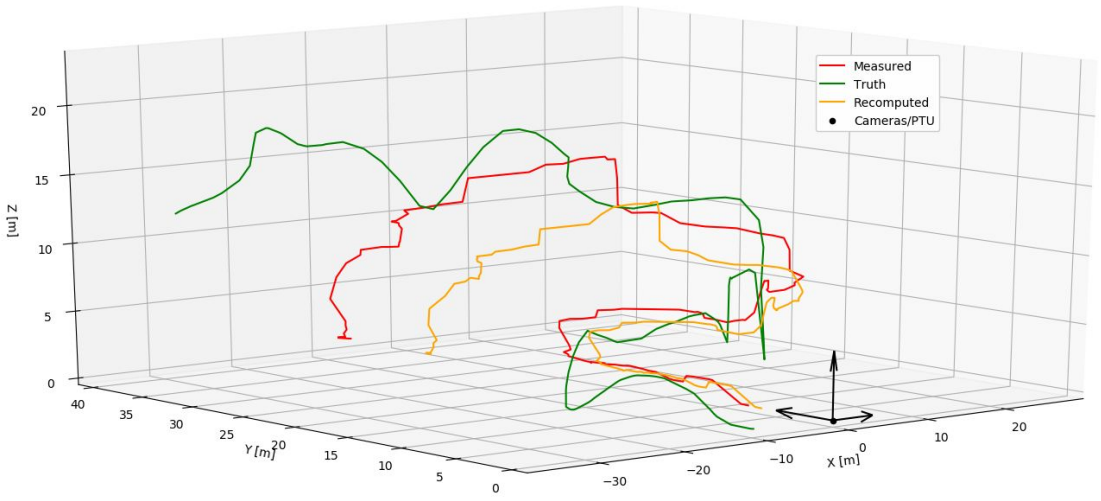


Figure 5.20: Truth, measured, and recalculated data, data set 1.

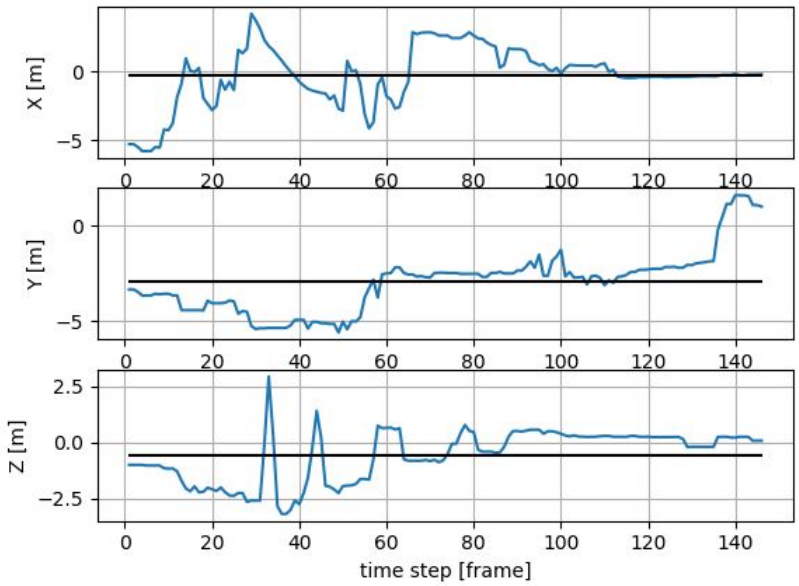


Figure 5.21: Error reduction in each Cartesian coordinate axis due to optimized focal lengths, data set 1.

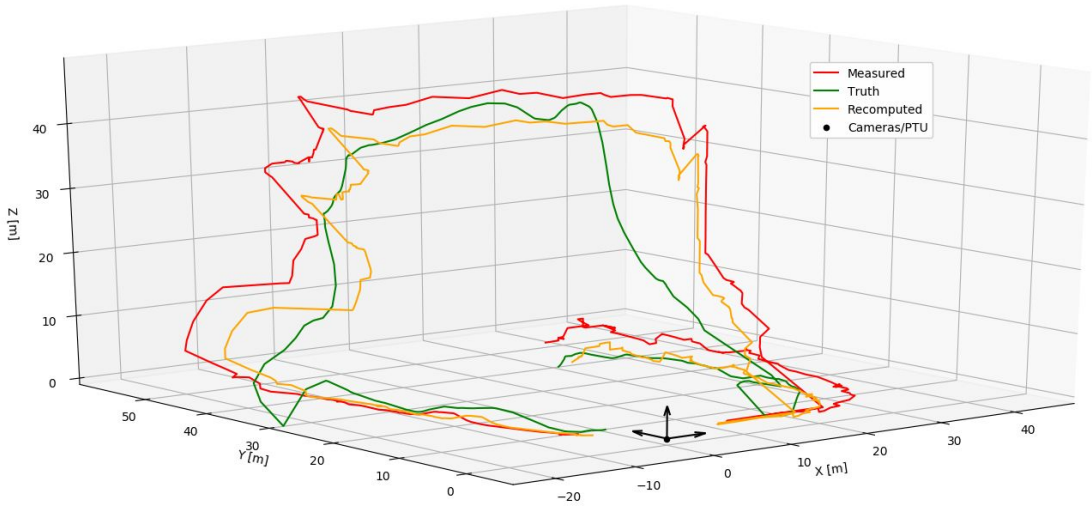


Figure 5.22: Truth, measured, and recalculated data, data set 2.

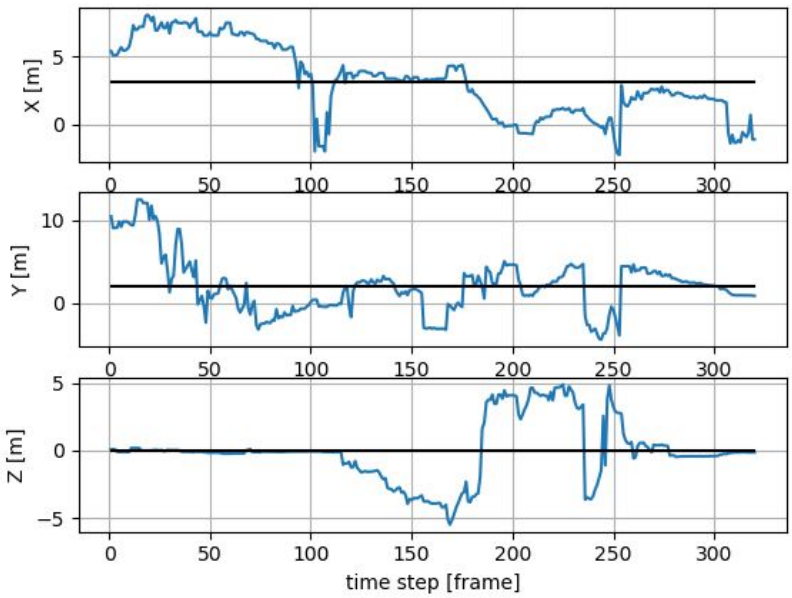


Figure 5.23: Error reduction in each Cartesian coordinate axis due to optimized focal lengths, data set 2.

Chapter 6

Conclusions and Future Work

The work in this thesis detailed the development of a computer vision tracking system that can be used to measure, in real time, the three dimensional movement of a small unmanned aerial system. The two dimensional tracking of the sUAS in the camera images is done with a mixtures of Gaussian background subtraction method coupled with several image pre- and post-processing techniques. To deal with camera motion, the background subtractor can simply be suspended during, and reinitialized after movement. Two methods of tracking throughout camera movement were explored briefly and proven in concept. The three dimensional tracking of the sUAS is accomplished with stereo vision enabled by the use of two cameras simultaneously. A linear least squares method is used to resolve the disparity between the two pointing vectors from the cameras to the sUAS. Accuracy of the system is analyzed by by comparing measurements against truth data generated by GPS measurements on-board the sUAS.

Several Kalman filter formulations are tested with the goals of improving the estimate of the in-image position of the tracked object, as well providing state predictions in the absence of measurements. With regards to predictive capabilities, higher-order Kalman filter formulations were shown to perform worse than the filter that kept track of only velocity. The Kalman filter formulations tested use diagonal matrices for the process and measurement noise covariance matrices, as is common in the computer vision community, but are not an accurate modeling of the uncertainties of the system. Future work could develop a more

realistic Kalman filter matrices and see how the predictive capabilities of the higher-order formulations are affected.

Through testing, the system is shown to be capable of continuously tracking a flying sUAS. The system is able to measure the position of the sUAS in-frame and calculate the PTU movements required to keep the sUAS within the FOVs of the cameras. In measuring the three-dimensional real-world position of the flying sUAS, the system requires improvement. The general form of the sUAS's flight path is preserved, but errors in all three Cartesian coordinate directions persist. It was shown that the errors increase with the distance to the sUAS.

The tracking system as a whole could be expanded and improved with different hardware. Wider FOV camera lenses would allow for measuring objects closer, and therefore allow for the use of taking truth data via an OptiTrack system which would provide more accurate and precise truth data. Wider FOV lenses would limit the effective tracking and ranging ranging distance of the system, however. Lenses with controllable mechanical zoom capabilities would allow for both testing with an OptiTrack system, as well as keeping track of objects at greater distances. In addition, rewriting the code in a compiled language such as C++, and running it on dedicated hardware could greatly increase the algorithm frame rate. Currently, the stereo vision 3D tracking system runs at roughly five FPS.

Future work for this line of research should include an exploration of the limitations of the methods used by varying the background environment and object-of-interest properties. Objects of different in-image size and speed should be tested, along with variations in the PTU speed settings. In addition, methods of discriminating between several moving objects in an image could be explored and implemented, limiting the loss of tracking due to other moving objects.

It is thought that another method for tracking a target during camera motion could involve maintaining and shifting the actual background model. This would involve saving the parameters that define the mixture of Gaussians probability density function at each pixel, and shifting them using the equations used in optical flow background estimation to move them by the correct amount in the image. This would allow for the continued use of the established background model throughout the camera motion, as long as the target does not leave the area of the scene that has already been modeled. The background model for the region of the image containing new parts of the scene will then have to be initialized and constructed.

Bibliography

[1] Megha. P. Arakeri and Lakshmana . Computer vision based fruit grading system for quality evaluation of tomato in agriculture industry. *Procedia Computer Science*, 79: 426–433, 12 2016.

[2] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In Aleš Leonardis, Horst Bischof, and Axel Pinz, editors, *Computer Vision – ECCV 2006*, pages 404–417, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[3] Alessandro Bevilacqua and Pietro Azzari. High-quality real time motion detection using ptz cameras. *2006 IEEE International Conference on Video and Signal Based Surveillance*, pages 23–23, 2006.

[4] George H. Born Bob E. Schutz, Byron D. Tapley. *Statistical Orbit Determination, 1st Edition*. Academic Press, 2004.

[5] Gary Bradski. Computer vision face tracking for use in a perceptual user interface. In *Proceedings of the Fourth IEEE Workshop on Applications of Computer Vision*, pages 214–219, January 1998.

[6] Ying Chen. A background subtraction algorithm for a pan-tilt camera. diploma thesis, University of Alberta, 2014.

[7] X. Clady, F Collange, F. Juire, and P. Martinet. Tracking with a pan-tilt-zoom camera for an acc system. In *Scandinavian Conference on Image Analysis*, pages 561–566, Bergen, Norway, 2001. IEEE.

[8] Xavier Clady, François Collange, Frédéric Juire, and Philippe Martinet. Object tracking with a pan-tilt-zoom camera: Application to car driving assistance. In *International Conference on Robotics & Automation*, Seoul, Korea, 2001. IEEE.

[9] Michel Couprie and Gilles Bertrand. Topological grayscale watershed transformation. In *SPIE Vision Geometry VI Proceedings*, volume 3168, pages 136–146, 1997.

[10] Musab Coşkun and Sencer Ünal. Implementation of tracking of a moving object based on camshift approach with a uav. *Procedia Technology*, 22:556 – 561, 2016. 9th International Conference Interdisciplinarity in Engineering, INTER-ENG 2015, 8-9 October 2015, Tirgu Mures, Romania.

[11] Daniel Doyle, Alan Jennings, and Jonathan Black. Optical flow background estimation for real-time pan/tilt camera object tracking. *Measurement*, 48:195–207, 02 2014.

[12] D. Eynard, P. Vasseur, C. Demonceaux, and V. Frémont. Uav altitude estimation by mixed stereoscopic vision. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 646–651, Oct 2010.

[13] Andrew B. Godbehere, A. Matsukawa, and K. Goldberg. Visual tracking of human visitors under variable-lighting conditions for a responsive audio art installation. In *American Control Conference (ACC)*, pages 4305–4312, June 2012.

[14] Lars J. Grimm, Jing Zhang, and Maciej A. Mazurowski. Computational approach to radiogenomics of breast cancer: Luminal a and luminal b molecular subtypes are associated with imaging features on routine breast mri extracted using computer vision algorithms. *Journal of Magnetic Resonance Imaging*, 42(4):902–907, 2015.

[15] İ. Güvenç, O. Ozdemir, Y. Yapici, H. Mehrpouyan, and D. Matolak. Detection, lo-

calization, and tracking of unauthorized uas and jammers. In *2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC)*, pages 1–10, September 2017.

[16] Markus Hägerstrand and Hjalmar Karlsson. Evaluation of background subtraction in pan-tilt camera tracking. diploma thesis, Chalmers University of Technology, 2016.

[17] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Proceedings of the 4th Alvey Vision Conference*, pages 147–151, January 1988.

[18] Eric Hayman and Jan-Olof Eklundh. Statistical background subtraction for a mobile observer. In *Proceedings Ninth IEEE International Conference on Computer Vision*, pages 67–74 vol.1, October 2003.

[19] A. Hernandez. Pan-tilt unit (e series) command reference manual, 2017. URL https://www.flir.com/globalassets/imported-assets/document/e-series-command-reference-manual.pdf. Last accessed 2 February 2019.

[20] Shengluan Huang and Jingxin Hong. Moving object tracking system based on camshift and kalman filter. In *2011 International Conference on Consumer Electronics, Communications and Networks (CECNet)*, pages 1423–1426, April 2011.

[21] Ramesh Jain and H. H. Nagel. On the analysis of accumulative difference pictures from image sequences of real world scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1:206–214, April 1979.

[22] Pakorn Kaewtrakulpong and Richard Bowden. An improved adaptive background mixture model for realtime tracking with shadow detection. *Proceedings of 2nd European Workshop on Advanced Video-Based Surveillance Systems; September 4, 2001; London, U.K*, 05 2002.

[23] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[24] Sangkyu Kang, Joonki Paik, Andreas Koschan, Besma Abidi, and Mongi Abidi. Real-time video tracking using ptz cameras. *Proceedings of SPIE - The International Society for Optical Engineering*, 5132, 04 2003.

[25] Dieter Koller, Joseph Weber, T Huang, J Malik, G Ogasawara, B Rao, and S Russell. Towards robust automatic traffic scene analysis in real-time. In *Proceedings of 1994 33rd IEEE Conference on Decision and Control*, volume 1, pages 126 – 131 vol.1, 11 1994.

[26] E. Komagal and B. Yogameena. Foreground segmentation with ptz camera: a survey. *Multimedia Tools and Applications*, 77:22489–22542, 2018.

[27] W. Kong, D. Zhang, X. Wang, Z. Xian, and J. Zhang. Autonomous landing of an uav with a ground-based actuated infrared stereo vision system. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2963–2970, Nov 2013.

[28] W. Kong, D. Zhou, Y. Zhang, D. Zhang, X. Wang, B. Zhao, C. Yan, L. Shen, and J. Zhang. A ground-based optical system for autonomous landing of a fixed wing uav. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4797–4804, Sep. 2014.

[29] Kenneth Kuhn. Small unmanned aerial system certification and traffic management systems, 2017. URL https://www.rand.org/pubs/perspectives/PE269.html.

[30] Jia-Guu Leu. A computer vision process to detect and track space debris using ground-based optical telephoto images. In *11th IAPR International Conference on Pattern Recognition*, The Hague, Netherlands, August 1992. IEEE.

[31] Da Li, Bodong Liang, and Weigang Zhang. Real-time moving vehicle detection, tracking, and counting system implemented with opencv. In *International Conference on Information Science and Technology*, Shenzhen, China, 2014. IEEE.

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

[33] Christine M. Belcastro, Richard Newman, Joni Evans, David Klyde, Lawrence C. Barr, and Ersin Ancel. Hazards identification and analysis for unmanned aircraft system operations. In *17th AIAA Aviation Technology, Integration, and Operations Conference, AIAA AVIATION Forum*, Denver, CO, June 2017.

[34] Carol Mejias, Iván Mondragón, Miguel Olivares-Mendez, Pascual Campoy, Luis Mejias, M Olivares, and Carol Martinez. Unmanned aerial vehicles uavs attitude, height, motion estimation and control using visual systems. *Autonomous Robots*, 29, 07 2010.

[35] LLC. Minitab. Minitab company webpage, 2019. URL http://www.minitab.com/en-us/. Last accessed 25 April 2019.

[36] Alexander Mordvintsev and Abid K. Revision. opencv python tutorials website, 2019. URL "https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html". Last accessed 28 February 2019.

[37] X. Pan, D. Ma, L. Jin, and Z. Jiang. Vision-based approach angle and height estimation for uav landing. In *2008 Congress on Image and Signal Processing*, volume 3, pages 801–805, May 2008.

[38] Zhigeng Pan, Yang Li, Mingmin Zhang, Chao Sun, Kangde Guo, Xing Tang, and Steven Zhiying Zhou. A real-time multi-cue hand tracking algorithm based on computer vision. In *2010 IEEE Virtual Reality Conference*, pages 219 – 222. IEEE, March 2010.

[39] Mitchel Pappot and Robert deBoer. The integration of drones in today's society. In *Proceedings of the 3rd European STAMP Workshop*, pages 54–63, Amsterdam, Netherlands, October 2015. Elsevier.

[40] Himani S. Parekh, Darshak G. Thakore, and Udesang K. Jaliya. A survey on object detection and tracking methods. *International Journal of Innovative Research in Computer and Communication Engineering*, 2, 2014 2014.

[41] Zhi Q. Qu, Dan Tu, and Jun Lei. Online pedestrian tracking with kalman filter and random ferns. *Applied Mechanics and Materials*, 536-537:205–212, April 2014.

[42] Ying Ren, Chin-Seng Chua, and Yeong-Khing Ho. Statistical background modeling for non-stationary camera. *Pattern Recognition Letters*, 24:183–196, January 2003.

[43] Rupesh Kumar Rout. A survey on object detection and tracking algorithms. diploma thesis, National Institute of Technology Rourkela, June 2013.

[44] Artem Rozantsev, Vincent Lepetit, and Pascal Fua. Flying objects detection from a single moving camera. In *Conference on Computer Vision and Pattern Recognition*, Boston, MA, 2015. IEEE.

[45] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: an efficient alternative to sift or surf. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2564–2571, 11 2011. doi: 10.1109/ICCV.2011.6126544.

[46] Afef Salhi and Ameni Yengui Jammoussi. Object tracking system using camshift, meanshift and kalman filter. *Journal of Science Education and Technology*, 6:674–679, 04 2012.

[47] Jianbo Shi and Carlo Tomasi. Good features to track. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, Seattle, WA, June 1994.

[48] Sam Siewert, Mehran Andalibi, Stephen Bruder, Iacopo Gentilini, and Jonathan Buchholz. Drone net architecture for uas traffic management multi-modal sensor networking experiments. In *2018 IEEE Aerospace Conference*, pages 1–18, March 2018.

[49] Sam Siewert, Mehran Andalibi, Stephen Bruder, Iacopo Gentilini, Aasheesh Dandpally, Soumyatha Gavvala, Omkar Prabhu, Jonathan Buchholz, and Dakota Burklund. Drone net, a passive instrument network diven by machine vision and machine learning to automate uas traffic management. In *AUVSI Xpotential*, May 2018.

[50] Chris Stauffer and W.E.L. Grimson. Adaptive background mixture models for real-time tracking. In *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 246–252, Boston, MA, June 1999. IEEE.

[51] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer, November 2010.

[52] Zygmunt L. Szpak and Jules R. Tapamo. Maritime surveillance: Tracking ships inside a dynamic background using a fast level-set. *Expert Systems with Applications*, 38(6): 6669 – 6680, 2011.

[53] Xun Wang, Jie Sun, and Hao-Yu Peng. Foreground object detecting algorithm based on mixture of gaussian and kalman filter in video surveillance. *Journal of Computers*, 8:693–700, March 2013.

[54] Greg Welch and Gary Bishop. An introduction to the kalman filter. *Proc. Siggraph Course*, 8, 01 2006.

[55] Karl Engelbert Wenzel, Andreas Masselli, and Andreas Zell. Visual tracking and following of a quadrocopter by another quadrocopter. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4993–4998, 2012.

[56] Christopher Richard Wren, Ali Azarbayejani, Trevor Darrell, and Alex Pentland. "pfinder: Real-time tracking of the human body". *IEEE Trans. Pattern Anal. Mach. Intell.*, 19:780–785, 1997.

[57] Serena Yeung. Stanford introduction to computer vision website, 2019. URL "http://ai.stanford.edu/~syyeung/cvweb/tutorial1.html". Last accessed 5 March 2019.

[58] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking: A survey. *ACM Computing Surveys*, 38:1–45, 01 2006.

[59] Sagi Zeevi. The impossible code, personal blog, 2017. URL "https://www.theimpossiblecode.com/blog/backgroundsubtractorcnt-opencv-3-3-0/". Last accessed 17 April 2019.

[60] Zoran Zivkovic. Improved adaptive gaussian mixture model for background subtraction. *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, 2:28–31 Vol.2, 2004.

[61] Zoran Zivkovic and Ferdinand van der Heijden. Efficient adaptive density estimation per image pixel for the task of background subtraction. *Pattern Recognition Letters*, 27: 773–780, 2006.

Appendices

Appendix A

Higher Order Kalman Filter Formulations

Just as acceleration a is the time derivative of velocity v, jerk j is the time derivative of acceleration, and snap s is the time derivative of jerk.

$$a = \frac{dv}{dt} = \frac{d^2x}{dt^2}$$

$$j = \frac{da}{dt} = \frac{d^2v}{dt^2} = \frac{d^3x}{dt^3}$$

$$s = \frac{dj}{dt} = \frac{d^2a}{dt^2} = \frac{d^3v}{dt^3} = \frac{d^4x}{dt^4}$$

Similar to the formulations of the velocity Kalman filter components in Section 2.2.4, formulations for the acceleration, jerk, and snap Kalman filters are given here. Of most importance are the state vector $\bar{x}$ and the state transition matrix F. The other components, Q, H, and R, remain diagonal matrices, with their dimensions adjusted according to need, and so are not included here.

Acceleration:

$$\bar{x} = \begin{bmatrix} x \\ y \\ v_x \\ v_y \\ a_x \\ a_y \end{bmatrix} \qquad F = \begin{bmatrix} 1 & 0 & t & 0 & \frac{1}{2}t^2 & 0 \\ 0 & 1 & 0 & t & 0 & \frac{1}{2}t^2 \\ 0 & 0 & 1 & 0 & t & 0 \\ 0 & 0 & 0 & 1 & 0 & t \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Jerk:

$$\bar{x} = \begin{bmatrix} x \\ y \\ v_x \\ v_y \\ a_x \\ a_y \\ j_x \\ j_y \end{bmatrix} \qquad F = \begin{bmatrix} 1 & 0 & t & 0 & \frac{1}{2}t^2 & 0 & \frac{1}{6}t^3 & 0 \\ 0 & 1 & 0 & t & 0 & \frac{1}{2}t^2 & 0 & \frac{1}{6}t^3 \\ 0 & 0 & 1 & 0 & t & 0 & \frac{1}{2}t^2 & 0 \\ 0 & 0 & 0 & 1 & 0 & t & 0 & \frac{1}{2}t^2 \\ 0 & 0 & 0 & 0 & 1 & 0 & t & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & t \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Snap:

$$\bar{x} = \begin{bmatrix} x \\ y \\ v_x \\ v_y \\ a_x \\ a_y \\ j_x \\ j_y \\ s_x \\ s_y \end{bmatrix} \qquad F = \begin{bmatrix} 1 & 0 & t & 0 & \frac{1}{2}t^2 & 0 & \frac{1}{6}t^3 & 0 & \frac{1}{24}t^4 & 0 \\ 0 & 1 & 0 & t & 0 & \frac{1}{2}t^2 & 0 & \frac{1}{6}t^3 & 0 & \frac{1}{24}t^4 \\ 0 & 0 & 1 & 0 & t & 0 & \frac{1}{2}t^2 & 0 & \frac{1}{6}t^3 & 0 \\ 0 & 0 & 0 & 1 & 0 & t & 0 & \frac{1}{2}t^2 & 0 & \frac{1}{6}t^3 \\ 0 & 0 & 0 & 0 & 1 & 0 & t & 0 & \frac{1}{2}t^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & t & 0 & \frac{1}{2}t^2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & t & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & t \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Appendix B

Template Feature Matching and Optical Flow Background Estimation Parameters

Template Feature Matching Parameters (Levels Tested)

- Template ROI Boundary: Minimum distance in pixels between the the target binary mask blob and the edges of the rectangular ROI template. (15, 25, 50, 75)
- Percent of Good Matches: The top percentage of features matched between the template and the new image to keep and use for target localization. (.1, .2, .25, .3, .5)
- Scale Factor: Reduction of pixels factor from one level of the image pyramid to the next. (1, 2, 3)
- Pyramid Levels: The number of image pyramid levels to use. (4, 6, 8, 10)
- Edge Threshold: The size of the border at the edge of the threshold to ignore when searching for features. (21, 31, 41)
- Patch Size: Size of the area surrounding each features to use to describe that feature's rotation. (21, 31, 41)
- WTA K: Number of points that make up each element of the part of each feature's description that defines the feature's rotation. (2, 3, 4)

- Score Type: Defines the method of ranking features by quality. The two options are the Harris algorithm scoring system or the FAST algorithm scoring system. (0, 1)

Optical Flow Background Estimation Parameters (Levels Tested)

- Bounding Radius: The radius in pixels of the circle, centered on the estimated location of a point, that defines the boundary beyond which a point is considered to be moving. (10, 15, 20)
- Max Corners: The maximum number of features to find and use for optical flow. (150)
- Quality Level: A measure of the minimum acceptable corner quality, as a ratio of the highest quality corner found. (.001, .01, .1, .3)
- Minimum Distance: The minimum distance in pixels between separate features. (3, 7, 10, 15)
- Block Size: Size of window over which to calculate the image intensity gradients to describe the points. (3, 9, 15, 25)
- Window Size: Size of the window to search to find point movement. (11, 21, 31)
- Max Image Pyramid Level: Number of image pyramid levels at which to search for good features to track. (3, 5, 7, 9)
- Termination Criteria Epsilon: Search is terminated when the search window becomes this small. (.001, .01, .1)
- Termination Criteria Count: Search is terminated when this many iterations have been completed. (20, 30, 40)
- Pixel Movement Maximum: A maximum bounds for target pixel movement. Pixels that move further than this bounds are classified as noise. (50, 100, 200)

Appendix C

Additional 3D Tracking Test Data

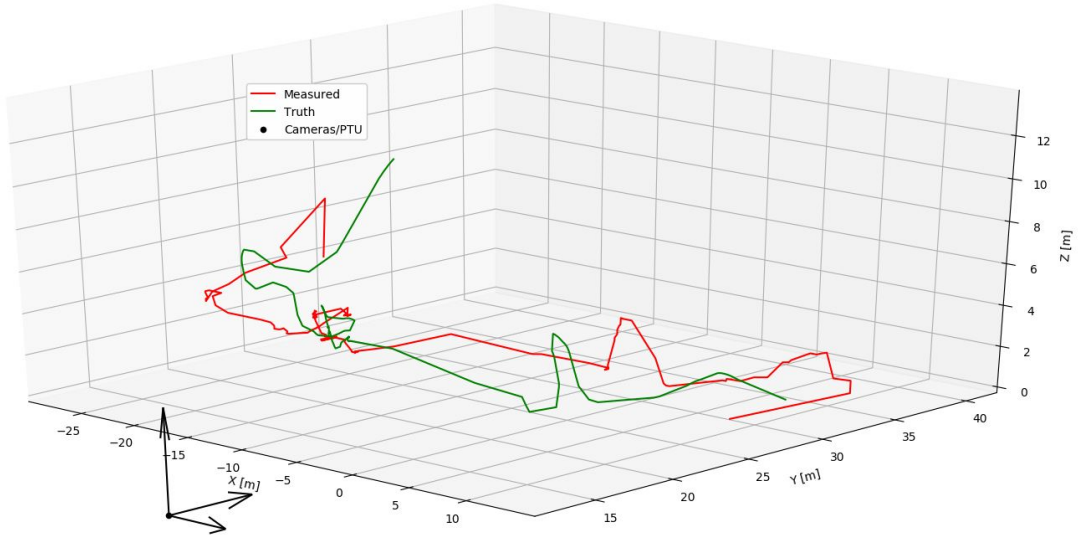


Figure C.1: Filtered measurements (red) plotted against truth (green), data set 3.

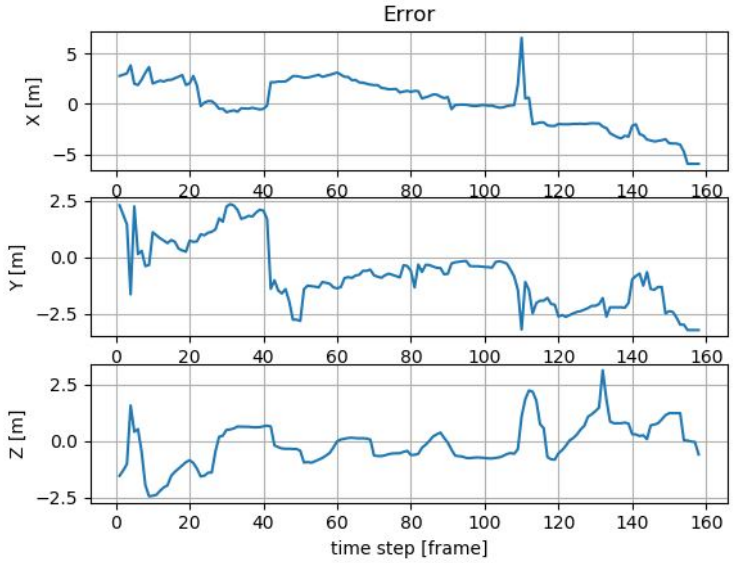


Figure C.2: Error (truth - measured and filtered) for each coordinate axis at each time step, data set 3.

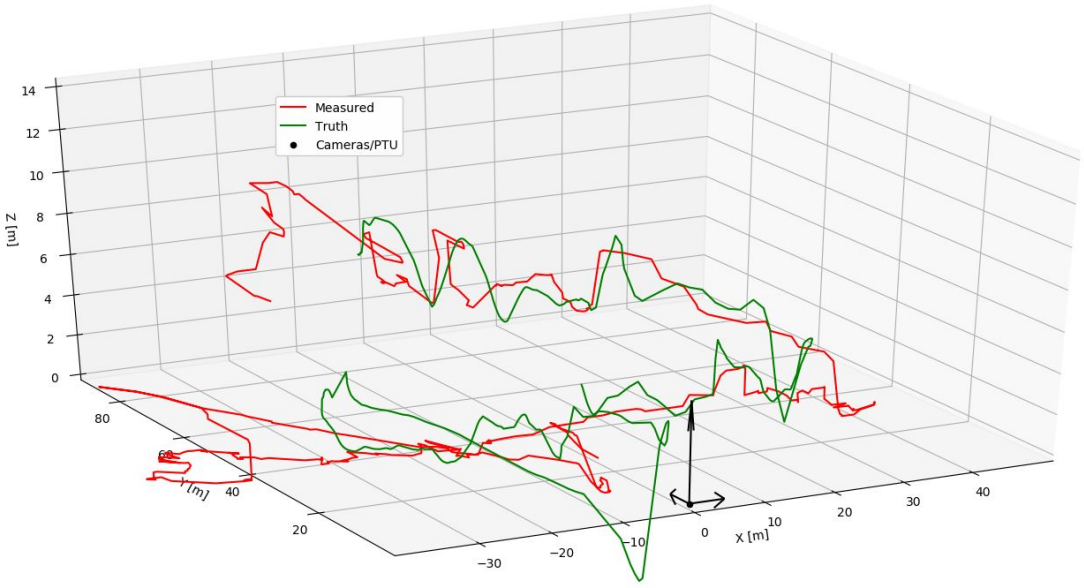


Figure C.3: Filtered measurements (red) plotted against truth (green), data set 4.

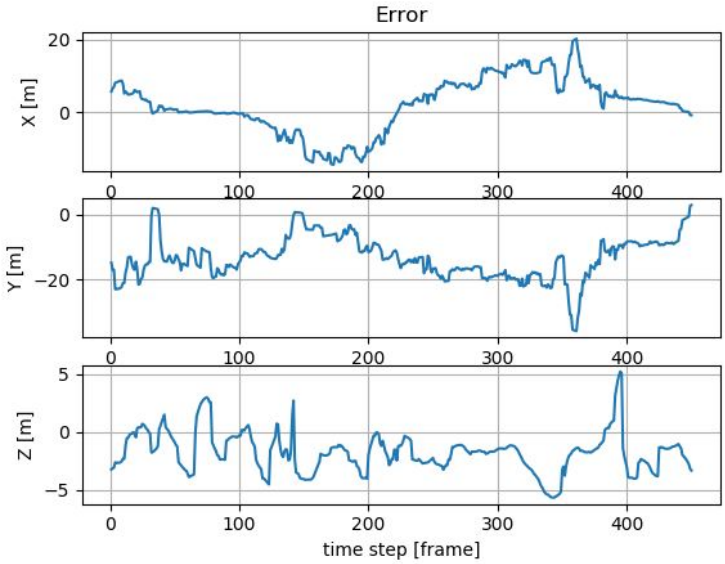


Figure C.4: Error (truth - measured and filtered) for each coordinate axis at each time step, data set 4.

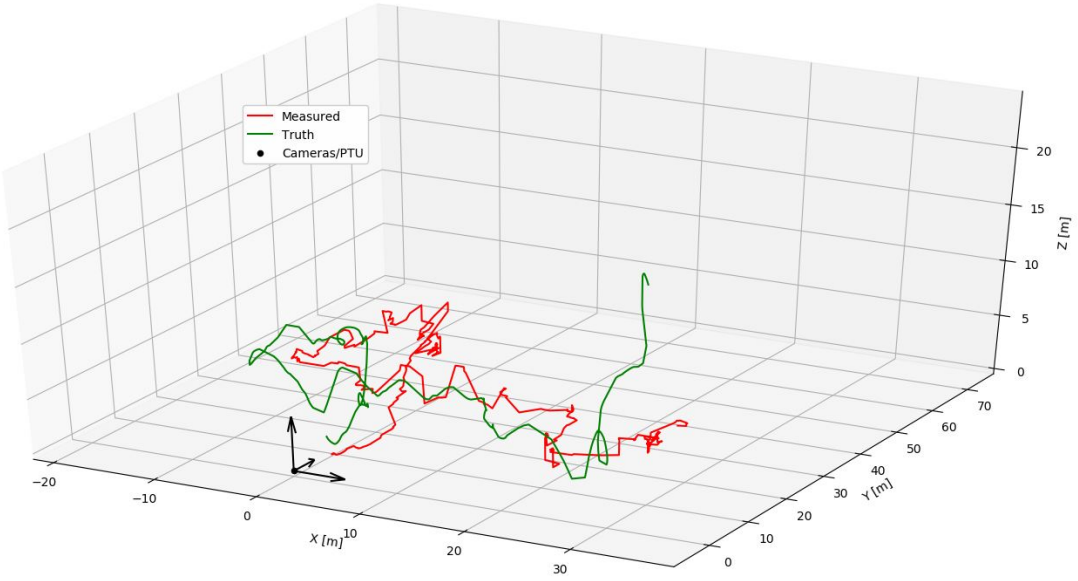


Figure C.5: Filtered measurements (red) plotted against truth (green), data set 5.

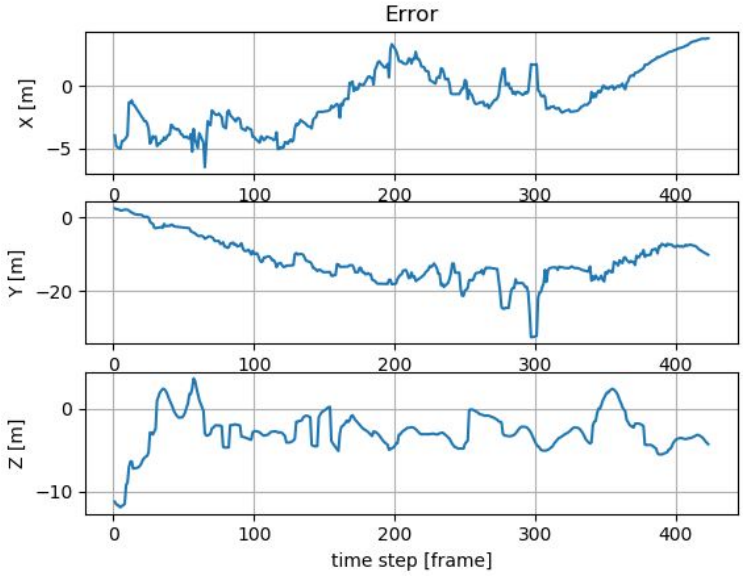


Figure C.6: Error (truth - measured and filtered) for each coordinate axis at each time step, data set 5.

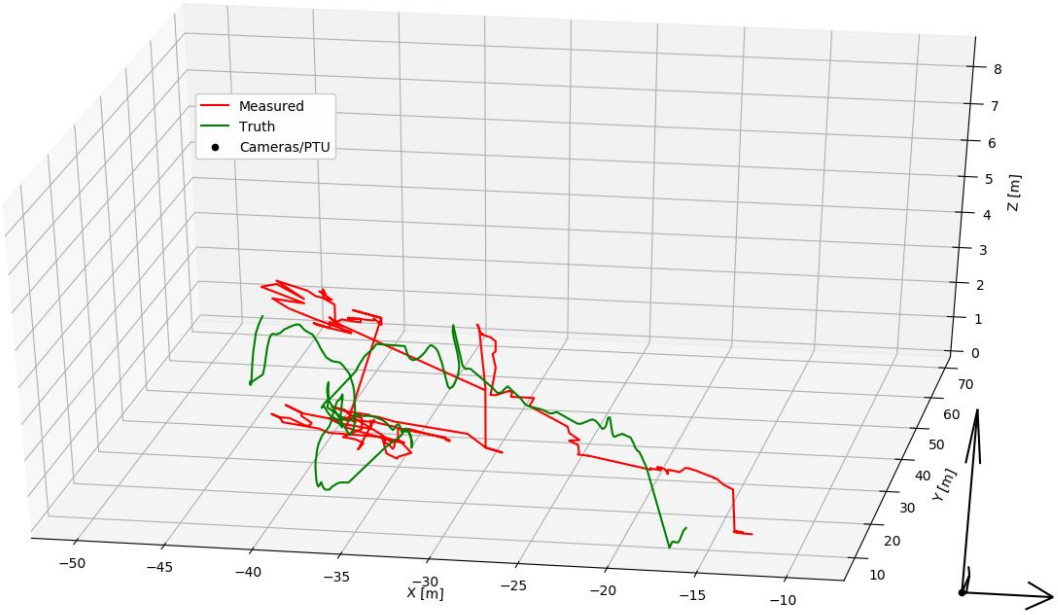


Figure C.7: Filtered measurements (red) plotted against truth (green), data set 6.

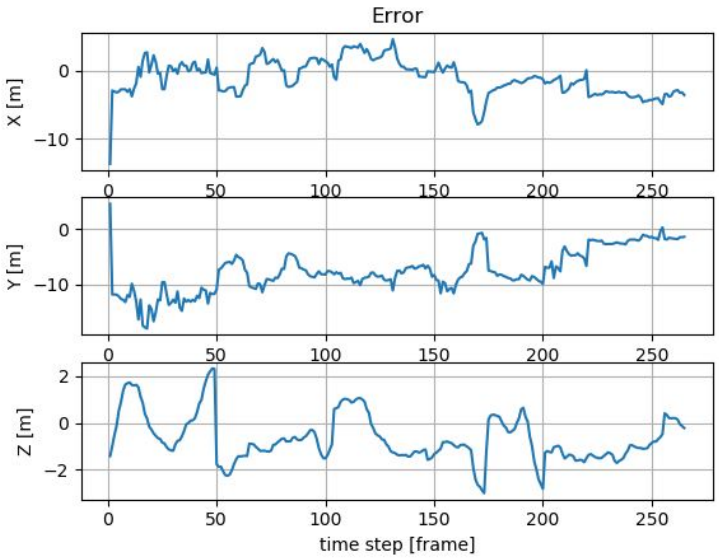


Figure C.8: Error (truth - measured and filtered) for each coordinate axis at each time step, data set 6.

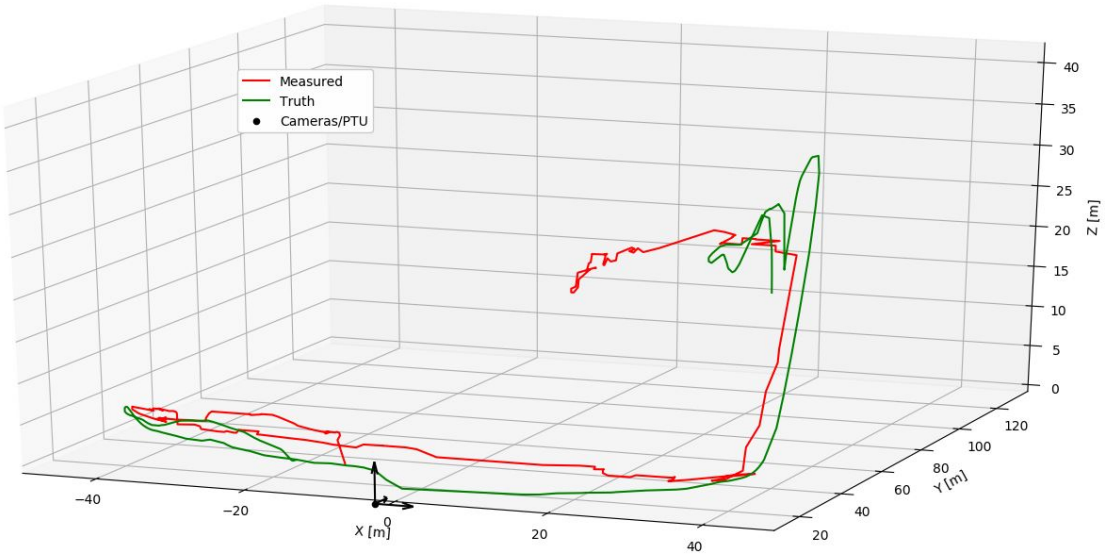


Figure C.9: Filtered measurements (red) plotted against truth (green), data set 7.

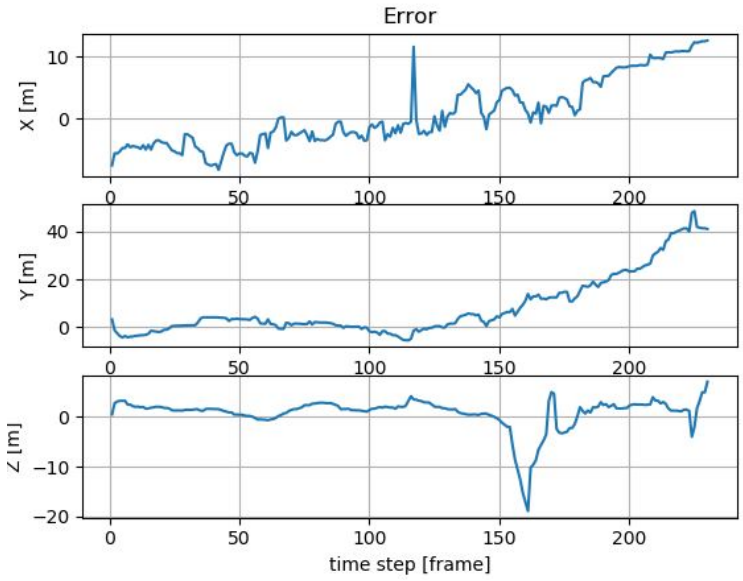


Figure C.10: Error (truth - measured and filtered) for each coordinate axis at each time step, data set 7.

Appendix D

Machined Parts Prints

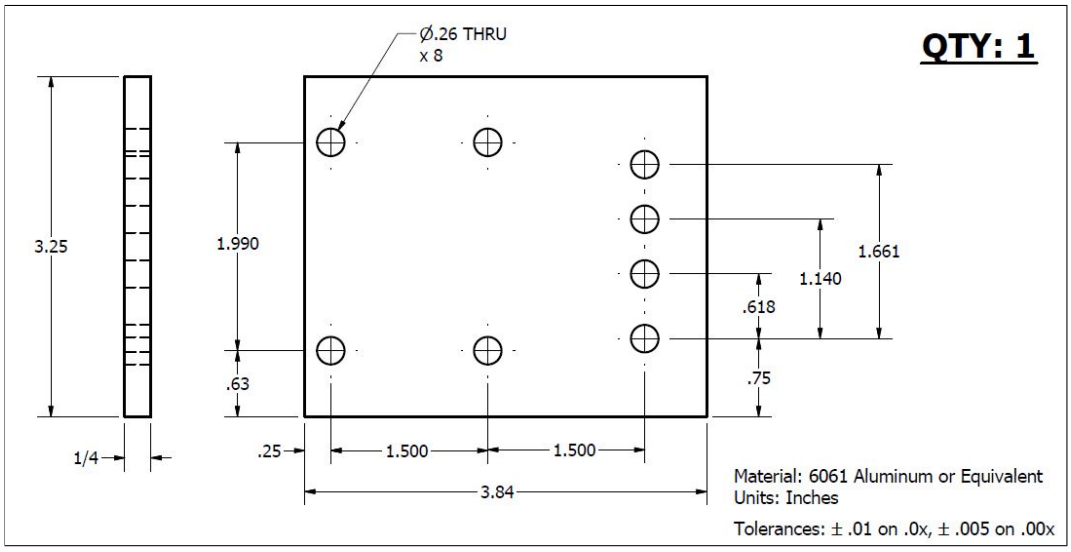


Figure D.1: Left camera horizontal mounting plate mechanical drawing.

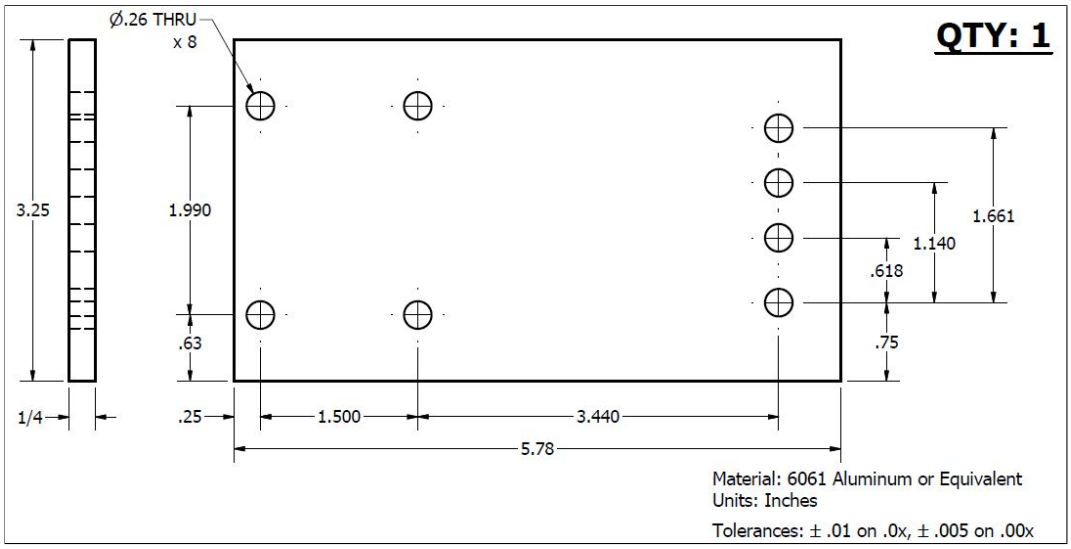


Figure D.2: Right camera horizontal mounting plate mechanical drawing.

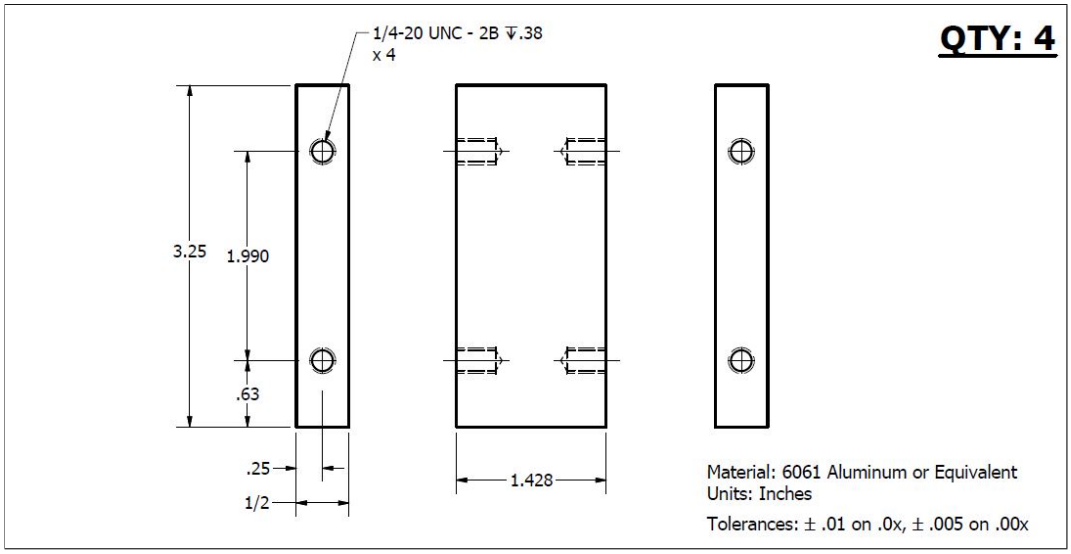


Figure D.3: Vertical mounting plates mechanical drawing.