NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

ASSESSMENT OF VISION-BASED TARGET DETECTION AND CLASSIFICATION SOLUTIONS USING AN INDOOR AERIAL ROBOT

by

Nicole R. Ramos

September 2014

Thesis Advisor:	Timothy H. Chung
Second Reader:	Oleg A. Yakimenko

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704–0188

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	09-30-2014	Master's Thesis 09-01-2012 to 09-01-2014

4. TITLE AND SUBTITLE

ASSESSMENT OF VISION-BASED TARGET DETECTION AND CLASSIFICATION SOLUTIONS USING AN INDOOR AERIAL ROBOT

5. FUNDING NUMBERS

6. AUTHOR(S)

Nicole R. Ramos

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

N/A

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views expressed in this document are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB protocol number: N/A.

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

The role of unmanned aerial vehicles (UAVs) in military, commercial and recreational applications is continuously evolving as developments in technology increase capabilities. The research herein presents an inexpensive computer-vision-based solution for detection and classification of a stationary target with a mobile aerial sensor as a prototyping platform. The main goal of this system is to use commercial-off-the-shelf and open-source components to reduce design complexity to provide a legacy product for future development of specific capabilities. Color imagery collected during flight using a low-resolution camera is used to test the application of a simple algorithm against a commercially available and low cost sensor. Original image processing algorithms that leverage the existing body of works in the open-source community are developed and tested within the Systems Engineering construct. System architecture leverages a modular approach that can be easily modified and adapted to changing requirements and objectives. Conclusions are drawn and recommendations for further study and system development are presented.

14. SUBJECT TERMS UAV, computer-vision, detection and classification, mobile aerial sensor, systems engineering

15. NUMBER OF PAGES 111

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2–89)
Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

ASSESSMENT OF VISION-BASED TARGET DETECTION AND CLASSIFICATION SOLUTIONS USING AN INDOOR AERIAL ROBOT

Nicole R. Ramos
Lieutenant, United States Navy
B.S., United States Naval Academy, 2008

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SYSTEMS ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 2014

Author: Nicole R. Ramos

Approved by: Timothy H. Chung
Thesis Advisor

Oleg A. Yakimenko
Second Reader

Clifford A. Whitcomb
Chair, Department of Systems Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The role of unmanned aerial vehicles (UAVs) in military, commercial and recreational applications is continuously evolving as developments in technology increase capabilities. The research herein presents an inexpensive computer-vision-based solution for detection and classification of a stationary target with a mobile aerial sensor as a prototyping platform. The main goal of this system is to use commercial-off-the-shelf and open-source components to reduce design complexity to provide a legacy product for future development of specific capabilities. Color imagery collected during flight using a low-resolution camera is used to test the application of a simple algorithm against a commercially available and low cost sensor. Original image processing algorithms that leverage the existing body of works in the open-source community are developed and tested within the Systems Engineering construct. System architecture leverages a modular approach that can be easily modified and adapted to changing requirements and objectives. Conclusions are drawn and recommendations for further study and system development are presented.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

List of Figures

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

ARSENL	Advanced Robotic Systems Engineering Laboratory
API	Application Program Interface
BGR	blue green red
CAS	close-in air support
COTS	commercial-off-the-shelf
CPA	closest point of approach
CPU	computer processing unit
csv	comma-separated value
DOD	Department of Defense
FAST	Features from Accelerated Segment Test
FOI	features of interest
FOV	field of view
FPS	frames per second
GESTALT	Grid-based Estimation of Surface Traversability Applied to Local Terrain
GPS	Global Positioning System
GUI	Graphic User Interface
HIS3	hue-in-saturation 3
HSV	hue saturation value
Hz	hertz

IMU	Inertial Measurement Unit
INS	Inertial Navigation System
IO	Information Operations
IP	Internet Protocol
IR	infrared
ISR	Intelligence, Surveillance and Reconnaissance
LA	landing area
NASA	National Aeronautics and Space Association
NPS	Naval Postgraduate School
OA	operating area
OpenCV	Open Source Computer Vision
PID	Proportional-integral-derivative controller
RGB	red green blue
RGB-D	red green blue depth
ROS	Robot Operating System
RPM	revolutions per minute
SIFT	Scale-Invariant Transform Feature
SURF	Speeded Up Robust Features
SWAP	size, weight and power
TAMD	threat air and missile defense
TCP	Transmission Control Protocol

UAV	unmanned aerial vehicle
UCAS	Unmanned Combat Air System
UCLASS	Unmanned Carrier-Launched Surveillance & Strike
UDP	User Datagram Protocol
USG	United States Government
USN	United States Navy
VTOL	Vertical Takeoff and Landing
2D	two-dimensional
3D	three-dimensional

THIS PAGE INTENTIONALLY LEFT BLANK

Executive Summary

An inexpensive computer-vision-based solution for detection and classification of a stationary target is developed and tested with a mobile aerial sensor as a prototyping platform. Original image processing algorithms that draw from the existing body of works in the open-source community are developed and tested within the Systems Engineering construct. Alternative solutions within the detection algorithm are analyzed against the baseline solution using the Systems Engineering approach.

The main goal of this system is to create a flexible and adaptable software framework for future computer-vision applications. The system architecture leverages a modular approach that can be easily modified and adapted to changing requirements and objectives. The project takes advantage of commercial-off-the-shelf (COTS) and open-source components to reduce design complexity and to provide a legacy product for future development of specific capabilities.

The computer-vision software developed consists of two parts: a detector and a classifier. The detector is decomposed into the steps from ingesting the original images from the video feed through to detecting an object of interest within the camera's field of view (FOV). The classifier takes the detector output and attempts to fit the detected object to a model of the target using a transformation matrix.

Color video footage is collected during flight of the prototyping platform, a Parrot AR.Drone, using the stock low-resolution camera. The AR.Drone's flight was constrained to an indoor Vicon System arena that provided near-real time ground truthing data. The target detection and classification algorithm is tested against the video collected from the AR.Drone. Modifications in the algorithm at various levels within the detection algorithm are made and the results are compared against the baseline algorithm, shown in Table 1.

	Baseline Algorithm	Alternative 1	Alternative 2	Alternative 3	
Pre-Processing	**HSV**	**RGB**			*PERCEPTION ALGORITHM*
Background Subtraction	**thresh_1** lower_G upper_G *[40,80,80] [70,255,255]*	**thresh_2** lower_G upper_G *[40,80,100] [70,255,255]*	**thresh_3** lower_G upper_G *[40,80,100] [80,255,255]*		
Image Filtering and Noise Reduction	**Bilateral Filter** *[3, 200, 0]*	**Gaussian Lowpass Filter** *[(3,3),0]*	**Bilateral Filter** *[5,150, 0]*		
Edge Detection	***None***	**Skeleton** *N/A*	**Canny Edge Detector** *[100,100]*	**Skeleton & Canny** *N/A [100, 100]*	
Line Detection	**HoughLinesP** *[1, math.pi/90, 40, 20, 35]*	**Blurring Function & HoughLinesP** *[5] [1, pi/90, 40, 20, 35]*			
Object Identifcation	**findContours** *cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMP*				CLASSIFICATION ALGORITHM
Object Classification	**warpPerspective**				

Table 1: Computer-vision algorithm alternatives are presented in table form to show a side-by-side comparison of changes in parameters and approach.

Conclusions are drawn and recommendations for further study and system development are presented.

Acknowledgments

First and foremost, I wish to thank my advisor, Professor Timothy Chung, for the last two years of guidance that have shaped my research. His optimistic attitude and tireless energy has made my work meaningful and especially interesting, and to him I owe a debt of gratitude. The advice I received from our discussions over the past 21 months will serve me well, both personally and professionally. To all the permanent members of the Advanced Robotic Systems Engineering Laboratory (ARSENL), especially Michael Clement and Michael Day, I am forever thankful. Without their support and assistance I would still be struggling to figure out how to connect to all the different systems I used along the way. Thanks to my second reader, Professor Oleg Yakimenko, for providing insight and advice. I would also like to thank every member of the Systems Engineering Department faculty and staff, for each member of the team has imparted wisdom and knowledge throughout the last two years that I will carry forward with me forever.

Thank you to all the students of ARSENL, both past and present, for providing me with a challenging and fun place to work, especially over the past summer. I enjoyed the challenges set forth everyday by my peers that worked alongside ARSENL, especially Brenton Campbell, Brad Davis and Blake Wanier, who collectively set the standards high and individually offered insight and tireless energy to help me along the way. You each truly are the embodiment of "Shipmates."

Finally, I would like to thank my family and friends, for their continued love and unwaivering support. Thank you to my San Diego family, notably Dana and Daneli, who were always ready and willing to host a stow-away for the weekend and distract me from my studies. Thank you to my Monterey family, especially the Vogts, Cariagas and O'Steens, who welcomed me into their homes on holidays, gave me a warm bed and delicious meals, and made me feel part of the family. Carson, you are the brother I always wanted! Erika, you kept me grounded and encouraged me to be more productive. Mallory, you helped bridge the gap between me and my little sister and gave me a feeling of belonging. To my older sister, Marie, and her son Fermin, who reminded me to work out when I hit a mental roadblock. To my sister Elena, who may be ten years my junior but who is catching up to me in experience. I enjoyed being a college student alongside you, little sis! To Jes-

sika Miller, Lauren Garcia and Caity White, my closest friends and confidants of ten years, who although scattered across the world, have made time to talk to me and stay in contact throughout this process. To my dad and his wife, Amy, I am pleased to have you in my life and I appreciate your patience and support along the way. And most of all, I would like to thank my mother; your patience and overwhelming support bolstered me when I was having a rough day and allayed my fears when self-doubt crept in. Thank you for the love and encouragement - it has made all the difference.

CHAPTER 1:
Introduction

Unmanned aerial vehicles (UAVs) are being used increasingly in military, commercial and recreational applications. The research herein presents an inexpensive computer-vision-based solution for detection and classification of a stationary target with a mobile aerial sensor as a prototyping platform. The main goal of this system is to use commercial-off-the-shelf (COTS) and open-source components as much as possible and to reduce design complexity to provide a legacy product for future development of specific capabilities. The system is comprised of the AR.Drone platform, and a linux host machine connected via a wireless connection. The computer vision program is demonstrated with a known target and validated using ground truth data provided by vicon data. This example illustrates some of the possible applications that can be easily implemented and that are advantageous for future research.

1.1 Motivation for Research

This research is motivated specifically by the development of UAV for use by U.S. naval forces. This research seeks to directly impact remotely piloted vehicles that incorporate the operator in the control loop, and completely autonomous vehicles that are capable of operations independent of direct user input. In particular, directional homing and navigation capability are common requirements for a wide variety of lightweight UAV military missions, and their development is the focus of this work.

1.2 Desired Capabilities

The current generation of Navy UAVs perform a wide variety of functions across many different missions: Intelligence, Surveillance and Reconnaissance (ISR), threat air and missile defense (TAMD), and Information Operations (IO) [1]. As of July 1, 2013, there are over 10,000 UAVs in the military's arsenal, the majority of which are classified as Class I UAVs, meaning they weigh under 20 lbs [2]. This class of UAVs is characterized by small, lightweight airframes with limited payload capacity and endurance. These vehicles

are equipped with a wide array of sensors and equipment, but almost all have a camera incorporated into the design.

As attractive as vision-based solutions appear to be for small UAVs, some challenges presented must first be overcome. Performance is hindered by limited bandwidth communication links and the stringent size, weight and power (SWAP) constraints of small UAVs [3,4], which can make on board real-time processing of imagery impossible. As computer processors improve and become more compact, the ability to conduct many of the low-level and computationally expensive pre-processing for visual data on board is emerging, presenting a potential solution to the current bandwidth issue presented by the previous methods.

The capability of autonomous navigation by means of a vision-based solution is an area of interest for this class of UAV due to the constraints imposed by payload capacity and cost. In the following sections, several illustrative examples of the need to perform directional, sensor-based navigation are presented. Though these applications may use a variety of sensors, common among them is the integration of relative pose detections and directional navigation of the aerial asset.

1.2.1 An Example: The Adversary's Warship Radar Exploitation

Understanding the weaknesses of the adversary's military forces, and how to exploit them is necessary for military superiority. Radar systems, used as the primary sensor for detection and tracking for many sophisticated weapons systems and military vehicles, are susceptible to exploitation of the geometry of the radar lobes.

For shipboard weapons systems, radar is the primary means of detection, tracking and target localization. A radar antenna will only have a clear field of view (FOV) when no obstructions on the ship in the path of the radiated energy on any relative bearing are blocking the transmission of the electromagnetic energy. Additional impediments, such as the metallic surface in or near the radar beam, will reduce the intensity of the radiated field [5].

Therefore, a target can be detected at a greater (and desired) range on bearings where the field of view is unobstructed than on bearings where metallic masses aboard the ship intercept part of the radiated energy. Almost all radars have areas of reduced coverage, and many have sectors that are completely blind. Since this is a function of the ship's

superstructure and physical location of the radar [5], it is completely unchangeable for a given system installed.

The image below shows the physical constraints on the radar coverage for the *U.S.S. Yorktown* due physical barriers.

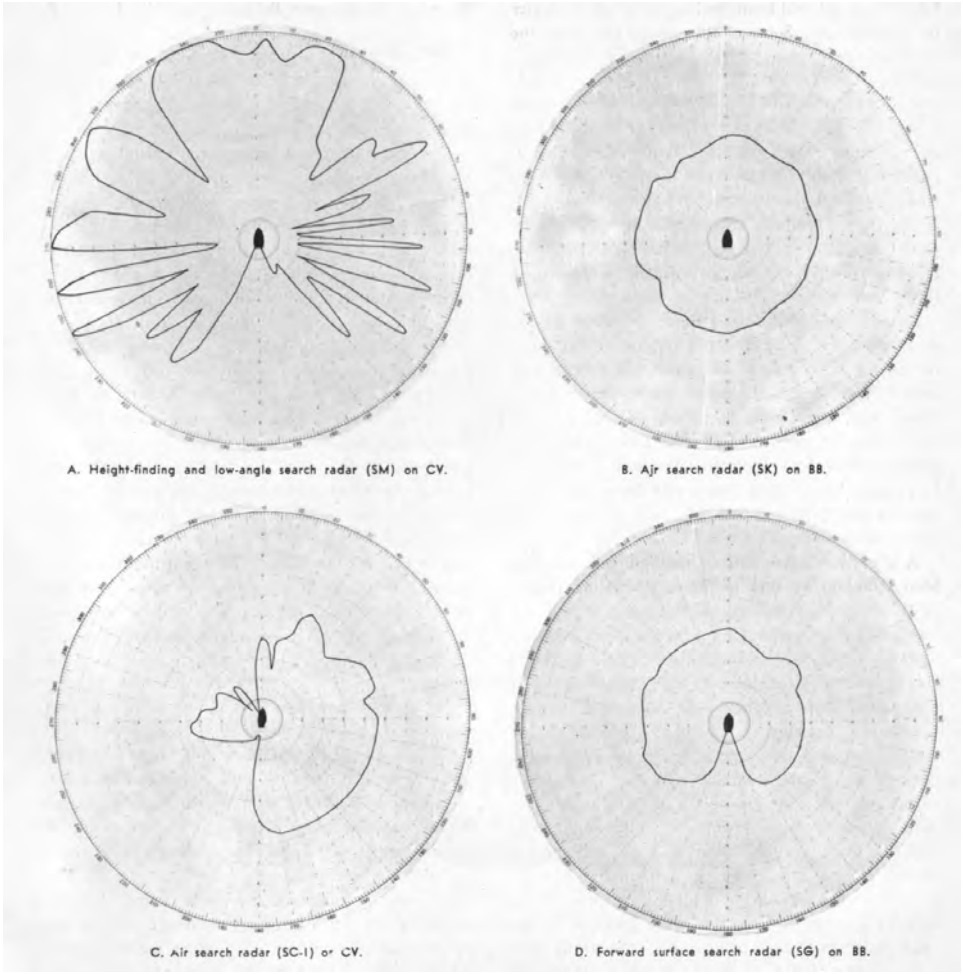


Figure 1.1: *U.S.S. Yorktown* radar coverage, interference due to superstructure. From [6].

Another type of blind zone for radar systems exists as a function of the vertical width of the antenna. This limits the maximum position angle that the radar can "see," and therefore detect targets. The size of this zone is dependent on the characteristics of the antenna itself.

Figure 1-27. Polar plot of antenna gain (in dB) versus azimuth angle for a hypothetical antenna, showing major antenna lobes. The peak and average sidelobe levels (as well as the main lobe antenna gain) are system level parameters that must be specified to the antenna designer.

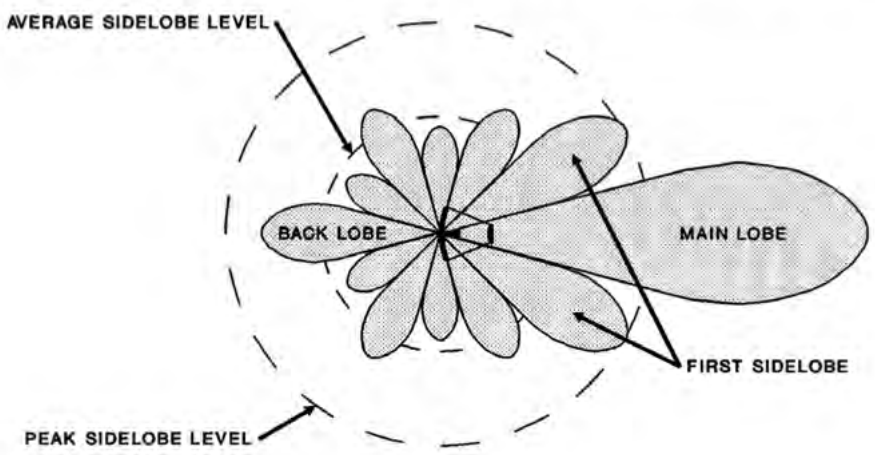


Figure 1.2: Radar geometry, sidelobes and baffles as a result of physical characteristics of the radar system, from [5].

Knowing this geometry, it is conceivable that aerial platforms could fly a profile that will exploit these weaknesses in the coverage by conducting an approach from the baffles of the radar, thus remaining undetected well within the average coverage envelope of the system. This targeting requires a fixed approach relative to the ship's orientation and is independent of the global orientation.

This research aims to use vision-based methods for target localization and intercept. Because the quad rotor (acting as the aerial platform) must pass through a hoop that is set relative to the ground robot (acting as the ship), the approach is limited as a function of approach angle relative to the vehicle, not to the global coordinates. Flight profiles driven by the closest point of approach (CPA) with the required close-in approach due to the fixed opening will be achieved through vision-based targeting.

1.2.2 Another Example: Carrier Landings

Aircraft carrier landings are a unique problem because the vehicle must account for its own position and the position of the landing area (LA) , which is a dynamic surface susceptible to environmental variables in all three space dimensions. Ships are limited in their mo-

tions; they are designed to move forward through the water with limited lateral and astern propulsion capabilities. This is both a blessing and a curse. They are predictable in their movements, with changes in course being gradual and easily identifiable, but they are also unable to compensate for gross error in estimation on the part of the vehicle attempting to land on the surface.

Conventional wisdom and current ship doctrine requires that all landings must be approached from astern of the ship. There are three types of recovery flight paths flown by a fixed-wing aircraft. The recovery class, or case, is determined by weather and background lighting conditions. Regardless of the recovery case, final approach is flown visually. Pilots use various visual markers and instruments to achieve alignment with the landing area [8].

The approach to an aircraft carrier, shown in Figure 1.3, is similar to the problem laid out in this thesis for various reasons. The quad rotor must approach the hoop from a certain angle relative to the platform, not to a global coordinate system. The platform is dynamic, but the constrained range of motion limit the turn radius and maximum velocity.

Recently, major strides in the UAV community have been made in the aircraft carrier aviation world. In August, 2014, the Northrop Grumman-built X-47/B Unmanned Combat Air System (UCAS) demonstrated launch and recovery capability from the flight deck of the *USS Theodore Roosevelt* (CVN 71) [9], shown in Figure 1.4. The UCAS is the precursor to the more capable class of UAVs emerging on the military landscape: the Unmanned Carrier-Launched Surveillance & Strike (UCLASS) aerial vehicle.

1.2.3 Another Example: Mountainous Terrain Search and Rescue

Current military campaigns place ground forces deep into enemy territory over widely variant topography with limited logistical support. A UAV capable of delivering payloads to the ground forces to provide close-in air support (CAS) or supplies would be beneficial. To minimize susceptibility to detection by adversarial forces, a flight profile that limits extended flight time in the mid-airspace is ideal. High altitude flight is advantageous because it is outside the weapons engagement range. A low-altitude flight profile that is close to the ground limits the detection over the horizon but also limits passive localization of the ground forces due to interference with structures. Mountainous missions are especially difficult for a low-altitude flight trajectory due to topography interfering with signal

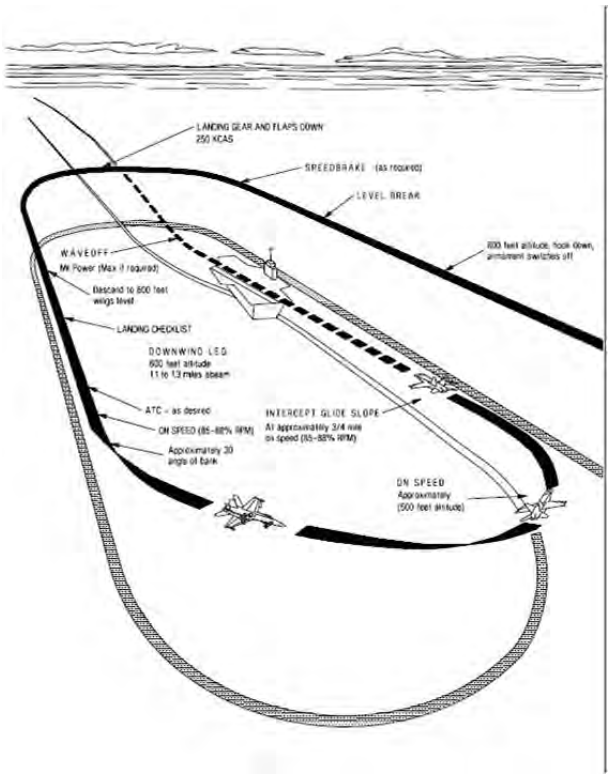


Figure 1.3: Daytime aircraft carrier landing profile, image taken from [7].

strength [10, 11]. Knowing the general operating area (OA) of the ground forces and the topography of the area, an optimized flight profile can be developed to maximize the likelihood of attaining a signal. Once a signal is achieved, the UAV should fly a profile such that it achieves the strongest return signal to reduce likelihood of loss of the signal.

1.3 Background

This section provides a background on the current state of the art vision-based solutions for control and navigation of unmanned platforms.

1.3.1 Fielded Navigation Technologies

The target tracking problem continues to be an area of interest for many research groups across the academic, commercial and military domains. Although the basic concept of identifying and tracking an object of interest remains unchanged, the scope, problem definition, and approaches all vary dramatically from one study to the next. For this reason,

(a) (b)

Figure 1.4: (a) X47/B UCAS and F/A-18 Super Hornet take off from the flight deck of the *USS Roosevelt* on 17 August, 2014 (b) X47/B landing on the *USS Roosevelt* on 17 August, 2014, from [9].

each study contributes something unique to the target localization problem landscape, but that alone is a significant body of research and development.

More narrowly, the number and type of fielded technologies for National Aeronautics and Space Association (NASA), Navy, Army, Air Force, and commercially available drones in computer vision applications are still very broad. Computer vision is rarely used as the sole source of information for navigation, but has been successfully employed as the primary means of gleaning information about the operating environment for navigation and object avoidance.

The proprioceptive sensor suite informs the best approach. In the case of this project, a lightweight quadrotor equipped with two cameras that provide non-overlapping field of views is used, as discussed in Section 2. During the preliminary literature review, many similar projects with various configurations are identified, including stereo vision, optical flow, and red green blue depth (RGB-D) sensors, briefly described in the following sections.

Stereo Vision

Stereo vision examines the relative positions of objects from two vantage points to extract three-dimensional (3D) information from the scene captured in the camera's field of view [12]. This method for localization and mapping has been used successfully in many military

applications. In 2004, NASA's Mars Exploration Rovers used stereo vision with great success to navigate safely through unknown terrain [13]. NASA developed the Grid-based Estimation of Surface Traversability Applied to Local Terrain (GESTALT) system, which relies on other sensors to correct for estimation errors.

Stereo vision is an attractive option for some applications but certainly not all. Traditional stereo imaging, which involves two or more cameras rigidly attached to the air-frame, is limited by the size of the airframe itself. For small airframes or large distances between the target and the airframe, stereo vision loses its effect [14] due to the geometry of the physical camera setup because success relies on a significant difference in the cameras' aspect and angle to the target. Motion stereo vision, developed by NASA primarily as a means to overcome the distance limitation introduced by traditional stereo vision [15], is largely inapplicable in the case of small UAVs because of the limitation imposed by weight and computational power required for the increased complexity of algorithms that are capable of compensating [16].

Optical Flow

Optical flow is induced by the apparent motion between the observer and the environment, measure the change in location over time of each discrete point in an image. Optical flow is often used to aid autonomous navigation in motion model estimation and low level navigational functions. Typical implementations, shown by [17–19], use optical flow for aerial stability and time-to-collision calculations, in addition to other computational methods for primary navigation [20, 21].

Optical flow has been shown to be capable of determining both the rotational quantity of motion and estimated translation [4]. These two capabilities combined create optical flow stereo vision, which leverage one of the previously mentioned advantages of stereo vision.

One advantage of optical flow is the ability to develop a sensor model and motion model from only one sensor, i.e. a single camera. This is an advantage for two reasons. Most drones already come equipped with a camera. Camera technology is rapidly developing, and as a result cameras are being produced that are smaller, lighter and cheaper. Apart from the sensor payload, the second advantage is the reduction in computational complexity that otherwise is increased when multiple sensors are used.

Other Sensor Technologies

A computer receiving both color and depth images may yield improved accuracy in detection, but at the cost of computational load [22]. There are many different sensors available that combine depth information with a two-dimensional (2D) color image. A survey of sensors that integrate 3D with images for localization and mapping can be can be found in [23].

The Microsoft Kinect is a widely used commercially-available example of a depth and color sensor. The Kinect sensor consists of an infrared laser emitter, an infrared camera and an RGB camera that capture red green blue (RGB) images along with per-pixel depth information to produce a combined image in the RGB-D image plane. The inventors describe the measurement of depth as a triangulation process [24]. Experiments conducted by [25] show that the random error of depth ranges from millimeters up to about 4 cm at the maximum range of detection for the Kinect. A summary of applications for these types of sensors can be found in [26].

1.3.2 Vertical Takeoff and Landing (VTOL) Aerial Robots

Traditionally, UAVs have been classified as tactical, VTOL, or endurance [27]. VTOL platforms are often selected for environments where landings and takeoff locations are reduced to small footprints. Unlike traditional fixed-wing aircraft, the VTOL aircraft can takeoff and land vertically and hover in place. VTOL are further divided into two broad categories: helicopter types, defined by a vertical thrust axis, and transitional types, which are capable of VTOL but then transition to a horizontal thrust axis for normal operations [27].

One unique advantage presented by a helicopter is the capacity for omnidirectional flight. Traditional helicopters have one primary rotor that creates vertical thrust. This type of helicopter relies on variable-pitch rotors for control and maneuverability. Other helicopter designs incorporate multiple rotors, and are commonly referred to by the number of rotors used to generate vertical thrust. One category within this design structure features four rotors, and these vehicles are commonly referred to as quadrotors or quadcopters.

Quadrotors are attractive test platforms for control and mapping algorithms in research applications because they offer many of the benefits of a helicopter with increased stabil-

ity and simplicity when scaled down in size. Unlike traditional helicopters, quadrotors typically employ fixed-pitch rotors and rely on differential motor speeds of the separate rotors for vehicle control and maneuverability [28]. This symmetry eliminates the mechanical control linkages required in the variable-pitch rotor that is on a traditional helicopter, which simplifies both the design and maintenance of the vehicle [28].

Vision-Based Navigation for a VTOL Vehicle

Some methods for vision-based navigation are inherently more applicable to helicopter-type platforms than others. Optic flow, which isolates features in the peripherals of a platform traveling at non-zero-velocity, to execute localization and/or mapping algorithms, is less advantageous for a rotor-wing aircraft than for fixed wing aircraft that require constant forward velocity to generate lift. This can be countered by applying optic flow to vertical motions in very specific tasks, such as landings and takeoffs. In [21], a visual control system is successfully implemented on a tethered rotorcraft that provided thrust inputs to minimize downward optic flow.

1.3.3 Vision-based Navigation

Vision-based systems present several inherent benefits in determining vehicle location relative to other objects, specifically for the purpose of navigation. Cameras are readily available, can be low cost, and are often already incorporated into the vehicle design, eliminating the need for extensive set up or modification of the existing system. The required calibration has been standardized and is oftentimes a built-in function through the drivers and software. Additionally, they are passive [29], potentially reducing the error, noise, susceptibility to electronic exploitation and power requirements introduced by two-way propagation.

Standing research has demonstrated that, when combined with other sensors available onboard for basic safety of flight (altimeter, whether via barometric pressure or Global Positioning System (GPS), and approximate location information provided via GPS), vision-based systems offer robust location and localization capabilities with reduced error. There are many examples of these efforts, but some interesting examples are briefly presented. The challenges presented by data uplink for vision-based systems onboard a small and agile aerial platform are shown to be overcome when visual information provided by a camera

is fused with inertial sensors [14]. Both [18,30] achieved improved target estimation performance with non-Gaussian, probabilistic vision information when incorporated with three position components in a particle filtering framework. The problem presented by [31] examines the challenges of a vision-based landing system on an aircraft carrier. Vision-based navigation approaches are also being extended to beyond aerial and terrestrial applications, but also underwater. In [29], the benefits of vision-based systems is demonstrated in an underwater environment when used as a mosaic overlay.

1.4 Scope of Thesis

The work herein addresses target-based approaches that are constrained by sensor detection profiles. The objective is to further investigate the robustness of vision-based solutions for object detection and classification. Additionally, this work explores solutions using open-source middleware to integrate the required hardware and software components. The scope of the work is as follows:

- Develop and integrate vision-based detection methods using the Systems Engineering process
- Investigate the performance of vision-based method using acceptable and understandable standards
- Perform real-time analysis of the object detector and prove feasibility for flight application
- Investigate the performance of the middleware to support software drivers and corresponding hardware components

1.4.1 Limitations

Computer vision is inherently limited. Camera sensors are sensitive to changes in ambient lighting and environmental visibility. Additionally, the sensor is susceptible to blurring or noise in the image plane as a result of jitter or other sudden movements while the image is captured.

As with any medium that involves reducing the "dimensionality" (dimension property) of the object, there are losses that cannot be fully recovered. Computer vision is the act of reducing 3D objects in the object space into 2D representations of the objects in the

image space (plane). This very act requires a certain tolerance for lost information. Further reductions occur as a function of resolution and range (distance) of the sensor to the target and/or objects in the camera field of view.

The platform used for this research is chosen due to availability and hardware/software support. The payload capacity, sensor capability and platform endurance and physical flight are unaltered and are limited by the chosen test platform. Therefore, the results of all experiments may be applied specifically only to this platform and other vehicles with similar characteristics. Broader application of the research may be applied to directional search and targeting solutions.

1.5 Main Contributions

The main contributions of this thesis include the moving object tracker design and its experimental evaluation, presented in Chapters 3 and 4, and the demonstration of possible improvements with underlying software architecture for future works, presented in Chapter 5.

1.5.1 Moving Object Tracker

This work addresses two key challenges for single robot-based object tracking: how to isolate the object of interest using computer vision techniques, and the design of a probabilistic filter to localize the object in the robot FOV. The proposed method is implemented and tested in a light-controlled indoor environment with varied object placement in the arena to ensure a robust solution. A ground-truthing system is utilized to measure the accuracy of the detection and classification software on a frame-by-frame basis for objective performance analysis.

1.5.2 Demonstration of Improvements

The system designed is limited in application and accuracy. However, the software architecture is highly adaptive and individual functions can be easily isolated and modified for further improvements. This is achieved by using standard system architecture, introduced and discussed in Chapter 2. Suggestions for further developments and improvements are provided in Chapter 5.

1.6 Organization of the Thesis

Having introduced the problem and identified related works, the remainder of the thesis is organized as follows. Chapter 2 provides the model formulation and construction, including descriptions of the proposed system architecture and experimental setup. System integration and implementation is presented in Chapter 3, followed by presentation and analysis of system performance and results in Chapter 4. Chapter 5 summarizes the presented work with discussion of impact and recommendations, as well as identification of further avenues of future work.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2:
System Architecture and Model Formulation

The motivation for the design of the scenario is based on military scenarios. Using the vignettes described in Section 1.2, a model is developed for simulation and experimentation. The presented work is based on a particular UAV platform available in the Naval Postgraduate School (NPS) Advanced Robotic Systems Engineering Laboratory (ARSENL). The presented work aims to establish a general methodology, which may be useful for future works utilizing platforms with similar control and payload capabilities.

2.1 Laboratory Scenario Description

The scenario explored involves an aerial chaser and a surface target. The ground target's detection ranges are assumed to be consistent with the radar detection profiles on a ship. This is simulated as a $1m \times 2m$ rectangular hoop vertically mounted on the superstructure of the ground robot at a height of $1.5m$, shown in Figure 2.1.

Figure 2.1: Rectangular hoop vertically mounted on the Clearpath Robotics Husky

For experimental purposes, the origin of the operating area is a pre-defined location at $0m$ elevation. The position of both the aerial chaser and the ground robot target are calculated

relative to the origin. The standard orientation with the right-hand rule is used throughout for computation.

The aerial chaser detects the hoop using computer vision-based methods, determines the approach angle and speed required to pass through the hoop (e.g., to satisfy kill criteria), and executes the commands. The complexity of the problem can be increased by implementing a run profile of the ground robot with various maneuvering strategies through speed and course changes. Initial run profiles for this foundational study involve a stationary target located at the approximate center of the arena ($0m \times 0m \times 1.5m$ from the origin).

2.1.1 The Agent: Parrot AR.Drone

To be effective, the chaser robot must exceed both maneuverability and speed of the ground robot. An aerial robot is an obvious choice, given the performance capabilities in both areas of interest. Fixed wing and helicopter designs offer different advantages. Fixed wing aircraft rely upon the forward flow of air over the cambered surface of the wings to generate lift. For a helicopter, lift is generated by air flow across the surface of the propeller blades, allowing it the distinct advantage of hovering and omnidirectional flight capabilities. In the remote control and hobbyist arena, the quadrotor is a natural follow-on to single-rotor craft. Similar to a traditional helicopter, the design features four horizontal blades surrounding a central body. The symmetry allows increased stability while reducing design complexity, making it ideal for research where control theory is not the main objective.

The Parrot AR.Drone, pictured in Figure 2.2, is a small quadrotor vehicle used for general research by the Systems Engineering Department of the NPS. The vehicle is manufactured by Parrot, a Paris-based company, initially designed as a consumer product for augmented reality applications such as video games [32].

The AR.Drone is advantageous for use in research because it is a robust, stable, low cost vehicle that is widely available commercially. Its popularity further boosts the company's resources for technical support, both through the vendors and the academic robotics community at large. The onboard processor and autopilot developed by Parrot employs proprietary algorithms that remains hidden from the user to ensure ease of operation for the casual hobbyist. This is useful as it eliminates the need for integrating a sophisticated flight

Figure 2.2: Top-down view of the Parrot AR.Drone, IR markers outlined in red for emphasis

control loop, making it easy to use and feasible to be modeled as a stable platform. The AR.Drone is driven by four single-blade propellers. Maneuvering is afforded by differential thrusts across each of the four rotors, and is controlled by varying individual rotor revolutions per minute (RPM). The vehicle has a forward-facing wide angle camera, a downward-facing camera, a sonar height sensor, and an onboard computer processing unit (CPU) running proprietary software for communication, low-level flight control and command handling [33].

For this research, no modifications are made to the stock sensor suite. After initial data collection, the image quality is considered satisfactory for desired research and goals set out by this thesis. Future works discusses potential improvements to the sensor suite and stock driver.

2.1.2 The Setup

The arena is approximately $15m \times 10m \times 10m$. Ambient light conditions are controlled by the overhead lighting system and remain fixed. A motion capture system is used to provide relative and absolute position ground truth data for measuring detection and classification performance. Specifically, the physical Vicon motion capture system comprises two types

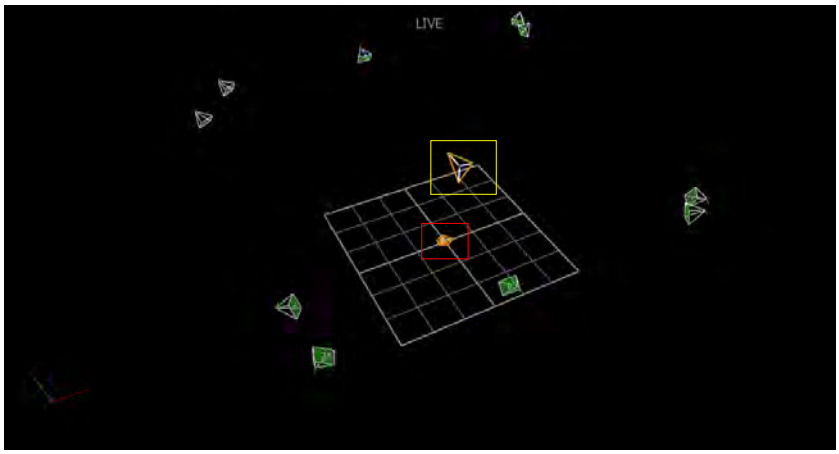
of components: ten “T-Series” cameras outfitted with infrared (IR) optical filters, shown in Figure 2.3), and an array of IR LEDs.

Figure 2.3: Image of one Vicon IR camera, image taken from the official Vicon webpage, from [34]

The software used, called Vicon Tracker 1.3, reconstructs the 3D representation of the markers from the images taken by all cameras with reference to a pre-determined origin, which is set by the user. A screen shot of the Graphic User Interface (GUI) showing the target and one of the AR.Drones being identified and tracked by the Vicon system is shown in Figure 2.4.

Vicon recognizes each object in the arena by identifying the unique constellation of (no fewer than three) IR markers that have been designated a priori to each object. Five reflective markers are placed in a unique pattern for each AR.Drone housing, shown in Figure 2.2. Three markers are used for the stationary target, shown in Figure 2.5.

The Vicon system relays the target and aerial chaser data in quaternion angles and vectors [34]. Quaternions are useful in robotics for eliminating singularities (e.g., gimbal lock) that are sometimes present with Euler angles in very complicated motions. However, due to the physical constraints imposed on the flight profile of the agent, it is highly unlikely that such an instance may occur in this research. For simplicity, these angles are converted to Euler prior to analysis of data. When properly calibrated, the Vicon system can provide an accurate measurement of spatial coordinates and position. Included in appendix B is a quick-start guide and introduction to the Vicon System. More details about Vicon and its technical specifications are found in [34].



(a)

(b)

Figure 2.4: The Vicon arena with target and aerial chaser. (a) Screenshot of the GUI for Vicon Tracker 1.3, showing the camera coverage of the ten IR cameras surrounding the arena. Within the arena are the constellations of the Parrot AR.Drone (shown sitting at the origin, outlined in red) and the target (shown offset and outlined in yellow) in the Vicon arena. The origin is set and calibrated by the user, with orientation indicated in the lower right corner. (b) Photo of the arena with same layout. Cameras are shown outlined in green and the arena corresponding coverage to the GUI is outlined in white

2.1.3 Challenges in the Model

There are inherent challenges with any model that implements localization without the presence of a ground truthing system, such as GPS. Camera image quality, which is, for

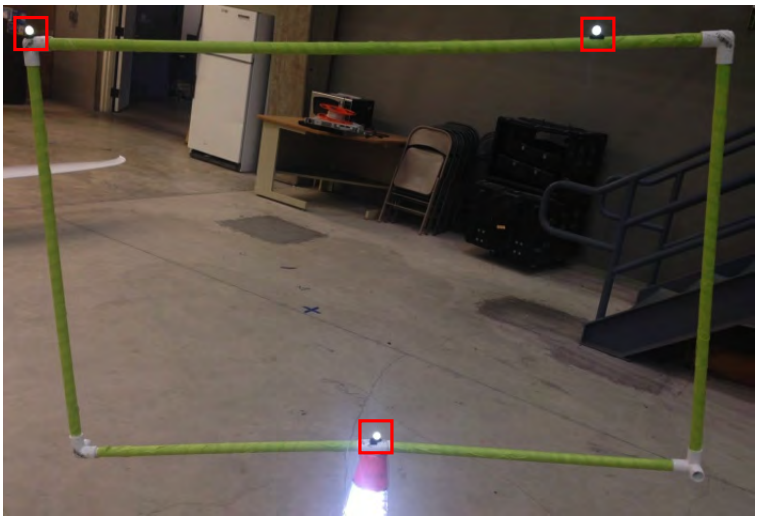


Figure 2.5: Head-on view of the target, IR markers outlined in red for emphasis

example, dependent on the resolution of the camera feed, may have an impact on the accuracy of the metric data. Inadequate lighting and contrast can impede the recognition of the target or may result in false targets. Own-ship knowledge of position and velocity is available from onboard proproiceptive sensors, but errors in the estimation of self state may be additive in nature when combined with target estimation in reference to the global reference grid.

Many computer-vision detectors use a pre-processing method or pre-filtering stage. Common pre-processing methods include blurring or background subtraction. Background subtraction is typically accomplished by removing the mean image color and intensity of the background over frame averaging from a number of subsequent frames. A moving sensor, such as the one used in this problem, creates apparent motion from one frame to the next, making background subtraction especially difficult and less likely to provide a reliable solution. In the event that the images are used to identify a target in a sky, background subtraction may provide a good avenue for pre-processing.

2.2 Functional Architecture

Functions occur across all the software and hardware used in the system. The main system functions fall into seven categories:

- Mission (High-level)
- Sensor
- Perceptor
- Mapping
- Estimation
- Graphics user interface
- Planning

This functional flow is shown in Figure 2.6 to illustrate the way the information interacts and crosses boundaries.

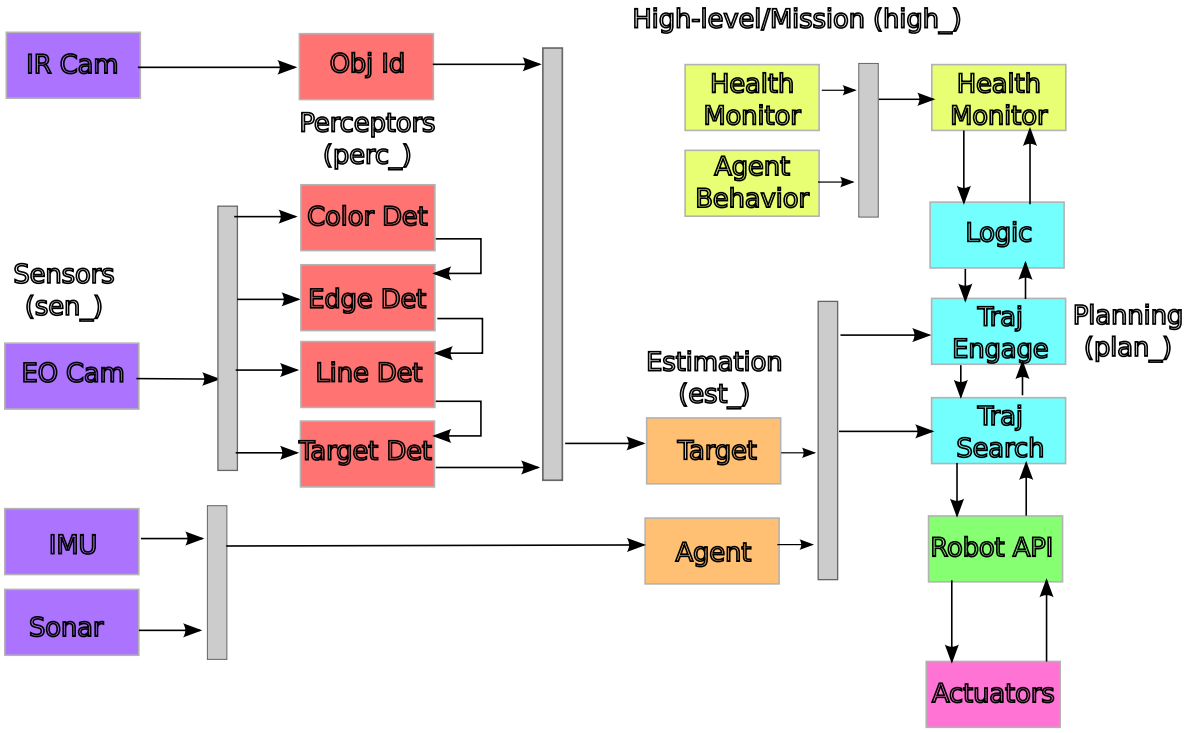


Figure 2.6: Software functional flow diagram

2.2.1 Mission

The mission encompasses all the high-level processes that occur for all physical components and software interface. All functions in Robot Operating System (ROS) that fall under this domain have the prefix "high_" preceding the name of the node to denote high-level behaviors or functionality. Functions that occur within the mission domain include:

- Health monitor: Maintains the high-level state awareness of all physical components and connectivity to the system. Monitors the Vicon system for connectivity and agent detection, WiFi for connectivity and port access, and the AR.Drone for receiving and delivering of messages across the network;
- Agent behavior: Maintains the AR.Drone state machine, such as battery power, internal connectivity and communications linkage. This occurs within the AR.Drone driver. Triggers prompt the AR.Drone to take action such as emergency land, automatically shut down, or publish an error through the user interface.

2.2.2 Sensor

The sensor domain is the interface of all sensors with the environment. In this project, most of these functions occur within the driver and proprietary software of each component. There are many more sensors that are important for the system that are not investigated, and are not discussed. In ROS, the prefix "sen_" precedes these functions to indicate they are sensor functions.

- Camera: The forward-looking and down-looking cameras onboard the agent are turned on and operate when the agent is powered on. The AR.Drone receives images with a resolution of 320 $\times$ 240 pixels from the forward-facing camera, and 88 $\times$ 72 pixels from the downward-facing camera [33, 10]. The forward-facing camera has a FOV of 92°. Camera selection is dependent on the scene content and the detection of the target in the FOV. This thesis focuses on the use of this forward facing camera.
- IR camera: The Vicon system uses IR cameras for object detection. There are ten IR cameras that are used in the system. Further description of the Vicon system is provided in Section 2.3.3.

2.2.3 Perceptor

The majority of the work for this thesis occurs in the perceptor domain. Raw data from the sensor domain is ingested to produce usable information. In ROS, the prefix “perc_” denotes these functions.

- Color detector: Streamed images are filtered for specific hues or colors. Rejected regions are removed and accepted regions are passed through for further data processing.
- Edge detector: Various methods are used to determine object boundaries within the image plane.
- Line detector: Lines within the image plane are isolated and characterized. The line detector takes edge filtered images and outputs an array of lines.
- Target detector: Outputs from the lower-level detectors are used to determine if criteria have been met to have positive identification of the target in the camera view. If a target is detected, the output of this function is the centroid and endpoints of the target in the image plane.
- Object identifier: In Vicon, known objects have fixed infrared reflectors in predetermined constellations. The software takes the IR camera feed and searches for expected constellations within the arena. If detected, the object is published by its name, as designated in the program. Vicon software updates the names of the detected constellations at 200 hertz (Hz).

2.2.4 Estimation

State estimation for the target and the agent is computed as new information comes in. Each function name follows the standard naming convention, which is preceded by "est_".

- Agent: State estimation for the agent is generated primarily using the proprioceptive sensors, including Inertial Measurement Unit (IMU) and ultrasonic range altimeter, standard with the AR.Drone. The inputs for the agent estimation are generated through the proprietary software driver and are not manipulated. Location and pose information from Vicon is also used to provide ground truth and eliminate any error within the agent's onboard close-loop control system.
- Target: Relative location and pose of the target is generated using the centroid and endpoints of the target in the image plane. Ground truth is provided from Vicon inputs.

2.2.5 Planning

The planning domain produces actionable commands for implementation by the agent. Perceptive and estimation outputs are used as the inputs in the planning domain. The naming convention for all functions within this domain is prefix "plan_."

- Logic: Logic acts as a switch to enable and disable certain control mechanisms, behaviors, and other executables within the planning code. It determines which sequence of functions to call.
- Engagement trajectory: Computes the agent's flight control and path for target intercept. Inputs the target state estimation information and the agent state estimation information and outputs flight commands for the agent.
- Search trajectory: Computes the agent's flight control and path when there is no detected target. Inputs last known target state estimation information, if available, and agent state information, and outputs agent behavior commands.

2.3 Physical Architecture

There are four major physical components for this project, the agent (AR.Drone), the target, the arena and the host station that acts as the central processing station.

2.3.1 Agent

The AR.Drone quadrotor, described in Section 2.2, comes equipped with stock sensors that are built-in to the driver. The following components are on the quadrotor:

- Front-facing camera
- Downward-facing camera
- Wi-Fi
- 3-axis gyroscope
- 3-axis accelerometer
- 3-axis magnetometer
- Pressure sensor
- Ultrasound sensors

With the exception of the front-facing camera, all these sensors are used to automatically control and stabilize the AR.Drone through flight.

2.3.2 Target

The target is stationary and remains within the arena. The target dimensions are assumed to be known for the problem laid out in this thesis.

2.3.3 Vicon System Arena

The arena is treated as an ellispoid in figures and graphic representations of the experiments, but the boundaries of the Vicon arena are highly variable both in the horizontal and vertical axes due to camera coverage geometries. Due to this variability, all datapoints taken when the quadrotor is outside the Vicon boundaries are discared during post processing. Only datapoints for which both the quadrotor's and the target's positions are reasonably known are included in the analysis and metrics for success. For this reason, the maximum ranges taken to measure detectibility are limited by the arena and not by the sensor.

2.3.4 Host Station

The remote operating station is a laptop computer running Linux Ubuntu 12.04. This host machine serves as the processing muscle for the computationally expensive computer vision algorithms. Commands and images are exchanged via a WiFi ad-hoc connection between the host machine and the AR.Drone. The connection with the Vicon system is hardwired in through a network cable.

2.4 Software

Proprietary software for each hardware component is wrapped in ROS to provide standard interfaces between components. Vicon and AR.Drone both have software drivers written in the open source community that serve as the bridge between the base software and the middleware. These software drivers are written in C++ and Python programming languages. ROS (version Groovy) is used for this project.

The custom computer vision code is written in Python programming language (version 2.7). Software development for Python presented here is done with Linux Ubuntu 12.04. Python is chosen because it provides a robust way to interact with the hardware components using free distributions and libraries, in contrast to the commercial software licenses required, for example, by Matlab.

In addition to Python implementation, Open Source Computer Vision (OpenCV) (version 2.4.8) software package is used [35]. ROS has an existing wrapper library that is used to convert OpenCV data types to ROS data types. This library in addition to OpenCV's Python bindings, is used for all results rendered that are pushed across the different interfaces.

Results are presented in graphical form using Matlab 2012b. Unlike Python and OpenCV, Matlab requires a license for use. The benefit Matlab provides to this project is the ease of generating sophisticated graphical oututs. Matlab is not fully integrated for the ROS version used in this project, but new interfaces are available between ROS and Matlab in more recent versions of ROS.

2.4.1 Robot Operating System (ROS)

ROS is open-source middleware originally maintained by Willow Garage [36] and now managed by the Open Source Robotics Foundation [37]. It provides a framework for finding, building and implementing control algorithms and code for this project [36]. ROS is a powerful tool because of the way it treats the interactions of data within the modular system, which in turn simplifies the transition between onboard and external computation. It modularizes the code in a way that removes the lower-level interactions, such as drivers and servers, from the user interface and places emphasis on task-oriented architecture [38]. This allows for increased flexibility in ways that are previously unattainable. Operation within ROS is built upon executable applications called "nodes", and data is transferred between nodes by the use of "messages".

ROSbag File Program ROS provides a way to record all data for replaying as if in real time through a unique file format called bag files. Bags are the primary mechanism in ROS for data logging, and allow the user to record datasets, then visualize, label them, and store them for future use [39]. There are several methods in the open-source community for handling `.bag` files, and the one used in this research is the rosbag package. The rosbag package provides a command-line tool for and interface creating and reading bag files in the Python programming language [40].

2.4.2 Robot Application Program Interface (API)

The robot API interfaces directly with the actuators physically on the robots. For this project, the API is embedded in the proprietary software and driver and is not modified. ROS allows common formats to be called using naming conventions that are standardized in the community. For simplicity and ease of use, existing API for the AR.Drone are leveraged in this project.

- Agent - Interfacing directly with the AR.Drone is the driver, which is responsible for converting ROS messages to and from Parrot's onboard computer processing unit. It manages low-level health and raw sensor messages from the AR.Drone and converts them into standard ROS messages that can easily be used and understood by the user. For example, the ROS driver ingests user messages that define the desired linear and angular velocities and converts them to discrete rotor spin rate commands. The driver for this agent takes the commands from the planning domain as inputs and outputs commands to hardware components on the robot.
- Vicon - The vicon API is the standalone machine that runs the program and all the components. It provides output of the pose estimation but no inputs from other nodes.

2.5 Computer Vision and Perception

Control of a robot via computer vision uses computer hardware to look for features of interest (FOI) within the FOV. There are a number of common algorithms and approaches which can be used, and different applications may call for different methodologies, which are briefly reviewed in this section.

2.5.1 Image Processing Background

Computer image processing occurs on different levels of the image. Examples of the type of feature that can be identified by common computer vision algorithms include edge detection, contour and corner identification, and template matching [35]. These approaches are considered lower and intermediate level operations, which typically occur on a pixel-level. More advanced algorithms utilize descriptors, which are features that are unique in the image space and are typically comprised of a large number of low-level features combined to create a robust feature space.

Pixel Operations

Most of the processes to extract key points transform the image from color intensity to gradients of intensity, typically on a pixel level. Features detected on a pixel level can be extremely useful when used properly. However, they are highly susceptible to changes in- and out-of-plane.

Model-Based Approach

Template matching is a simple concept but can be computationally expensive to execute in real-time. The object of interest is assumed known and is represented in an image patch or "template." The template is then convolved across images searching for the least difference or best match. Depending on the pre-processing steps taken and method used to compute the difference, template matching can be robust to changes in saturation and illumination and in-plane rotation and translation, but the overall effectiveness of the template is largely dependent on the uniqueness of the feature used in the template [41].

One method used to reduce the number of operations is the pyramid method. For each image searched, a number "n" of reduced-scale representations of the image are created using a fixed reduction ratio. This results in a multi-dimensional data structure similar to a pyramid in concept, with the image represented in a sequence of copies with decreasing pixel dimensions, where the nth-scaled image is represented in the nth matrix array. The template is held at a fixed size and convolved with each image to locate the best match across the entire array. This method is equivalent to varying the template size and convolving each template against the image of a set resolution. The benefit is that this approach offers the same effect with an exponential decrease in computations [42].

Classifiers

The purpose of a classifier algorithm is to be able to reliably identify objects in a noisy image in a way that is robust to in-plane rotation, scale, lighting and out-of-plane rotation. Many of the classifiers investigated for this project are built upon pixel-level operations, such as the Harris corner detector [43] or Canny edge detector. One such example is Features from Accelerated Segment Test (FAST), which are built upon Harris and proved to be more robust to invariants than previous methods using Harris at the time of publication, in 2006 [44]. Some other feature detectors/descriptors available in computer vision are

Speeded Up Robust Features (SURF) [4] and Scale-Invariant Transform Feature (SIFT) [45].

For the simple target design used in this project, a feature detector may not provide more information than provided by a simple color or edge detector. Additionally, many feature detectors (such as SIFT) are computationally intensive and the latency experienced during real-time application makes them ineffective [44]. Therefore, the use of SURF and SIFT is left to future investigation.

2.5.2 Computer Vision Techniques

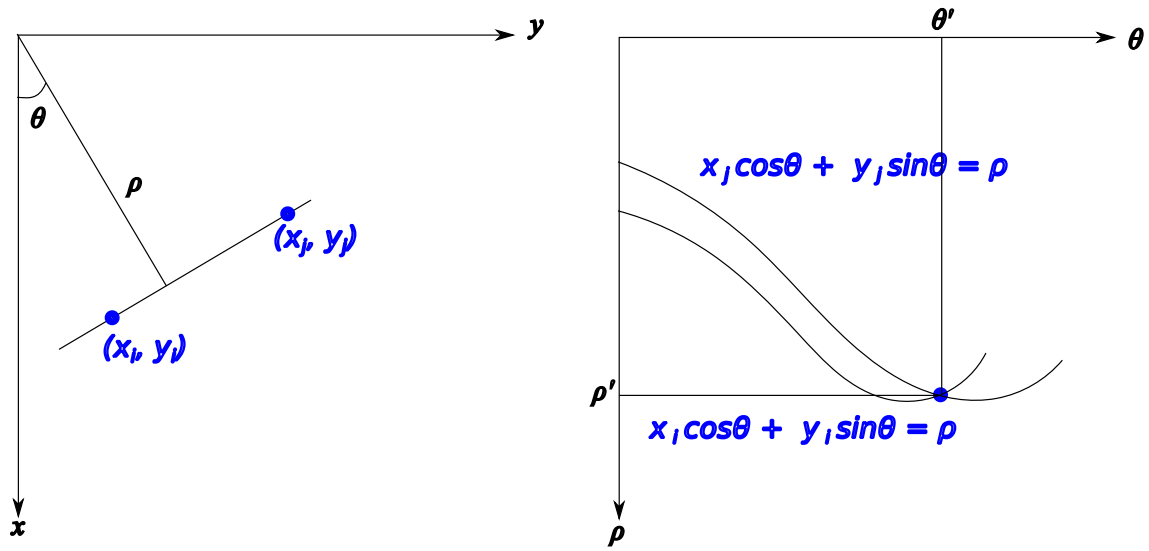
This project uses open-source OpenCV libraries to conduct image processing. OpenCV is a function library that provides many computer vision operations that can be called with real-time results [35]. It is compatible with both in C++ and Python [35]. In this thesis, the target's color is used to achieve recognition criteria. There are many methods that have been developed for target recognition using color-based criteria. Color detection and description are then explored using edge detection methods and region segmentation methods.

Edge Detector

Common edge detectors such as the Sobel and Canny methods use convolution masks to identify gradients or boundaries of edges [41]. The specific detector is defined by the thresholding criteria used to determine an edge. If the threshold criteria is met, a pixel is considered a boundary pixel in the image plane and remains a binary value (1) [35]. Otherwise, the pixel is rejected and given a binary value (0).

Hough Line Detector

Lines in an image plane can be expressed using two variables in the polar coordinate system: a distance (ρ) and an angle (θ) from the reference point in the image plane to the normal intersection with the line [41], given by the relation, $y = -\frac{\cos\theta}{\sin\theta}x + \frac{\rho}{\sin\theta}$. This conversion is shown in Figure 2.7. In the $xy-$plane, two points on a given line are expressed by their (x_i, y_i) and (x_j, y_j) coordinates. This these two lines can then be expressed as the intersection of two sinusoidal lines with the expressions $x_i\cos\theta + y_i\sin\theta = \rho$ and $x_j\cos\theta + y_j\sin\theta = \rho$.



(a) (b)

Figure 2.7: Parameterization of lines in the $xy-$plane. (a) $(\rho, \theta$ can be expressed in the $xy-$plane and then translated to their corresponding (b) sinusoidal curves in the $\rho\theta-$plane, with the points of intersection being (ρ', θ'), which correspond to the parameters of the line intersecting (x_i, y_i) and (x_j, y_j), after [41].

For each point in the image plane that is thresholded to be an edge, there are a finite number of lines that the point may fall on, which when plotted turns out to be a trigonometric relationship between ρ and θ. For a pure Hough Transform, every edge point is plotted and each point that exceeds the threshold number of registered "hits" is considered a line [46].

The Probabilistic Hough Line Detector uses Hough Lines but reduces the computational load further by only sampling the image space to populate the θ vs ρ plot [35] shown in Figure 2.7(b). Section 3.3.5 provides further detail of this method.

2.5.3 Color Models

Color images are represented in different ways. Images can be segmented by color planes, commonly referred to as the RGB or blue green red (BGR) color models, and also in hue saturation value (HSV), and hue-in-saturation 3 (HIS3). The camera physically uses one specific color plane but images can be converted into any other method using software.

The Color Plane Method

The color plane model is an additive color model in which different color lights are summed (with scaled weighting) to produce a comprehensive array of colors. Most commonly, the camera decomposes each pixel in an image into three primary colors: red, green and blue. Secondary colors of yellow, cyan and magenta are produced by equal parts of two of the three colors.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3:
System Development and Integration

As described in Chapter 2, the target tracking algorithm for an individual robot is decoupled into two phases: target identification and target localization. This chapter describes the problem statement and specific solution for the robot tracking algorithm as a foundation for the system architecture in a Systems Engineering framework.

3.1 Methodology

The Systems Engineering method is implemented for this project. Because it is results driven, the Systems Engineering approach is useful because it provides a discipline to facilitate a functioning product and then allows for further improvements to the existing structure. The project methodology, risk assessment, time line are all considered when planning and executing the research herein.

3.1.1 Systems Engineering: The Vee Model Approach

A structured and articulate approach to research, experimentation and testing will assist in meeting time lines and achieving goals. To this end, the Systems Engineering approach is adopted, using the "Vee" Model for project development and experimental phases. The Vee Model, presented in Figure 3.1, so-called due to the way it graphically presents the design and integration process, provides a top-to-bottom-to-top approach to design and development of a system. The left side steps through the decomposition process, where top-level end-user desires are decomposed and mapped to function and design specifications. As the process approaches the documentation and inspection plan phase, the process enters the "Design Engineering" phase, shown in Figure 3.1 as the three bottom blocks in the Vee. The bottom of the Vee is the build and assemble phase. The right side of the Vee Model steps back to perform verification and validation on the component, sub-system, and system level [47].

For this project, the Vee model was modified to meet the scope of this project. The adaptation for this project produces a method that is as follows:

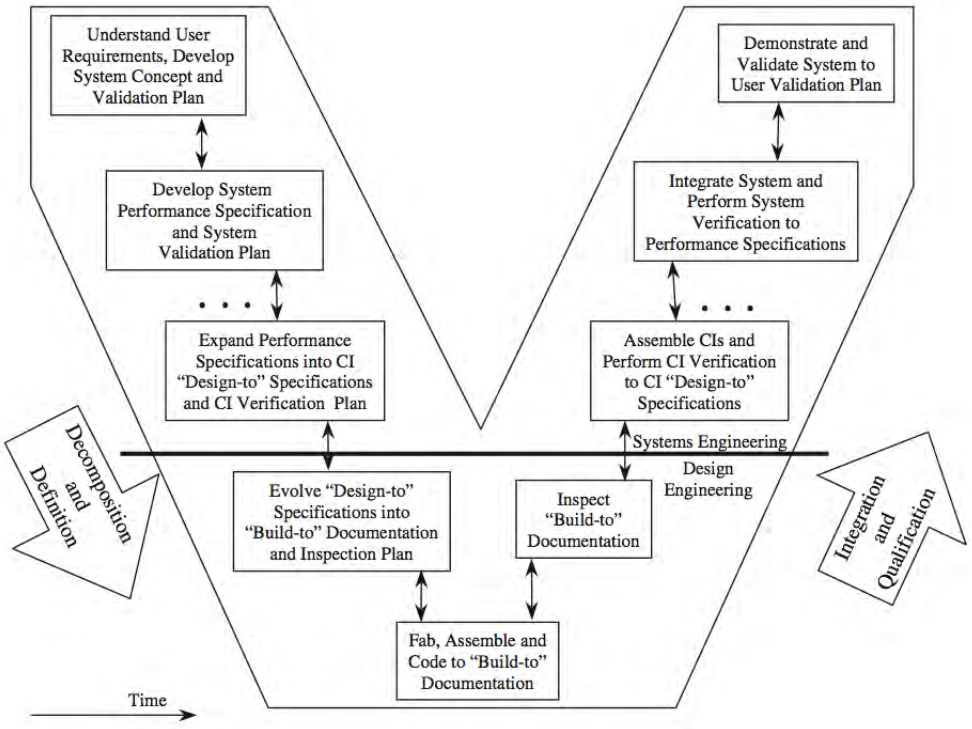


Figure 3.1: The traditional Systems Engineering approach begins with the Vee Model, from [47].

- Conduct a survey on the existing body of research in the field of vision-based solutions for navigation and localization. This is addressed in Chapter 2
- Determine the objectives of the project. Use vignettes, operational concepts and other methodologies to determine system boundaries and functional analysis to establish the effective need. Chapter 1 discusses this in further depth.
- Develop the software and hardware components to deliver the objective, described in Section 3.3.
- Determine the desired objectives and thresholds to be met. This is addressed in Section 3.4. Conduct analysis of alternatives and develop software and hardware modifications as time permits to improve the product.
- Develop the scenario for experimentation, addressed in Section 2.1. Consider the target characteristics and construction, the arena, environmental conditions and obstacles or other extraneous interferences. Identify the prerequisite infrastructure re-

quired for the experiment. Auxiliary equipment, measuring devices and any other subordinate equipment shall be considered.

- Develop the criteria for success and develop experiment, described in greater detail in Section 3.3.
- Execute experiments, analyze and compile results. This process is addressed in Section 4.1 and the results are discussed in Section 4.2.

3.2 Perception Algorithm Implementation

The perception algorithm for this project focuses on the ability of the computer vision software to detect and classify the object of interest. For this project, a specific object of interest is designated ahead of time as the "target" with known qualities and features. The challenge presented is to use real-time capable vision-based algorithms to isolate and identify the object of interest in a dynamic environment.

3.2.1 Target Characteristics

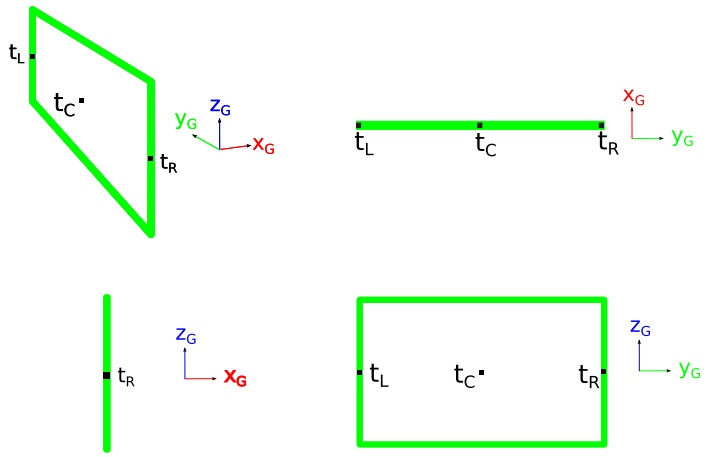


Figure 3.2: View of the target, Clockwise from Top Left: 3D side view, 3D top-down view, 2D side view, 2D front view

The target is a $1m \times 2m$ rigid frame constructed out of 1.5" diameter cylindrical PVC piping. The target is vertically mounted to orient the plane formed by the four sides of the frame normal to the AR.Drone's forward-looking camera's field of view, shown graphically in Figure 3.2. Matte tape covers the PVC piping to provide color contrast from the

surroundings. For preliminary investigation, the target is stationary during each experimental run, but the location in the arena is changed to different locations within the arena to provide different background challenges and additional opportunities for varied detection ranges and geometries.

Color, Contrast in Computer Vision

Different variations of colors were used. Initial trials were conducted with a white frame. Blue and green colors were introduced to enhance contrast against the background, shown in Figure 3.3. After analyzing the camera frames from the three colors, the green frame is selected due to the highest level of contrast with the background.

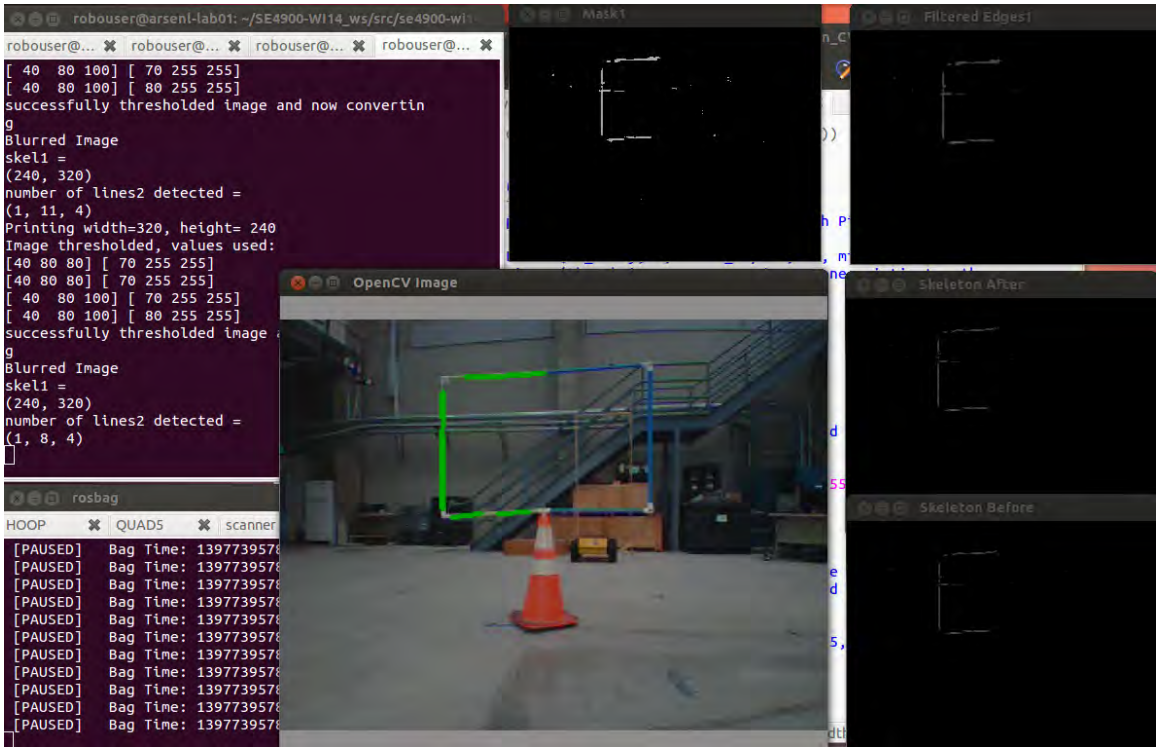
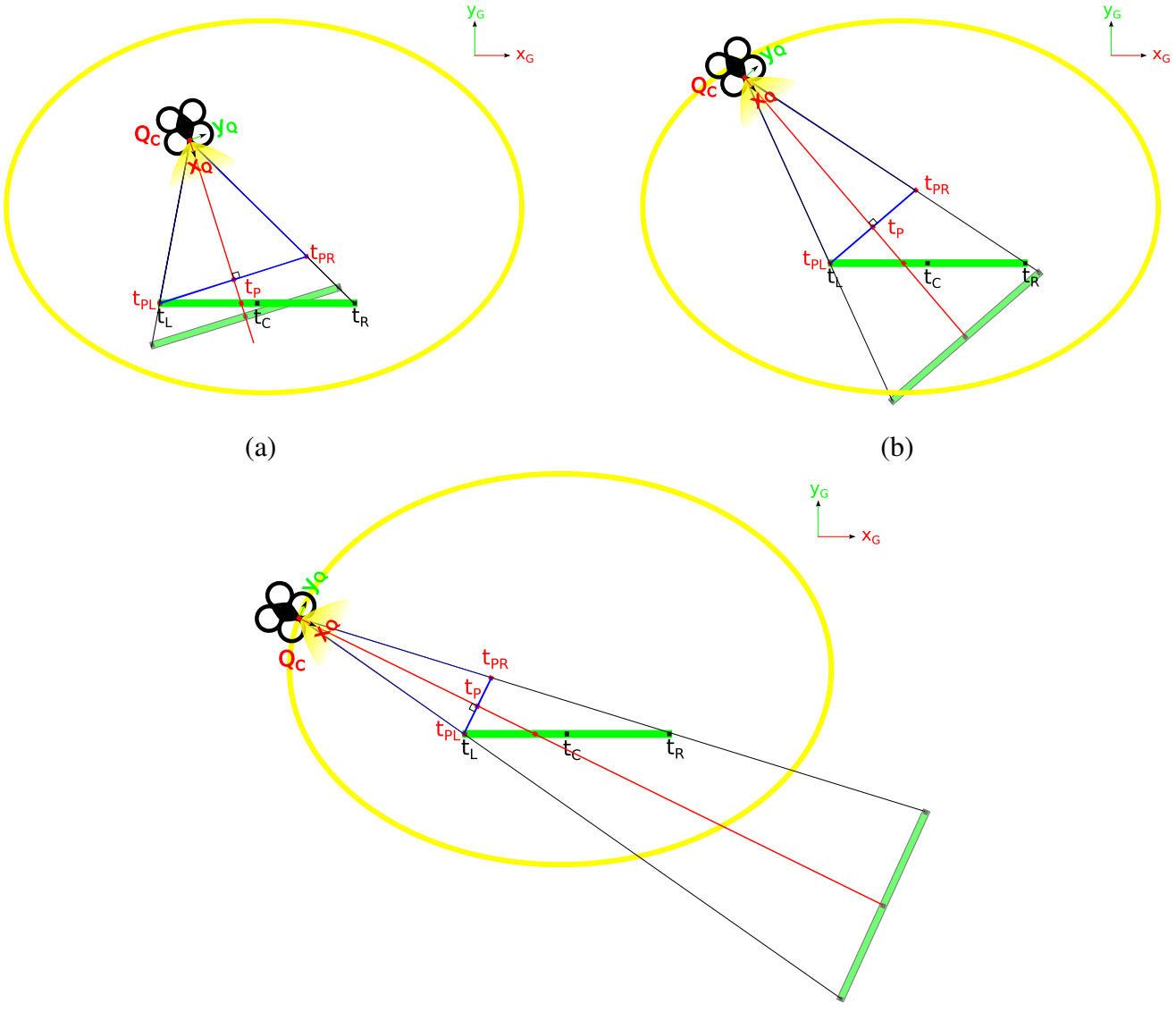


Figure 3.3: Target with blue and green frame. The green portion of the frame has higher contrast with the surroundings and is detected more easily. Clockwise from top left: (a) The algorithm provides cues to the user as it executes the color-vision algorithm. (b) The "Mask" shows the background eliminated by the color thresholding. The image remaining from the background elimination is passed through a Canny Edge Detector (c) and a Skeletonization algorithm (d&e). The edges are then passed through a Probabilistic Hough Transform and the output is overlayed on the original OpenCV image (f). (g) Timestamps are shown for the rosbag replay.

3.2.2 Implementation of Computer Vision Techniques

In this scenario, the target's centroid position is assumed unknown. Pose estimation and tracking is accomplished using only the information that can be gleaned from the agent's sensor suite, specifically, its onboard camera.



(a) (b)

(c)

Figure 3.4: Camera field of view and perceived target center and range

The camera provides the angular size of the target without reference to the centroid of the target. This gives a partial solution for the location of the target. Unless the target orientation is completely orthogonal to the camera's bearing, shown as (x_Q) in Figure 3.4, the target will be closer than it appears in the view.

The more obtuse the incident angle between the camera and the target plane, the smaller the target appears and the more inaccurate the target localization estimate becomes when based on the size of the target alone. Other methods must be explored to extrapolate the location of the target to determine the proper intercept trajectory, the determination of which is a goal for the sensor-based navigation applications presented in this thesis.

3.3 Description of Software Implementation

The core of the computer vision code is written in three Python scripts that are accessed as nodes in ROS. Modifications were made throughout the code to produce improved results, but the order of operations remained constant. First, the video frame is selected. Second, pre-processing operators are performed on the raw image. The color map of the image defaults to BGR and must be converted if a different color map is desired. Noise in the image is removed using a filter. Third, the background is removed using a series of color filters. Fourth, edge detection software is applied to find the boundaries of the object of interest. Fifth and finally, a transformation matrix is applied to localize the target relative to the camera.

3.3.1 Video Frame Selection

The AR.Drone front-facing camera captures images at a maximum rate of 30 frames per second (FPS) [33]. The camera footage is streamed to the host machine via a wireless connection established with the AR.Drone proprietary software. ROS automatically detects and recognizes common data types, including many standard image formats. The streaming images are published to the topic `ardrone/image_raw`. The ROS node, encoded in `color_vision_node.py`, subscribes to the topic and selects the most recent frame using code in Algorithm 3.1:

```
    def __init__(self):
        rospy.init_node('hsv1_vision_node')
```

```
        """ Give the OpenCV display window a name. """
        self.cv_window_name = "OpenCV Image"
        """ Create the window and make it re-sizeable (second parameter = 0) """
        cv.NamedWindow(self.cv_window_name, 0)

        """ Create the cv_bridge object """
        self.bridge = CvBridge()

        """ Subscribe to the raw camera image topic """
        self.image_sub = rospy.Subscriber("/camera/image_raw", Image, self.callback)

    def callback(self, data):
        try:
            self.last_image_header = data.header #this is for my csv file
            """ Convert the raw image to OpenCV format """
            cv_image = self.bridge.imgmsg_to_cv(data, "bgr8")
```

Algorithm 3.1: Video selection code written in Python

3.3.2 Pre-Processing

After the frame is selected, the image is converted to be compatible with OpenCV. For the presented approach, the color model is converted from BGR to HSV for follow-on processes. This is accomplished by indexing the image and using a simple conversion, available in OpenCV library. This operation is conducted in the `thresh_im` function, shown below in Algorithm 3.2.

```
    ''' Separate the color image by Hue Sat and Color, threshold each channel'''
    def thresh_im(self, cv_image):
      (width, height) = cv.GetSize(cv_image)
      channel_h = cv.CreateMat(height, width, cv.CV_8UC1)
      channel_s = cv.CreateMat(height, width, cv.CV_8UC1)
      channel_v = cv.CreateMat(height, width, cv.CV_8UC1)
      '''Convert the image to HSV'''
      cv_array = cv2.cvtColor(numpy.asarray(cv_image), cv2.COLOR_BGR2HSV)
```

Algorithm 3.2: One version of the color selection function code written in Python

3.3.3 Background Subtraction

Various methods were implemented and investigated with to find the best method for isolating the target from the background. The target's color provides greatest contrast with the

background and therefore is an obvious criterion for isolation. To accomplish this, the target is treated as a flat monochromatic object, allowing for a simple single-color extraction method to be employed. To accomplish this, the color feature space must be reduced to a single dimension. For this purpose, the HSV color model provided the best opportunity by partitioning color, hue and value as separate entities. This color model is attractive for this application because any artifacts and color distortions, such as variations as a result of lighting and shading, are mostly isolated in the value channel [48].

In the HSV color format, pure green is $[60, 255, 255]$ in the color map. To filter out other colors, a mask is applied that searches for colors that fall within a certain tolerance of green. The first element indicates the hue channel and the second and third elements are the saturation and value, where 0 is none at all and 255 is maximum. Because of the high light saturation in the FOV due to direct exposure to overhead halogen lights in the laboratory, the mask had restricted upper limits to eliminate the color white, which would otherwise satisfy the lower limits. Therefore, unless another green object is introduced into the arena, the resulting image represents a scene that has filtered out all non-green objects, with any remaining noise resulting from color distortion at the boundaries of high-light levels where the transition appears to have green transitional pixels as a result of sensor limitations. After exploring various iterations to refine these upper and lower threshold limits, (some of which are shown below in code as `thresh1_im`, `thresh2_im`, and `thresh3_im`, the values are selected as: $[40, 80, 80] < p < [80, 255, 255]$. The code is shown in Algorithm 3.3.

```
#First color threshold parameters to try:
lower_G1 = numpy.array([40,80,80])
upper_G1 = numpy.array([70,255,255])
thresh1_im = cv2.inRange(cv_array,lower_G1,upper_G1)
#Second color threshold parameters to try:
lower_G2 = numpy.array([40,80,100])
upper_G2 = numpy.array([70,255,255])
thresh2_im = cv2.inRange(cv_array,lower_G2,upper_G2)
#Third color threshold parameters to try:
lower_G3 = numpy.array([40,80,100])
upper_G3 = numpy.array([80,255,255])
thresh3_im = cv2.inRange(cv_array,lower_G3,upper_G3)
```

Algorithm 3.3: Different threshold parameters are tried in the HSV color format to find the best fit for the application

3.3.4 Image Filtering and Noise Reduction

Noise is initially reduced from the image using a Gaussian low-pass filter, but in later versions of code, a bilateral low-pass filter is employed. Both are described further in this section. A low-pass filter eliminates excessive noise using local mean and variance [49]. Gaussian masks nearly perfectly simulate optical blur, so they are intuitively ideal for image processing. The Gaussian low-pass filter, often also referred to as a "Gaussian smoothing filter" is good for removing Gaussian noise or noise drawn from a normal distribution.

Gaussian Low-pass Filter

Although common and the basic underpinning of many more complicated filtering processes, including the bilateral filter and the common Kalman filter, a brief overview of the Gaussian smoothing filter is provided, referencing [46]. The Gaussian function consists of a single lobe, so a Gaussian filter smooths in the image plane by replacing each pixel with the weighted average of the neighboring pixels, with the priority decreasing as the distance from the center pixel increases. Gaussian filtering operates off the assumption that elements in images typically vary slowly over space, so near pixels are likely to have similar values, and it is therefore appropriate to average them together. The noise values that corrupt these nearby pixels are mutually less correlated than the signal values, so noise is averaged away while signal is nominally preserved.

A 2D Gaussian function is characterized by the property of rotational symmetry, and it therefore does not bias subsequent edge detection in any particular direction, because smoothing occurs consistently in every direction. Although it does not bias the filtered image in any given orientation for edge detection, it does weaken the relative signal of all edges, because the pixels at the edge boundaries are averaged together. The assumption of slow spatial variations fails at edges, which are consequently blurred by linear low-pass filtering. In OpenCV, the Gaussian filter is standardized in the library with adjustable parameters of the mask size and filter center, implemented as shown in Algorithm 3.4.

```
#first, blur to remove some noise
im_blur = cv2.GaussianBlur(numpy.asarray(thresh_im),(3,3),0)
#from CVMat datatype to numpy array datatype
```

Algorithm 3.4: Code for the Gaussian Low-Pass Filter with Parameters

Bilateral Filter

Given the limitations of the Gaussian filter discussed in Section 3.3.4, a bilateral filter is advantageous because it effectively reduces unwanted noise without blurring edges excessively. Additionally, when applied to color images, it does not create "phantom" colors and actually suppresses such "phantom" colors that may be present in the original image as a result of sensor flaws or limitations [50]. The trade off is that it is slower in terms of processing than most filters due to its computational load. A brief overview of the fundamental operation is provided, referencing [50].

Essentially, the bilateral filter inspects the pixels in the spatial domain and creates a filtering distribution as a function of both proximity and similarity to the neighboring pixels. Unlike the Gaussian filter, it is not linear and therefore non-uniform from pixel to pixel. When the neighboring pixels are similar and there is no discernible sharp boundary, the filter acts as a normal linear transformation, typically defaulting to a Gaussian distribution. However, when a sharp boundary is detected for the pixel values in the small neighborhood, the boundary is defined and the weights of pixels that are on the dissimilar side are suppressed nearly to zero. This results in a distribution that closely resembles a Gaussian distribution (centered on the central pixel) on the similar side of the boundary and the tail of a Gaussian distribution on the dissimilar side of the boundary, illustrated in Figure 3.5. The final step normalizes all the weights across the image to ensure they sum to one.

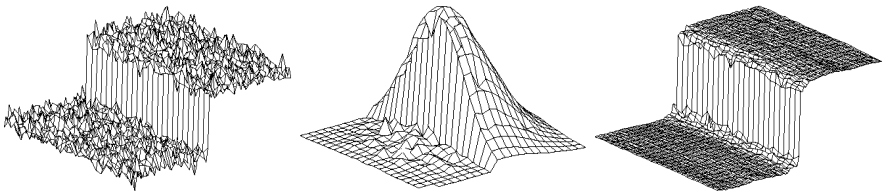


Figure 3.5: An image boundary (left) is filtered with a bilateral low-pass filter (center), resulting in Gaussian noise reduction without blurring the boundary, from [50].

As with Gaussian filters, the OpenCV library provides a function for bilateral filtering as shown by the code segment in Algorithm 3.5 from `color_vision_node.py`. The parameters for the filter are mask size, color space sigma, and physical space sigma. For the latter two, the larger sigma, which may vary depending on the feature space, indicates that values further from the centroid in each domain will be mixed together, resulting in larger areas of blurring [51].

```
combo = cv2.cvtColor(combo, cv2.COLOR_BGR2GRAY)
im_blur1 = cv2.bilateralFilter(combo, 3, 200, 0)
print "Blurred Image"
```

Algorithm 3.5: Code for the bilateral filter with parameters

3.3.5 Edge Detection

Several methods were investigated for edge detection. The output from the low-pass filter is used as the input for the new node, called `houghtrans`. Initial implementations experiment with the canny edge detector to find the edges of the target in the image plane. The OpenCV library does not have a "skeleton-ization" function (discussed in greater detail below) but instead offers two functions which, when iterated through multiple times accomplish this task on the detected shapes in the image plane to find boundaries. This is used in conjunction with the Canny edge detector for a refined implementation of the edge detector. The method selected for the final version of code employed a Probabilistic Hough Transform on the filtered image to find edges and lines. These different edge detection methods, and their respective strengths are weaknesses, are highlighted in this section.

Canny Edge Detector

The Canny edge detector was developed by John Canny in 1986 as a multi-stage algorithm that serves as a detection operator. Canny held that three criteria must be met to satisfy the requirements for a "good" edge detector [52]:

- Low error rate: Reduced occurrence of a missed detection (false negatives) and reduced occurrence of "spurious" responses (false positives);
- Good localization of edge points: minimal distance between the points marked by the detector and the center of the true edge;
- Only one edge detected for each edge. No double positives;

Referencing his first published paper on the topic [52], and follow-on works that implement the detector in current algorithms [53–55], below is a quick overview of the way the operator functions.

The original Canny edge detector is performed in four subordinate steps on a grayscale image: (1) excessive normally-distributed noise is removed using a Gaussian low-pass fil-

ter (2) a gradient operator calculates intensity/magnitude and direction in the image plane, (3) non-maximum suppression determines the "best" pixel for an edge boundary in each neighborhood, (4) double thresholding the image to remove false positives; and (5) hysteresis thresholding determines the line boundaries and eliminates false edges. Improved Canny edge detectors have been proposed in recent years, including modifications to the filtering process [53, 55].

In the OpenCV library, the Canny edge detection function uses a 5×5 Gaussian filter. It uses two orthogonal Sobel kernels for gradient detection, and the specific example of the function from the code is shown in Algorithm 3.6. A complete description of the methodology for the function is available in [54].

```
#now use canny to isolate lines
      im_canny = cv2.Canny(skel,100, 100)
```

Algorithm 3.6: Canny edge detection function in the OpenCV Library written in Python

Morphological Skeletonization

Skeletonization in the context of digital image processing is a vague term that can refer to several different methods. [56] defines two categories in this domain: one category is based on distance transforms, producing a subset of points of a given component that represent the center of a circle of a given radius contained in the given component. The second category is the one this research employed, and is defined by thinning until the median is reached, with the end result of the morphology being a connected set of digital curves, arcs or lines.

There is no specific function in OpenCV for a morphological skeleton. Instead, the `dilation` and `erosion` functions were used in a `while` loop to affect the function, shown in 3.7.

```
#Try to make a skeleton: "Skeletonization using OpenCV-Python"
#rename the output of the Bilateral or Gaussian filter to preserve original version
blurry = im_blur1
#declare variables
size = numpy.size(blurry)
skel = numpy.zeros(blurry.shape,numpy.uint8)
element = cv2.getStructuringElement(cv2.MORPH_CROSS,(3,3))
done = False
```

```
#Execute a Loop to continue to dilate and erode the lines until single pixel in width
while (not done):
        eroded = cv2.erode(blurry,element)
        temp = cv2.dilate(eroded,element)
        temp = cv2.subtract(blurry,temp)
        skel = cv2.bitwise_or(skel,temp)
        blurry = eroded.copy()
        zeros = size - cv2.countNonZero(blurry)
        if zeros ==size:
                done = True
```

Algorithm 3.7: Python code for the skeletonization function

At the conclusion of the morphological skeletonization, the Canny edge detector is applied to produce the correct syntax of edges in the image plane, as seen in Algorithm 3.6.

Hough Transform

As discussed in Section 2.5.2, the Hough line transform uses the polar coordinate system to express a line. It is applied to images which have already been pre-processed to remove irrelevant pixels by means of filtering, thresholding and edge detection (or a combination of all three).

In the image plane, every line can be expressed from a discrete reference point (typically either the origin/upper left hand corner or center of the image). The Hough Transform redefines the line from the Cartesian space and a parameter space in which a straight line (or other boundary formulation) can be defined. For every point (pixel) in the image plane, a discrete number of lines exist for which it can belong. This equation, written from the reference point, can be expressed as a given angle and distance.

Probabilistic Hough Transform The Probabilistic Hough Line Detector function in OpenCV is used for this research. All small artifacts are discarded by setting the threshold for criteria to a minimum number of pixels. This presented several issues with the detection capability, discussed in further detail in Chapter 4 and Chapter 5.

3.3.6 Target Detection

If the results of the Hough Transform yield a closed geometric shape, the computer algorithm registers that the target is in the field. The handoff is then made to determine

target location and pose relative to the aerial camera sensor. This function occurs in the `corner_det` function in `color_vision_node_HSV1_record_data.py` file. For a given frame, this function first determines the number of lines detected using the Hough Line Detector, shown below in Algorithm 3.8.

```
#find the number of lines detected
(r,c,n) = lines2.shape
#v = zeros(
print "r, c, n="
print r
print c
print n
print lines2[0]
#print lines2[0][0]
#print lines2[0][0][0]
c = c-1 #because the index starts with 0
```

Algorithm 3.8: Code for the Gaussian low-pass filter with parameters

The lines are then extended out to the boundaries of the image, shown in Algorithm 3.9.

```
        for i in range (0,c):
           v = lines2[0][i]
           dy=v[1]-v[3] #rise
           dx=v[0]-v[2] #run
           if  (dx != 0 and dy !=0): #if the line is not horizontal or vertical
                m=dy/dx
                #xmid=(v[0]+v[2])/2
                #ymid=(v[1]+v[3])/2
                yint=m * -v[0] +v[1]            # (0,yint)
                xint=v[0]-(v[1]/m)              # (xint,0)
                ymax=m*(im_col-v[0])+v[1]       # (im_col,ymax)
                xmax=v[0]+(im_row-v[1])/m       # (xmax,im_row)
                if yint==0 or im_col==xmax or yint == im_row or im_col == xint: #if the ↩
                    line extends through corner
                    if yint ==0:
                        lines2[0][i][0]=0
                        lines2[0][i][1]=0
                    if im_col == xmax:
                        lines2[0][i][2] = im_col
                        lines2[0][i][3] = im_row
                    if im_row==yint:
                        lines2[0][i][0]=0
                        lines2[0][i][1]=im_row
```

```
            if im_col==xint:
                lines2[0][i][2]=im_col
                lines2[0][i][3]=0

        if (yint<im_row and yint>0):
            lines2[0][i][0]=0
            lines2[0][i][1]=yint
            if (ymax<im_row and ymax>0):
                lines2[0][i][2]=im_col
                lines2[0][i][3]=ymax
            elif (xint<im_col and xint>0):
                lines2[0][i][2]=xint
                lines2[0][i][3]=0
            elif (xmax<im_row and xmax>0):
                lines2[0][i][2]=xmax
                lines2[0][i][3]=im_row

        elif (ymax<im_row and ymax>0):
            lines2[0][i][2]=im_col
            lines2[0][i][3]=ymax
            if (xint<im_col and xint>0):
                lines2[0][i][0]=xint
                lines2[0][i][1]=0
            elif (xmax<im_row and xmax>0):
                lines2[0][i][0]=xmax
                lines2[0][i][1]=im_row

        elif (xint<im_col and xint>0):
            lines2[0][i][0]=xint
            lines2[0][i][1]=0
            if (xmax<im_row and xmax>0):
                lines2[0][i][2]=xmax
                lines2[0][i][3]=im_row
            elif (yint<im_row and yint>0): #Had to add this because if the ↩
                conditions are met
                lines2[0][i][2]=0          #Possible that its overwritten
                lines2[0][i][3]=yint
        elif (xmax<im_row and xmax>0):
            lines2[0][i][2]=xmax
            lines2[0][i][3]=im_row


    elif (dx == 0): #vertical line
        #lines2[0][i][0] = v[0]
        #lines2[0][i][2]=v[2]
        lines2[0][i][1]=0
        lines2[0][i][3]=im_row
    else: #horizontal line
        #lines2[0][i][1] = v[1]
```

```
            #lines2[0][i][3]=v[3]
            lines2[0][i][0]=0
            lines2[0][i][2]=im_col

    for line in lines2[0]: #the number of lines detected
       pt1v = (line[0],line[1])
       pt2v = (line[2],line[3])
       #cv2.line(numpy.asarray(cv_image),pt1v,pt2v,(0,0,255),2)
       cv2.line(numpy.asarray(im_size),pt1v,pt2v,(0,0,255),2)
```

Algorithm 3.9: Lines are extended to the image boundaries to show intersecting points

Once the lines have been extended out, the lines that are very close to each other are eliminated and the remaining lines are considered for intersecting points.

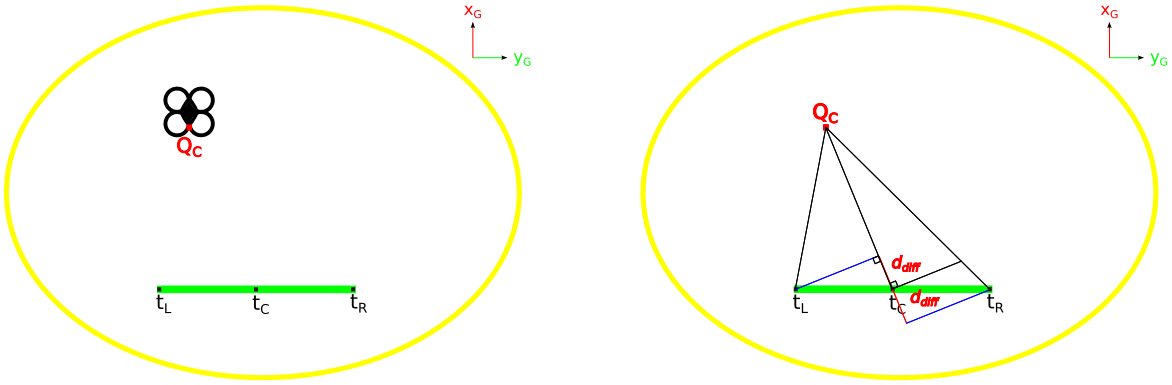
3.3.7 Pose and Location Estimation: The Transformation Matrix

The relationship between the 3D coordinates of a point in a space and the corresponding 2D coordinates of of that point in the image plane can be expressed in matrix form, known as the transformation matrix [57]. The transformation matrix is derived analytically when certain camera properties are known, including position, orientation, focal length [57]. This problem is simplified with ROS because many of these camera properties are automatically ingested and therefore automatically available for calibration. The transformation code utilized leverages existing code written in the open source community, available at [58].

Mathematical Explanation of Pose Estimation

Suppose the quadrotor's camera is considered to be a distinct point in space, and let us refer to that as Q_C in the global coordinates. If the camera is omnidirectional, the quadrotor would be able to see the target (with centroid t_C) within the arena for any range and bearing where the view of the target is unobstructed.

The range and bearing vector is given by identifying the difference in the position of the center of the camera (Q_C) and the center of the target (t_C). The size of the target (measured in degrees within the field of view) in the camera image plane is a function of distance and angle relative to the camera. The distance is given by the length of the vector formed $\vec{Q_C t_C}$ and the angle is the projection of the target onto the plane normal to the vector $\vec{Q_C t_C}$, shown mathematically in Eguation 3.1. This projection is shown in Figure 3.6 as the blue line formed at the different distances. The resulting size in the field of view (θ_Q), is the



(a) (b)

Figure 3.6: (a) Top-down view of the target and aerial chaser in the arena. (b) Top-down view of corresponding geometries for the target and aerial chaser. The angular size of the target in the camera plane is a function of target distance and orientation.

sum of the the view to the left (θ_L) and right θ_R of the centerline, can be calculated through a series of steps in Equations 3.1, 3.2, and 3.3.

$$d_{diff} = \vec{Q_C}\, t_C \cdot \vec{t_C t_L} \tag{3.1}$$

$$\alpha = t_C - d_{diff} \tag{3.2}$$

$$\theta_Q = \arctan \frac{\overline{\alpha t_L}}{\overline{Q t_C} + d_{diff}} + \arctan \frac{\overline{\alpha t_L}}{\overline{Q t_C} - d_{diff}} \tag{3.3}$$

The camera is a forward looking camera with a 92° FOV, and the center of the field of view is denoted is as x_C in the camera's local coordinates. Because the camera is not omnidirectional, it is possible that the quadrotor will not detect the target due to camera orientation. The edges of field of view are denoted as vectors that extend from the camera, 46° to the left and right of its centerline, called $\vec{v}_L$ and $\vec{v}_R$ to refer to the camera's left and right side of the frame. To determine whether the target is in the camera field of view, the orientation must be converted from camera coordinates to global coordinates and compared to the minimum and maximum angular position of the target, or $\vec{Q t_L}$ and $\vec{Q t_L}$.

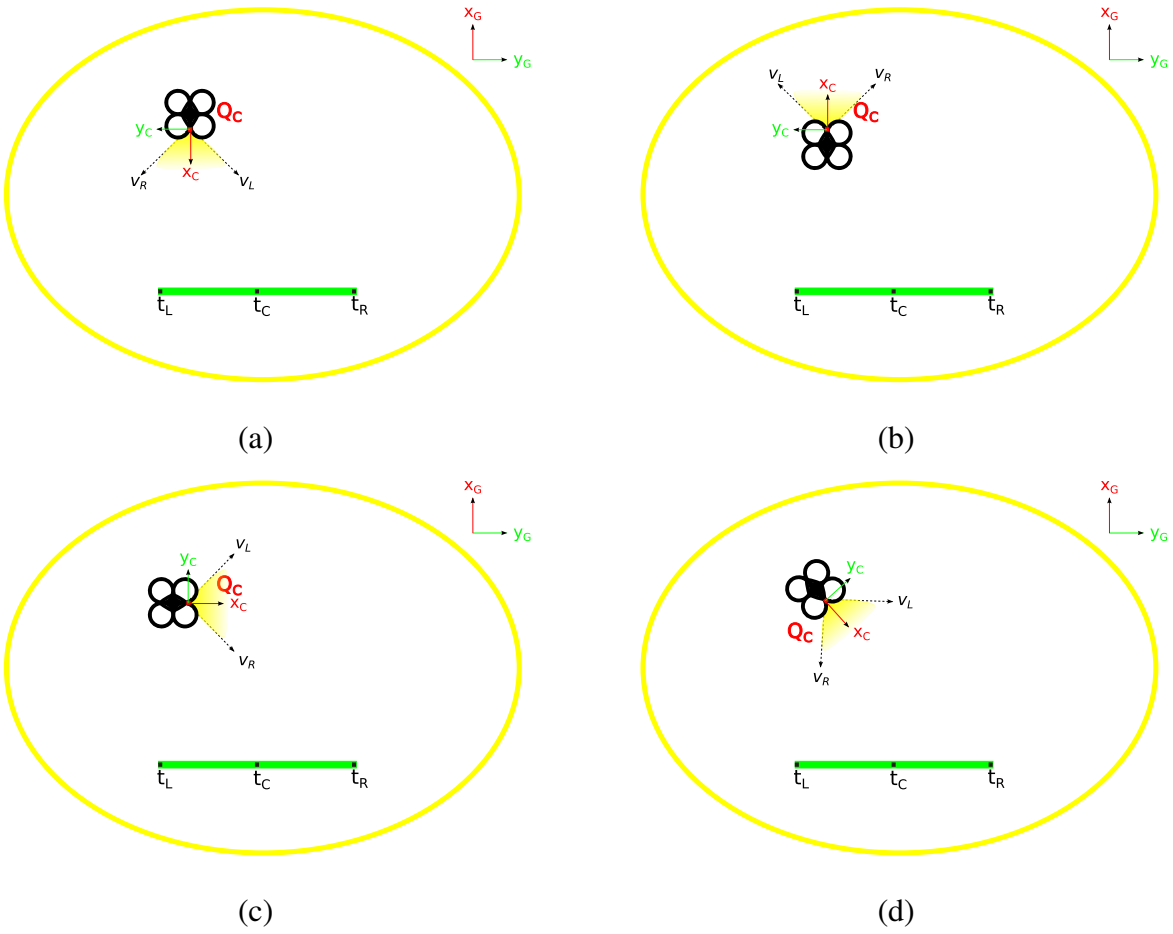


Figure 3.7: Camera FOV at a fixed location and varied orientation, relative to the target in the arena. This demonstrates the importance of the orientation of the aerial chaser. (a) Target is completely within the camera FOV; (b) Target is completely outside the camera FOV; (c) Target is mostly outside camera FOV; (d) Target is mostly inside camera FOV

The smaller the target appears in the camera frame, the larger the permissible angular range ($\nu_Q - \theta_Q$). This is affected by both distance and orientation of the camera relative to the target. The further away from the target the camera is, the smaller the target appears to be. Similarly, at more oblique angles, the target has a smaller angular size.

For the target to be fully within the field of view of the camera, the vectors that denote the maximum angular boundaries must be greater or less than θ_R and θ_L. Although not fully developed in the computer-vision algorithm, these boundaries are shown visually in

Figure 3.7. Only for these cases were the conditions for detection considered to be met and performance measured. The criteria is determined using the ground truth provided by Vicon data.

3.4 Expected Results

It is postulated that the closer the quadrotor is to the target, the more consistently the target detection software will work. When the camera is further away, the target contrast with the background appeared less pronounced.

	Baseline Algorithm	Alternative 1	Alternative 2	Alternative 3	
Pre-Processing	**HSV**	**RGB**			*PERCEPTION ALGORITHM*
Background Subtraction	**thresh_1** lower_G upper_G *[40,80,80] [70,255,255]*	**thresh_2** lower_G upper_G *[40,80,100] [70,255,255]*	**thresh_3** lower_G upper_G *[40,80,100] [80,255,255]*		
Image Filtering and Noise Reduction	**Bilateral Filter** *[3, 200, 0]*	**Gaussian Lowpass Filter** *[(3,3),0]*	**Bilateral Filter** *[5,150, 0]*		
Edge Detection	***None***	**Skeleton** *N/A*	**Canny Edge Detector** *[100,100]*	**Skeleton & Canny** *N/A [100, 100]*	
Line Detection	**HoughLinesP** *[1, math.pi/90, 40, 20, 35]*	**Blurring Function & HoughLinesP** *[5] [1, pi/90, 40, 20, 35]*			
Object Identifcation	**findContours** *cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMP*				CLASSIFICATION ALGORITHM
Object Classification	**warpPerspective**				

Table 3.1: Computer-vision algorithm alternatives are presented in table form to show a side-by-side comparison of changes in parameters and approach.

For the sake of completeness, a solution is presented to accomplish the desired detector, shown in Table 3.1. Each detector and filtering process is chosen based on a notional decision of what appeared to be the best fit. After a complete solution is accomplished, the Systems engineering spiral approach discussed in Section 3.1 is utilized to do post processing and analysis of alternatives. Below are the assumptions that drove the decision for the initial solution to the problem and the hypothesis for each test.

3.4.1 Pre-Processing

BGR is an attractive option because OpenCV imports images in this format automatically. The image quality appears to be degraded when it is converted to HSV. We postulate that the color will be more intact if the BGR solution is implemented, but opt to use HSV due to the ease of partitioning for color.

In the HSV color domain, we select the color parameters shown in the code below, Algorithm 3.10. Adjusting the parameters in both directions appears to give either too many false positives or failed to detect the target at close range.

```
# Define Range of Green Colors in HSV (GREEN IS 60,255,255)
lower_G = numpy.array([40,80,80])
upper_G = numpy.array([70,255,255])
thresh_im = cv2.inRange(cv_array,lower_G,upper_G)
# Threshold the HSV image to get only green colors
```

Algorithm 3.10: Parameters chosen for the base version of the algorithm

3.4.2 Image Filtering and Noise Reduction Results

The bilateral filter is selected for the final version because it appeared to preserve edges with no noticeable increase in noise. Using the bilateral filter did not appear to make an impact on the latency, which, as discussed in Section 3.3.4, is the most notable drawback to bilateral filtering over Gaussian.

3.4.3 Edge Detection

The final version of code employs a Probabilistic Hough Transform on the filtered image to find edges and lines. There was not sufficient time to explore alternatives to the Hough transform, but the solution space explores several methods for edge detection. Canny, Skeleton-ization and simple color isolation are all investigated as potential inputs for the Hough Transform. It is believed that the best results are rendered from the filtered image directly into the Hough Transform, and the results of this experiment seek to validate that postulation.

3.4.4 Target Detection and Pose Estimation

The target is deemed to exist when a geometric shape can be extracted from the line function. From this, the pose may be estimated by locating the centroid and conducting the transformation as described in Section 3.3.7. The final experiment for this thesis is to quantify the target detection success rate and pose estimation accuracy.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4:
Experimental Results

The experimental results for the algorithm are presented below. The initial background filtering and edge detection components appear to be relatively robust, but the hand-off to the target detection algorithm is less effective. Frames with successfully filtered edges do not always have successfully detected targets. Results for the transformation matrix and location algorithms are inconclusive and require further analysis.

4.1 Methodology for Analysis

As mentioned in Section 2.4.1, ROS provides an existing structure for recording data and replaying it as if it were real-time in the `rosbag` function. This allowed for identical data sets to be used across all different iterations of code and methodologies for image processing. There were seven recorded runs using a quadrotor and the green target in the arena. Each test is performed against these seven data sets.

To accurately quantify the results of the code, some data points are eliminated for which there is incomplete data collected. Although the Vicon arena is roughly sketched out on the ground for reference, the arena is not a perfectly formed geometric shape and coverage is subject to signal returns to a minimum number of IR cameras from all the IR markers for a given constellation. Any data collected from the camera while the quadrotor operates outside the Vicon arena boundaries is discarded as invalid due to these limitations.

4.1.1 Collecting Data

One benefit of using middleware such as ROS is that it provides an easy way to retrieve data from multiple sources in parallel. The Vicon system updates positions on all detected constellations at a rate of approximately 100 Hz. Video frames from the quadrotor, however, are captured at a rate of approximately 20 Hz. To get positional data for each instance a video frame is captured, every new image prompts ROS to record the positional data for both the quadrotor and the target and write to a $*.csv$ file using standard Python I/O and file handling methods [59]. An example of this is shown in Algorithm 4.1

```
def callback(self, data):
    try:
        self.last_image_header = data.header #this is for my csv file
        d_row = (self.last_image_header.seq, self.last_image_header.stamp.secs, self↩
            .last_image_header.stamp.nsecs)
        writer = csv.writer(open(os.environ['HOME'] + "/Desktop/GRN04_def.csv",'a'))
        if len(d_row) > 0:
            writer.writerow(d_row) #If there is a line detected, write a new row
    except CvBridgeError, e:
      print e
```

Algorithm 4.1: One example of the csv writer function in the algorithm

4.1.2 Analysis of Results

Frame breaks for transitions to target in or out of the scene is conducted manually by analyzing the videos and recording the frames for which the target is fully (F), partially (P) or not at all visible (N) in the quadrotor camera FOV. Streaming video is manually paused at each transition and the time stamp is annotated on the reference comma-separated value (csv) file for the given video. Each frame that is assigned (F) is given a numerical value of 1, each frame that is assigned (P) for having the target partially present is given a numerical value of 0.5, and all frames for which there is no target present (N) are given a value of 0. For analysis, any nonzero frame, which is any frame that has partial (P) or full (F) view of the target, is considered a positive presence of the target in the FOV, regardless of how much or little of the frame is visible. Chapter 5 discusses potential improvements for this process.

A Matlab script analyzes the results. The csv files created using the Python script described in Section 4.1.1 are imported into Matlab and any headers are stripped off to leave the raw data. In the csv files, any data that is outside the Vicon system boundaries will simply show the last known position. These data points are removed from the arrays using the custom written Matlab function `Data_Run_Edit.m`.

Once the spurious data is stripped and only useful information remains, the next step for analysis is to determine the true range and bearing from the sensor centroid to the target centroid. Vicon provides the centroid of the constellations, which is deemed to be satisfactory for this application. Section 5.2.1 discusses improvements for data synthesis. The

Euclidean distance is easily computed using the difference between the two centroids in the x, y, z axes, as shown in Figure 4.1 by X_t, Y_t, Z_t and X_Q, Y_Q, Z_Q.

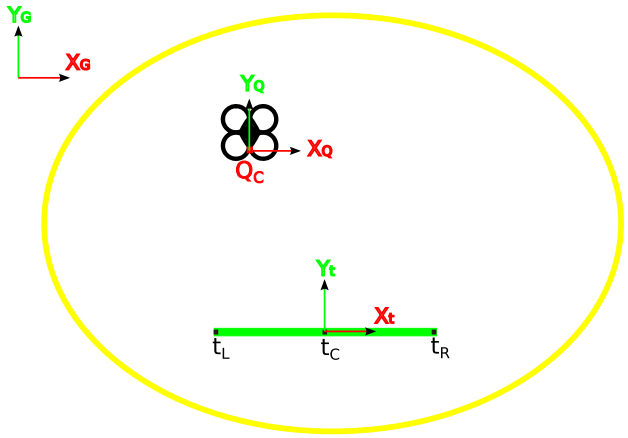


Figure 4.1: Euclidean distance is computed by calculating the distance between each centroid in the 3D space

Euclidean distances are computed by this simple computation:

$$\text{distance} = \sqrt{(X_t - X_Q)^2 + (Y_t - Y_Q)^2 + (Z_t - Z_Q)^2}$$

This computation is carried out in the Matlab script `Data_Run_Edit.m`, shown in Algorithm 4.2.

```
rowC = size(im_seq,1); %Number of rows in Vicon-Boundaries
EucDist = zeros(rowC,1);%The Euclidean Distance, in Meters
for i = 1:rowC
  %Break up each translational vector component and caluculate distance in each ↩
      dimension
    x_HOOP = HOOP_trans_x(i,1); y_HOOP = HOOP_trans_y(i,1); z_HOOP = HOOP_trans_z(i,1);
    x_QUAD = QUAD_trans_x(i,1);y_QUAD = QUAD_trans_y(i,1);z_QUAD = QUAD_trans_z(i,1);
    x_diff = x_HOOP - x_QUAD; y_diff = y_HOOP - y_QUAD; z_diff = z_HOOP - z_QUAD;
    %Take the square root of the sum of squares of the difference
    x = (x_diff)^2; y = (y_diff)^2; z = (z_diff)^2; EucDist(i,1) = sqrt(x+y+z);
end
```

Algorithm 4.2: Code from `Data_Run_Edit.m` that computes the Euclidean distance of the target to the sensor

Finally, the data points and their corresponding distances are sorted and indexed based on the detection and ground truth data. This classification sorts them based on two criteria, shown graphically in Figure 4.2.

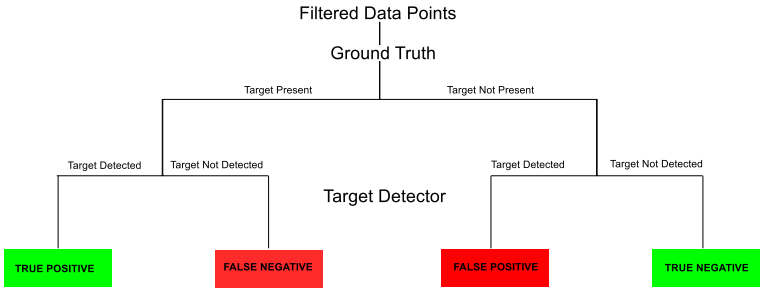


Figure 4.2: Flow chart representing the logic for determining classification criteria for analysis. True Positives are positive detections during which the target is in the FOV. False Negatives are when the algorithm fails to detect the target in the camera's FOV.

First, ground truth of whether the target is in the FOV. If the target is in the FOV, it is indexed into the first category. Within this category, detections are classified as either "true positives", meaning the target is detected and is in the camera's FOV or "false negatives", which means the target is in the camera's FOV but no detections are registered. The second category is for all data points remaining where there is no target in the camera's FOV. In this case, all positive returns from the detection software are classified as "false positives." Once normalized, "False Positive" and "True Negative" probabilities should sum to one. Similarly, "True Positive" and "False Negative" probabilities should also sum to one.

The function in the Matlab script `Data_Run_EDIT.m` completes this task. Classification decomposition can be accomplished in any order, and in this function the first branch is positive and negative detections, shown in Algorithm 4.3 as the "while" loop. All non-detections go to the "while" loop and are classified as either a "False Negative" or a "True Negative." If there is a detection for the indexed image, the while loop will be invalid and the function will drop down to the "if" statement that classifies the non-detection as either a "True Positive" or "False Positive."

```
%   Declare Variables
TPos = zeros(1,1); FNeg = zeros(1,1); %Pos Det, Neg Det: Hoop IN FOV
FPos = zeros(1,1); TNeg = zeros(1,1); %Pos Det, Neg Det: Hoop NOT in FOV
```

```
rowPD = size(im_pos_seq,1); %For Loop will go through every line of the positive ↩
    detection array "im_pos_seq"
rowDef = 1; %the row of the sequenced def files
rowTPos = 1; rowFPos = 1; rowFNeg = 1; rowTNeg = 1; %initialize rows for new matrices
for i = 1:rowPD
    while im_seq(rowDef,1)<im_pos_seq(i,1) %No detections from Algorithm
        if HOOP_in_frame(rowDef,1)>0 %Target IS PRESENT in FOV (No Detection Made)
            FNeg(rowFNeg,1) = EucDist(rowDef,1); %record Euc Dist to the False Negative ↩
                Matrix
            rowFNeg = rowFNeg+1; %add a new row
        else %if the target is NOT in FOV, (No Detection Made)
            TNeg(rowTNeg,1) = EucDist(rowDef,1); %record Euc Distance to the True ↩
                Negative Matrix
            rowTNeg = rowTNeg+1; %add a new row to matrix TNeg
        end
        rowDef = rowDef+1; %Now go to the next indexed image number
    end
    if im_pos_seq(i,1) == im_seq(rowDef,1) %Algorithm Made Detection
        if HOOP_in_frame(rowDef,1) > 0 %Target IS PRESENT in FOV (Det Made)
            TPos(rowTPos,1) = EucDist(rowDef,1); %record the Euc Distance in current row↩
                for True Positive Classification
            rowTPos = rowTPos+1; %add a row to matrix TPos
        else %if the target is NOT in FOV (Detection Made)
            FPos(rowFPos,1) = EucDist(rowDef,1); %record Euc Distance in FPos Matrix, ↩
                current row
            rowFPos = rowFPos+1; %add a new row to matrix FPos
        end
        rowDef = rowDef+1; %Now go to the next indexed image number
    end
end
```

Algorithm 4.3: Data is sorted based on classification category in `Data_Run_Edit.m`

The resulting arrays from this final step are used to create the graphical displays of the results shown in Section 4.2.

4.2 Perception Algorithm Results

The baseline perception algorithm used for analysis of alternatives is shown side by side with the alternatives in Table 4.1 has the results shown below in Figure 4.3 and Figure 4.4.

	Baseline Algorithm	Alternative 1	Alternative 2	Alternative 3	
Pre-Processing	**HSV**	**RGB**			*PERCEPTION ALGORITHM*
Background Subtraction	**thresh_1** lower_G upper_G *[40,80,80] [70,255,255]*	**thresh_2** lower_G upper_G *[40,80,100] [70,255,255]*	**thresh_3** lower_G upper_G *[40,80,100] [80,255,255]*		
Image Filtering and Noise Reduction	**Bilateral Filter** *[3, 200, 0]*	**Gaussian Lowpass Filter** *[(3,3),0]*	**Bilateral Filter** *[5,150, 0]*		
Edge Detection	***None***	**Skeleton** *N/A*	**Canny Edge Detector** *[100,100]*	**Skeleton & Canny** *N/A* *[100, 100]*	
Line Detection	**HoughLinesP** *[1, math.pi/90, 40, 20, 35]*	**Blurring Function & HoughLinesP** *[5]* *[1, pi/90, 40, 20, 35]*			

Table 4.1: Perception algorithm alternatives are presented in table form to show a side-by-side comparison of changes in parameters and approach.

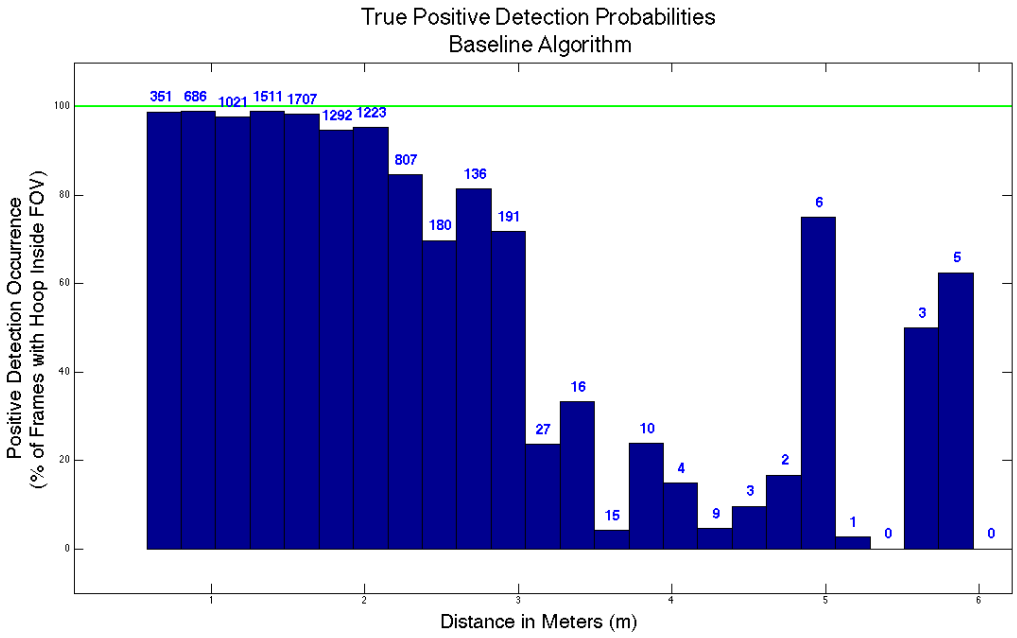


Figure 4.3: **Baseline detection software model results.** True positive detection probabilities shown in histogram

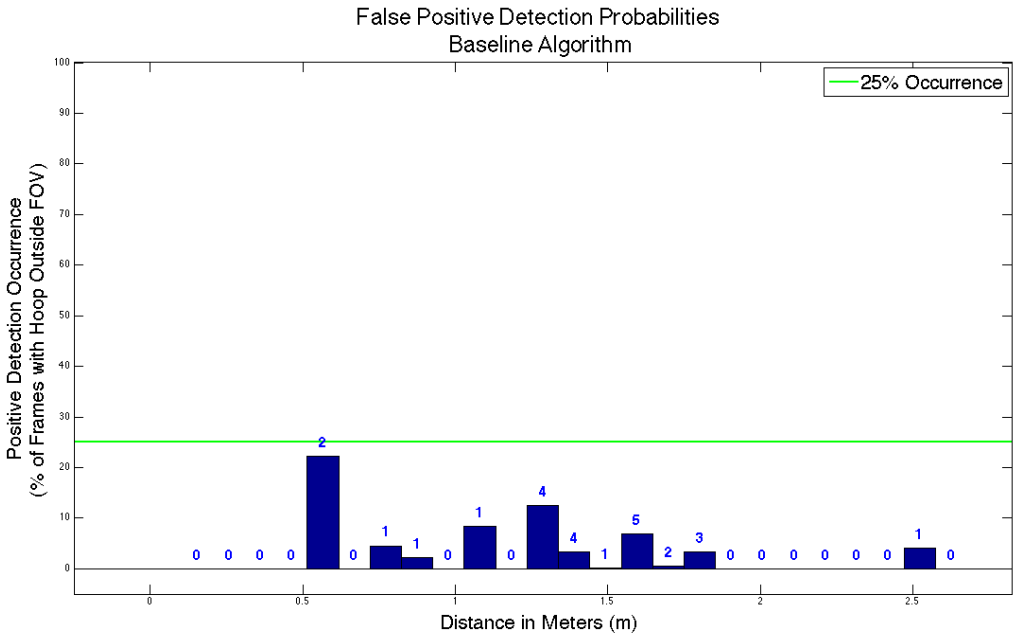


Figure 4.4: **Baseline detection software model results.** False positive probabilities shown in histogram

Some emerging trends remained consistent across every version of the software implemented. As was expected, the closer the quadrotor is to the target, the more consistently the target detection software worked. When the camera is further away, the target contrast with the background appears less pronounced. This likely resulted in intermittent target detection at further ranges and then consistent target detection at close-in ranges. Below we discuss the results for each individual experiment to validate the model chosen initially.

4.2.1 Pre-Processing

BGR is an attractive option because OpenCV imports images in this format automatically. The image quality appears to be degraded when it is converted to HSV, so for this reason, initial efforts are made to filter the image in the BGR format. BGR is sensitive to light changes and is difficult to normalize for changes in the scene based on high illumination areas such as white back lights. While possible, the additional filtering required is cumbersome and time consuming.

Time constraints and the difficulties in isolating the colors accurately discussed above resulted in using HSV, making it impossible to assess if there is a loss of information or color clarity when the images are reformatted.

Another area in pre-processing explored is the thresholds set in the HSV domain. Four parameter sets, shown in Algorithm 4.4, produced seemingly acceptable filters, but the results were too close to determine heuristically. Instead, the algorithm is run against all four versions. All other aspects of the algorithm remain unchanged.

```
# Define Range of Green Colors in HSV (GREEN IS 60,255,255)
''' Baseline Threshold Parameters are shown as thresh_im.
Alternative solutions are thresh1_im, thresh2_im, and thresh3_im '''
lower_G = numpy.array([40,80,80])
upper_G = numpy.array([80,255,255])
thresh_im = cv2.inRange(cv_array,lower_G,upper_G)

lower_G1 = numpy.array([40,80,80])
upper_G1 = numpy.array([70,255,255])
thresh1_im = cv2.inRange(cv_array,lower_G1,upper_G1)

lower_G2 = numpy.array([40,80,100])
upper_G2 = numpy.array([70,255,255])
thresh2_im = cv2.inRange(cv_array,lower_G2,upper_G2)

lower_G3 = numpy.array([40,80,100])
upper_G3 = numpy.array([80,255,255])
thresh3_im = cv2.inRange(cv_array,lower_G3,upper_G3)
```

Algorithm 4.4: Color threshold upper and lower boundary settings

For the first algorithm, which is the baseline model for all other experiments conducted, the results against all seven data sets are shown in Figure 4.3 and Figure 4.4 as histograms. The results of alternative color thresholds are overlayed on the baseline threshold results in Figure 4.5 and Figure 4.6.

The first alternative color algorithm, `thresh1_im` (data points shown as green triangles in Figure 4.5 and Figure 4.6) showed similar performance to the baseline. In some instances, it out-performed the baseline model (baseline data points are blue cirlces in Figure 4.5 and Figure 4.6) because it had a larger tolerance for the upper limits of defining the color green. The baseline color model outperformed both `thresh2_im` (data points are displayed in pink

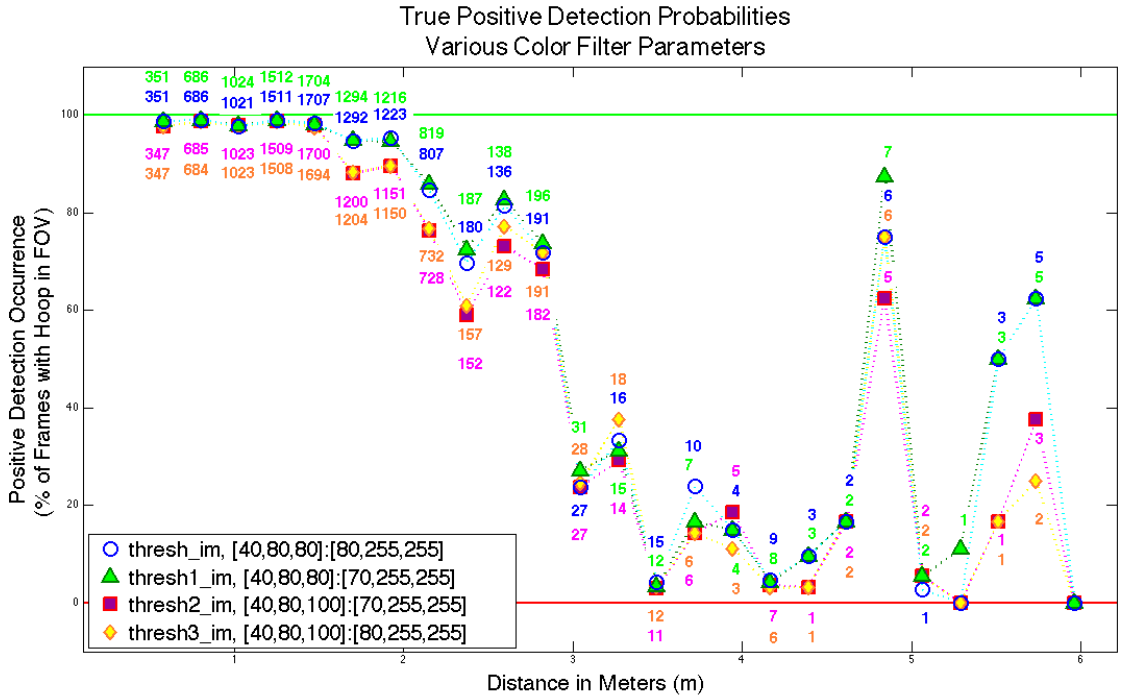


Figure 4.5: **True positive detection probabilities, color filter threshold `thresh_im` varied as indicated.** Baseline is shown in blue and is labeled `thresh_im`. Bin sizes are shown for corresponding data points to provide context of the number of data points for a given range.

squares in Figure 4.5 and Figure 4.6) and `thresh3_im` (data points are yellow diamonds in Figure 4.5 and Figure 4.6) at distances less than three meters. Overall, the baseline color model was more robust. For both `thresh2_im` and `thresh3_im` color models, the lower thresholds were more constrained than the baseline model, and from this data it is clear that this was a poorer fit for the color of the target.

The first thresholding shows best results with the overall algorithm. These results are specific to this algorithm and this experiment. Overall, the detections are only moderately impacted by the changes in the parameters. This indicates that there are limitations of the entire algorithm that changes in parameter settings for the color thresholding cannot impact.

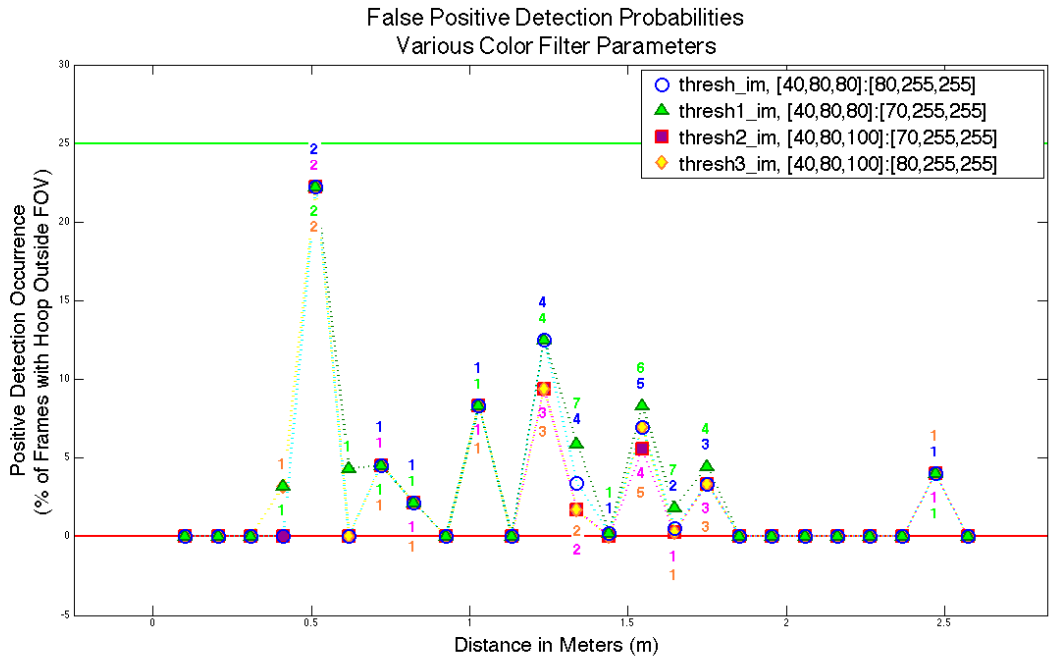


Figure 4.6: **False positive detection probabilities, color filter threshold `thresh_im` varied as indicated.** Baseline is shown in blue and is labeled `thresh_im`. Bin sizes are shown for corresponding data points to provide context of the number of data points for a given range.

4.2.2 Image Filtering and Noise Reduction Results

The filtering method and parameter settings used to eliminate the background resulted in a much higher occurrence of false negatives than false positives. For seven trial runs involving all three above described arena configurations, the data is compiled and analyzed. The results of the algorithm with Gaussian blur as compared to the baseline using a bilateral filter are shown in Figure 4.7.

The algorithm performs better overall with the Gaussian filter. The detection performance while the target is in the FOV either meets or exceeds the bilateral filter, shown in Fig-

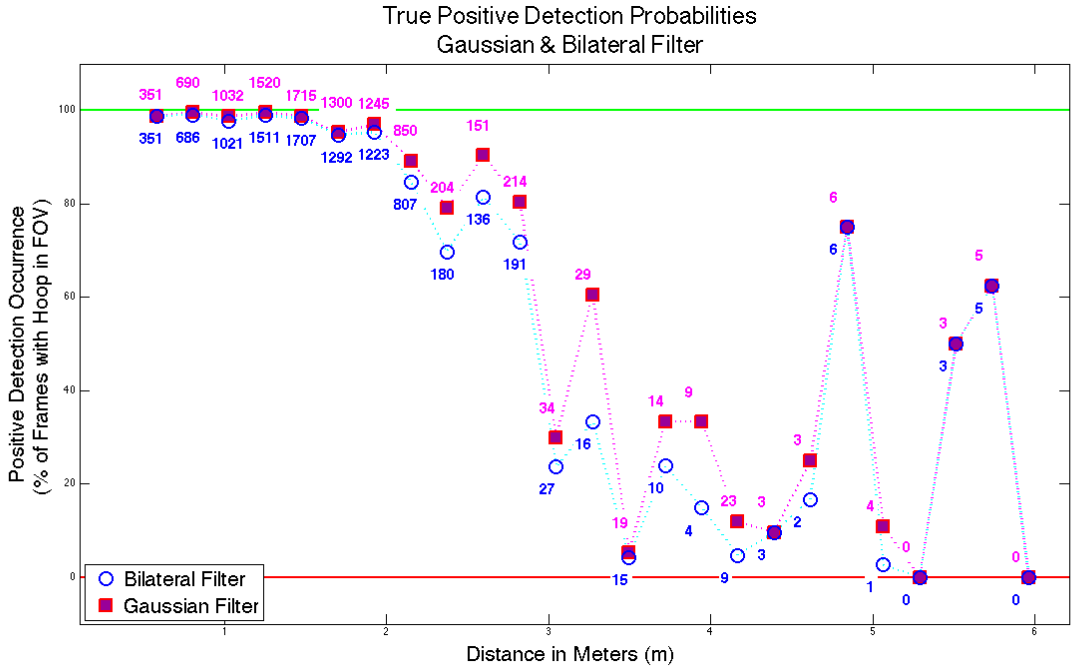


Figure 4.7: **True positive detection probabilities, bilateral filter and Gaussian blur.** Bin sizes are shown for corresponding data points to provide context of the number of data points for a given range.

ure 4.7, and the degradation in performance for registering false positives is not significant, shown in Figure 4.8.

4.2.3 Edge Detection

The baseline metric for the edge detector performance is the probabilistic Hough line detector run against the bilateral filter input. The Hough line detector was not modified. Rather, the inputs were varied as discussed in Section 3.4 to see if there was an increase in performance. The results for the baseline performance are shown in Figure 4.3. Compared to the Canny edge detector and Skeletonization edge detector, the raw input of a filtered image performed the best with the Hough Line detector. This is likely due to the fact that

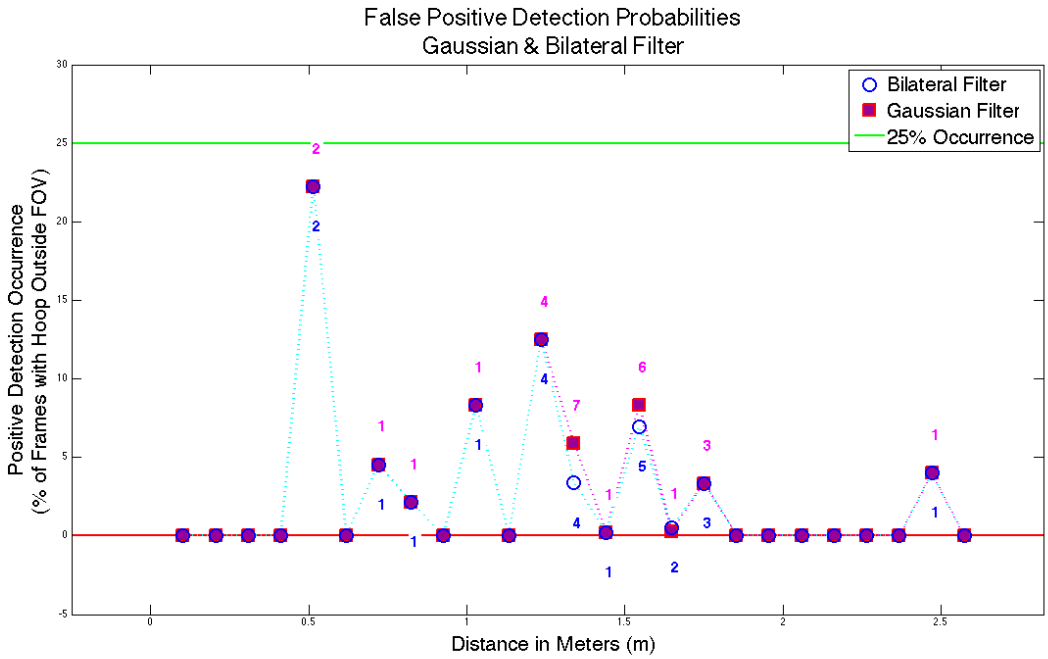


Figure 4.8: **False positive detection probabilities, bilateral filter and Gaussian blur.** Non-zero bin values are shown to provide reference of scale.

the Hough Line detector performs a Canny operator on the input as part of the steps, which are outlined in Section 3.3.

The skeletonization algorithm described in Section 3.3.5 performed very poorly. When the Hough transform was run against the "skeletonize" edge detector, the positive returns were extremely sparse. In the images shown in Figure 4.9, the detector clearly filters the edges in a gray-scale image but the Hough Line function returns very few positive detections, even when the returned edge signal appears very strong. There was some returns, but these are still insufficient to produce an object that would provide a centroid or geometry. Often the returns were only a partial side, shown in Figure 4.10.

Ultimately, the results speak for themselves. Looking at the graphical display of the true positive detections and the false positive detections, it becomes quite apparent that

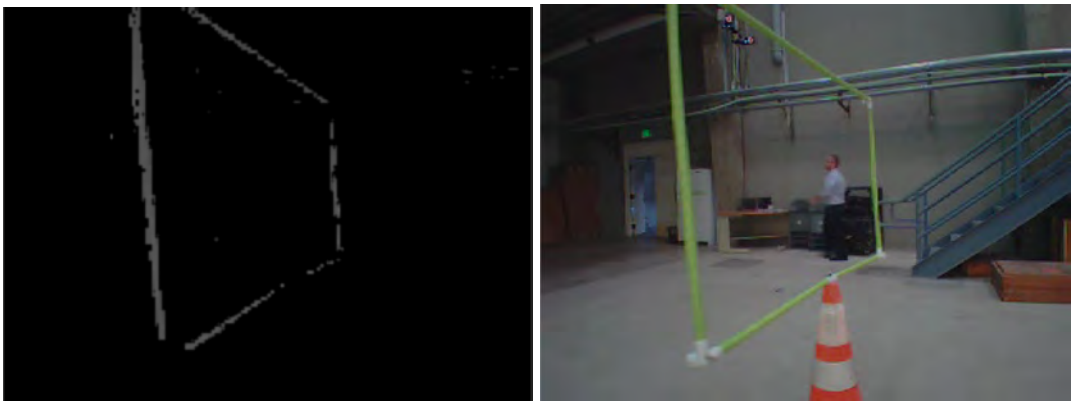


Figure 4.9: A frame of the video is analyzed using the target detection software with the skeletonize function enabled. (a) Edges of the target appear to have a strong return in the filter. (b) Despite the strong edge signals shown, no positive target detection.

Figure 4.10: A frame of the video is analyzed using the target detection software with the skeletonize function enabled. Despite the strong edge signals, detection using the skeleton function was sporadic and inconclusive. Shown here, only partial positive detection of one side.

the `Skeletonize` function did not perform as the user intended. Figure 4.11 and Figure 4.12 show the poor detection performance achieved when the results are analyzed against the experimental data.

4.2.4 Target Detection

Due to the high variability of the target size in the image frame, limitations imposed by the approach taken make it difficult to create a robust target detector that detected the object from the detected edges. Figure 4.13 illustrates a critical weakness in the algorithm pre-

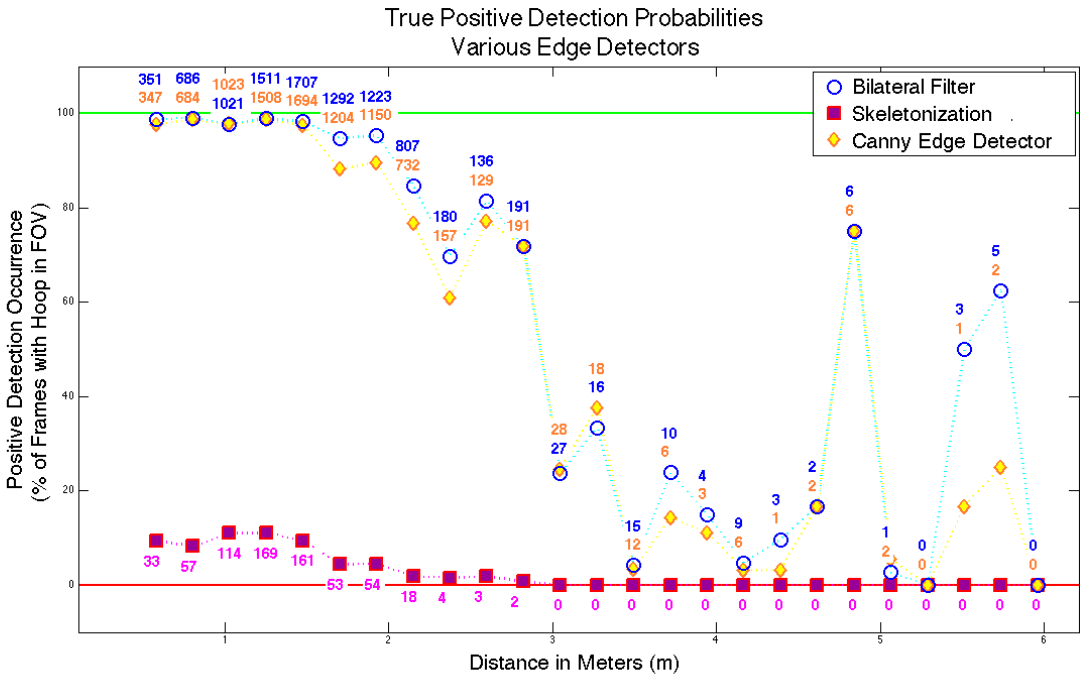


Figure 4.11: **True positive detection probabilities, edge detection methodology varied as indicated.** Baseline is shown in blue and is labeled `Bilateral Filter`. Bin sizes are shown for corresponding data points to provide context of the number of data points for a given range.

sented. There are many occurrences where the edges of the target are detected but the target itself is not detected. This is largely because of the inflexibility of the algorithm used to adapt to the number of edge pixels detected, which is a function of the target distance and aspect relative to the camera, and not a function of detection. This is specifically showcased in Figure 4.13, where it is obvious to the casual observer that edges correlating to the target are detected, but fail to be registered as the target in the hand-off to the Hough transform. Although the edges are detected and all background is filtered out of the image plane, the edge signal is not strong enough to be detected as an object by the target detection criteria and is discarded.

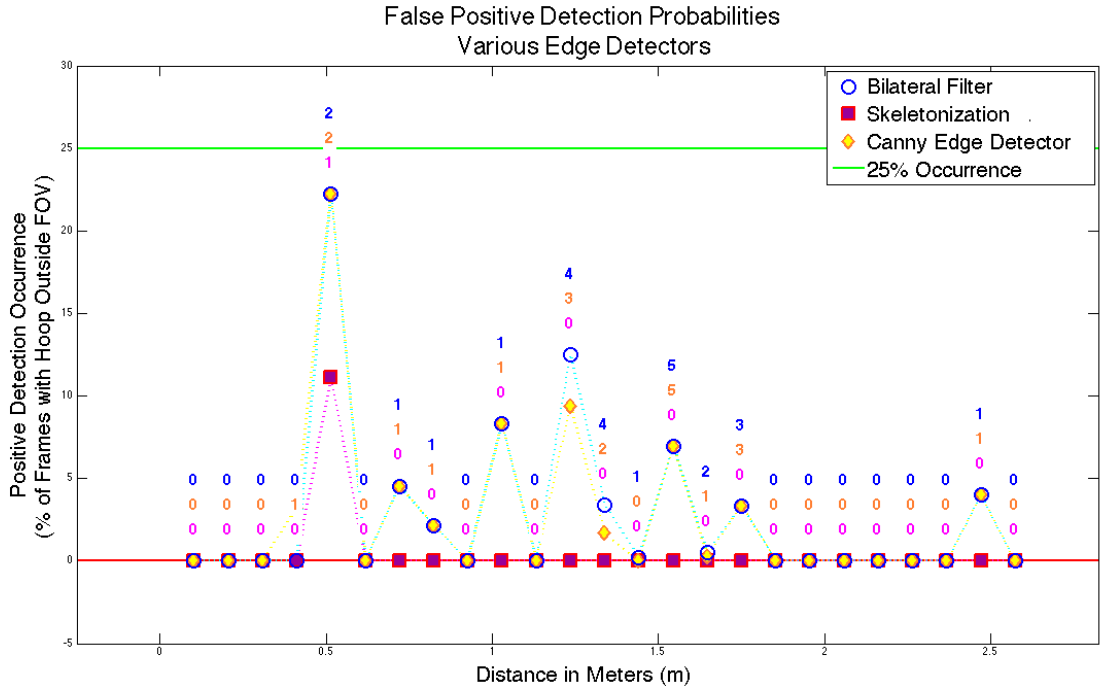
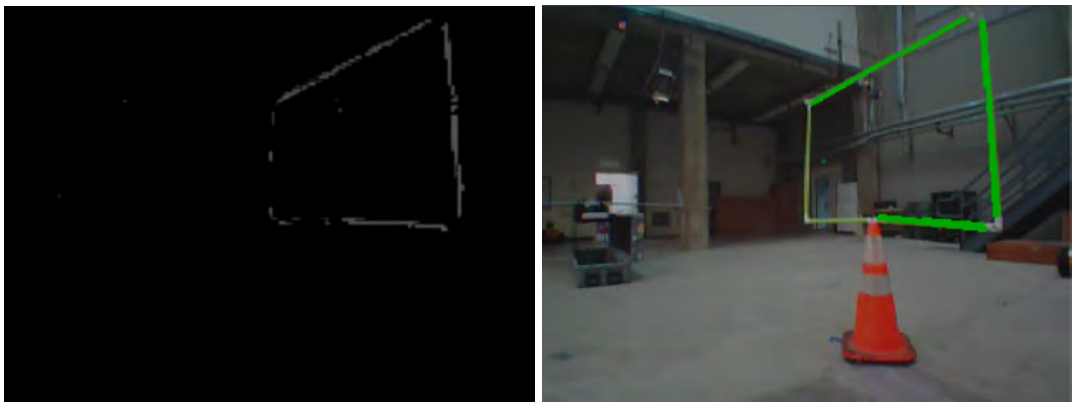


Figure 4.12: **False positive detection probabilities, edge detection methodology varied as indicated.** Baseline is shown in blue and is labeled `thresh_im`. Bin sizes are shown for corresponding data points to provide context of the number of data points for a given range.

There are some cases where the results of the detection software was surprisingly robust. In some cases, positive detection is still achieved with reduced clarity from the edge detection software to the target detection software, as shown in Figure 4.14. Although the filtered signal is weak, there are sufficient edge pixels and the target is close enough to register valid lines when the results are run through the Hough transform function. This observation further supports the conclusion that the algorithm's critical weakness is heavily weighted by the number of pixels correlating to the size of the target in the camera view, and not as impacted by the color contrast diminishing due to distance or aspect, demonstrated in Figure 4.15.

Figure 4.13: A frame of the video is analyzed using the target detection software. (a) Edges of the top and bottom sides of the frame are detected. (b) No positive target detection.



(a) (b)

Figure 4.14: A frame of the video is analyzed using the target detection software. (a) All four edges of the top and bottom sides of the frame are detected. (b) Only sides of the target are detected.

4.2.5 Issues of Latency

There were no observed issues as a result of latency during the one-way transmission of data, but could be an issue with the processing speed required for some of the operations. There were delays observed for some of the computer vision processing applications, but these are reduced greatly when they are not displayed to the graphic user interfaces. For ease of understanding, a display of the lines detected using `HoughLinesP.py` is created in which the line segments are extended out to the boundaries of the image. The display is cumbersome and did appear to lag behind the stream rate due to the computational load.

(a) (b)

Figure 4.15: Two separate frames from the same video taken from the first series of experiments (target at origin) are analyzed using the target detection software. In both frames, the target has the same aspect but is shown at varying range. (a) Closer range to target: Edges of all four sides of the frame are detected. (b) Further range from target: No positive target detection.

Additional investigation, although beyond the scope of this thesis, is warranted since the ultimate goal is to enable closed-loop feedback control using these results.

4.2.6 Varying Angles and Approaches

To determine the robustness of the algorithm, one may implement the vision software across a series of experiments that vary the location of the target in the arena. The first data set collected places the target in the center of the arena, nearly at the origin in the *xy*-plane. For the second and third series of experiments, streaming video data is collected with the target located near the boundaries of the arena (as dictated by the Vicon coverage) to provide maximum range between the quadrotor and the target.

During each experiment series, the quadrotor moves freely around the target to gather data at continuously varying ranges and angular approaches from the target. Vicon markers affixed to both the quadrotor and the target provide distance and aspect information, while pose information gathered from the quadrotor shows the expected FOV.

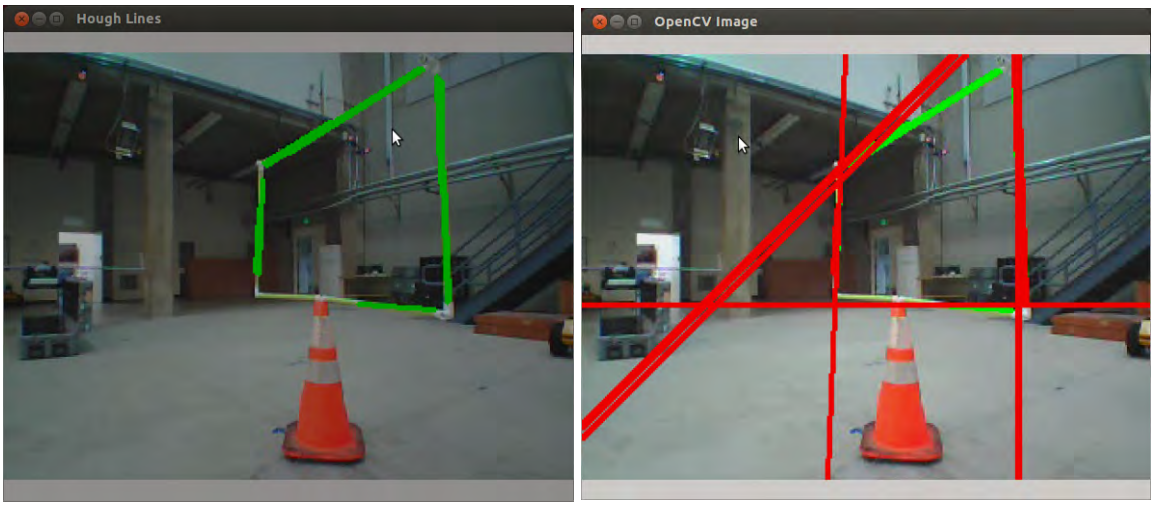


Figure 4.16: A frame of the video is analyzed using the target detection software. (a) All four target sides are detected (b) The Line Extension Display shows as an overlay on the image, showing inaccuracies and errors.

Initial observations of the algorithm running indicate that the target detection software success is less impacted by the angle of the target than the distance of the target. The observations for the target classification software are less conclusive. It appears that the larger target angle offset is correlated to lower classification likelihood. Chapter 5 discusses improvements and additional results that should be analyzed for future works.

CHAPTER 5:
Conclusions

Final summaries and findings of this project are presented herein, alongside suggested future work and improvements.

5.1 Summary

The purpose of this thesis was to develop a flexible architecture using open-source programs to build a vision-based algorithm to assist in the automated navigation of an AR.Drone. This was accomplished by implementing pre-existing functions in several libraries.

Using the Systems Engineering approach presented in Chapter 3, we were first able to determine effective needs and from those establish threshold and objective requirements. A rudimentary algorithm is developed using assumptions drawn from the literature review in Chapters 1 and 2. After the first operational model based on these generalizations and assumptions, alternatives are examined and some experiments are conducted where needed and able so that we could validate the assumptions made and potentially improve performance of the algorithm.

The performance of the algorithm presented in this thesis did not provide the desired results in all areas assessed. However, the underlying structure and software architecture remains valid and flexible and is the greatest contribution of this body of work.

5.2 Lessons Learned and Short-Term Recommendations

Throughout the process, there were many occasions where a modified approach would have likely resulted in a better end product. These are captured in Section 5.2.1. Short term recommendations do not seek to modify the algorithms but instead to better improve the analysis approach to provide better methods for data consolidation and analysis.

5.2.1 Areas for Immediate Improvement

More time could be dedicated to determine threshold and objective requirements for detection software suitability. There was very little analysis completed in the early stages to

determine what the application of this detector would ultimately simulate. The initial undertaking was that this target might be a target of interest for prosecution. Implementing a weapons system based on the detections registered with this software would be catastrophic if it misidentified and registered a target present that was indeed not.

Conversely, if the target is a potential target of interest for surveillance, higher false detections may be acceptable to reduce the risk of a missed detection or false negative.

Dynamic Classifier

The classifier used in this research is very basic and simple. Trying to detect a target using other means would have probably rendered improved results. Also, last known data for the target was never integrated into the algorithm to drive the search criteria.

For the target detection software specifically, the classifier must be made to be more robust. The current target classifier is very sensitive to the size and spacing in the image plane. An alternative would be to instead accommodate the scaled size of the objects detected instead of number of pixels, due to the changes in the distance/size of the target.

Automated Vicon System Integration for Analysis

In many respects, data analysis is limited by the ability to utilize all the information provided by the sensors. Although the quaternion positions and angles of the target and quadrotor constellations are available through the Vicon system, additional investment of time is necessary to learn how to translate that into usable information for analysis. Determining the aspect of the quadrotor, and the resulting FOV of the quadrotor camera, would have allowed for an automated process of determining ground truth for the presence of the target in the camera's FOV. Additionally, further analysis could then be conducted to determine the limitations of detection as a function of both distance and aspect instead of distance alone. Much information is lost, especially when considering a target such as this that is so different from one aspect to the other.

Additional Data Collection with Varying Angles and Approaches

To determine the robustness of the algorithm, the vision software is implemented during three series of experiments with different target location within the arena. The first data set collected placed the target in the center of the arena, nearly at the origin in the xy-plane. For the second and third series of experiments, streaming video data is collected with the

target located near the boundaries of the arena (as dictated by the Vicon system coverage) to provide maximum range between the quadrotor and the target.

During each experiment series, the quadrotor moved freely around the target to gather data at continuously varying ranges and angular approaches from the target. Vicon markers affixed to both the quadrotor and the target provide distance and aspect information, while pose information gathered from the quadrotor show the expected FOV. Once the data was compiled and analyzed, it was apparent that the majority of the data points collected were at close range to the target. Additional data collection efforts at the boundaries of the Vicon system arena to allow for more comprehensive data analysis could directly benefit the end user with more distilled information to draw consolidated conclusions.

5.3 Future Work

As mentioned in Section 4.2.1, reformatting the images qualitatively appeared to potentially degrade color resolution. There was not sufficient time to explore this, and further work could be done to analyze if this is indeed the case.

Chapter 4 notes that the transformation algorithm was not implemented entirely correctly and provided inconclusive results. Modifying the existing transformation algorithm or incorporating a functioning transformation algorithm to determine the orientation/distance of the target would allow for further investigation into the controller and complexity of the problem. From this one could determine desired course and speed change to intercept target and incorporate controller (P or PI or PID) for implementation. The vision for the project initially began with a mobile target that moved within the arena and was prosecuted by the aerial target. This is still a long way off from that with the current target detector.

5.3.1 Improved Algorithm

As mentioned in Section 4.2, target detection was intermittent at times, especially when the target is at a further distance. Assuming the data update rate far exceeds the speed of the target, thus making it essentially stationary from frame to frame, it is feasible for the target to be considered at a known position given "last known position" and then using a dead-reckoning scheme of maneuvers to focus the search area and potentially increase the detector sensitivity to allow for a higher rate of detection.

The Probabilistic Hough Transform function in OpenCV makes the criterion for a line adjust based on the size of the lines detected, not on the hard-fast rule of number of pixels. Long lines have a higher likelihood of being picked and therefore need less votes before the corresponding accumulator bin reaches a count that is not accidental [60]. For shorter lines a much higher proportion of supporting points must vote. This is relative to the entire image plane space. This means that the closer range target views will provide strong returns from the `PHoughLines.py` but for the further distance, the voting of the lines requires a larger number of samples to provide a positive. Dynamic adjustment of the parameters would provide a sliding rule to lower the required number of samples (and therefore delay) for larger line segments. One alternative may be to section off the boundaries where any positives edges are registered as part of the pre-processing stages, so the lines returned are stronger relative to the entire image space considered.

Following on the improvements to the Hough Transform, there is a lot of room to improve the transformation matrix so it can be implemented in a way that provides useable results. Currently, the transformative matrix only works when the Houghline output meets very specific criteria. Increasing the fidelity of the transformative matrix or improving the `Hough Line` function output are both recommended for consideration.

5.3.2 Target Modification

The target used is elementary by design and also holds many opportunities for improvement. Altering the shape, feature space, and target characteristics are all areas that could be improved.

Other Targets/Shapes

Another alternative is to utilize a more interesting feature space for target. Fiduciary markers or a more complex target feature space will both present interesting alternatives for the methodology of vision. Incorporating the ground robot initially presented in Chapter 2 as the base for the target could present an interesting feature space without much additional modifications.

Target Mobility

Once the algorithm has been improved, it would be useful to incorporate a moving target, as the design of experiments was initially envisioned. One method for doing so is to increase

the complexity in phases. First, incorporate a ground robot base with constant velocity and constant direction, then increase the complexity by varying velocity, direction and acceleration.

5.3.3 Other Subsystem Integration

Potential exists in this project to expand from a simple color detector and target classifier to a closed-loop controller.

Trajectory Planning

Using the estimated target position gleaned from computer image processing, the computer processing unit can determine the desired course and speed for intercept based on a static target or a target of a known velocity vector. A closed-loop system employing a simple desired position and current position estimate would be roughly equivalent to a proportional controller (P). Executing more complicated control loops, such as a proportional-derivative (PD) or proportional-integral (PI) controller to manipulate dampening and speed of arrival may produce better solutions. For the purpose of this thesis, this is outside the scope of work but would be an interesting addition.

State Estimator

Using own-ship knowledge from proprioceptive sensors and the relative position of the target from one frame to the next may produce an accurate estimation of the target's velocity vector. This becomes increasingly important for fast moving targets where the speed of the target may impact the intercept velocity and direction for the chaser.

Driver Modification

The closed-loop control loop for the flight controls onboard were not modified. The driver employs stock velocity and turns that impose transfer and rotational velocity parameters (that act as the governor for safety purposes) that the robot is capable of exceeding. Other efforts to implement a controller for the Parrot AR.Drone indicated that it may become necessary to modify the stock driver to override undesirable safety mechanisms and short-cuts for mobility to to account for changes in air density and fluid dynamics as a result of "rotor wash" or other phenomena that occurred during the testing that were difficult to account for as the user. Although many of the effects have been discussed in the helicopter literature reviewed, their influence on quadrotors and specifically the AR.Drone, has not

been comprehensively explored. In [28], the operator found that at moderate and increased velocities, the controller was highly variable in effectiveness, which he attributed to the driver.

Incorporation of a GUI

Additional improvements could be made with the user interface of the program. One might incorporate a GUI to show the user what the algorithm "sees." This would be especially useful once a robust detector has been implemented.

APPENDIX: Associated Programs and Python Code

The algorithm is executed by running `color_vision_node_HSV1_record_data.py`. The results are computed in Matlab. Each section uses a unique master file and a common function. The master file for color thresholding analysis is `Master_Data_Color4.m`. The master file for filtering analysis is `Master_Data_Filter.m`. The master file for edge detection analysis is `Master_Data_Edge2.m`. The underlying function is `Data_Run_EDIT.m`.

All source code and files used is available for download at: `https://wiki.nps.edu/display/~thchung/Resources`

THIS PAGE INTENTIONALLY LEFT BLANK

References

[1] "Assessments of Selected Weapon Programs," Washington, D.C., Tech. Rep. March, 2013. [Online]. Available: http://www.gao.gov/products/GAO-13-294SP

[2] "Unmanned Systems Integrated Roadmap FY2013-2038," Department of Defense, Washington, D.C., Tech. Rep., 2014. [Online]. Available: http://www.defense.gov/pubs/DOD-USRM-2013.pdf

[3] "Unmanned Systems Integrated Roadmap FY2011-2036," Department of Defense, Washington, D.C., Tech. Rep., 2011. [Online]. Available: http://www.acq.osd.mil/sts/docs/UnmannedSystemsIntegratedRoadmapFY2011-2036.pdf

[4] T. Mori and S. Scherer, "First results in detecting and avoiding frontal obstacles from a monocular camera for micro unmanned aerial vehicles," in *2013 IEEE International Conference on Robotics and Automation*. Karlsruhe, Germany: IEEE, May 2013, pp. 1750–1757. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6630807

[5] R. C. N. P. S. Harney, "Conventional Radars. I. Implementation," in *Combat Systems*, Monterey, CA, 2004, vol. 2, ch. 1, pp. 5–92.

[6] L. Jewell and P. Clancey, "Operational Limitations of Shipborne Radar," Washington, D.C., pp. 111–130, 1945. [Online]. Available: http://www.ibiblio.org/hyperwar/USN/ref/RADONEA/COMINCH-P-08-06.html

[7] "CV NATOPS MANUAL," U.S. Department of the Navy, Washington, D.C., Tech. Rep., 2007.

[8] "NATOPS General Flight and Operating Instructions," Department of the Navy, Washington, D.C., Tech. Rep., 2009. [Online]. Available: http://doni.daps.dla.mil/Directives/03000NavalOperationsandReadiness/03-700FlightandAirSpaceSupportServices/3710.7U.pdfhttp://doni.daps.dla.mil/

[9] S. J. Freedberg Jr., "X-47B Drone & Manned F-18 Take Off & Land Together In Historic Test « Breaking Defense - Defense industry news, analysis and commentary," Aug. 2014. [Online]. Available: http://breakingdefense.com/2014/08/x-47b-drone-manned-f-18-take-off-land-together-in-historic-test/

[10] I. F. Mondragón, P. Campoy, C. Martinez, and M. Olivares, "Omnidirectional vision applied to Unmanned Aerial Vehicles (UAVs) attitude and heading estimation," *Robotics and Autonomous Systems*, vol. 58, no. 6, pp. 809–819, Jun. 2010. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0921889010000576

[11] Y.-C. Liu and Q.-h. Dai, "A survey of computer vision applied in aerial robotic vehicles Yu-chi," in *2010 International Conference on Optics, Photonics and Energy Engineering*, no. 201. Beijing, China: IEEE, 2010, pp. 277–280.

[12] K. Khateeb, M. Awang, and O. Khalifa, "Intelligent auto tracking in 3D space by image processing," in *2008 IEEE International Conference on Robotics and Biomimetics*. Bangkok: IEEE, Feb. 2009, pp. 1744–1749. [Online]. Available: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=4913265

[13] S. B. Goldberg, M. W. Maimone, and L. Matthies, "Stereo vision and rover navigation software for planetary exploration," in *IEEE Aerospace Conference Proceedings,*, vol. 5. Big Sky, MT: IEEE, Mar. 2002, pp. 2025–2036. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1035370

[14] M. Achtelik, M. Achtelik, S. Weiss, and R. Siegwart, "Onboard IMU and monocular vision based control for MAVs in unknown in- and outdoor environments," in *IEEE International Conference on Robotics and Automation*. Shanghai: IEEE, May 2011, pp. 3056–3063. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5980343

[15] A. E. Johnson, Y. Cheng, and L. H. Matthies, "Machine Vision for Autonomous Small Body Navigation," in *Aerospace Conference Proceedings*, vol. 7, no. 8 18. Big Sky, MT: IEEE, Mar. 2000, pp. 661–671. [Online]. Available: http://ieeexplore.ieee.org/xpl/login.jsp?tp=\&arnumber=879333\&url=http\%3A\%2F\%2Fieeexplore.ieee.org\%2Fxpls\%2Fabs_all.jsp\%3Farnumber\%3D879333

[16] L. Doitsidis, S. Weiss, A. Renzaglia, M. W. Achtelik, E. Kosmatopoulos, R. Siegwart, and D. Scaramuzza, "Optimal surveillance coverage for teams of micro aerial vehicles in GPS-denied environments using onboard vision," in *Autonomous Robots*, vol. 33, no. 1-2. Taipei, Taiwan: Springer Science+Business Media, LLC, Mar. 2012, pp. 173–188. [Online]. Available: http://link.springer.com/10.1007/s10514-012-9292-1

[17] R. Jurriaans, "Optical Flow Based Obstacle Avoidance for Real World Autonomous Aerial Navigation Tasks," Ph.D. dissertation, University of Amsterdam, 2011. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.415.5558\&rep=rep1\&type=pdf

[18] E. N. Johnson, A. J. Calise, A. R. Tannenbaum, A. J. Yezzi, Jr, S. Soatto, G. Barbastathis, and N. Hovakimyan, "Active-Vision Control Systems for Complex Adversarial 3-D Environments," Georgia Institute of Technology, Arlington, VA, Tech. Rep., Mar. 2009. [Online]. Available: http://oai.dtic.mil/oai/oai?verb=getRecord\&metadataPrefix=html\&identifier=ADA532870

[19] F. Kendoul, I. Fantoni, and K. Nonami, "Optic flow-based vision system for autonomous 3D localization and control of small aerial vehicles," in *Robotics and Autonomous Systems*, vol. 57, no. 6-7. Elsevier B.V., Jun. 2009, pp. 591–602. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0921889009000396

[20] J. Courbon, Y. Mezouar, N. Guénard, and P. Martinet, "Vision-based navigation of unmanned aerial vehicles," in *Control Engineering Practice*, vol. 18, no. 7. Elsevier Ltd., Jul. 2010, pp. 789–799. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0967066110000808

[21] F. Ruffier and N. Franceschini, "Optic flow regulation: the key to aircraft automatic guidance," in *Robotics and Autonomous Systems*, vol. 50, no. 4. Elsevier B.V., Mar. 2005, pp. 177–194. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0921889004001733

[22] F. Faion, M. Baum, and U. Hanebeck, "Tracking 3D shapes in noisy point clouds with Random Hypersurface Models," in *Information Fusion (FUSION)*, vol. 5. Singapore: IEEE, Jul. 2012, pp. 2230–2235. [Online]. Available: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=6290575

[23] A. Ajay and D. Venkataraman, "A survey on sensing methods and feature extraction algorithms for SLAM problem," *Inernational Journal of Computer Science, Engineering and Applications*, vol. 3, no. 1, pp. 59–64, Mar. 2013. [Online]. Available: http://arxiv.org/abs/1303.3605

[24] Y. Freedman, B. Shpunt, A. Machline, and Y. Arieli, "Depth Mapping Using Projected Patterns," p. 13, May 2010. [Online]. Available: http://www.google.com/patents/US20100118123

[25] K. Khoshelham and S. O. Elberink, "Accuracy and resolution of Kinect depth data for indoor mapping applications." in *Sensors*, vol. 12, no. 2. Basel: Molecular Diversity Preservation International, Jan. 2012, pp. 1437–1454. [Online]. Available: http://www.mdpi.com/1424-8220/12/2/1437/htm

[26] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 647–663, Apr. 2012. [Online]. Available: http://ijr.sagepub.com/content/31/5/647.short

[27] R. M. Howard and I. Kaminer, "Survey of unmanned air vehicles," in *American Control Conference, ...*, vol. 5, no. June. Seattle, WA: IEEE, Jun. 1995, pp. 2950–2953. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=532054

[28] G. M. Hoffmann, H. Huang, S. L. Waslander, and C. J. Tomlin, "Quadrotor Helicopter Flight Dynamics and Theory and Experiment," in *AIAA Guidance, Navigation and Control Conference and Exhibit*, vol. 6461. Hilton Head, South Carolina: American Institute of Aeronautics and Astronautics, Incorporated, Aug. 2007, pp. 1–20. [Online]. Available: http://arc.aiaa.org/doi/pdf/10.2514/6.2007-6461

[29] S. D. Fleischer, "Bounded-error vision-based navigation of autonomous underwater vehicles," Ph.D. dissertation, Stanford University, Palo Alto, CA, 2000.

[30] S.-M. Oh, "Nonlinear estimation for vision-based air-to-air tracking," Ph.D. Dissertation, Georgia Institute of Technology, Atlanta, GA, 2007. [Online]. Available: https://smartech.gatech.edu/handle/1853/19882http://hdl.handle.net/1853/19882

[31] L. Coutard and F. Chaumette, "Visual detection and 3D model-based tracking for landing on an aircraft carrier," in *IEEE International Conference on Robotics and Automation*. Shanghai: IEEE, May 2011, pp. 1746–1751. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5979771

[32] "About Parrot," 2014. [Online]. Available: http://blog.parrot.com/about/

[33] S. Piskorski, N. Brulez, P. Eline, and F. D'Haeyer, "Parrot AR.Drone developer guide," pp. 1–105, May 2012. [Online]. Available: http://www.msh-tools.com/ardrone/ARDrone_Developer_Guide.pdf

[34] "Software," 2014. [Online]. Available: http://www.vicon.com/Software/Tracker

[35] "OpenCV (Open Source Computer Vision)," 2014. [Online]. Available: http://opencv.org/

[36] "ROS," 2014. [Online]. Available: http://www.willowgarage.com/pages/software/ros-platform

[37] B. Gerkey, "Open Source Robotics Foundation," 2014. [Online]. Available: http://www.osrfoundation.org/

[38] "ROS," 2014. [Online]. Available: http://www.ros.org/

[39] "Bags," 2014. [Online]. Available: http://wiki.ros.org/Bags

[40] D. Thomas, "rosbag," Dec. 2013. [Online]. Available: http://wiki.ros.org/rosbag

[41] R. C. Gonzalez, R. E. Woods, and S. L. Eddins, *Digital Image Processing Using Matlab*, 1st ed., M. Horton, Ed. Upper Saddle River, NJ: Pearson Prentice Hall, 2004.

[42] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt, and J. M. Ogden, "Pyramid methods in image processing," *RCA Engineer*, vol. 29, no. 6, pp. 33–41, Nov. 1984.

[43] C. Harris and M. Stephens, "A combined corner and edge detector," in *Proceedings of the 4th Alvey Vision Conference*. Plessey Company plc., 1988, pp. 147–151. [Online]. Available: http://courses.daiict.ac.in/pluginfile.php/13002/mod_resource/content/0/References/harris1988.pdf

[44] T. Drummond and E. Rosten, "Machine learning for high-speed corner detection," in *9th European Conference on Computer Vision*, ser. Lecture Notes in Computer Science, A. Leonardis, H. Bischof, and A. Pinz, Eds., vol. 3951, no. 3. Berlin, Heidelberg: Springer Berlin Heidelberg, May 2006, pp. 430–443. [Online]. Available: http://www.springerlink.com/index/10.1007/11744023

[45] V. Ghadiok, J. Goldin, and W. Ren, "On the design and development of attitude stabilization, vision-based navigation, and aerial gripping for a low-cost quadrotor," *Autonomous Robots*, vol. 33, no. 1-2, pp. 41–68, Aug. 2012. [Online]. Available: http://link.springer.com/10.1007/s10514-012-9286-zhttp://download.springer.com/static/pdf/653/art\%253A10.1007\%252Fs10514-012-9286-z.pdf?auth66=1389456217_71e8e2fd4c7eb9c5f06040243a5d286e\&ext=.pdf

[46] R. Jain, R. Kasturi, and B. G. Schunck, "Image Filtering," in *Machine Vision*, 1st ed., E. M. Munson, Ed. New York: McGraw-Hill, Inc., 1995, ch. 4, pp. 112–139. [Online]. Available: http://www.cse.usf.edu/~r1k/MachineVisionBook/MachineVision.pdfhttp://www.cse.usf.edu/~r1k/MachineVisionBook/MachineVision.files/MachineVision_FrontMatter.pdf

[47] D. Buede, *The Engineering Design of Systems Models and Methods*, 2nd ed. Hoboken, NJ: John Wiley & Sons, Inc, 2009.

[48] J. R. Smith and S.-F. Chang, "Single color extraction and image query," in *International Conference on Image Processing*, vol. 3. Washington, D.C.: IEEE, Oct. 1995, pp. 528–531. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=537688

[49] J.-s. Lee, "Digital image enhancement and noise filtering by use of local statistics," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-2, no. 2, Naval Research Laboratory. Washington, D.C.: IEEE, Mar. 1980, pp. 165 – 168. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=\&arnumber=4766994

[50] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Sixth International Conference on Computer Vision*, vol. 6, Stanford University. Bombay:

IEEE, Jan. 1998, pp. 839–846. [Online]. Available:
http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=710815

[51] "Image filtering," Apr. 2014. [Online]. Available:
http://docs.opencv.org/modules/imgproc/doc/filtering.html

[52] J. Canny, "A Computational Approach to Edge Detection," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, Massachusetts Institute of Technology, Cambridge, MA. IEEE, Nov. 1986, pp. 679–698. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4767851

[53] J. Li and S. Ding, "A research on improved Canny edge detection algorithm," in *Applied Informatics and Communication International Conference*, ser. Communications in Computer and Information Science, J. Zhang, Ed., vol. 228. Xi'an: Springer, Aug. 2011, pp. 102–108. [Online]. Available:
http://www.springerlink.com/index/10.1007/978-3-642-23223-7

[54] "Canny Edge Detection — OpenCV-Python Tutorials 1 documentation." [Online]. Available: http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html

[55] B. Wang and S. Fan, "An Improved CANNY Edge Detection Algorithm," in *2009 Second International Workshop on Computer Science and Engineering*, vol. 1. IEEE, 2009, pp. 497–500. [Online]. Available:
http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5403419

[56] G. Klette, "Skeletons in Digital Image Processing," University of Auckland, Tamaki, Tech. Rep., Jul. 2002. [Online]. Available:
http://citr.auckland.ac.nz/techreports/2002/CITR-TR-112.pdfhttp://hdl.handle.net/2292/2853

[57] S. Ganapathy, "Decomposition of transformation matrices for robot vision," *Pattern Recognition Letters*, vol. 2, no. 6, pp. 401–412, Dec. 1984. [Online]. Available:
http://www.sciencedirect.com/science/article/pii/0167865584900072

[58] N. Amin, "Automatic perspective correction for quadrilateral objects," Jan. 2013. [Online]. Available: http://opencv-code.com/tutorials/automatic-perspective-correction-for-quadrilateral-objects/

[59] "13.1. csv — CSV file reading and writing," 2014. [Online]. Available:
https://docs.python.org/2/library/csv.html

[60] J. Matas, C. Galambos, and J. Kittler, “Robust detection of lines using the progressive probabilistic Hough transform,” *Computer Vision and Image Understanding*, vol. 78, no. 1, pp. 119–137, Apr. 2000. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1077314299908317

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California