A Real-Time Vision System for a Semi-Autonomous Surface Vehicle

by

Collin Walker

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved July 2014 by the
Graduate Supervisory Committee:

Baoxin Li, Chair
Pavan Turaga
David Claveau

ARIZONA STATE UNIVERSITY

August 2014

ABSTRACT

In the sport of competitive water skiing, the skill of a human boat driver can affect athletic performance. Driver influence is not necessarily inhibitive to skiers, however, it reduces the fairness and credibility of the sport overall. In response to the stated problem, this thesis proposes a vision-based real-time control system designed specifically for tournament waterski boats. The challenges addressed in this thesis include: one, the segmentation of floating objects in frame sequences captured by a moving camera, two, the identification of segmented objects which fit a predefined model, and three, the accurate and fast estimation of camera position and orientation from coplanar point correspondences. This thesis discusses current ideas and proposes new methods for the three challenges mentioned. In the end, a working prototype is produced.

LEGAL

Patent Applied For.

TABLE OF CONTENTS

Page

CHAPTER Page

CHAPTER Page

CHAPTER Page

CHAPTER Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Figure Page

Figure Page

1 INTRODUCTION

Vision continues to be an instrumental sense in many forms of robotic automation. In the field of autonomous vehicles, researchers have made an ongoing effort to integrate optical sensors into their autonomous machines. While researchers often utilize other sensors like radar, LIDAR, GPS, etc. [1], there is still a large effort from the community to use camera based systems due to their low cost. In this thesis, the design of a semi-autonomous surface vehicle, constructed to operate under a specific set of constraints, is presented. The design consists of a segmentation, identification, and pose estimation module. The chapters of the thesis discuss each component.

The system presented is developed in response to unresolved issues in the sport of slalom waterskiing. To understand the issues, one must first develop a basic understanding of the sport itself. With basic knowledge of the sport, one can identify how certain human influences, specifically human drivers, can cause issues in terms of competition and training. Once a basic understanding of skiing is developed, it will be easy to understand the need for an automated system within the sport. The following sections introduce the sport, discuss the importance of a boat driver in slalom competition, and provide a high level outline of system design.

1.1 Slalom Waterskiing

The International Water Ski Federation (IWSF) is the governing body of competitive water skiing throughout the world. The IWSF specifically defines the objective of the slalom event as “The contestant shall follow the towboat through the entrance gate of the slalom course, pass around the outside of any or all six buoys and proceed through the far end gate” [2]. Figure1.1 illustrates a slalom course. The red

dotted line shows the centerline of the slalom course and expected path of the towboat. The bold blue line shows the expected path of a skier.

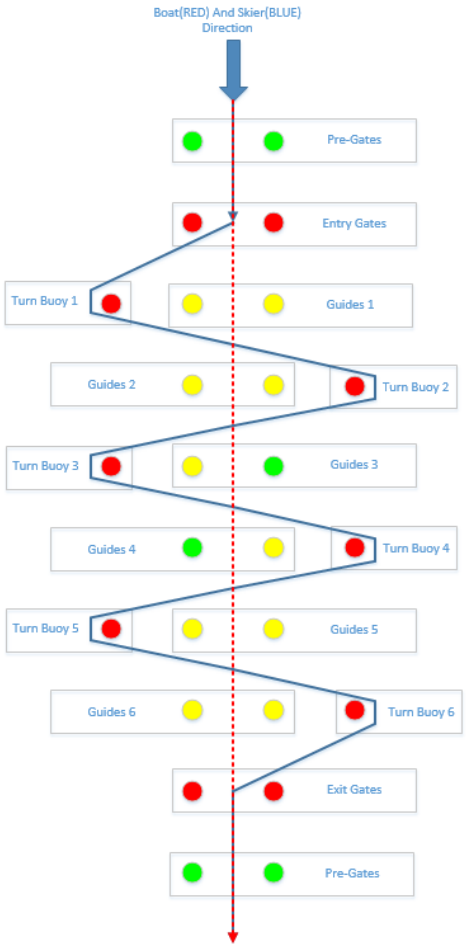


Figure 1.1: Boat and Skier Paths.

When a skier successfully navigates the slalom course he or she has completed a pass. If a skier completes a pass, then the skier challenges himself or herself further by: (1) increasing the boat speed by a predefined amount, or (2) shortening the rope by some predetermined amount. A skier will continue to challenge himself or herself with the previous two options until her or she finds the limit of his or her ability. A skier will

normally run anywhere from four to eight passes before tiring. A collection of passes is referred to as a set.

1.2 Importance of the Boat Driver

The current method for boat steering control is by human driver. The ability to properly drive a boat, in terms of slalom driving, is a skill itself. If one wants to drive a sanctioned waterski tournament, then it is usually required that he or she have a drivers rating. Ratings are acquired by successfully completing driving clinics. Actual invitations to drive at tournaments are only given to those who have both the required rating and a positive reputation. The types of people who receive invitations to drive at tournaments are usually those who have been involved with the sport for anywhere from years to decades. Despite the restrictions set by the IWSF and efforts of tournament coordinators to ensure high quality veteran drivers, the drivers are still human. A human driver may suffer from distractions, boredom, and fatigue, which may inhibit performance. When a boat driver's performance suffers, a skier's performance may also suffer due to both physical and psychological issues associated with low quality boat driving.

1.2.1 Physical Effect of the Boat Driver

As defined by the IWSF rulebook, the distance from the centerline of the slalom course to the center of a turn buoy is 11.5 meters. In some situations, an experienced skier may attempt to complete passes where the rope length is near or even less than 11.5 meters. In these types of situations, it is essential that the boat be as close to the slalom course centerline as possible. If the boat path deviates from the center line of the slalom course, then the skier may have an unfair advantage or disadvantage.

Along the lines of boat path deviation, the IWSF has a specification which outlines the maximum allowable deviation of a boat path when attempting to set a world record. When a skier completes a pass that may constitute a world record, the boat path of all passes in the set are reviewed. A record is only accepted if all boat path deviations in all passes of the set fall within the allowable tolerances set by the IWSF. In some cases, records are rejected due to an out of tolerance boat path. At this time, supervised software is the current method of boat path examination. The software works with video captured from cameras located on the centerline of the slalom course outside of the course itself. Figure 1.2 illustrates camera location for boat path validation.

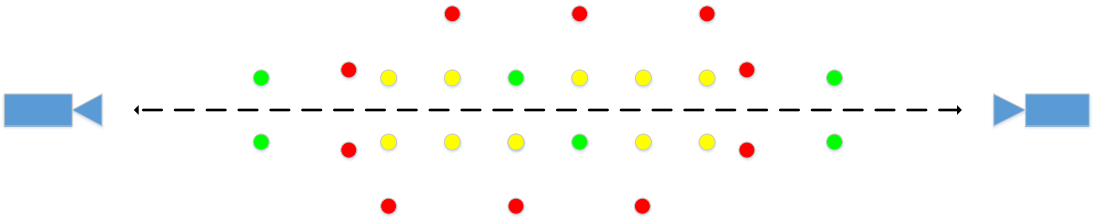


Figure 1.2: Camera Setup for Record Validation.

1.2.2 Psychological Effect of the Boat Driver

As with any sport, becoming a talented skier involves a fitness element and a skill element. The skill element is gained through repetition, in this case, completing slalom passes. As a skiers total water time increases, his or her awareness of minute changes also increases. Changes of this sort include differences in the composition of the lake water, ski settings, boat wakes, boat drivers, etc. As with any sport, when the conditions are familiar, an athlete can expect consistent performance. When conditions vary, an athlete's performance may suffer.

When a boat driver with a bad reputation, or no reputation at all, is driving a skier, the skier may suffer psychological issues which keep he or she from focusing on the task at hand. This effect may be true regardless of the drivers actual ability or performance. For example, a skier may perform well behind a highly reputable driver who is actually driving bad, and perform poorly behind a low reputation driver who is driving perfectly. The effect is entirely psychological.

1.3 Problem Statement

The issues surrounding tournament slalom driving all relate to the errors made by human drivers. By replacing human drivers with some type of automated system, all previously discussed issues may be circumvented. With an automated system, any person would be a capable driver, and any skier would be confident in the ability of any machine assisted human. This thesis aims at providing a basic design for the suggested system.

1.4 Overall Design

The overall goal of current and future research is to create a machine that resides inside the boat and can autonomously steer the boat when it is passing through the slalom course. The decision to place all components of the system inside the boat is made in order to avoid issues that come with using external components such as the lack of infrastructure at lakes, the chance of vandalism to shore installed components, and the challenge of wireless communication between external components and components residing in the boat.

1.4.1 Selection of Sensors

The IWSF tournament rulebook specifies the allowable error in buoy position for each buoy in a slalom course. Since there is an allowable error in buoy placement, it is

expected that position data generated from buoys will also be erroneous. Furthermore, it is expected that position estimates generated from many buoys will have less error than estimates generated from fewer buoys. Due to the need for an accurate estimate, and thus the desire for more buoys, it is necessary to examine sensors that either perceive many buoys, or do not rely on slalom course buoys at all. Some sensors are examined in terms of cost, resolution, and required processing power.

GPS is an example of a sensor that does not rely on slalom course buoys. It is inexpensive, robust, and thus a potential sensor for the stated steering problem. Current commercial waterski boat speed control systems rely on GPS and accelerometers. The control systems have been successful, but the sensors used are not appropriate for the stated steering problem. The accuracy of a GPS accelerometer combination does not meet the accuracy requirements of the proposed system at this time.

In terms of sensors that utilize buoy information, LIDAR, SONAR, and camera based systems are examined. LIDAR or SONAR based solutions may offer a means of simple and robust tracking for buoys located near the sensor, however, they lack the cost effective ability to sense buoys at a distance. Sensing objects at a distance is an ability that camera based systems can achieve. In terms of camera based systems, there are a variety of wavelengths that an optical sensor can detect. The most obvious optical solution is one which uses an infrared or thermal camera. In this type of solution, a water surface appears in captured frames as one temperature, and buoys appear as a separate temperature. The identification of floating objects is transformed to a simple task. The issue with high resolution IR cameras, however, is the considerable cost.

Cameras which rely on the visual spectrum of EMR avoid the cost issues associated with IR cameras. Both monocular and stereo systems seem to be the most appropriate sensors for the stated problem, however, stereo systems appear to have issues that monocular camera systems avoid. Some difficulties with a stereo setup are: one, issues associated with calibrating a stereo rig, two, the requirement for more equipment, and three, the extra computation required for stereo processing. With all considerations, a monocular system appears to be the best sensor to use.

A camera, computer, and a mechanical interface provide all necessary hardware for the system and all physical components can reside inside of a boat itself. Figure 1.3 displays the basic design. The software components driving the first three "Computer" modules, segmentation, identification, and pose estimation, are the main topics of this thesis.

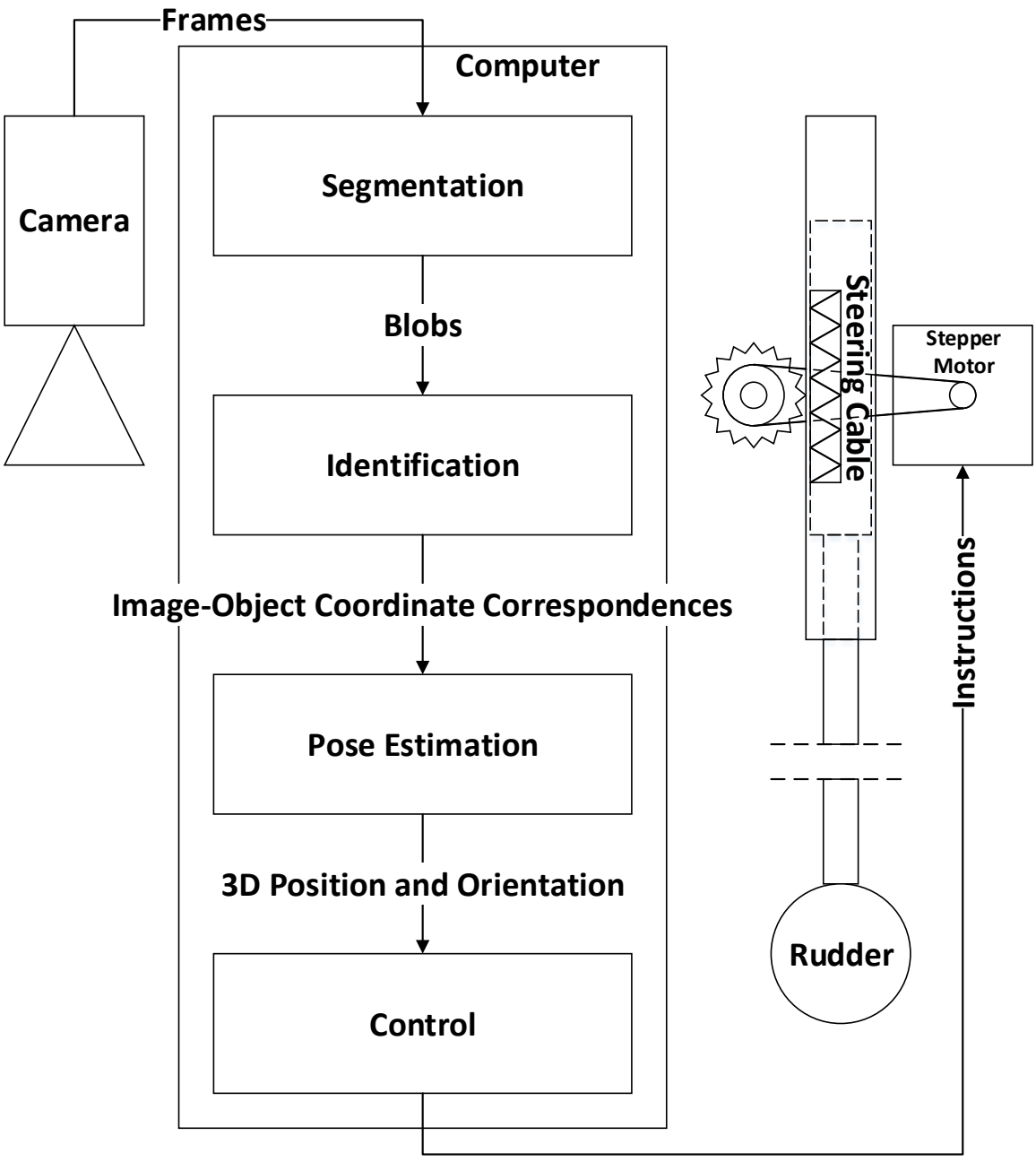


Figure 1.3: A Diagram of the Entire System.

1.5 Equipment

All experiments discussed in the thesis use the same equipment. The boat selected for experimentation is a 2006 Correct Craft Ski Nautique. Nautique is an competitive

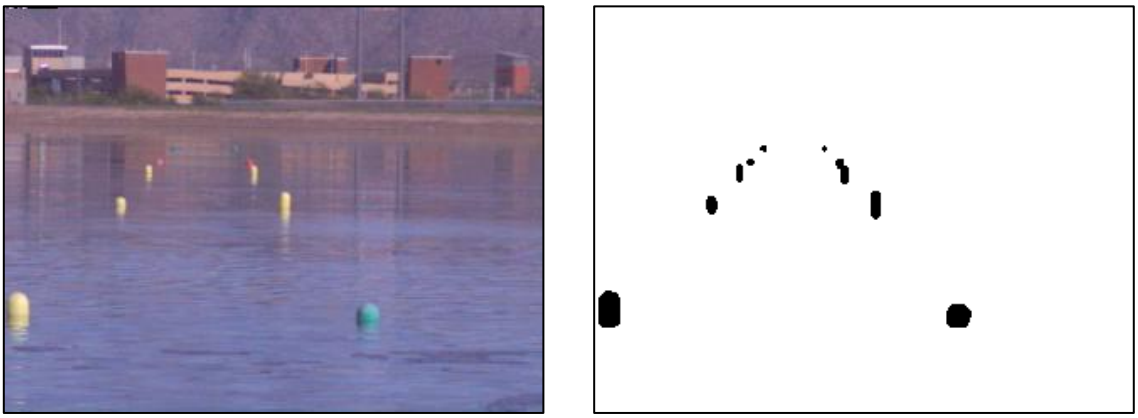
brand in terms of tournament water ski boats. There are many design changes between the test vehicle and Nautique's current tournament water ski boat model, however, the changes are negligible in terms of the proposed control system. The computer used in all experiments is a Dell Vostro 1500. In terms of performance, the machine has 4 GB of RAM and an Intel Core 2 Duo CPU running each core at 2.0 GHz. The computer runs Ubuntu 12.10 as its OS and all implemented software is heavily dependent on OpenCV 2.4.8 as well as Boost 1.49.0.1. The selected camera is a Logitech Quickcam Pro 4000. The stock lens is replaced with an aftermarket lens whose focal length is 3025 pixels when the resolution of the captured frames is set to 320 x 240. The camera is accessed through OpenCV with the Linux V4L driver.

2 SEGMENTATION

2.1 Introduction

2.1.1 General Goal of the Module

The system uses a forward mounted camera to capture images of its environment. Each image, or frame, may contain regions of pixels representing buoys and regions of pixels corresponding to objects other than buoys. The system's first task, and the goal of the segmentation module, is to label pixels as representative of either buoys or non-buoy objects. Figure 2.1 (a) shows a typical frame captured from the system's camera. Figure 2.1 (b) displays an example of the segmentation mask the module aims to produce.



(a) (b)

Figure 2.1: Segmentation Module Goal.(a) A Typical Frame. (b) The Desired Segmentation Mask. In addition to accurately labeling incoming frames, the system must also perform the segmentation task in an efficient manner so that real-time constraints can be met.

2.1.2 Significance of the Module

The segmentation stage of processing is necessary as modules further in the processing pipeline, identification, and pose estimation, depend heavily on the blobs generated from the result of the segmentation stage's segmentation masks. Out of the

three major modules discussed in this thesis, the segmentation module requires the best performance outcome. Any errors made in the segmentation module will only propagate through the rest of the processing pipeline. Additionally, the overall frame rate of the complete system is determined entirely by the computationally intensive algorithms in the segmentation module.

To the machine vision community, the problem of segmenting floating objects on a aquatic surface is not new [3,4,5,6,7]. The knowledge found in developing solutions for the segmentation module furthers the body of knowledge associated with the task of segmenting floating objects. Furthermore, if one is to look at the segmentation module's goal from a slightly higher level of abstraction, then the problem becomes one of identifying and labeling pixel regions that satisfy a certain set of properties, a fundamental problem in computer vision.

2.1.3 Assumptions

Creating a generic solution to satisfy the stated goal which operates under any circumstance is appealing, however, the time requirements of completing such a task are outside the scope of this thesis. For this reason, the major focus of this chapter is finding a solution to a constrained version of the presented problem that operates under a justifiable set of assumptions. First, any incoming frame is expected to have enough perceptual contrast such that a human could perform the segmentation operation. At a minimum, a buoy should appear differently than that of the water surface that surrounds it. Second, it is assumed that the illumination of all important objects in an incoming frame sequence will not change dramatically in a small time period. This is a justifiable assumption as ski lakes are generally large bodies of water with no natural or manmade

light obstructions in the middle of the them. Third, it is assumed that the aquatic surface under consideration is relatively calm, i.e. no five foot waves or 60 mph winds. This assumption is acceptable as people generally prefer skiing on calm flat water during the windless parts of the day. Finally, it is expected that the position and color of the buoys satisfies the specification set forth in the IWSF tournament rule book [2].

2.1.4 Difficulty

The segmentation problem presented has two major difficulties associated with it. First, any usable solution is required to function on power-up regardless of current or previous lighting conditions. In other words, the system can only assume that the current lighting conditions fall within the range of conditions specified by the assumptions. Furthermore, the module must assume that each pass through the slalom course will present a lighting condition that is independent of all other passes. The system is not continuously running and therefore cannot rely on the commonly made assumption of slow illumination changes between passes consecutive runs. Second, any solution used needs to appropriately deal with shadows and reflections of shoreline objects like trees, houses, and lights.

2.2 Review of Related Literature

The problem of accurately segmenting a known grid of buoys for use in a real-time control system has not been directly dealt with in previous literature, however, work has been done on challenges similar in nature. Of the analogous problems, two tasks of similarity are the landing of a UAV [8,9,10,11,12,13] and the detection of floating mines, marine buoys, or floating objects on a sea surface [3,4,5]. Beyond methods developed specifically for these problems, the problem of visual lane marker detection shares many

similarities to the presented problem. In lane detection tasks [1], the algorithms usually try to segment regions of the road that follow certain appearance properties and fit a specified model. The remainder of this section presents a review of previously proposed solutions.

2.2.1 UAV Landing

The problem of landing a UAV is similar to the stated steering problem. Usually, the UAV must recognize either a runway or designated landing area and safely land on it. Many proposed systems use a multitude of sensors, yet some systems use visual sensors exclusively. Some vision-based systems are examined due to the considerable relationship between the UAV landing problem and the stated problem.

One attempt at UAV landing [8] is made by using a forward mounted camera and attempting to land the aircraft on a large dome shaped airbag. The authors decide that position estimates generated by a GPS are not accurate enough to guide an aircraft onto the airbag and turn their attention to visual processing. Since the landing zone is a large red dome, the authors process the images by first labeling pixels whose red component is larger than their green and blue components. This yields an object set containing both the landing airbag and other red objects. The moments of the objects are analyzed in order to discriminate between the dome shaped airbag and other red objects. Once the airbag is identified, control is achieved by applying control commands that orient the aircraft such that the targeted airbag is in the center of incoming frames. Another attempt at UAV landing is made in [9]. In this work, the authors attempt to land a helicopter type UAV on a planar landing pad. They use a specially designed pattern on the landing pad to ease the

complexity of visual computation. Both of these systems are successful vision-based control systems, however, they both require the addition of specially designed markers.

Other than systems that rely on specially made markers, some systems attempt to visually detect and track runways with no extra markers. These types of systems either use a reference image[10,11] or detect runway edges [12,13]. Since The proposed system is intended to work with any slalom course, the proposed image registration methods may not be useful. Furthermore, a slalom course consists of a series of buoys and does not contain the edges that [12,13] require.

2.2.2 Floating Mine Detection

A problem that is most similar to the goals of the segmentation module is the detection of floating mines. There is a body of research associated with this subject and some of the methods presented in literature are presented in the following section.

2.2.2.1 Block-Level Segmentation

On the problem of floating mine detection, the authors of [5] approach the task under the assumption that the appearance of a floating mine will differ from the appearance of its surrounding area. Because they do not know the exact size of the mine they are trying to detect, they process each frame on multiple scales by down-sampling incoming frames by a factor of two. Pixel level segmentation at each scale is achieved by first dividing their frame and down-sampled frames into 80x80 pixel blocks. Then, after block formation, the mean, b_k, and standard deviation, σ_k, of each block, $\boldsymbol{B_k}$, is calculated. Finally the block level statistics are used to identify contrasting pixels within each block. They use

$$\boldsymbol{C}(r,c) = \begin{cases} 1 \; if \; |\boldsymbol{P}(r,c) - \; b_k(r,c)| \geq n_\sigma * \sigma_k \\ 0 \; if \; |\boldsymbol{P}(r,c) - \; b_k(r,c)| < n_\sigma * \sigma_k \end{cases}$$

Equation 2.1

to create a binary segmentation mask representing candidate pixels. In the equation $\boldsymbol{P}(r,c)$ is the intensity of the pixel at position (r, c), and $\boldsymbol{C}(r,c)$ is the candidate mask at position (r, c). The user assigned value of n_σ allows the algorithm to select more or less candidate pixels. If the value of n_σ is large, fewer pixels are selected from the block. If n_σ is small, more pixels are selected.

After candidate pixel selection is performed, the authors use a motion correspondence algorithm they name "motion projection" as well as spatial-temporal smoothening to refine their results. As the motion projection algorithm and spatial-temporal smoothing require multiple frames as input, they are outside the scope of the module's frame-by-frame task and not discussed. Furthermore, the authors do not include details on how they fuse results processed at different scales.

Overall, the algorithm achieves a high level of accuracy in terms of avoiding false negatives at each scale. Two key ideas behind the presented algorithm are first, the use of multiple scales in the process of pixel level segmentation, and second, the use of diverse features, contrast and motion estimates in their case, to generate more robust results.

2.2.2.2 Pixel-Level Segmentation

Outside of the work in [5] the authors in [3] present an evaluation of pixel-level floating object detection methods. They discuss using background subtraction techniques such as a running mean [15], running single Gaussian [15,16], Gaussian mixture model [17], and Extended Gaussian mixture model [18,19] as the means to classify pixels as

either representative of sea surface or floating mine. In addition to testing statistical background based methods, the authors also discuss and experiment with the visual background extractor algorithm [20,21] and behavior subtraction [22]. Some of the algorithms tested in [3] are discussed in this thesis, but only a select few are tested due to results in [3,14,23,24] as well as analysis of the algorithms. Some of the methods will be discussed in the following sections.

2.2.2.3 Behavior Subtraction

Upon examination of the proposed method in [22] the authors explicitly provide frame sequences of aquatic surfaces as a means of demonstrating the strength of the algorithm and its ability to correctly classify glint and other temporary activity. At first glance the algorithm seems to be a promising solution to the module's problem but after inspection it can be seen that it is unsuitable for the presented problem due to its use of the temporal domain.

The behavior subtraction algorithm works similarly to any background subtraction algorithm but employs a temporal window as a means of improving robustness. It creates a model of normal background activity by using a two step approach. First, every frame in a training sequence of frames undergoes background subtraction in order to segment moving foreground objects from the background. Next, the segmentation masks, or motion detection labels, created by the background subtraction process are converted into "behavior descriptors" based on the cardinality of detections at each pixel location over a temporal window. Simply put, behavior descriptors are created for each pixel based on the number of times a pixel is marked as foreground over a window of frames. Figure 2.2 shows an example of converting a

training sequence for one pixel into a behavior descriptor. In this case, the "maximum activity" descriptor could be generated by looking for the maximum number of positive detections within the window for every possible window position.

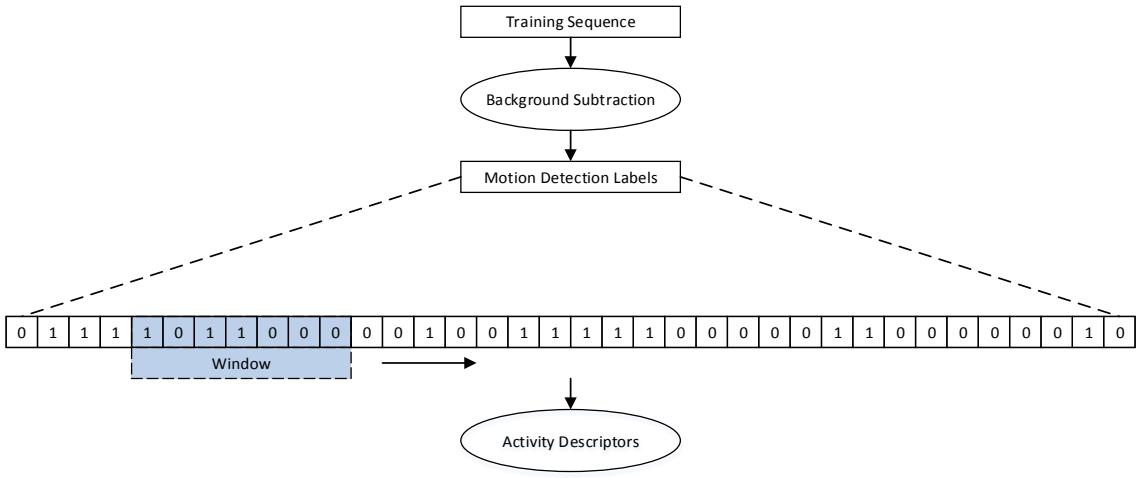


Figure 2.2: Generating Activity Descriptors from a Training Sequence for One Pixel.

The authors also propose the "average activity" descriptor. This descriptor is simply the number of positive detections over the entire training sequence divided by the number of frames in the sequence, or

$$\boldsymbol{B}(r,c) = \frac{1}{M}\sum_{j=1}^{M} \boldsymbol{L}(r,c,j)$$

Equation 2.2

where M is the number of frames in the training sequence, **B** is the behavior descriptor image, **L** is the motion labels produced from a background subtraction algorithm, and r, c and j are the row, column and frame indexes.

Once training is complete, an incoming frame sequence can be turned into synonymous behavior descriptors using methods similar to those used to create the behavior descriptors from the training sequence. Again, they are based on cardinality of

detections over a window of recent frames. Without getting into further detail about behavior descriptors, or how the final segmentation mask is produced, it can be seen that the algorithm relies on the use of a temporal window. The use of the temporal window mechanism is undesirable for the task of buoy segmentation. It causes the detections of abnormal behavior, or objects on the water's surface, to arrive at a later time. A delayed response, and therefore a pose estimate based on old data could be used by the control module in the system. It is assumed, however, that a control module operating on younger data will perform better and require a simpler design than a module operating on older data.

2.2.2.4 Pixel-Level Background Subtraction

The authors of [3] focus on background subtraction techniques as a means of segmenting floating mines from sea surface. For this reason, the background subtraction techniques mentioned in [3] as well as others mentioned in the surveys [14,23,24] are examined. In the subsequent text $\boldsymbol{I_t}$ will represent the incoming frame at time t. $\boldsymbol{B_t}$ will represent an image of the background model at time t, and the resulting segmentation mask produced by each method will be referenced as $\boldsymbol{F_t}$.

2.2.2.4.1 Image Differencing

Probably the simplest approach to background subtraction is image differencing. As the name suggests, the foreground mask $\boldsymbol{F_t}$ is created by simply differencing the current frame $\boldsymbol{I_t}$ with an image of the current background model $\boldsymbol{B_t}$. A pixel is said to be part of the foreground if its difference from the corresponding value in the background is larger than some threshold T. Equation 2.3 gives the formal definition of image differencing.

$$\boldsymbol{F_t} \leftarrow (|\boldsymbol{I_t} - \boldsymbol{B_t}| > T)$$

Equation 2.3

2.2.2.4.2 Adaptive Mean

One problem with image differencing is that it fails to supply a method of generating or updating a suitable background model. An improvement to image differencing is made by modeling background pixels as continuously updated averages of all previous frames. A mention of the adaptive averaging filter described in this text is made in [14]. A mention of a similar method which takes the average over a window of frames is made in [25]. A background model that is updated with a running mean is updated by adding the weighted value of a current pixel to the value in the background model or

$$\boldsymbol{B_{t+1}} = \boldsymbol{\mu_{t+1}} = \alpha * \boldsymbol{I_t} + (1 - \alpha) * \boldsymbol{\mu_t}$$

Equation 2.4

In Equation 2.4, α is a learning factor that determines the rate at which the background model adapts to current conditions. This method is a step above image differencing as it allows the background image to adapt to events like illumination changes or background object position changes.

2.2.2.4.3 Adaptive Single Gaussian Model

Using an adaptive mean to model background pixels improves foreground segmentation but neglects to define the threshold at which an incoming pixel is said to be outside of the background model. This problem is alleviated by modeling each pixel as a continuously updated Gaussian distribution [15,16]. The adaptive single Gaussian

background model updates its mean using Equation 2.4 and updates its variance measure for each pixel using Equation 2.5.

$$\boldsymbol{\sigma_{t+1}^2} = a * (\boldsymbol{\sigma_t^2}) + (1 - \alpha) * (\boldsymbol{I_t} - \boldsymbol{\mu_t})^2$$

Equation 2.5

In Equation 2.5 α is a learning factor that controls the rate at which the background model adapts. In using the adaptive single Gaussian model of a background pixel, the foreground mask is generated by replacing the threshold in Equation 2.1

with a Mahalanobis distance or

$$\boldsymbol{F_t} \leftarrow \left(|\boldsymbol{I_t} - \boldsymbol{\mu_t}| > (c * \boldsymbol{\sigma_t})\right)$$

Equation 2.6

where c dictates how many standard deviations away from the mean an incoming value must be within to be considered background.

2.2.2.4.4 Adaptive Gaussian Mixture Model

Sometimes a single Gaussian distribution cannot completely describe background dynamics. To deal with this, a background is modeled as multiple Gaussian distributions [17]. A background described with the Gaussian mixture model uses k distributions to model the multiple color ranges a true background pixel can fall into. Each distribution has a mean μ, standard deviation σ, and weight ω. The Gaussian mixture model processes each incoming frame by first sorting all distributions for a pixel in order of decreasing evidence ω/σ. A new pixel value $\boldsymbol{I_t}(r, c)$ is compared to the sorted Gaussian distributions until a distribution that explains the value is found. If a match is found then the mean μ and standard deviation σ of only the matching distribution is updated with

Equation 2.4 and Equation 2.5 respectively. The weight of the matching distribution is updated with

$$\omega_{t+1} = a + (1 - a)\omega_t$$

Equation 2.7

Any unmatched distributions are updated so that their weights decrease. Equation 2.8 gives this updating function.

$$\omega_{t+1} = (1 - a)\omega_t$$

Equation 2.8

As consistent background values are added, the weights of the true background distributions increase while the variance of these distributions decrease resulting in higher evidence for true background distributions. In the case that a incoming value does not belong to any of the distributions, the distribution with the lowest evidence is replaced with a new distribution whose mean is the current pixel value, standard deviation is set high, and weight is set low.

Finally, the distributions that describe the background are selected with

$$\boldsymbol{B}_t = min\left(\sum_{k=1}^{k} \omega_k\right) > T$$

Equation 2.9

where the value of T is set to the percentage of the incoming data that is expected to be background. Once the background distributions for a pixel have been selected, the foreground mask is created from those pixels that match non-background distributions, or don't match any distributions at all.

2.2.2.4.5 Adaptive Median Filter

The adaptive median filter is used and tested in many works [23,24,26,27,28]. It is similar to the aforementioned pixel-based background subtraction methods, however, its updating mechanism works by incrementing or decrementing the value in the background model by one based on the incoming image as in Equation 2.10.

$$\boldsymbol{B_{t+1}} = \boldsymbol{B_t} \begin{Bmatrix} +1 \; if \; (\boldsymbol{I_t} < \boldsymbol{B_t}) \\ -1 \; if \; (\boldsymbol{I_t} > \boldsymbol{B_t}) \end{Bmatrix}$$

Equation 2.10

The incoming frame is then compared to the background and segmented using the thresholding method of Equation 2.3.

2.2.2.4.6 Visual Background Extractor

The Visual Background Extractor (ViBe) [20,21] is an approach to background subtraction that relies on a set of retained samples rather than statistical information. The algorithm assigns to each pixel a set **s** of n samples hereafter indexed as $\boldsymbol{S_q}(\boldsymbol{r}, \boldsymbol{c})$ where r and c are coordinates within sample q. The entire sample set (q = 1..N) is initialized by setting the values of $\boldsymbol{S_q}(r, c)$ to a randomly selected neighbor of the corresponding pixel in an initialization frame $\boldsymbol{I_0}$ or

$$\boldsymbol{S_q}(\boldsymbol{r}, \boldsymbol{c}) = \boldsymbol{I_0}\big(randNeighbor(r, c)\big)$$

Equation 2.11

The function $\mathrm{randNeighbor(r, c)}$ randomly selects one of pixel (r, c)'s 8-connected neighbors. Note that the $\mathrm{randNeighbor(r, c)}$ function is fired independently for each pixel-sample index.

Once the background model has been initialized, an incoming frame $\boldsymbol{I_t}$ is segmented by comparing the values of a pixel in $\boldsymbol{I_t}$ to all values in the sample set corresponding to the pixel. If the distance, between $\boldsymbol{I_t}(r, c)$ and some sample $\boldsymbol{S_q}(r, c)$ is below a threshold, then the pixel's positive background matches count increments. After comparing to all of the pixel's samples, if the number of matches is greater than a threshold, then the pixel is marked as background. If the number of matches does not break the threshold then the pixel is marked as foreground.

One distinct difference between this method and previously mentioned methods is its use of a conservative updating scheme and a spatial updating scheme. In a conservative updating scheme only data classified as background is used to update the background model. In a spatial updating scheme, an incoming pixel not only updates its corresponding background value, but also updates a neighboring pixel's corresponding background value. In the case of ViBe, if a pixel in an incoming frame is classified as background, then its value is used to update the corresponding pixel sample set at some randomly selected sample index. It is also used to update one of the corresponding pixel's randomly selected neighbors at a random sample index.

2.3 Proposed Methods

Methods available in literature for floating object detection have been presented. Methods not specifically purposed for floating object detection will now be presented with modifications for tackling the problem.

2.3.1 Difference of Gaussians

While difference of Gaussian (DoG) operators are not specifically edge detectors, they are related [29]. One benefit of the DoG operator is that it not only detects edges, but

also detects the weight associated with the structure of an edge [29]. Difference of Gaussian operators are also used, in conjunction with other mechanisms, for dynamic range compression in images [30]. Difference of Gaussian operators can also be viewed as band-pass filters in the spatial domain. These characteristics of the operator make it a potential solution to the module's problem. If the operator can be tuned to reject spatial frequencies unrelated to possible buoy sizes, then the operator has the potential of becoming a powerful frame-by-frame segmentation method. For this reason it is examined in further detail.

The simplest method of applying a difference of Gaussians operator is to first smooth an image with two differently sized Gaussian kernels. After smoothing, the operator output is found by taking the absolute difference of the output of the convolution between each kernel and the image. A segmentation mask can be generated by looking at the difference image and comparing it to a threshold.

One detail evident in all test sequences is that normal water surface activity produces images in which only changes parallel with the horizon are detected by the camera. Figure 2.3 shows an example of this property. In the figure, the original image appears normal, but filtering by a Sobel operator shows that the camera picks up disturbances that project onto the image plane as horizontal lines. Due to this observation, only Gaussian kernels built to detect vertical lines are selected for testing.

(a) (b) (c)

Figure 2.3: The Effect of Scene Geometry on Perceived Spatial Frequencies. (a) Original Image. (b) Output of a Sobel Filter Designed to Respond to Vertical Edges. (c) Output of a Sobel Filter Designed to Respond to Horizontal Edges.

2.3.2 Color Based Methods

All segmentation methods presented have depended on differences in color instead of color itself. Using color contrast rather than specific color is desirable as it relaxes the assumptions related to specific buoy colors and eases any concern related to camera specific intrinsic color calibration. Despite these attractive features, segmentation based on specific colors is still a viable method for image segmentation.

One of the stated assumptions is that the buoys will follow the color specification set forth in the current IWWF rulebook. Due to this, segmentation masks can be created by comparing pixel color values to accepted buoy color values. This is an approach that has been used to detect buoys in [6]. Using specific buoy colors can have advantages. For instance, if a segmentation method produces only one mask, and it is noisy, then the identification module may have a difficult time locking onto buoys. On the other hand, if multiple segmentation masks are created for each color and only one is noisy, then the other non-noisy masks can be used to acquire predictive knowledge about the environment and lock onto buoys in the noisy mask. Figure 2.4 shows this type of instance.

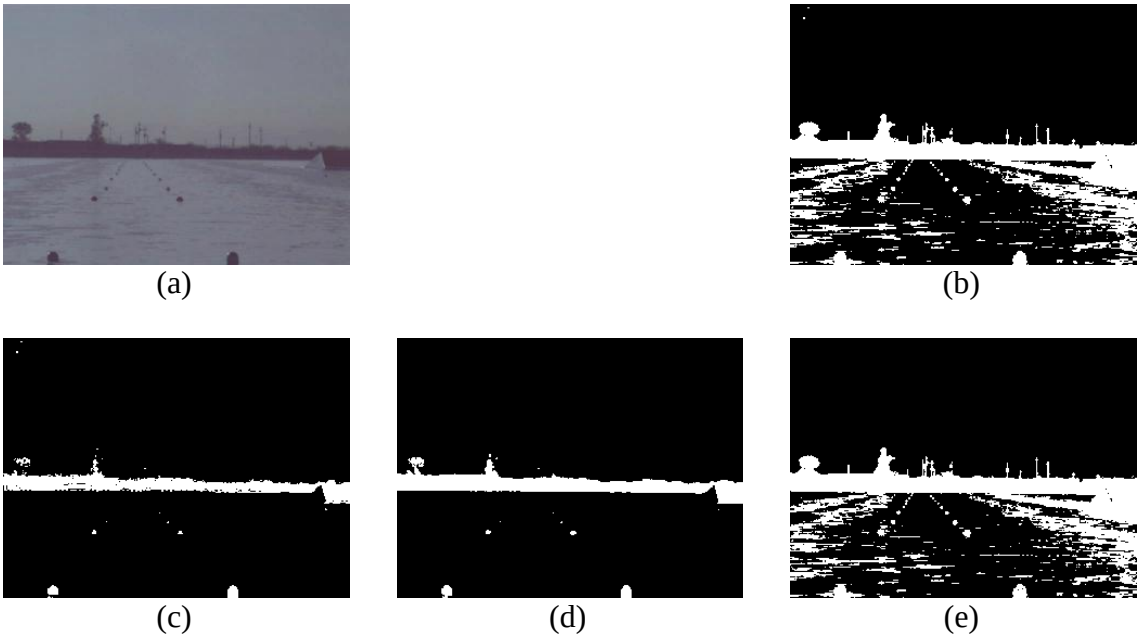


Figure 2.4: A Frame and Its Associated Color Segmentation Mask Outputs. (a) Original Image. (b) Combined Segmentation Mask. (c) Green Segmentation Mask. (d) Red Segmentation Mask. (e) Yellow Segmentation Mask.

The main question when using color as a feature for segmentation, is what color space is appropriate? In [31] the author argues that $L^*u^*v^*$ and $L^*a^*b^*$ are too susceptible to noise. In [32] an illumination invariant color space is used to avoid the misclassification of shadows. There a wide variety of proposed color spaces [33,34,35], and a variety of spaces have been utilized [36]. For simplicity RGB space is used in the following methods as frame data from the camera is retrieved as values in RGB space. In the subsequent text, two methods are presented based on color. The first is a classifier that searches for specific buoy colors and is trained prior to system operation. The second is a pixel classifier that uses water color and is updated online.

2.3.2.1 Offline Buoy Color

The assumptions state that the color of buoys will be that which is specified in the IWSF rulebook. Based on this assumption, a classifier can be created which labels each

pixel based on its similarity to known buoy colors. In order to developed such a classifier two questions must first be answered. First, what data structure is appropriate to precisely model the wide range of colors a buoy may appear as? And second, how does one populate the data structure?

2.3.2.1.1 Data Structure

The classifier is attempting to label pixels based on a set of color indexes in RGB space. In the background subtraction techniques presented, a minimal representation of colors was formulated using either second order statistical or similar approaches. This minimal-memory representation was necessary because each of the pixels needed an independent model. In the case of classifying buoy color, there is only one system wide model for each possible buoy color. For this reason, more memory resources can be allocated to the data structure used to hold buoy color. Instead of using space conserving statistical models, a bitset is utilized. It provides superior performance in precisely classifying pixels based on color.

A bitset is simply an array of bits. It is included in the C standard library. In the case of classifying color, a 256^3 bit large structure is used, one bit for each possible index in 8-bit per channel RGB space. As stated before, the module could use a second order statistics to model possible buoy colors, however, the entire dynamic range of buoy colors is not describable by one spectral band or by a simple shape in RGB space. A buoy can appear as a florescent color under sunny front lit conditions, a darker color under backlit conditions, or a less intense color under low light cloudy conditions. A bitset data structure can precisely capture these color qualities at fine grain resolution.

2.3.2.1.2 Training

The 27 test videos can be divided into 6 classes of lighting conditions which vary on 3 dimensions. These dimensions are: sunny vs. cloudy, front lit vs. backlit buoys, and camera auto features on vs. off. The video class described by the dimensions cloudy with automatic exposure control off is missing from the test set, hence only six classes, instead of eight, exist in the total test set. One video from each of the six video classes is randomly selected to be the training sequence for that class. For each of the six training sequences the user manually labels one pixel in each buoy of the next three or more pairs of buoys every ten frames. The user has the option of labeling buoy color as either green, red, or yellow.

2.3.2.1.3 Data Structure Population

After manual labeling is complete, the labels are used to generate RGB indexes by referencing the frame-row-column index of the label and retrieving the color value at that pixel. The RGB index is used to populate the label's corresponding bitset according to Equation 2.12.

$$(\boldsymbol{Bitset_A}[R,G,B] = true) \leftarrow \big(\boldsymbol{L}(r,c,t,A,V)\big) \wedge \big(\boldsymbol{I_{(t,V)}}(r,c) = [R,G,B]\big)$$

Equation 2.12

In Equation 2.12 $\boldsymbol{L}(r,c,t,A,V)$ is the index of a manually labeled pixel located at row r, column c, frame t, video number V expressing that the point is of buoy color A (green, red, or yellow).

2.3.2.1.4 Application

A segmentation mask is produced from a frame and bitsets by checking the RGB index of individual pixels within the frame against all three buoy color bitsets as in Equation 2.13.

$$\boldsymbol{F}_{(t,A)}(r,c) \leftarrow (\boldsymbol{I}_t(r,c) = [R,G,B]) \wedge \boldsymbol{Bitset}_{\boldsymbol{A}}[R,G,B]$$

Equation 2.13

In Equation 2.13, $\boldsymbol{F}_{(t,A)}$ is the foreground mask at time t for buoy color A. After application, an individual frame produces three separate segmentation masks; one for each possible buoy color.

2.3.2.2 Online Water Color

The system's assumptions dictate two rules about buoy color. First, the color of a buoy must match that which is specified in the IWSF rulebook. Second, the color of a buoy and the water that surrounds it must appear perceptually different. Following the second assumption, a pixel level classifier can be created which operates on the belief that buoy color will be unexplainable by a model built to describe water color. In creating such a classifier one must answer the same two questions presented for the buoy color based classifier.

2.3.2.2.1 Data Structure

The bitset data structure used for classifying buoy colors is selected to represent water color as well. It provides superior performance in precisely classifying pixels as a water surface contains not only the generic blue color one associates with water color, but also the colors of shoreline object reflections and specularities.

2.3.2.2.2 Data Structure Population

One must fill a bitset with a complete description of water color possibilities. In a model based on second order statistics, missing data is usually filled in if neighboring data (in 3D RGB space) has support. In a bitset type data structure, no such courtesy is given. Even if there is support from neighboring data, missing data, or holes in the bitset, are not filled. For this reason, a very complete understanding of possible water colors must be created.

Bitset data population for water color is done using safe window methods from [37,38]. In these works, the authors use a safe window to generate information about traversable paths. Furthering the ideas in [37,38], safe window construction is dynamic and relies on feedback from the identification module. It is constructed so that it covers as much known water surface as possible. For simplicity, however, only a simple rectangular shaped window is presented. Figure 2.5 shows an example of a simple safe window.

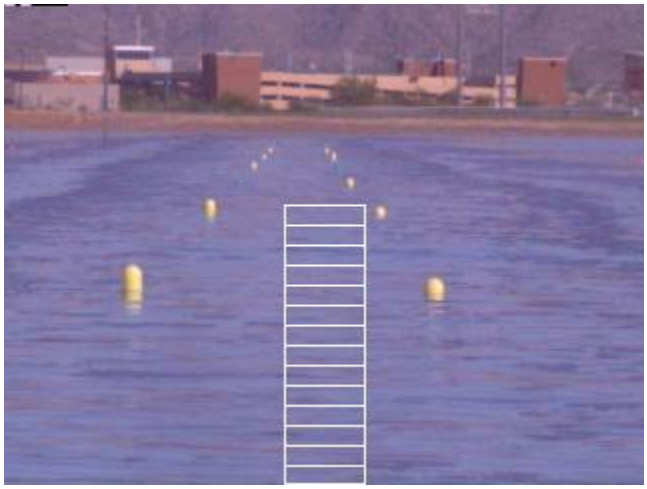


Figure 2.5: Water Color Safe Window.

2.3.2.2.3 Application

The water color based classifier works similarly to the buoy color based classifier. The RGB index of a pixel is tested in the water color bitset. If it corresponding index is set in the bitset, then the pixel is classified as water. If it is not set, The pixel is classified as non-water.

2.3.2.2.4 Error Protection

A tradeoff is made between maximizing safe window size, and thus forming a more complete model of water color, and increasing the chances of incorrectly setting an RGB index within the bitset. For example, Figure 2.6 shows a safe window overlapping shoreline objects prior to boat entrance into the slalom course.

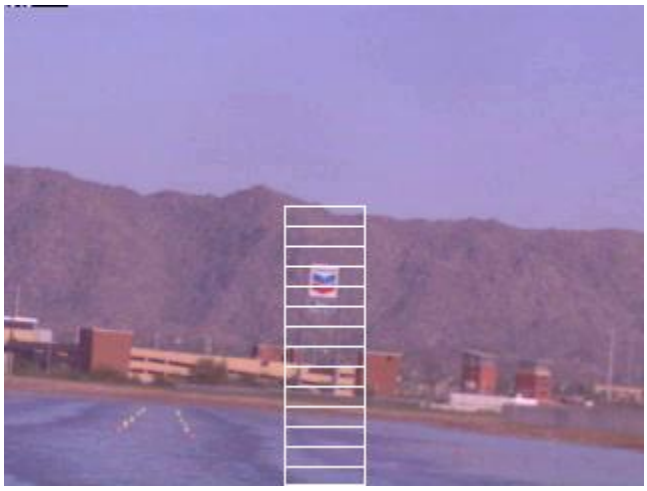


Figure 2.6: Safe Window Incorrectly Overlapping Shoreline Objects.

To avoid retaining incorrectly set RGB indexes, a small portion of all indexes within the entire bitset are randomly unset. This procedure is done using the pseudocode below.

1. $numResets = 256^3 * k$
2. $for\ i = 1{:}numResets$

3. $randIndex = random(1, 256^3)$

4. $waterColor[randIndex] = false$

The main idea of randomly unsetting bitset indexes is that color indexes with greater support will persist in the bitset while indexes set in error will reset after a few iterations. In the pseudocode, the value of k controls the rate of index resetting.

2.4 Experiment Design

Each of the proposed methods is tested with varied parameter settings on 27 frame sequences. From each test, precision, recall, and "percentage of correct classification"[21] (PCC) scoring metrics are produced by looking at the differences between manually labeled and method produced frames. The testing scheme devised takes into account the ability of higher level modules and the tests are designed with the intent of feeding the segmentation mask blobs into the next module. The remainder of the chapter discusses the testing techniques and results of the tests.

2.4.1 Experiment Setup

Three frames from each of the 27 test sequences are manually labeled. Manual labeling involves a user selecting all pixels representative of buoys in a frame. The specific frames selected for manual labeling are not selected at random, they are selected to maximize the variability of the test. Figure 2.7 shows a diagram of which frames are selected as those to be manually labeled. In Figure 2.7 a colored point at zero meters indicates that the frame selected for labeling is the frame in which at least one buoy of the point's color is completely within the frame bounds. The non-zero meter points show the

relative distances of other buoys in the frame. The colors of the point indicate the colors of the buoys.

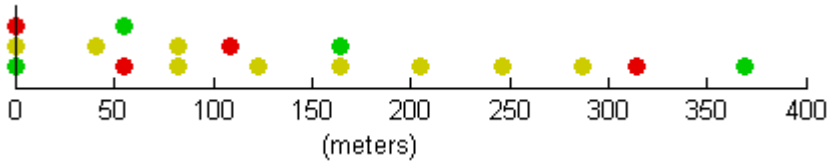
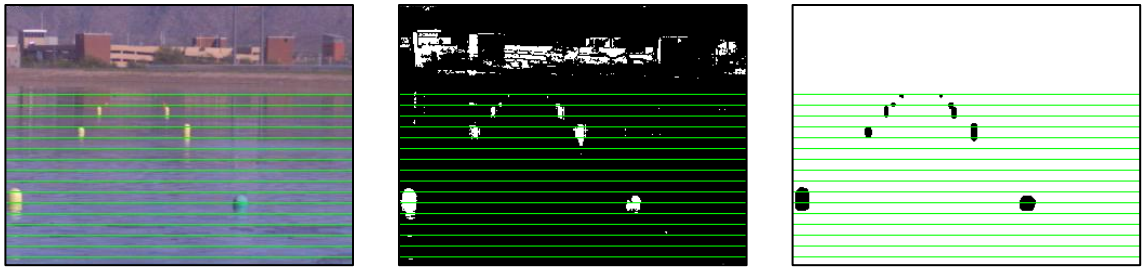


Figure 2.7: The Spread of Points in Tested Poses

From Figure 2.7 it can be seen that the tests are selected so that green, red, and yellow buoys are tested at short range (0-50 meters) medium range(50-150m), and long range (150+ meters).

2.4.2 Data Collection

Each of the presented algorithms is tested by using varied parameter settings for each algorithm and using each of the 27 video sequences as input. At the manually labeled frame indexes, the algorithm's segmentation masks are compared to the manually labeled segmentation masks. From both the method frame and the manually labeled frame, the number pixels classified as true positive, false positive, true negative, and false negative are calculated. One important difference between this test and others in related works is that the region of the image scored is the region below the highest manually labeled pixel rather than the entire frame. Using the reduced region for scoring ensures that shoreline objects do not influence the results of the comparison. Figure 2.8 shows an example of the scoring region.



(a) (b) (c)

Figure 2.8: Scoring Regions. (a) Original Image. (b) Sample Segmentation Mask. (c) Manually Labeled Segmentation Mask. Scoring Region Denoted By Green Horizontal Lines.

2.4.3 Scoring Metric

The experiment is designed to quantitatively test the ability of each proposed algorithm. Precision and recall measurements are used to provide a numeric representation of each algorithm's capability. Precision is a ratio that defines what portion of the selected pixels actually belong to buoys. It is defined in Equation 2.14.

$$precision = \frac{tp}{tp + fp}$$

Equation 2.14

Recall is the portion of target pixels actually selected by the algorithm. It is defined in Equation 2.15.

$$recall = \frac{tp}{tp + fn}$$

Equation 2.15

Percentage of correct classification [21] is a score that gives number of correct classifications compared to the total number of classifications. Equation 2.16 shows how to produce this score.

$$pcc = \frac{tp + tn}{tp + tn + fp + fn}$$

Equation 2.16

In all of these equations *tp* and *tn* represent true positives and true negatives. *fp* and *fn* represent false positives and false negatives.

2.4.4 Varied Parameters

The parameters of the tested methods are varied along certain dimensions in order to ensure a thorough examination of each algorithm's performance. The subsequent table gives the varied and static parameters of each tested methods.

Running Mean BGS			
Varied Parameters			Static Parameters
Conservative Update	Spatial Update	Threshold Distance	Learning Factor
{ True, False }	{ True, False }	{ 20, 40, 60 }	{ .05 }
Running Median BGS			
Varied Parameters			Static Parameters
Conservative Update	Spatial Update	Threshold Distance	Learning Factor
{ True, False }	{ True, False }	{ 20, 40, 60 }	{ .05 }
Running Single Gaussian BGS			
Varied Parameters			Static Parameters
Conservative Update	Spatial Update	Mahalanobis Distance	Learning Factor
{ True, False }	{ True, False }	{ 1.5, 2.0, 2.5, 3.0 }	{ .05 }
Gaussian Mixture Model			
Varied Parameters			Static Parameters
Mahalanobis Distance	Learning factor	Max Modes	Background Percent
{ 1.0, 1.5, 2.0, 2.5, 3.0, 3.5 }	{ .001, .005, .01, .05}	{ 2, 3, 4 }	{ .75 }

Visual Background Extractor					
Varied Parameters			Static Parameters		
Radius	Subsample Frequency	Threshold Distance	Conservative Update	Spatial Update	Number of Samples
{ 1.0, 1.5, 2.0, 2.5, 3.0 }	{ 8, 16 }	{ 20, 30, 40, 50 }	{ True }	{ True }	{ 20 }

Buoy Color
Varied Parameters
Bitset Source
{ Total, Camera Setting Specific}

Water Color		
Varied Parameters		
Forget Factor	Safe Window Width	Safe Window Height
{ 0.5/256, 1.0/256, 1.5/256, 2.0/256 2.5/256, 3.0/256 }	{ 10, 40 }	{ 40, 160 }

Difference Of Gaussians		
Varied Parameters		
σ_x	σ_y	Threshold Distance
{ 0, 1, 2, 4 }	{ 0, 1, 2, 4 }	{ 3, 8, 13 }

Table 2.1: Varied Parameters of the Segmentation Experiment.

2.5 Results

The precision and recall values are computed for each method using various parameter settings. The PCC, precision and recall plots for six identified video classes are presented. Only five sets of sample output are provided due to the similarity of the "shadow side" and "foam" class videos. The PCC scores presented are the best overall parameter setting of each method presented.

2.5.1 PCC for Best Parameter Settings

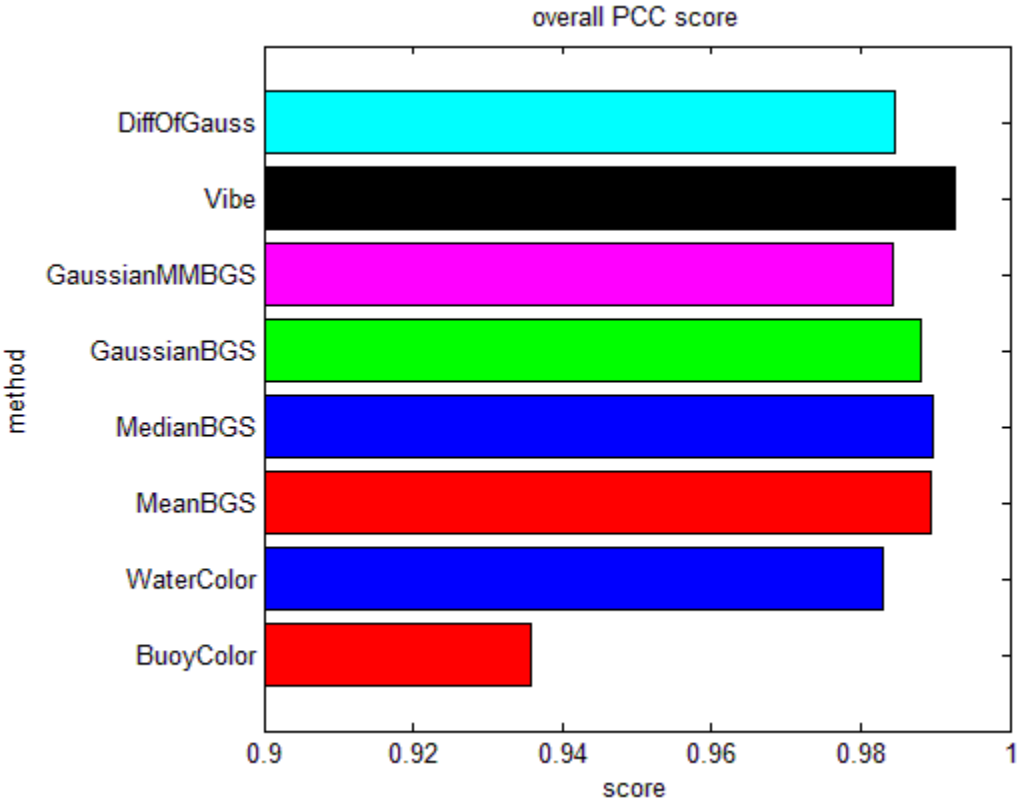


Figure 2.9: Best PCC Score for each Method.

2.5.2 Precision Recall Plots for Various Parameter Settings

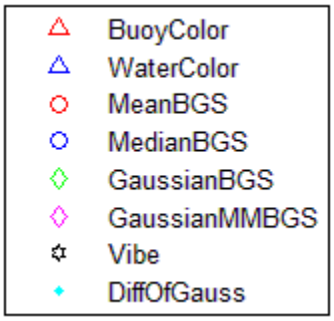


Figure 2.10: Legend of Plots.

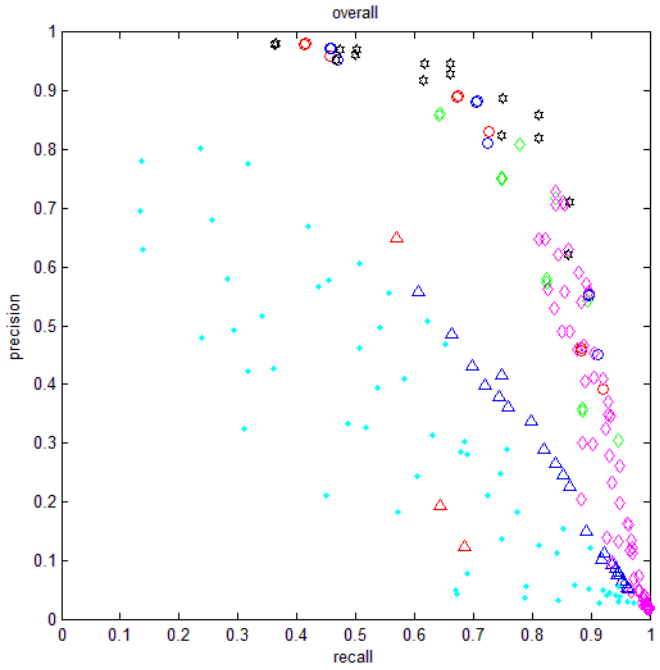


Figure 2.11: Precision-Recall Plot for Methods Tested on All Classes of Video.

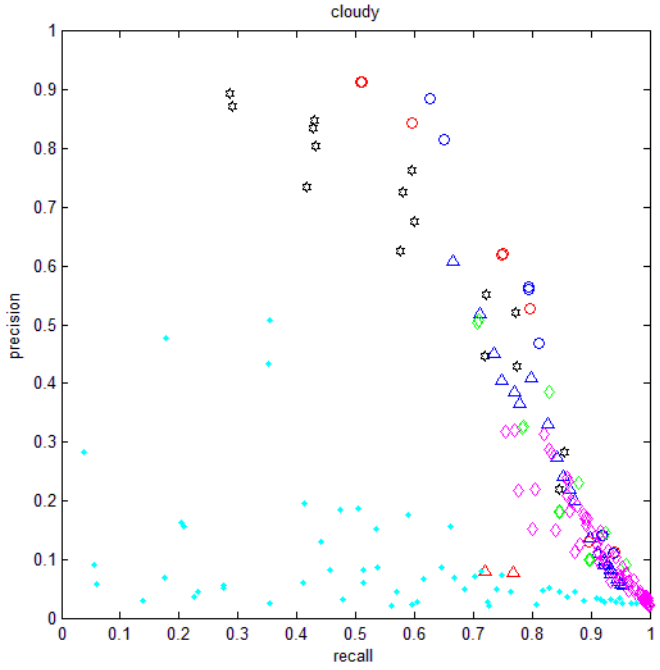


Figure 2.12: Precision-Recall Plot for Methods Tested on Cloudy Class Videos.

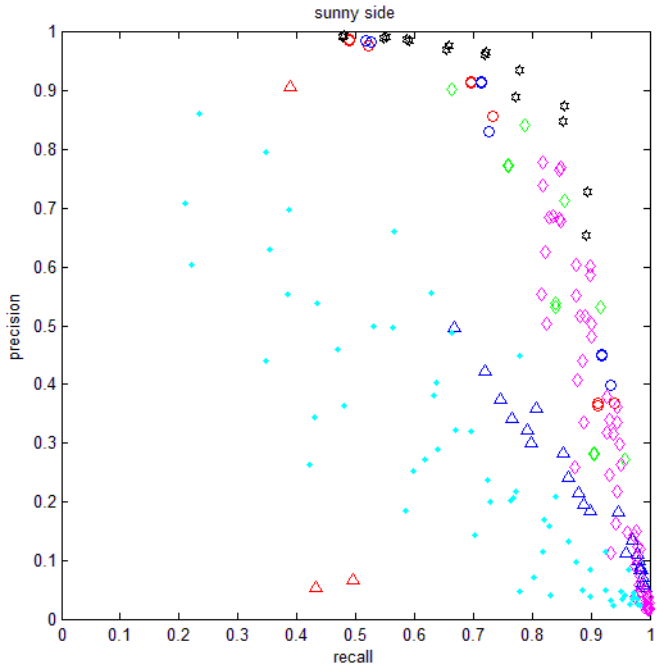


Figure 2.13: Precision-Recall Plot for Methods Tested on Sunny Side Videos.

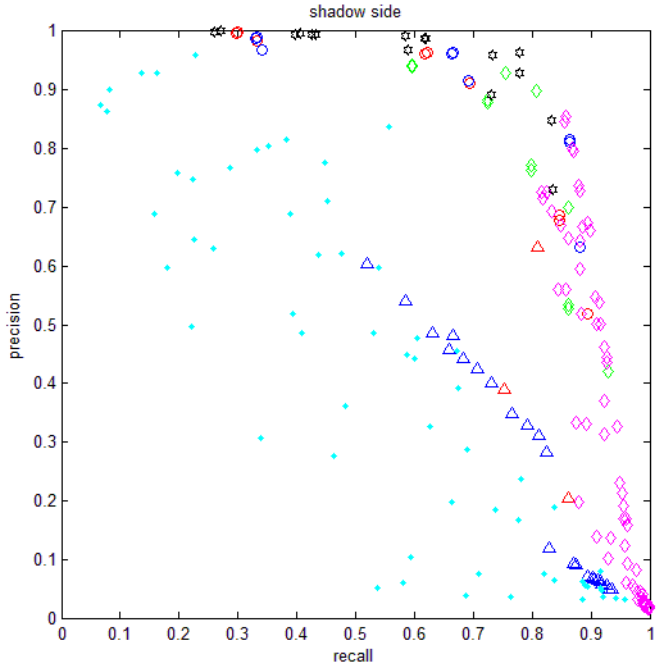


Figure 2.14: Precision-Recall Plot for Methods Tested on Shadow Side Videos.

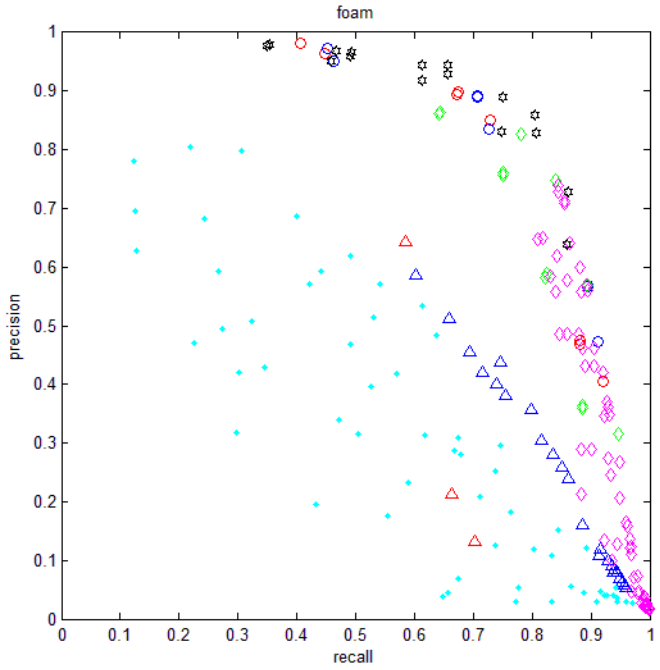


Figure 2.15: Precision-Recall Plot for Methods Tested on Foam Class Videos.

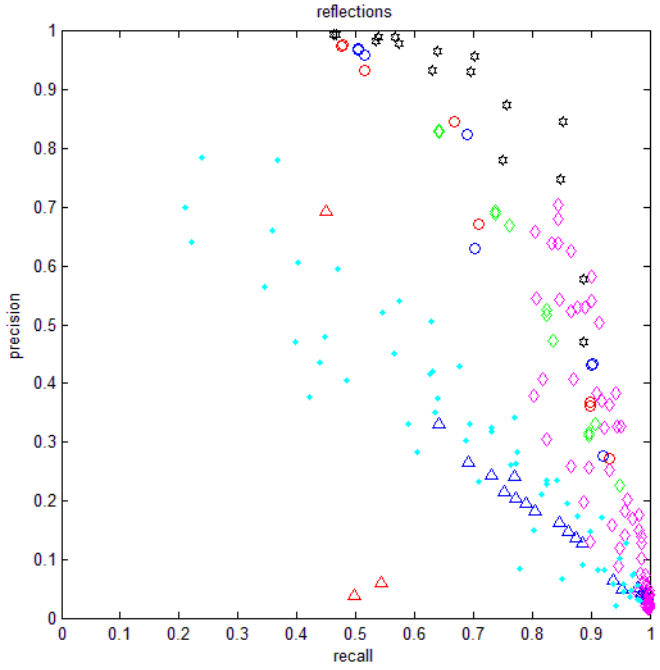
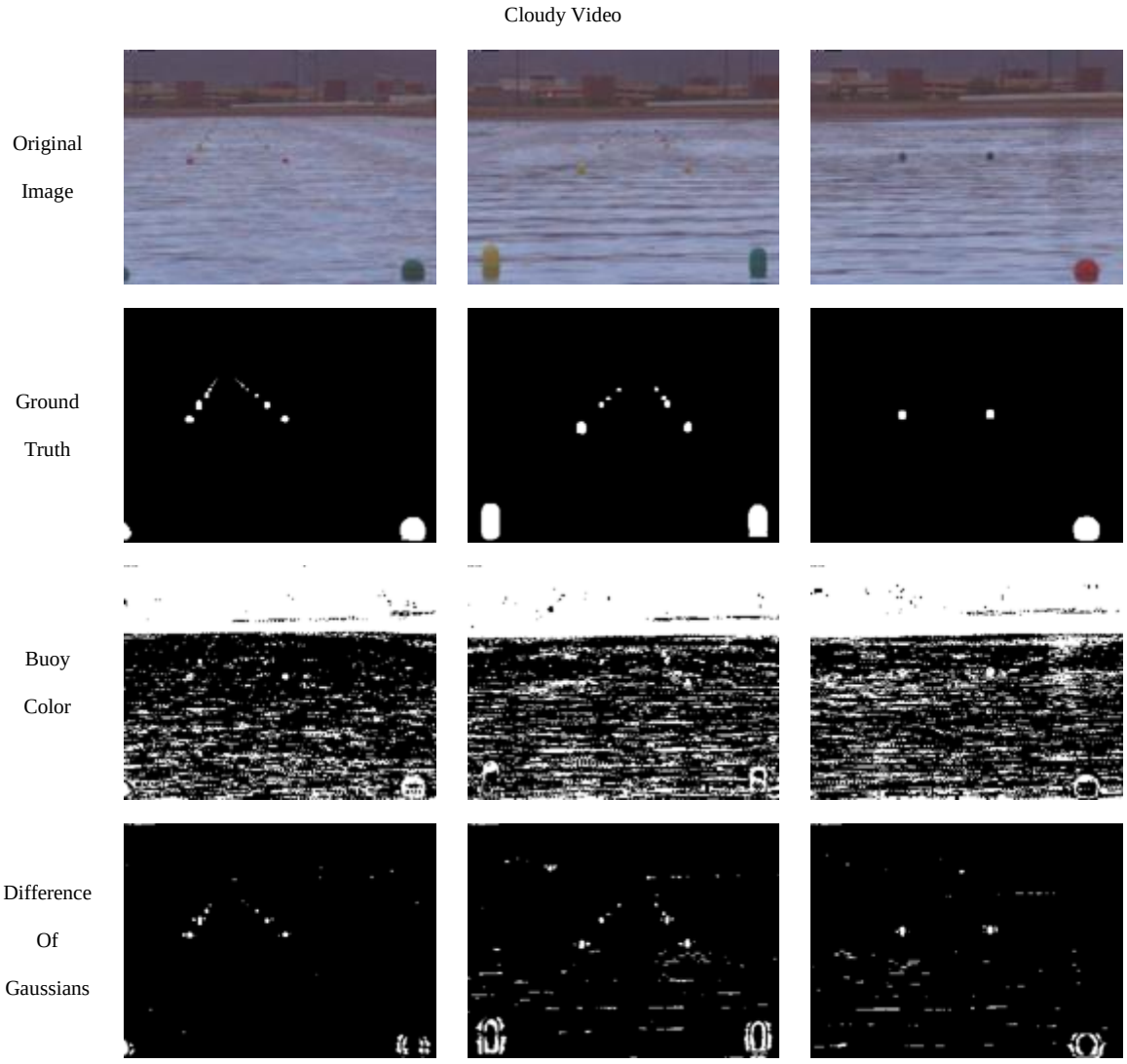
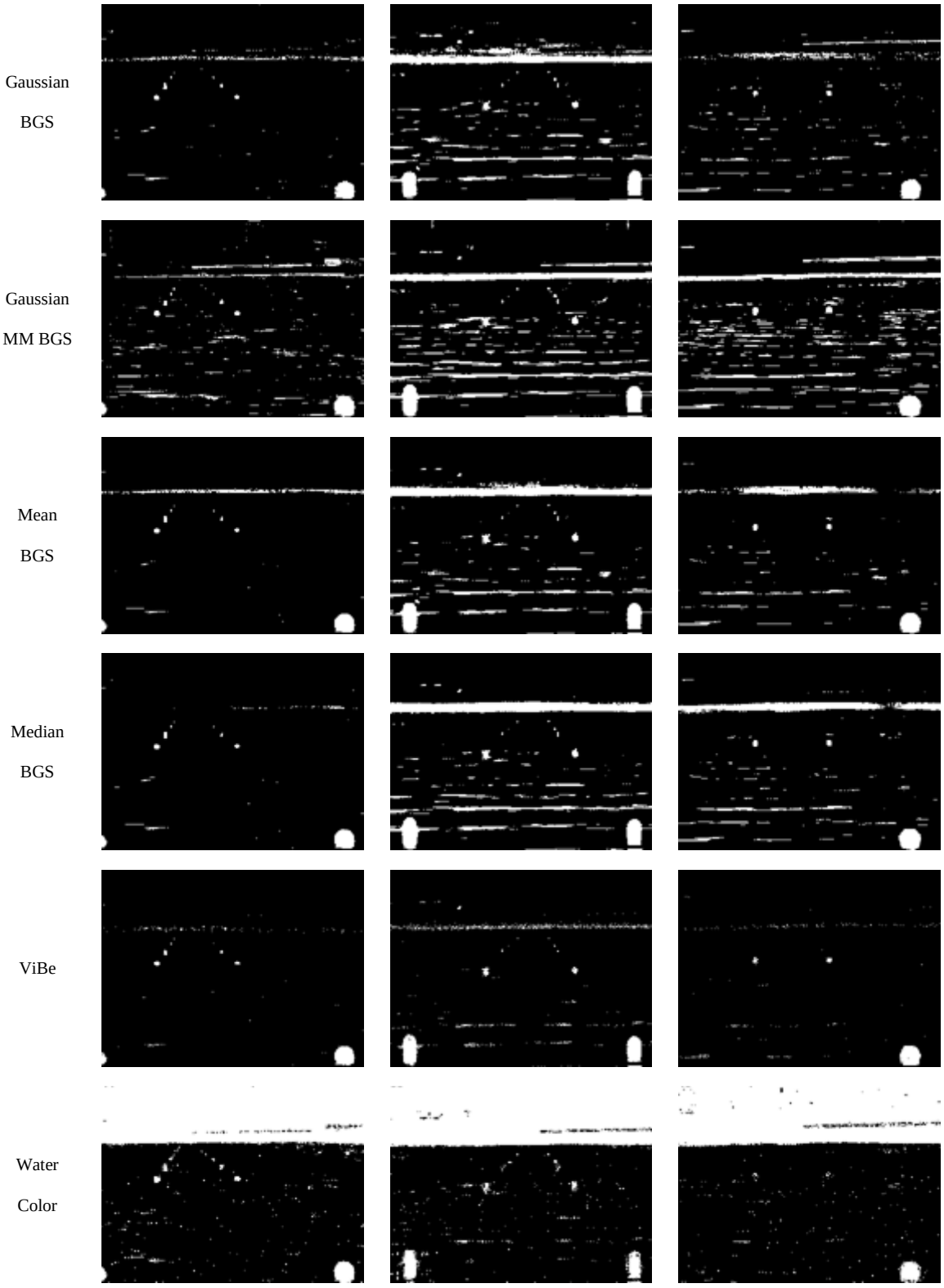


Figure 2.16: Precision-Recall Plot for Methods Tested on Reflection Class Videos.

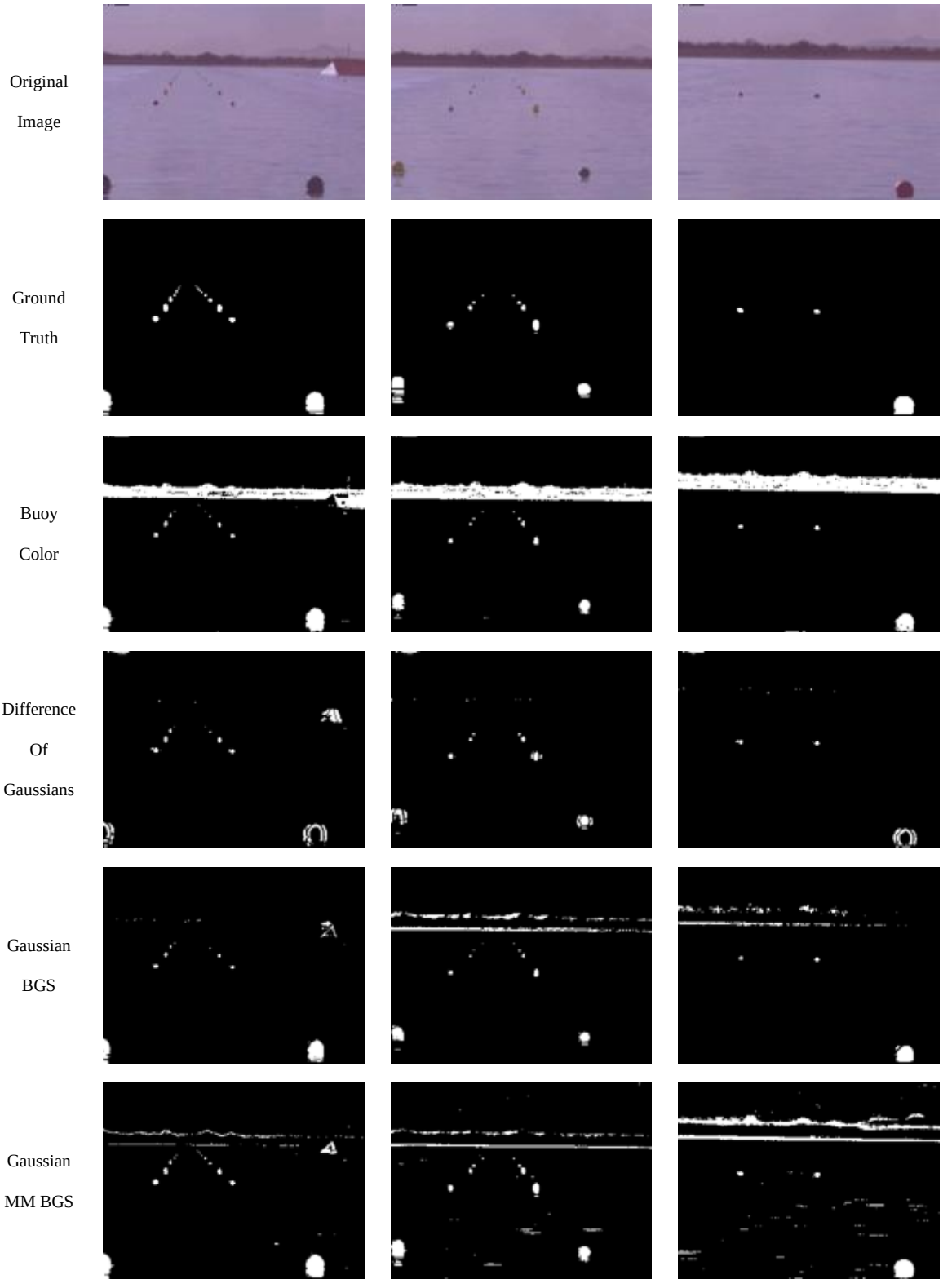
2.5.3 Sample Output

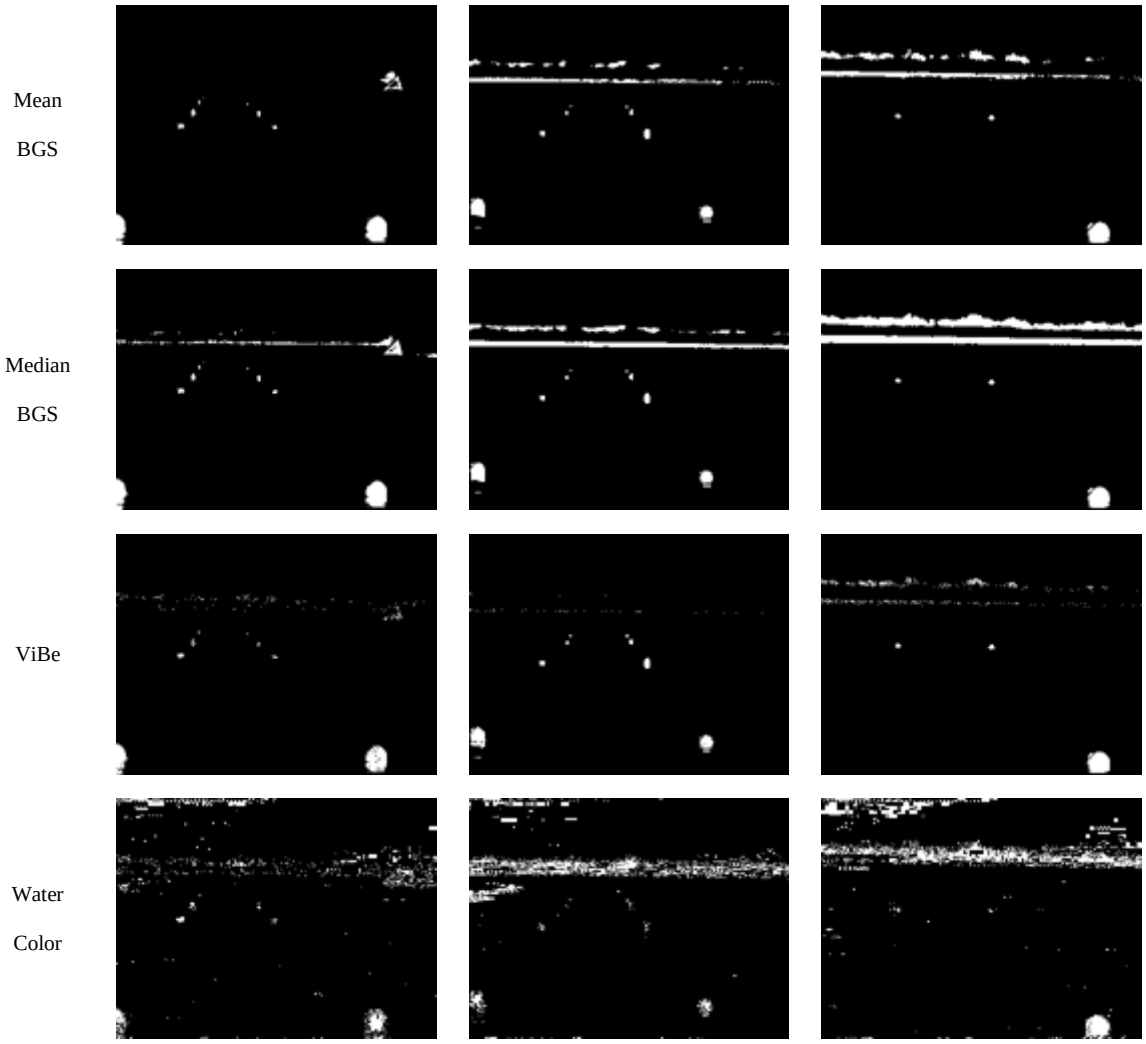
For five of the six video sequence categories presented in the aforementioned plots, sample output is provided which shows the method's best overall parameter setting output in comparison to the original frame and ground truth. The "Foam" category of video sequences is excluded due to its similarity to the "Shadow Side" and "Sunny Side" class of videos.



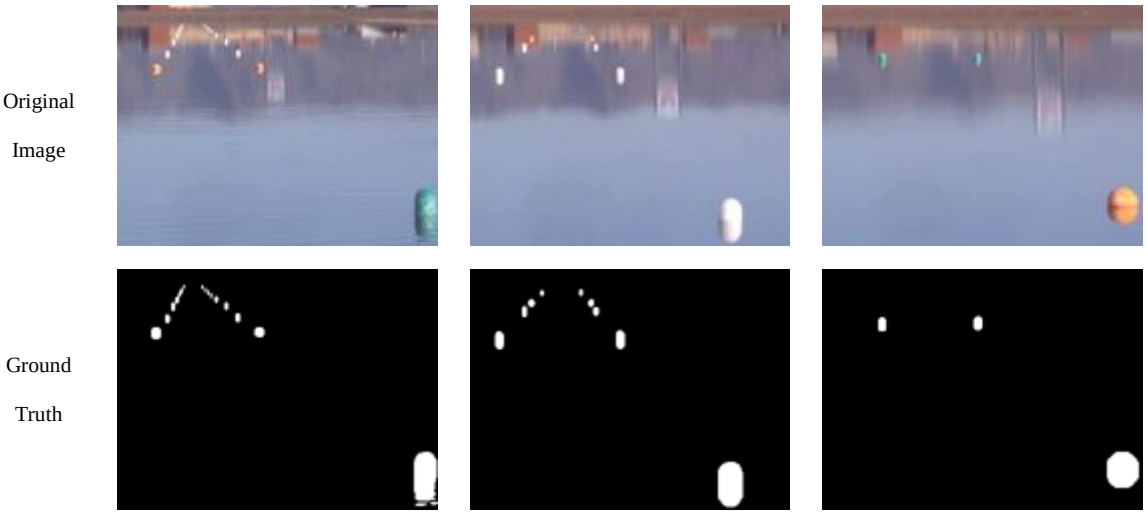


Shadow Side Video


Original Image
Ground Truth
Buoy Color
Difference Of Gaussians
Gaussian BGS
Gaussian MM BGS

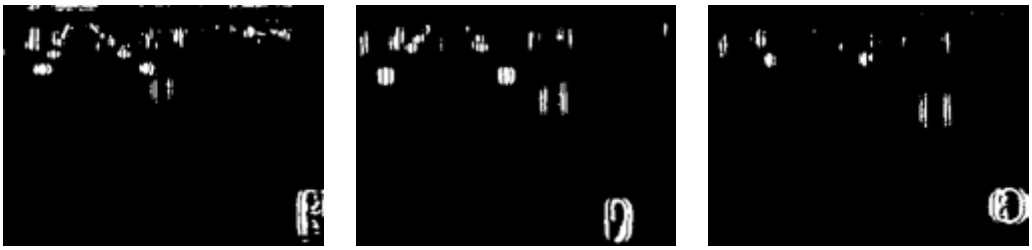


Reflections Video



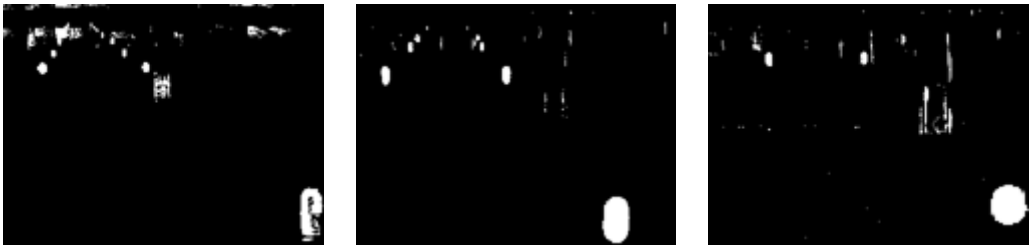
Buoy Color

Difference Of Gaussians



Gaussian BGS

Gaussian MM BGS

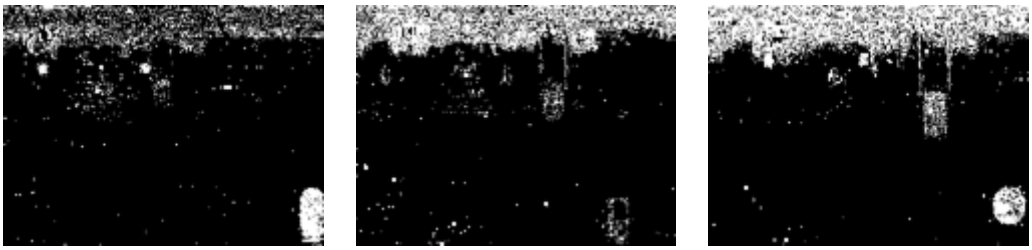


Mean BGS

Median BGS

ViBe

Water Color



Sunny Side Video

Original Image

Ground Truth

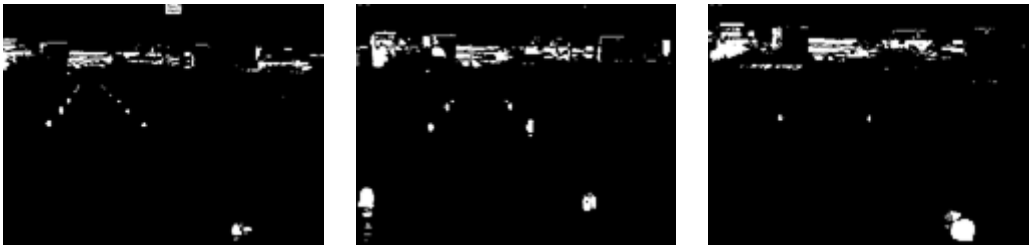
Buoy Color

Difference Of Gaussians

Gaussian BGS

Gaussian MM BGS

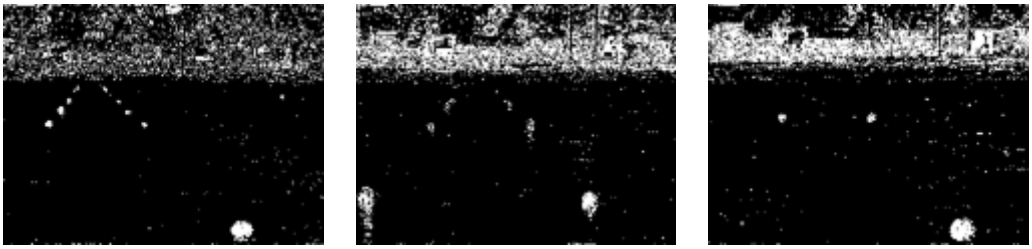
Mean BGS



Median BGS

ViBe

Water Color



2.5.4 Limitations of Experimental Evidence

The method of testing proposed solutions has multiple weaknesses. First, only 27 frame sequences are used in quantitatively scoring the abilities of each algorithm. From these 27 sequences only a small subset of all possible lighting and weather conditions are captured. Second, only three frames for each of the 27 sequences are used in testing as manually labeling frames is an extremely tedious and time intensive. Finally, the precision and recall scoring schema used may not be the best indicator of each algorithm's ability when considering the module's purpose within the scope of the complete system. For example, an algorithm which receives high precision and recall scores, but tends to segment only one side of a buoy due to lighting conditions may not be as the best algorithm in comparison to an algorithm that segments buoys from edge to edge but produces additional unconnected noise. In the first case, a pose estimation algorithm would bias its result towards the fully segmented side of the buoys. In the second case, the pose estimation module would not bias its result assuming that the identification module produces a correct blob-to-buoy mapping.

2.5.5 Discussion

Upon examination of the experimental results, specifically the precision-recall plots, it can be seen that the ViBe method appears to be the overall best segmentation technique. Its superior performance appears to stem from its ability to hold a large number of color values for each pixel, a property that differentiates the ViBe method from the other background subtraction techniques. This multi-color mechanism becomes especially useful in situations where camera jitter, and thus movement of reflections within the image, is present. Furthermore, the ability to store multiple colors per pixel

allows the algorithm to adequately model the wide variety of color possibilities present on a aquatic surface.

Along the same lines as the ViBe technique, the Gaussian mixture model background subtraction technique also performs well. Its performance is again most likely due to its ability to hold multiple color models for each pixel. In reviewing the experimental results, it is understandable that the Gaussian mixture model background subtraction technique performs comparatively worse than the ViBe method. The parameter settings used in the experiment allowed the ViBe method to hold up to 20 unique color values per pixel as background possibilities. The Gaussian mixture model was only able to hold up to four models of color per pixel.

The mean and median background subtraction algorithms had similar experimental performance. Surprisingly, the algorithms performed well on the cloudy class videos. This effect is most likely related to the fact that these algorithms had statically defined thresholds. The other two related background subtraction techniques had thresholds that were dependent on the variance of incoming values. The video sequences categorized as cloudy class had two common properties present in all videos. They had lower overall contrast in every frame and higher noise per pixel due to the camera's struggling internal dynamic range compression logic. These two properties, along with the statically defined threshold, could provide reason for the performance gap between the methods with statically defined thresholds and the methods with dynamic thresholds.

The difference of Gaussians method performed relatively poor in comparison to the other tested methods. It failed to fill in the inner regions of buoys and its precision

and recall scores suffered. Nevertheless, it did a superb job of segmenting the edges of buoys in all video classes and allowed for the adjustment of response based on spatial frequencies. Contrary to most of the other tested algorithms, it functioned as a frame-by-frame algorithm and required no prior training, something that may be important when trying to relax more assumptions about full system use.

The algorithm based solely on buoy color did not perform well. The poor performance was most likely related to the fact that one data structure was used to hold all possible buoy color indexes under all possible lighting conditions. The use of one data structure becomes a major hindrance to the method's performance due to the fact that buoy color under one lighting condition matches water color under another lighting condition. Specifically, the color indexes of yellow buoys in sequences in which the perceived buoy surfaces are illuminated indirectly (shadow side) are almost identical to the color indexes of water in other sequences. Overall the buoy color based method does not seem worthy of further pursuit as it does not allow the system to relax assumptions about buoy color.

The water color based method appears to be stronger than the buoy color based method. It could, however, perform better if a smarter safe window was used. A smart safe window would be a window in which the area of the window is maximized using feedback from higher level modules. This would allow the algorithm to build a more complete model of water color.

2.6 Conclusion

A usable method of buoy segmentation, the ViBe algorithm, has been found by testing multiple segmentation methods with varied parameter settings on real test

sequences. Each parameter setting for each method was quantitatively evaluated by finding precision and recall scores. While the strongest method found, ViBe, is not a perfect solution to the problem, it is a usable solution that will allow for the continued development the system.

The experiment relied on 27 test sequences. While these sequences where not extensive enough to include every possible environment, they were varied enough to expose the strengths and weaknesses of each of the tested methods. Furthermore, they were varied enough to reveal possible faults within the algorithms that make them unsuitable as a long term solutions.

In general, it appears that using specific, offline trained, colors as the means to classify pixels is not a robust technique. There will always be situations in which accepted color in one condition becomes rejected color in another condition. This type of issue could be dealt with by understanding lighting conditions and their effects on perceived color, but implementing a system that used this knowledge would shift the module away from using specific colors and move it towards using the relationships between colors under lighting conditions.

The water color based classifier, which relaxes assumptions about specific colors and determines the classification of colors on the fly, looks like a slightly better solution in comparison to the buoy color based method. It too, however, is susceptible to conditions that can cause erroneous output. For example, suppose a large green shoreline object projected a reflection onto the water surface that the camera perceived as in the same spatial region as yellow buoys. The safe window would absorb this color and incorrectly classify any green pair of buoys. A human operator would be able to

understand this situation, but a watercolor based method would not. This leads to the conclusion that work should go into a water color based algorithm that takes into account both colors and the spatial location of used colors.

The background subtraction techniques work well on the presented segmentation problem. Their success is most likely due to the geometry of the problem itself. In the problem, a camera experiences significant one-dimensional motion along its optical axis. Because the shoreline objects are located significantly further away from the camera than the buoys, the shoreline objects grow at a slower rate become part of the accepted background. The only issues with background subtraction methods are their need for a training period, and their inability to deal with camera jitter. A solution for the training period problem may be to force a high adaptation rate until the system is certain that it has stabilized and is looking at a scene it can use for further processing. In terms of camera jitter, an image stabilization procedure prior to segmentation may mitigate the effects of camera motion.

The difference of Gaussians method gives the most insight into possibly strong solutions. It can be seen that the binary masks generated by the difference of Gaussians method are inadequate at supplying what is truly needed. While the produced binary masks effectively show changes in image color with respect to spatial location, they do not show from what color or to what color the change represents. A more appropriate solution would be one that not only detects color change, but also gives details on the direction and location of the change within a color space. This would allow the module to differentiate between a water-to-buoy change and a water-to-reflection change. Figure 2.17 shows how an improved solution may work.

(a) (b) (c)

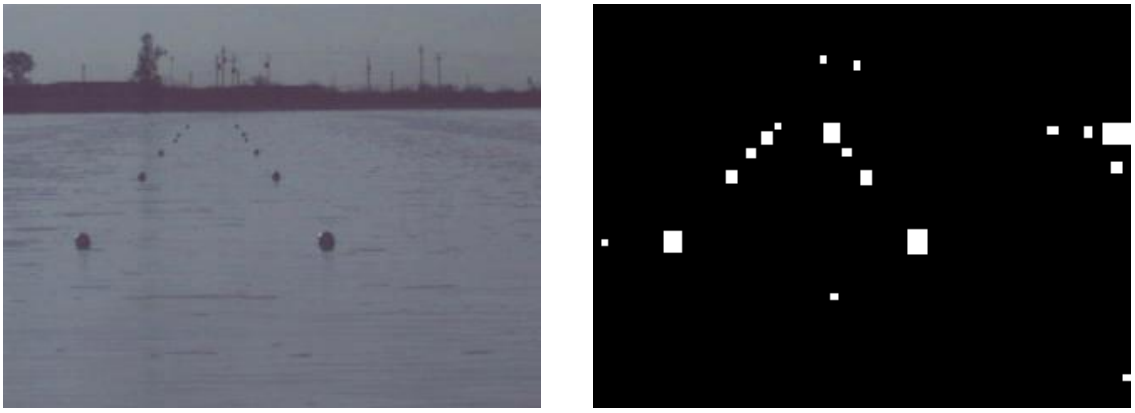
Figure 2.17: Continued Research. (a) Normal Image. (b) Binary Segmentation Mask. (c) A More Complex but Robust Solution.

Overall, it can be seen that all presented methods offer simple solutions to the problem but suffer from issues that make them unfit as all encompassing solutions. It appears the output of the segmentation module must move away from binary masks to more complex masks. Nonetheless, a usable method, ViBe, has been experimentally found and progress on the entire system can continue.

3 IDENTIFICATION

3.1 Introduction

In the prior stage of processing, the segmentation module labeled pixels representative of buoys. A connected-component labeling algorithm then grouped the pixels into blobs and all blobs were placed in a blob set. The system must now label each blob according to which buoy it represents. If the blob does not represent a buoy, then it must be labeled as such. Figure 3.1 (a) displays a typical frame captured by the system's camera. Figure 3.1 (b) shows the bounding box representation of the blob set produced by the segmentation module. The identification module is expected to work with sets of blobs such as the one presented in and Figure 3.1 (b).



(a) (b)

Figure 3.1: A Frame and Its Associated Blobs. (a) A Frame Captured under Normal Conditions. (b) The Blobs Produced by the Segmentation Module.

A naive approach to the labeling task would be a brute force solution that attempts to test every permutation of blob to buoy labeling. This type of approach would be too computationally expensive to meet real-time constraints. For example, consider Figure 3.1. If there were 18 blobs present, as there are in Figure 3.1 (b), and it was decided that

there were 12 buoys to map to, as in Figure 3.1 (a), then the system would need to test a total of

$$\frac{\boldsymbol{n}!}{(\boldsymbol{n}-\boldsymbol{r})!} = \frac{18!}{(18-12)!} = 8.8 * 10^{12}$$

configurations in order to find the best blob to buoy fit. The calculation presented only represents the required number of tests for mapping 18 blobs to 12 buoys. As seen in Figure 3.1 some of the blobs do not represent buoys. To ensure the globally best blob labeling, the system would also need test sets of 17 blobs, sets of 16 blobs, and so on. The number of actual test necessary is not important. It is only important to realize that this approach is not computable in real-time with a modern personal computer.

To avoid the large number of tests associated with the above brute force approach, the system attempts to label blobs by examining blobs in pairs. Furthermore, constraints are enforced which limit the actual number of tests. The end result is an algorithm that is significantly less complex.

3.2 High Level Overview

At the highest level the entire identification module can be described with the pseudocode:

```
1.  for pairIndex= 1:num_pairs
2.      [buoy0, buoy1] = buoysAtpair(pairIndex)
3.      [blob0, blob1] = detectOrTrack(buoy0,  buoy1)
4.      updateBlobAndState(buoy0, blob0)
5.      updateBlobAndState(buoy1, blob1)
6.      updateBlobset(blobset, buoy0.newState, buoy0.newBlob)
7.      updateBlobset(blobset, buoy1.newState, buoy1.newBlob)
```

In the pseudocode, line 1 is a for loop that iterates through all 10 buoy pairs. Line 2 retrieves the data structures associated with the two buoys in the current buoy pair. Line 3 is the execution of a detection or tracking algorithm. The detection and tracking algorithms use all available knowledge to select the blob pair that best represents the buoy pair. Lines 3 and 4 update the buoy data structures with the newly selected blobs. This includes updating a buoy's list of assigned blobs, and assigned states. Finally, lines 6 and 7 remove the blob selected by the detection or tracking algorithm from the set of usable blobs.

It is important to note that buoy pairs are processed in the order of closest from camera to furthest from camera. The ordering is due to the fact that buoys closer to the camera generally appear in captured frames as objects with higher resolution and better consistency in terms of size and position.

3.3 Assumptions

The identification module works under assumptions similar to those already stated. In terms of camera orientation, it is assumed that the camera has been mounted on the boat such that: (1) The camera is above the water, (2) The optical axis of the camera is oriented within five degrees of parallel to the direction of boat travel, and (3) The camera's rotation about the optical axis is within five degrees of upright. When the module is searching through blob sets and trying to identify blobs that represent buoys, it is assumed that the boat position and orientation abides by the following: (1) The boat is positioned within .5 meters of the slalom course centerline when it is at a distance of 50 meters or less from the first set of pre-gates, and (2) The boat is oriented within five degrees of parallel to the slalom course when it is within 50 meters of the first set of pre-

gates. Finally, the module assumes that the buoys are: (1) anchored to the world at the positions specified in the IWSF tournament rulebook, and (2) sized and shaped according to the rules specified in the IWSF rulebook.

3.4 Chapter Notation

The remaining sections of this chapter are written in a consistent notation so that the concepts described are easier to understand. The two primary data types used in the module are buoys and blobs. Each data type has its own associated predicates, functions, and symbols. It is important to note that the process of blob labeling is further referred to as blob assignment or assigning blobs to buoys.

3.4.1 Buoy Notation

Buoys are referred to as either actual buoys, image buoys, or virtual buoys. Actual buoys are the tangible objects in the real world, image buoys are the buoys seen in a frame, and virtual buoys are the abstract, programmatic, data structures used in the system. All three types of buoys are connected in the sense that they all reference the same object. The symbols $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{\gamma}$, and $\boldsymbol{\delta}$ represent individual virtual buoys. The virtual buoys $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{\gamma}$, and $\boldsymbol{\delta}$ have indexes that range from 0 to 19 as the system works with 20 actual buoys. The symbols $\boldsymbol{\varepsilon}$, $\boldsymbol{\theta}$, $\boldsymbol{\lambda}$, and $\boldsymbol{\sigma}$ represent virtual buoy pairs. Virtual buoy pairs consist of unique individual virtual buoys, a virtual buoy cannot belong to multiple virtual buoy pairs. The virtual buoy pairs $\boldsymbol{\varepsilon}$, $\boldsymbol{\theta}$, $\boldsymbol{\lambda}$, and $\boldsymbol{\sigma}$ have indexes that range from 0 to 9 as there are 10 pairs of 20 actual buoys. The predicate $\text{pair}(\boldsymbol{\alpha}, \boldsymbol{\beta})$ is true when virtual buoy $\boldsymbol{\alpha}$ and virtual buoy $\boldsymbol{\beta}$ are in a pair. The notation $\boldsymbol{\varepsilon} = \{\boldsymbol{\alpha}, \boldsymbol{\beta}\}$ indicates that virtual buoy $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are in virtual buoy pair $\boldsymbol{\varepsilon}$. Note that the pair $\{\boldsymbol{\alpha}, \boldsymbol{\beta}\}$ is different from the pair $\{\boldsymbol{\beta}, \boldsymbol{\alpha}\}$. The first set denotes that virtual buoy $\boldsymbol{\alpha}$ is a port side buoy and $\boldsymbol{\beta}$ is a starboard

side buoy. The second pair denotes the opposite. As seen in Figure 3.2, a pair of buoys is what the boat travels through while driving through a slalom course.

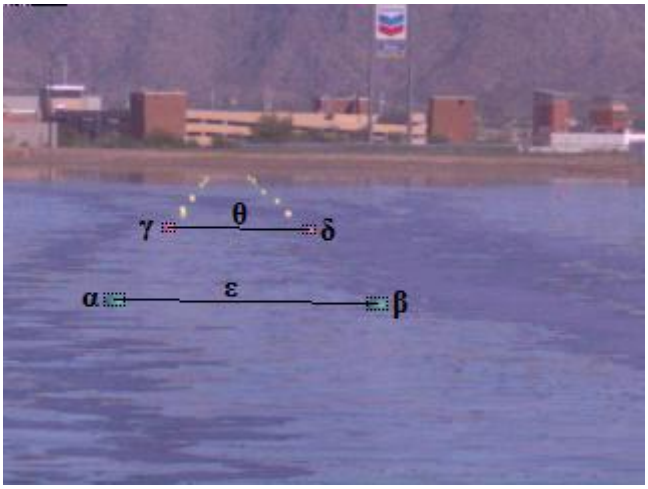


Figure 3.2: A Frame Marked in Notation. Dotted Boxes Mark Some of the Individual Buoys and Solid Lines Denote Some of the Buoy Pairs.

The predicate $\mathrm{tracking}(\boldsymbol{\alpha}, \mathrm{t})$ is true when virtual buoy $\boldsymbol{\alpha}$ is in a tracking or partially occluded state at time t. The function $\mathbf{LTIGT}(\boldsymbol{\alpha}, \mathrm{t})$ (Least Tracking Index Greater Than) is a function that finds the individual tracking virtual buoy at time t with smallest index greater than $\boldsymbol{\alpha}$'s index which is not in $\boldsymbol{\alpha}$'s pair. For example, if in Figure 3.2 the buoys with dotted boxes represented tracking virtual buoys then the function $\mathbf{LTIGT}(\boldsymbol{\alpha}, \mathrm{t})$ would return a reference to a virtual buoy the next furthest virtual buoy pair from the camera. In this case, either $\boldsymbol{\gamma}$ or $\boldsymbol{\delta}$. Formally the function is defined as

$$\boldsymbol{LTIGT}(\boldsymbol{\alpha}, t) = \boldsymbol{\gamma} \leftarrow \big(tracking(\boldsymbol{\gamma}, t)\big) \wedge (\boldsymbol{\gamma}.index > \boldsymbol{\alpha}.index) \wedge \big(!\, pair(\boldsymbol{\alpha}, \boldsymbol{\gamma})\big)$$
$$\wedge \big(\nexists\, \boldsymbol{\delta} : \big[\big(tracking(\boldsymbol{\delta}, t)\big) \wedge (\boldsymbol{\delta}.index > \boldsymbol{\alpha}.index) \wedge (\boldsymbol{\delta}.index < \boldsymbol{\gamma}.index)\big]\big)$$

Equation 3.1

The function $\mathbf{GTILT}(\boldsymbol{\alpha}, \mathrm{t})$ (Greatest Tracking Index Less Than) is the largest indexed tracking virtual buoy with an index less than $\boldsymbol{\alpha}$'s index not in $\boldsymbol{\alpha}$'s pair. As an

example, if the dotted boxes in Figure 3.2 represented tracking individual virtual buoys, and the solid lines represented pairs, then virtual buoy $\mathbf{GTILT}(\boldsymbol{\gamma}, \mathrm{t})$ references either $\boldsymbol{\alpha}$ or $\boldsymbol{\beta}$. Formally this function is defined as

$$\boldsymbol{GTILT}(\boldsymbol{\gamma}, t) = \boldsymbol{\alpha} \leftarrow (tracking(\boldsymbol{\alpha}, t)) \wedge (\boldsymbol{\alpha}.index < \boldsymbol{\gamma}.index) \wedge (!\, pair(\boldsymbol{\alpha}, \boldsymbol{\gamma}))$$
$$\wedge (\nexists\, \boldsymbol{\beta} : [(tracking(\boldsymbol{\beta}, t)) \wedge (\boldsymbol{\beta}.index < \boldsymbol{\gamma}.index) \wedge (\boldsymbol{\beta}.index > \boldsymbol{\alpha}.index)])$$

Equation 3.2

Both of the functions $\mathbf{GTILT}(\quad)$and $\mathbf{LTIGT}(\quad)$ can be applied to pairs of virtual buoys rather than individual virtual buoys.

A virtual buoy $\boldsymbol{\alpha}$ has memory of its previously assigned blobs and its previous states. The function $\boldsymbol{\alpha}.\mathbf{blob}(\mathrm{t})$ returns the blob selected to represent virtual buoy $\boldsymbol{\alpha}$ at time t. The function $\boldsymbol{\alpha}.\mathrm{state}(\mathrm{t})$ similarly returns the state of the virtual buoy $\boldsymbol{\alpha}$ at time t. Finally, the function $\boldsymbol{\alpha}.\mathbf{blobset}(\mathrm{t})$ returns the set of usable blobs available to virtual buoy $\boldsymbol{\alpha}$ at time t. Synonymously, the function $\boldsymbol{\varepsilon}.\mathbf{blobset}(\mathrm{t})$ returns the blobs the module can map to virtual buoy pair $\boldsymbol{\varepsilon}$ at time t.

3.4.2 Blob Set Notation

The symbols **d**, **e**, **f**, and **g** represent individual blobs and the symbols **j**, **k**, and **m** represent pairs of blobs. While a virtual buoy only belongs to one virtual buoy pair, a blob may belong to any and all possible blob pairs, even the pair that includes the blob twice. Like the pair predicate for virtual buoys, the predicate $\mathrm{pair}(\mathbf{d}, \mathbf{e})$ is true when blobs **d** and **e** are in a pair together. The statement $\mathbf{j} = \{\mathbf{d}, \mathbf{e}\}$ states that blobs **d** and **e** are in blob pair **j.** The pair $\{\mathbf{d}, \mathbf{e}\}$ also expresses that blob **d** is supposed to represents the port side blob while blob **e** represents the starboard side blob.

3.4.3 Buoy and Blob Set Formulas

The system often uses the Euclidean distance formula to find the distance between a pair of blobs or a pair of virtual buoys. For a pair of blobs, the distance formula is defined as

$$dist(\boldsymbol{j}) = dist(\boldsymbol{d}, \boldsymbol{e}) = \sqrt{(\boldsymbol{d}.\boldsymbol{center}.x - \boldsymbol{e}.\boldsymbol{center}.x)^2 + (\boldsymbol{d}.\boldsymbol{center}.y - \boldsymbol{e}.\boldsymbol{center}.y)^2}$$
$$\leftarrow (\boldsymbol{j} = \{\boldsymbol{d}, \boldsymbol{e}\})$$

Equation 3.3

For a pair of virtual buoys, the distance between them is the distance between their blob representations, or

$$dist(\boldsymbol{\varepsilon}, t) = dist(\boldsymbol{\alpha}, \boldsymbol{\beta}, t) = dist\big(\boldsymbol{\alpha}.\boldsymbol{blob}(t), \boldsymbol{\beta}.\boldsymbol{blob}(t)\big) \leftarrow (\boldsymbol{\varepsilon} = \{\boldsymbol{\alpha}, \boldsymbol{\beta}\})$$

Equation 3.4

3.5 Virtual Buoy States and State Transitions.

A virtual buoy exists in one of five states. In order to understand buoy states, transition conditions, assignment acceptance rules and occlusion preparation rules must first be discussed.

3.5.1 Blob Labeling Acceptance

A virtual buoy is assigned a unique blob every frame. Sometimes the system incorrectly labels a blob. Two observable conditions in all video sequences allow the system to detect incorrect blob assignments. First, image buoys have predictable velocities and positions through a sequence of frames. Second, the pixel areas of image buoys are consistent and predictable. Classifiers built on these observations dictate transitions in and out of tracking states at any point in time.

3.5.1.1 Position Consistency

An image buoy will traverse a sequence of frames at a slow pixel per frame rate. The system can classify a virtual buoy's blob history based on this fact. The proposed system uses a function that checks the position of a virtual buoy's blob at time t against its blob at time t-1. If the distance between the blob centers is under a threshold, then the blobs at those times are classified as image buoys. This process is repeated for a distance R frames in the past. Formally, a virtual buoy is classified as a representative of an image buoy in terms of center position if

$$\bigwedge_{t=0}^{-R}\left(dist\left(\boldsymbol{\alpha}.\boldsymbol{blob}(t), \boldsymbol{\alpha}.\boldsymbol{blob}(t-1)\right) < c_{motion}\right)$$

Equation 3.5

A hardcoded value that is small enough to detect jumps in blob positions, yet large enough to accept normal image buoy motion is assigned to the threshold c_{motion}. In the tested system, this value is 10 pixels.

3.5.1.2 Area Consistency

An image buoy's pixel area will grow at a slow and consistent rate until the image buoy becomes occluded due the frame bounds. The system can detect inconsistent blob assignments by comparing the area of a virtual buoy's blob at time t to its previous frame blob. If the ratio of areas is larger than a threshold, then the blob history is classified as non-representative of an image buoy. Formally, a virtual buoy is classified as a representative of an image buoy if

$$\bigwedge_{t=0}^{R}\left(\frac{max(\boldsymbol{\alpha}.\boldsymbol{blob}(t).area, \boldsymbol{\alpha}.\boldsymbol{blob}(t-1).area)}{min(\boldsymbol{\alpha}.\boldsymbol{blob}(t).area, \boldsymbol{\alpha}.\boldsymbol{blob}(t-1).area)} < c_{area}\right)$$

Equation 3.6

The member $\boldsymbol{\alpha}.\boldsymbol{blob}(t).area$ is the area of the blob assigned to buoy $\boldsymbol{\alpha}$ at time t. The use of the min( ) and max( ) functions ensure that the ratio produced is always greater than 1. A pixel value large enough to accept an image buoy's area growth, yet small enough to detect changes in blob selection or merged blobs is assigned to the threshold c_{area}. A ratio of 3 is used in the implemented system.

3.5.2 Preparing for Frame Occlusion

The blobs representing an image buoy become absent from the usable blob sets over a sequence of frames for two reasons. Either the image buoy itself has traversed off of the frame or noise in the optical sensor and segmentation module have blocked the image buoy's blob representation. Knowing the reason for the absence of a representative blob is used to select the appropriate detection or tracking algorithm. If the image buoy is known to be close to the frame bounds then an algorithm suited for the possibility of occlusion is selected. Flagging a virtual buoy as close to the frame bounds can be done by examining the virtual buoy's currently assigned blob. A flag is set if a virtual buoy's current blob is close to the frame bounds. The predicate used to denote that a blob is near the frame bounds is generated with

$$nearBounds(\boldsymbol{\alpha}, t) \leftarrow \left(\boldsymbol{\alpha}.\boldsymbol{blob}(t).minX < c_{pad}\right) \vee \left(\boldsymbol{\alpha}.\boldsymbol{blob}(t).minY < c_{pad}\right)$$
$$\vee \left(\boldsymbol{\alpha}.\boldsymbol{blob}(t).maxX > \left(\boldsymbol{frame}.width - c_{pad}\right)\right) \vee \left(\boldsymbol{\alpha}.\boldsymbol{blob}(t).maxY > \left(\boldsymbol{frame}.height - c_{pad}\right)\right)$$

Equation 3.7

The pixel distance between a blob and frame bounds at which the blob is flagged is assigned to c_{pad}. A value of 5 pixels is used for c_{pad} in the tested implementation of the system.

3.5.3 States of a Virtual Buoy

Virtual buoy states are used throughout the system to select appropriate detection or tracking algorithms, generate constraints, and select the correct blob set updating function. A virtual buoy is assigned one of five states at each time interval. The most common transition between states relies on the logic shown in Figure 3.3. In Figure 3.3 the "Stable" conditional diamond represents the functionality described in the Blob Labeling Acceptance section. The "In Bounds" conditional diamond represents the function discussed in the Preparing for Frame Occlusion section.

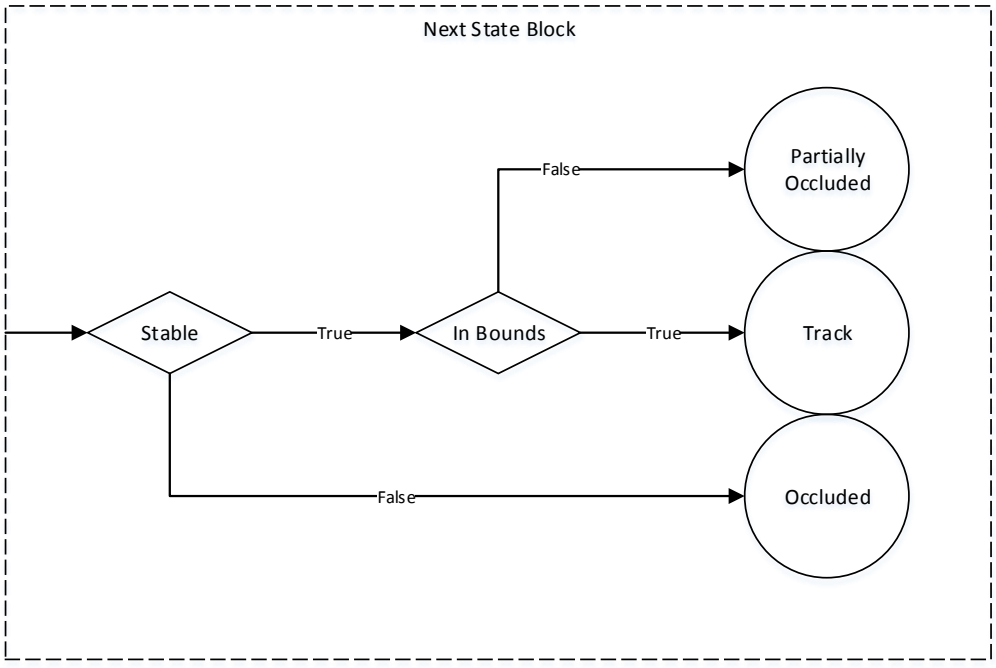


Figure 3.3: The Logic Defining Most State Transitions in the System.

3.5.3.1 Track State

A virtual buoy enters the tracking state when its recent blob history has been deemed representative of an image buoy. The system can use information from a buoy in the track state to increase its chances of correctly tracking or detecting the same or other buoys. The transition into the track state is shown in Figure 3.3.

3.5.3.2 Partially Occluded State

The partially occluded state is almost identical to the track state. The only difference between the two states is the flag that is set when a virtual buoy is near a frame bound. The separate outcomes of the "In Bounds" condition shown in Figure 3.3 illustrate this difference. This state's main purpose is to signal to the system that the image buoy associated with a virtual buoy has a higher chance of becoming occluded.

3.5.3.3 Search State

The system cannot use any information from a virtual buoy in the search state to detect or track image buoys. Transitions out of a search state occur under two conditions. Either the system determines that a virtual buoy pair is permanently occluded or both virtual buoys in a pair pass the stability tests described in the Blob Labeling Acceptance section. If both virtual buoys in a pair are deemed stable, then the virtual buoy receives the state described by the general transition block in Figure 3.3. Figure 3.4 displays the logic for a pair of buoys transitioning out of the search state. One difference between search state transitions and other transitions is that search state transitions only happen in pairs.

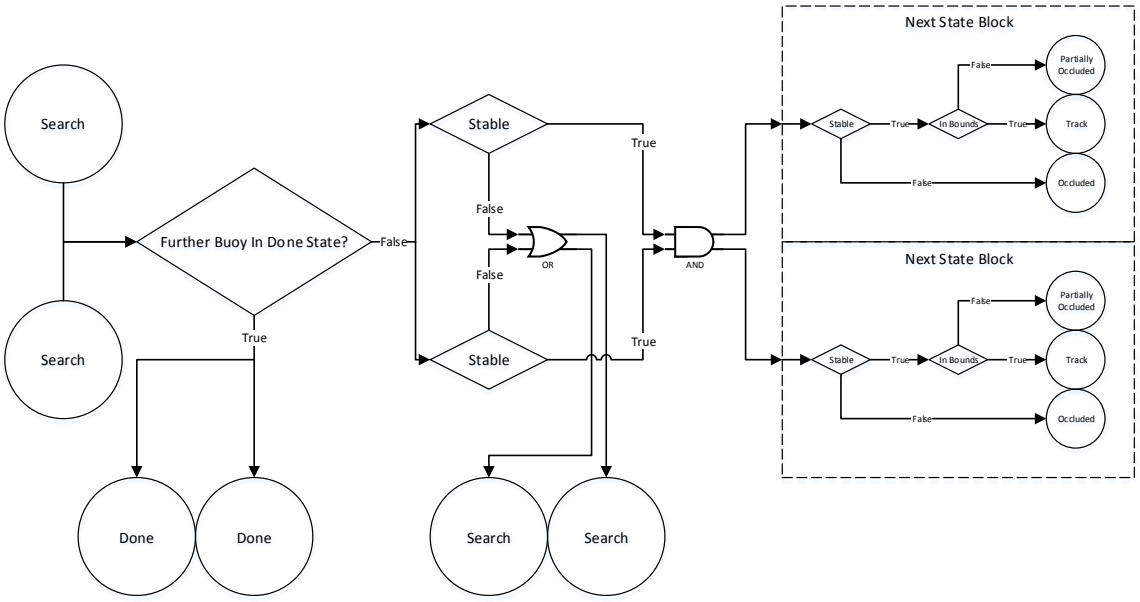


Figure 3.4: The Transition Diagram for a Pair of Virtual Buoys Transitioning out of Search State.

In Figure 3.4 the conditional "Further Buoy In Done State" is similar to the $\mathrm{LTIGT}(\alpha, t)$ function with the exception that the condition is looking for the least indexed done state virtual buoy.

3.5.3.4 Occluded State

A buoy in an occluded state is similar to a buoy in the search state. The main difference between the occluded state and the search state is that a virtual buoy in the search state has not yet been in the track or partially occluded state. A virtual buoy can only transition into the occluded state from the tracking or partially occluded state. The purpose of the occluded state is to signal to the system that a virtual buoy has gone through the expected transitions. Figure 3.5 shows the logic for transitioning out of the occluded state. Note that only a pair of virtual buoys can enter the done state from the occluded state.

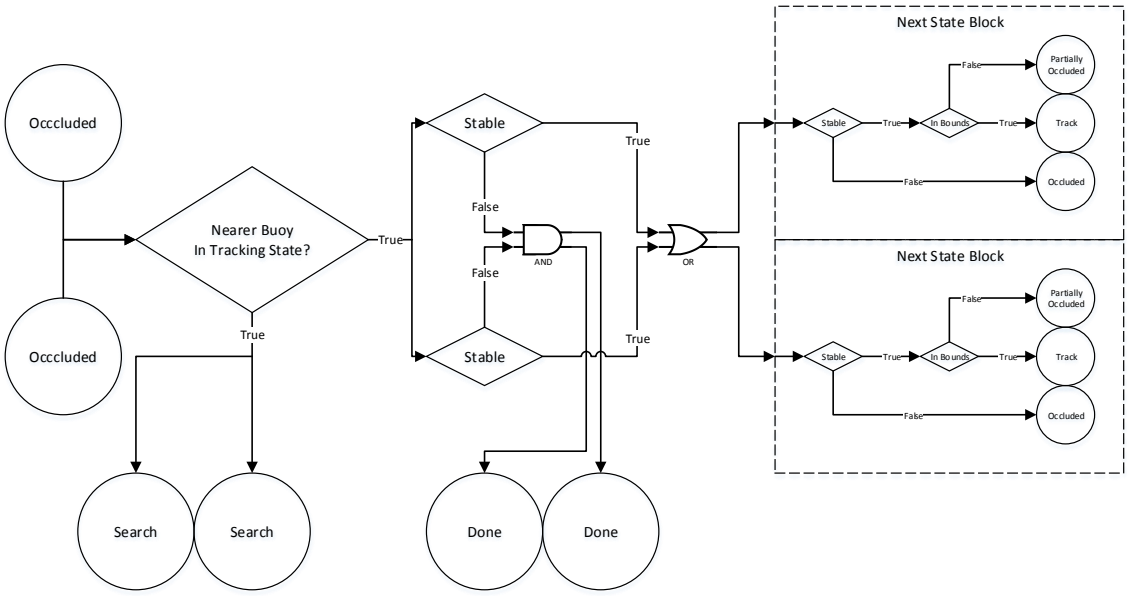


Figure 3.5: Transitions Diagram for a Pair of Virtual Buoys Transitioning out of Occluded State.

3.5.3.5 Done State

The done state signals that the actual buoy is out of the camera's field of view and will never return until system reset. The system cannot use any information from a virtual buoy in the done state to track or detect other buoys. Virtual buoys only transition out of the done state on reset. Resets are performed when the system is no longer tracking any buoys. Figure 3.6 shows the transition. Whenever one virtual buoy transitions out of the done state all virtual buoys transition out of the done state.

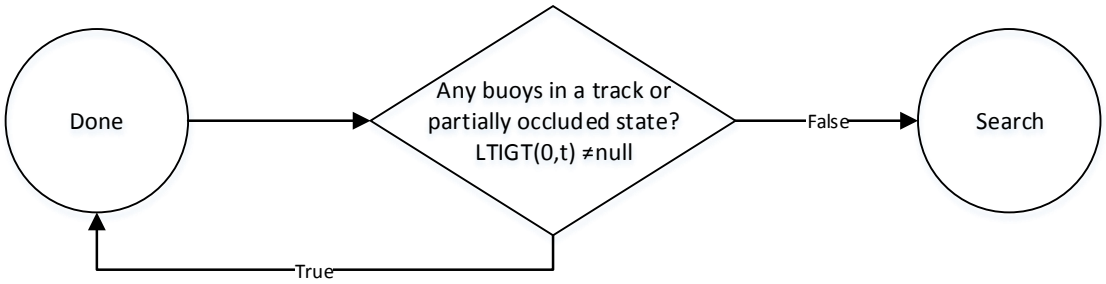


Figure 3.6: Transitioning out of Done State.

3.6 Blob Assignment

The system must assign each virtual buoy a unique blob every frame. The system attempts to accomplish this goal by processing virtual buoys in pairs. First the usable blob set(s) are permuted into every possible blob pair. Then every pair is ranked based on its suitability with the virtual buoy pair under consideration. The system ranks blob pairs using scoring functions. Scoring functions quantitatively assess blob pair to virtual buoy pair fitness based on a set of metrics. In addition to the scoring functions, the system also flags invalid blob pairs with constraint functions.

3.6.1 Constraining Functions

The segmentation module produces noisy sets of blobs. A blob set is noisy in the sense that some image buoys do not have representative blobs and some blobs in the set do not represent image buoys. Regardless of the noise, the blobs must be correctly labeled. A blob **d** or a pair of blobs **j** can be ruled as an invalid representation of virtual buoy **α** or pair of virtual buoys **ε** through the use of constraints. Two types of constraints, single constraints and pair constraints, are used to determine the validity of a blob or blob pair. Single constraints invalidate individual blobs, while pair constraints invalidate pairs of blobs. If a blob fails any of the multiple single blob constraints, it is removed from the usable set. If a blob pair fails any blob pair constraint, then the pair is removed from the set of pairs used for scoring. The end result is that only blob pairs that pass all single and pair constraints get scored.

3.6.1.1 Notation

The predicate $v_{(\mathbf{d},\boldsymbol{\alpha},\text{type},t)}$ is true (valid) when the blob **d** passes the constraint type for virtual buoy **α** at time t. $v_{(\mathbf{j},\boldsymbol{\varepsilon},\text{type},t)}$ synonymously means that the blob pair **j** is an

acceptable representation of virtual buoy pair $\boldsymbol{\varepsilon}$ using constraint type at time t. The symbol $c_{(\boldsymbol{\alpha},type,t)}$ is used to describe a constant, usually a threshold, for the virtual buoy $\boldsymbol{\alpha}$ and constraint type at time t. Similarly $c_{(\boldsymbol{\varepsilon},type,t)}$ describes a constant for constraint type for virtual buoy pair $\boldsymbol{\varepsilon}$ at time t.

3.6.1.2 Single Blob Constraints

Single blob constraints, also referred to as single constraints, invalidate individual blobs as potential representations of image buoys. If a single constraint invalidates a blob, then it is removed from the set of usable blobs available to the virtual buoy it failed under. Single constraints both optimize the labeling process and improve the consistency of the labeling module. They optimize the module by reducing the number of blobs the system must test as pairs and improve the overall robustness of the module by removing blobs that pair constraints might fail to remove. The single blob constraints presented were generated by making intuitive rules based on observations about image buoys over sequences of frames.

3.6.1.2.1 Unlocked Constraints

The system cannot obtain any useful blob validation information from its virtual buoys if none of its virtual buoys are in the track or partially occluded state. The system is said to be in an unlocked state under these circumstances. When the system is in the unlocked state, it is prone to making two types of errors. The first type of error occurs when the segmentation module produces a blob set that includes blobs representative of shoreline objects or significantly sized non-buoy floating objects. These types of objects often trick the scoring functions into producing scores that, as a result, make the system believes the objects are image buoys. The second type of error occurs when the incoming

blob set is large. If the incoming blob set is large, then the number of blob pairs to test is large resulting in the failure to meet real-time processing demands. To reduce these types of errors, a subclass of the single constraints was developed called unlocked constraints. Unlocked constraints are fired when the entire system is in an unlocked state. It is important to note that the unlocked constraints were designed to allow the system to successfully track only one pair of image buoys. Once the system has acquired tracking status on a pair of virtual buoys, it is out of the unlocked state and has enough information to successfully avoid the above two errors. The following unlocked constraints are used by the implemented system and help deal with the above two issues.

3.6.1.2.1.1 Unlocked Dimensionality

The blobs representative of the closest actual buoys have proportional widths and heights due to their assumed constant sizes. This observation is used to make a constraint that is true regardless of the cameras position and orientation in the world. Formally, a blob is validated with the rule

$$V_{(\boldsymbol{d},\boldsymbol{\alpha},uDimension,t)} \leftarrow \left(\boldsymbol{d}.width < \left(c_{(\boldsymbol{\alpha},uDimension,t)} * d.height\right)\right)$$
$$\wedge \left(\boldsymbol{d}.height < \left(c_{(\boldsymbol{\alpha},uDimension,t)} * \boldsymbol{d}.width\right)\right)$$

Equation 3.8

where $c_{(\boldsymbol{\alpha},\mathrm{uDimension,t})}$ is a hardcoded max dimensionality proportion. $c_{(\boldsymbol{\alpha},\mathrm{uDimension,t})}$ is necessary because the blobs given by the segmentation module are almost never completely square. A value of 3 has been found to be the best value for $c_{(\boldsymbol{\alpha},\mathrm{uDimension,t})}$. Figure 3.7 illustrates the result of applying the constraint. In this example, oblong blobs are removed from the set.

(a) (b) (c) (d)

Figure 3.7: The Result of the Unlocked Dimensionality Constraint. (a) Original Frame. (b) Original Blob Set. (c) Valid Blobs under the Unlocked Dimensionality Constraint. (d) Invalid Blobs.

3.6.1.2.1.2 Unlocked Area Less Than

As an actual buoy gets closer to the camera its corresponding image buoy also grows due to perspective projection. An image buoy can only grow to a specific maximum pixel area due to the camera's position above the water and the camera's field of view. Any actual buoys close enough to produce image buoys larger than the maximum pixel area cannot exist because they fall outside of the camera's field of view. This observation allows for the construction of an unlocked constraint based on area. Formally the Unlocked Area Less Than constraint is defined as

$$v_{(\boldsymbol{d},\boldsymbol{\alpha},uMaxArea,t)} \leftarrow \left(\boldsymbol{d}.area < c_{(\boldsymbol{\alpha},uMaxArea,t)}\right)$$

Equation 3.9

where $c_{(\boldsymbol{\alpha},\mathrm{uMaxArea,t})}$ is assigned the maximum pixel area the system would ever expect an image buoy to grow. The tested implementation of the system uses a value off 500 pixels for $C_{(\alpha,\mathrm{uMaxArea,t})}$. This value is for a constant frame size of 320 x 240 and a focal length of 3025 pixels. Figure 3.8 shows the result of applying the unlocked area constraint. In this example the blobs which are too large to be image buoys are removed from the usable blob set.

(a) (b) (c) (d)

Figure 3.8: The Result of Applying the Unlocked Area Less Than Constraint. (a) Original Frame. (b) Original Blobs. (c) Valid Blobs under the Constraint. (d) Invalid Blobs.

3.6.1.2.1.3 Unlocked Area Greater Than

The segmentation module generally produces consistently positioned and sized blobs for actual buoys closer to the camera. Actual buoys at a distance generally are usually converted into inconsistently positioned and sized blobs. Due to this observation, the system attempts to assign blobs representative of the closest virtual buoys first. The Unlocked Area Greater Than constraint is executed in the hope that the system will track nearer blobs. The cost of using the constraint is the potential rejection of blobs associated with actual buoys at a distance. Formally this constraint is

$$v_{(\boldsymbol{d},\boldsymbol{\alpha},uMinArea,t)} \leftarrow \left(\boldsymbol{d}.area > c_{(\boldsymbol{\alpha},uMinArea,t)}\right)$$

Equation 3.10

The system uses the threshold $c_{(\boldsymbol{\alpha},\mathrm{uMinArea,t})}$ to filter out blobs too small to ever represent close actual buoys. The tested system uses a value of 9 pixels for $c_{(\boldsymbol{\alpha},\mathrm{uMinArea,t})}$. Figure 3.9 shows the resulting sets of blobs before and after applying the constraint. In the example blobs too small to be image buoys are rejected.

(a) (b) (c) (d)

Figure 3.9: The Result of the Unlocked Area Greater Than Constraint. (a) Original Frame. (b) Original Blobs. (c) Blobs Valid under the Constraint. (d) Invalid Blobs.

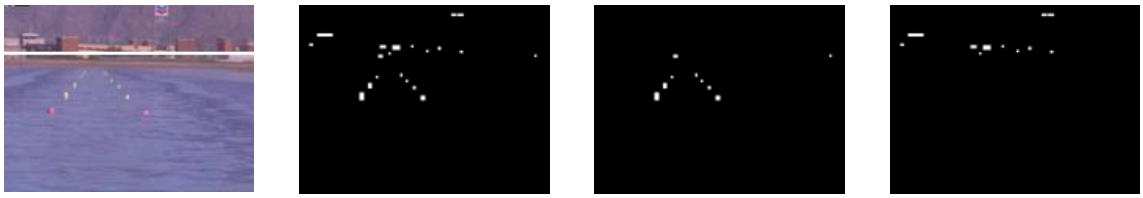
3.6.1.2.1.4 Center Below Shore Line

Actual buoys in the water are associated with image buoys that appear below the projected shoreline due to camera position and orientation assumptions. This observation in conjunction with a consistently estimated shoreline row allows for the system to invalidate blobs based on their positions relative to the shoreline. The row of the shoreline in a frame can filter individual blobs by the using the rule

$$v_{(\boldsymbol{d},\boldsymbol{\alpha},uShoreline,t)} \leftarrow \left(\boldsymbol{d}.\boldsymbol{center}.y > c_{(\boldsymbol{\alpha},uShoreline,t)}\right)$$

Equation 3.11

The threshold $c_{(\boldsymbol{\alpha},\mathrm{uShoreline,t})}$ is set to the row returned by the shoreline estimation algorithm for the frame being processed at time t. Figure 3.10 displays the effect of the constraint. In the example, blobs above the white shoreline row are rejected.



(a) (b) (c) (d)

Figure 3.10: Results of the Shoreline Row Constraint. (a) Original Frame with the Shoreline Row Drawn in White. (b) Original blobs. (c) Valid blobs. (d) Invalid blobs.

3.6.1.2.2 Locked Constraints

The system uses unlocked constraints to help correctly assign blobs and track at least one image buoy pair. Once a pair of virtual buoys moves into a tracking state, the unlock constraints are disabled and a new class of constraints fire. The new class of constraints allows the system to use information from tracking virtual buoy pairs in the constraining portion of detection and tracking methods.

3.6.1.2.2.1 Area Less Than

An actual buoy further from the camera always appears as image buoy whose area is less than the area of an image buoy associated with an actual buoy closer to the camera. This effect is due to perspective projection and the assumption that all actual buoys are the same size. If the system is constraining the blob set available to a virtual buoy associated with an actual buoy in the distance, and in the same frame is tracking an actual buoy closer to the camera, then the system can use the area assertion to its advantage. Blobs can be validated using the rule

$$v_{(\boldsymbol{d},\boldsymbol{\alpha},maxArea,t)} \leftarrow \left(\boldsymbol{d}.area < c_{(\boldsymbol{\alpha},maxArea,t)}\right)$$

Equation 3.12

The value $c_{(\boldsymbol{\alpha},\text{maxArea,t})}$ is set to the maximum pixel area a blob representative of virtual buoy $\boldsymbol{\alpha}$ can have. $c_{(\boldsymbol{\alpha},\text{maxArea,t})}$ is found by using the rule

$$\left(c_{(\boldsymbol{\alpha},maxArea,t)} = \boldsymbol{e}.area\right) \leftarrow (GTILT(\boldsymbol{\alpha},t) = \boldsymbol{\gamma}) \wedge (\boldsymbol{\gamma}.\boldsymbol{blob}(t) = \boldsymbol{e})$$

Equation 3.13

Figure 3.11 shows the result of this type of constraint on a blob set. In Figure 3.11 (a), the value of $c_{(\boldsymbol{\alpha},maxArea,t)}$ comes from the buoys denoted by the solid white

boxes and line. The dotted white line in the distance represents the virtual buoy pair the system is examining.

(a) (b) (c) (d)

Figure 3.11: The Result of the Area Less Than Constraint. (a) The Original Frame with a White Solid Line Signifying a Tracking Virtual Buoys Pair and a White Dotted Line Indicating the Buoy Pair under Consideration. (b) Original Blobs. (c) Valid Blobs under the Constraint. (d) Invalid Blobs.

3.6.1.2.2.2 Area Greater Than

Similar to the Area Less Than constraint, the Area Greater Than constraint invalidates blobs which are too small to be considered buoys based on the positions of actual buoys relative to the camera. The constraint only fires if the system is tracking a further actual buoy while constraining blobs for a nearer actual buoy. The actual buoys closer to the camera appear as image buoys that are larger than images of actual buoys further from the camera. This property is used to validate blobs with the rule

$$v_{(\boldsymbol{d},\boldsymbol{\alpha},minArea,t)} \leftarrow \left(\boldsymbol{d}.area > c_{(\boldsymbol{\alpha},minArea,t)}\right)$$

Equation 3.14

The system finds $c_{(\boldsymbol{\alpha},minArea,t)}$ using a tracking virtual buoy further in the distance or

$$\left(c_{(\boldsymbol{\alpha},minArea,t)} = \boldsymbol{e}.area\right) \leftarrow (LTIGT(\boldsymbol{\alpha},t) = \boldsymbol{\gamma}) \wedge (\boldsymbol{\gamma}.\boldsymbol{blob}(t) = \boldsymbol{e})$$

Equation 3.15

Figure 3.12 demonstrates the result of applying the constraint to individual blobs. In the example the system is constraining blobs for the image buoy pair marked with dotted

lines. The constraint generates its value of $c_{(\boldsymbol{\alpha},minArea,t)}$, from the buoys denoted by the white solid boxes and solid white line.

(a) (b) (c) (d)

Figure 3.12: The Result of the Area Greater Than Constraint. (a) Original Frame. (b) Original Blobs. (c) Valid Blobs. (d) Invalid Blobs.

3.6.1.2.2.3 Center Less Than

The system can invalidate blobs based on their positions in the frame relative to tracking image buoys. If the system is tracking an image buoy, then image buoys associated with actual buoys further from the camera should appear higher in the frame. This observation is used to validate individual blobs with the rule

$$v_{(\boldsymbol{d},\boldsymbol{\alpha},maxRow,t)} \leftarrow \left(\boldsymbol{d}.\boldsymbol{center}.y < c_{(\boldsymbol{\alpha},maxRow,t)}\right)$$

Equation 3.16

The row value of $c_{(\boldsymbol{\alpha},maxRow,t)}$ is found by using the rule

$$\left(c_{(\boldsymbol{\alpha},maxRow,t)} = \boldsymbol{e}.\boldsymbol{center}.row\right) \leftarrow (GTILT(\boldsymbol{\alpha},t) = \boldsymbol{\gamma}) \wedge (\boldsymbol{\gamma}.\boldsymbol{blob}(t) = \boldsymbol{e})$$

Equation 3.17

Figure 3.13 presents an example output of this constraint. The value $c_{(\boldsymbol{\alpha},maxRow,t)}$ comes from the solid white boxes in Figure 3.13 (a). In the example, all blobs below the highest (in terms of row) white box are removed from the usable set.

(a) (b) (c) (d)

Figure 3.13: The Result of the Center Less Than Constraint. (a) The Original Frame with a Solid White Line Denoting a Tracking Buoy Pair and a Dotted Line Signifying the Pair under Consideration. (b) The Original Blob Set. (c) The blobs Valid under the Constraint. (d) The Invalid Blobs.

3.6.1.2.2.4 Center Greater Than

Similar to the center less than constraint, blobs representative of actual buoys closer to the camera appear lower (in terms of row number) in the frame than image buoys further from the camera. If the system is tracking a virtual buoy in the distance and processing a closer pair of virtual buoys, then it can validate blobs constrained under the current pair with the rule

$$v_{(\boldsymbol{d},\boldsymbol{\alpha},minRow,t)} \leftarrow \left(\boldsymbol{d}.\boldsymbol{center}.row > c_{(\boldsymbol{\alpha},minRow,t)}\right)$$

Equation 3.18

The row value of $c_{(\boldsymbol{\alpha},minRow,t)}$ comes from the rule

$$\left(c_{(\boldsymbol{\alpha},minRow,t)} = \boldsymbol{e}.\boldsymbol{center}.row\right) \leftarrow (LTIGT(\boldsymbol{\alpha},t) = \boldsymbol{\gamma}) \wedge (\boldsymbol{\gamma}.\boldsymbol{blob}(t) = \boldsymbol{e})$$

Equation 3.19

Figure 3.14 shows an example of applying the constraint. In the example the solid white boxes represent a pair of tracking virtual buoys. In the example the blobs above the highest white box are removed from the usable set.

(a) (b) (c) (d)

Figure 3.14: An Example of Firing the Center Greater Than Constraint. (a) The Original Frame with White Boxes Representing the next Furthest Pair of Tracked Buoys and a Dotted Line Denoting the Pair under Consideration. (b) Original Blobs. (c) The Valid Blobs under the Constraint. (d) Invalid Blobs.

3.6.1.2.2.5 Lanes

If not apparent from given images of the slalom course, all buoys are nearly collinear with two lines. The exception is the gate buoys which are ten cm wide of perfect collinearity. Due to the near collinear nature of all buoys, the system can assume that the positions of all actual buoys can be described as points on one of two parallel lines. These parallel lines in the world project onto the image plane as lines which intersect at the point at infinity. Intuitively, one can start making rules about the positions of image buoys based on the observation that all buoys are on one of two parallel image lines. The system does just this. If the system is tracking two image buoys on one side of the slalom course, it expects all buoys on that side of the slalom course have positions somewhere along the line created by the centers of the two tracked buoys. The concept of lanes enforces this rule. A lane is a region in the frame where the system expects valid image buoys to exist. Figure 3.15 shows a typical frame captured by the system's camera and a lane constructed using two tracking image buoys on one side of the slalom course. The valid region is the region inside the two white lines where all image buoys are found.

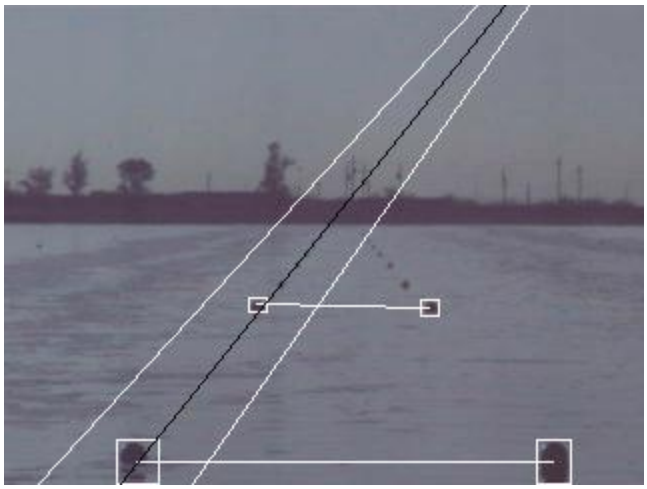


Figure 3.15: A Visual Representation of a Lane.

A lane is constructed by first performing a Deming regression on the centers of all tracking buoys on one side of the slalom course. The resulting best fit line is padded with a constant pixel value and angle. In Figure 3.15 the white lane bounds are not parallel lines due to the addition of the pad angle. In the system, the predicate $inLane(\boldsymbol{d}.\boldsymbol{center}, \mathbf{l_l}, t)$ is true if $\boldsymbol{d}$'s center is within the left lane's bounds. $\mathbf{l_r}$ denotes the right lane.

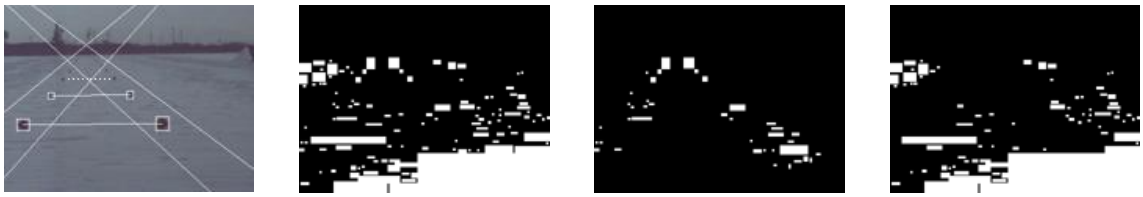
3.6.1.2.2.6 In Lane

If the system is tracking two or more image buoys on one side of the course then the system can construct lanes and validate blobs using the rule

$$v(\boldsymbol{d}, \boldsymbol{\alpha}, inLane, t) \leftarrow inlane(\boldsymbol{d}.\boldsymbol{center}, \boldsymbol{l_l}, t) \vee inlane(\boldsymbol{d}.\boldsymbol{center}, \boldsymbol{l_r}, t)$$

Equation 3.20

Figure 3.16 shows the effect of applying the constraint. In the example, blobs outside of the lanes are rejected.



(a) (b) (c) (d)

Figure 3.16: The Result of the In Lane Constraint. (a) The Original Frame. The Solid White Boxes Denote Tracking Buoys. The Diagonal Lines Illustrate the Lanes. (b) Original Blobs. (c) Blobs Validated under the Constraint. (d) Invalid Blobs.

3.6.1.3 Blob Pair Constraints

The blobs that pass single constraints are permuted into all possible pairs. Since the system uses blob pairs as it's basis for blob selection, the system must remove blob pairs that cannot represent image buoy pairs. The removal of invalid blob pairs is done with pair constraints. Like single constraints, pair constraints are based on observations made about sequences of frames.

3.6.1.3.1 Left Right

A blob pair denoted as {**d**, **e**} is different than the blob pair indicated by {**e**, **d**}. The first pair signifies that blob **d** is representative of the left image buoy and that blob **e** is representative of the right image buoy. The second pair indicates the opposite. Under the assumption that the camera is upright, blobs associated with actual buoys on the port side of the boat should appear on the left side of the frame while blobs representative of actual buoys on the starboard side of the boat should appear the right side of the frame. This constraint is enforced by checking the centers of the blobs in a pair and ensuring that the port side blob is positioned further left in the frame than the starboard side blob. This constraint is enforced by the rule

$$v_{(\boldsymbol{j},\boldsymbol{\varepsilon},leftRight,t)} \leftarrow (\boldsymbol{d}.\boldsymbol{center}.x < \boldsymbol{e}.\boldsymbol{center}.x) \wedge (\boldsymbol{j} = \{\boldsymbol{d},\boldsymbol{e}\})$$

Equation 3.21

Figure 3.17 shows the result of applying this constraint to pairs constructed using a specified port side blob. In the example, the invalidated blobs are the blobs located to the left of the specified port side blob.

(a) (b) (c) (d)

Figure 3.17: The Result of the Left Right Constraint. (a) Original Frame with a White Dotted Line between the Buoy Pair under Consideration. (b) The Original Blob Set with the Blob the System Considers the Left Blob in White. (c) The Blobs That Can Be in a Pair with the Specified Blob. (d)The blobs That Cannot Be in a Pair with the Specified Blob.

3.6.1.3.2 At Least One Inside

Under the assumptions of system operation, at least one image buoy from a pair of actual buoys further from the camera should lie inside the vertical frame bounds created with the centers of image buoys associated with a pair of closer actual buoys. In Figure 3.18, the vertical bounds mentioned are drawn in black on the original frame. Failure to meet this constraint means that either the camera's yaw is outside of the bounds stated in the assumptions, or the boats lateral displacement relative to the slalom course centerline is outside the bounds stated in the assumptions. This constraint is enforced by comparing the centers of a nearer tracking pair to the centers of a further pair. Formally, this rule is implemented using

$$v_{(\boldsymbol{j},\boldsymbol{\theta},oneInside,t)} \leftarrow \Big(\big(\boldsymbol{e}.\boldsymbol{center}.x < c_{(\boldsymbol{\theta},oneInsideMax,t)}\big) \wedge \big(\boldsymbol{e}.\boldsymbol{center}.x > c_{(\boldsymbol{\theta},oneInsideMin,t)}\big)\Big)$$
$$\vee \Big(\big(\boldsymbol{d}.\boldsymbol{center}.x < c_{(\boldsymbol{\theta},oneInsideMax,t)}\big) \wedge \big(\boldsymbol{d}.\boldsymbol{center}.x > c_{(\boldsymbol{\theta},oneInsideMin,t)}\big)\Big)$$

Equation 3.22

where the blob pair $\boldsymbol{j} = \{\boldsymbol{d}, \boldsymbol{e}\}$ and the values of $c_{(\boldsymbol{\theta},oneInsideMax,t)}$ and $c_{(\boldsymbol{\theta},oneInsideMin,t)}$ are produced using

$$\big(c_{(\boldsymbol{\theta},oneInsideMax,t)} = \boldsymbol{\beta}.\boldsymbol{blob}(t).\boldsymbol{center}.x\big) \leftarrow (\boldsymbol{GTILT}(\boldsymbol{\theta},t) = \boldsymbol{\alpha}) \wedge pair(\boldsymbol{\alpha},\boldsymbol{\beta})$$

Equation 3.23

$$\big(c_{(\boldsymbol{\theta},oneInsideMin,t)} = \boldsymbol{\alpha}.\boldsymbol{blob}(t).\boldsymbol{center}.x\big) \leftarrow (\boldsymbol{GTILT}(\boldsymbol{\theta},t) = \boldsymbol{\alpha}) \wedge pair(\boldsymbol{\alpha},\boldsymbol{\beta})$$

Equation 3.24

In these equations $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are the buoys from the nearer pair of tracking buoys. The nearest tracking pair is ordered as $\{\boldsymbol{\alpha}, \boldsymbol{\beta}\}$ meaning that the virtual buoy $\boldsymbol{\alpha}$ is expected to be represented by a blob whose column position is less than that of $\boldsymbol{\beta}$'s. Figure 3.18 displays the result of applying this constraint to blob pairs which include a specified blob (in white in Figure 3.18 (b)). In the example, the specified blob is purposefully selected as a blob outside of the bounds so that the effect of the pair constraint can be seen. The result is that blob pairs where at least one blob is inside the black vertical bounds of Figure 3.18 (a) are rejected.

(a) (b) (c) (d)

Figure 3.18: The Result of the At Least One Inside Constraint. (a) The Original Frame with White Solid Boxes Denoting a Tracking Pair and Black Lines Indicating the Bounds. (b) The Original Blob Set with a Blob in White Indicating the Specified Blob. (c) The Blobs That Can Be in a Pair with the Blob in White. (d) The Blobs That Cannot Be in a Pair with the Specified Blob.

3.6.1.3.3 At Least One Outside

At least one of the image buoys in a closer pair will lie outside of the vertical frame bounds constructed with the centers of a image buoys associated with an actual buoy pair further from the camera. This constraint is valid for the same reason that the At Least One Inside constraint is valid. Figure 3.19 shows the vertical bounds mentioned in black. This rule is enforced by checking candidate blob pairs against the centers of tracking virtual buoys associated with actual buoys further from the camera, or

$$v_{(\boldsymbol{j},\boldsymbol{\theta},oneOutside,t)} \leftarrow \Big(\big(\boldsymbol{d}.\boldsymbol{center}.x > c_{(\boldsymbol{\varepsilon},oneOutsideMax,t)}\big) \vee \big(\boldsymbol{d}.\boldsymbol{center}.x < c_{(\boldsymbol{\varepsilon},oneOutsideMin,t)}\big)\Big)$$
$$\vee \Big(\big(\boldsymbol{e}.\boldsymbol{center}.x > c_{(\boldsymbol{\varepsilon},oneOutsideMax,t)}\big) \vee \big(\boldsymbol{e}.\boldsymbol{center}.x < c_{(\boldsymbol{\varepsilon},oneOutsideMin,t)}\big)\Big)$$

Equation 3.25

where the blob pair $\boldsymbol{j} = \{\boldsymbol{d}, \boldsymbol{e}\}$ and the virtual buoy pair $\boldsymbol{\theta} = \{\boldsymbol{\gamma}, \boldsymbol{\delta}\}$. The values of $c_{(\theta,oneOutsideMax,t)}$ and $c_{(\theta,oneOutsideMin,t)}$ are produced using

$$\big(c_{(\boldsymbol{\theta},oneOutsideMax,t)} = \boldsymbol{\beta}.\boldsymbol{blob}(t).\boldsymbol{center}.x\big) \leftarrow (\boldsymbol{LTIGT}(\boldsymbol{\theta},t) = \boldsymbol{\alpha}) \wedge pair(\boldsymbol{\alpha},\boldsymbol{\beta})$$

Equation 3.26

$$\left(c_{(\boldsymbol{\theta},oneOutsideMin,t)} = \boldsymbol{\alpha}.\boldsymbol{blob}(t).\boldsymbol{center}.x\right) \leftarrow (\boldsymbol{LTIGT}(\boldsymbol{\theta},t) = \boldsymbol{\alpha}) \wedge pair(\boldsymbol{\alpha},\boldsymbol{\beta})$$

Equation 3.27

where the tracking pair is ordered as $\{\boldsymbol{\alpha}, \boldsymbol{\beta}\}$.Figure 3.19 shows the result of applying the constraint to blob pairs that include a specified blob. In the example, the blob selected to be in all pairs is purposefully picked as a blob inside the bounds so that the effect of the constraint can be seen. In the example the blobs rejected are the blobs that lie inside the vertical frame bounds in Figure 3.19 (a).

(a) (b) (c) (d)

Figure 3.19: The Result of Testing Blob Pairs under the At Least One Outside Constraint. (a) The Original Frame. (b) Original Blobs with the Specified Blob in White. (c) Blobs That Can Be in a Pair with the Specified Blob. (d) Blobs That Cannot Be in a Pair with the Specified Blob.

3.6.1.3.4 Distance Less Than

The distance, in pixels, between a pair of image buoys associated with an actual buoy pair at a distance from the camera is expected to be less than the pixel distance between a pair closer to the camera. If the system is tracking a pair of image buoys closer to the camera than the pair currently under consideration, it can use the distance between the close pair to filter invalid pairs. This constraint is

$$v_{(\boldsymbol{j},\boldsymbol{\theta},maxDist,t)} \leftarrow \left(dist(\boldsymbol{j}) < c_{(\boldsymbol{\theta},maxDist,t)}\right)$$

Equation 3.28

where the function d$ist(\boldsymbol{j})$ is defined in the Buoy and Blob Set Formulas section. The threshold $c_{(\boldsymbol{\theta},maxDist,t)}$ is obtained from a pair of tracking image buoys closer to the camera than the pair the system is processing, or

$$\left(c_{(\boldsymbol{\theta},maxDist,t)} = dist(\boldsymbol{\alpha},\boldsymbol{\beta})\right) \leftarrow (\boldsymbol{GTILT}(\boldsymbol{\theta},t) = \boldsymbol{\alpha}) \wedge pair(\boldsymbol{\alpha},\boldsymbol{\beta})$$

Equation 3.29

Figure 3.20 is an example of constraining blob pairs constructed with a specific blob. In the example, the blobs rejected as potential pair candidates with the blob in white are the blobs whose distance from the white blob is greater than the distance between the buoys in the tracking pair.

(a) (b) (c) (d)

Figure 3.20: The Result of Applying the Distance Less Than Constraint. (a) The Original Frame with a White Solid Line Representing a Tracking Pair and a White Dotted Line Indicating the Pair under Consideration. (b) The Original Blob Set with the Blob All Pairs Are Built with in White. (c) The Blobs Allowed to Be in a Pair with the Specified Blob. (d) The Blobs That Cannot Be in a Pair with the Specified Blob.

3.6.1.3.5 Distance Greater Than

The Distance Greater Than constraint is the opposite of the Distance Less Than constraint. It states that an image buoy pair associated with an actual buoy pair closer to the camera should have a greater distance between its elements than an image buoy pair associated with actual buoy pair further from the camera. The constraint is enforced with the rule

$$\boldsymbol{v}_{(\boldsymbol{j},\boldsymbol{\theta},maxDist,t)} \leftarrow \left(dist(\boldsymbol{j}) > c_{(\boldsymbol{\theta},maxDist,t)}\right)$$

Equation 3.30

The threshold $c_{(\boldsymbol{\theta},maxDist,t)}$ is obtained from a further tracking pair as

$$\left(c_{(\boldsymbol{\theta},maxDist,t)} = dist(\boldsymbol{\alpha},\boldsymbol{\beta})\right) \leftarrow (\boldsymbol{LTIGT}(\boldsymbol{\theta},t) = \boldsymbol{\alpha}) \wedge pair(\boldsymbol{\alpha},\boldsymbol{\beta})$$

Equation 3.31

Figure 3.21 shows an example of applying the constraint to all pairs built with a specific blob. In the example the blobs rejected as potential partners are the blobs whose distance from the specified blob is less than the distance between the elements of the tracking pair.

(a) (b) (c) (d)

Figure 3.21: The Result of the Distance Greater Than Constraint. (a) The Original Frame with a White Dotted Box Denoting the Buoy Pair under Consideration and White Solid Boxes Representing a Tracking Pair.(b) The Original Blob Set with the Blob Tested in All Pairs in White. (c) The Blobs That Can Be in a Pair with the Specified Blob. (d) The Blobs That Cannot Be in a Pair with the Specified Blob.

3.6.1.3.6 Slope Greater Than

The slopes of the lines between members of image buoy pairs are parallel regardless of the camera's rotation about the optical axis so long as the boat's position is within the bounds set forth in the assumptions section. The system uses tracking image buoy pairs to invalidate blob pairs by comparing their pair slopes to one another. If a blob pair angle, with respect to the frame rows and columns as axis, is far enough away from

the average tracking angle value, then the pair is rejected. To use this constraint, the system must first find the average slope of all tracking pairs with the formulas

$$m_{(\boldsymbol{\theta},avg,t)} = \frac{1}{numTracking(t)} \sum_{\boldsymbol{\theta}=0}^{numPairs} \begin{cases} m_\theta & \boldsymbol{\theta}.state(t) = track \\ 0 & \boldsymbol{\theta}.state(t) \neq track \end{cases}$$

Equation 3.32

$$numTracking(t) = \sum_{\theta=0}^{numPairs} \begin{cases} 1 & \boldsymbol{\theta}.state(t) = track \\ 0 & \boldsymbol{\theta}.state(t) \neq track \end{cases}$$

Equation 3.33

Once the system finds the average slope of all tracking pairs, it finds the slope of the line joining elements in a blob pair with

$$m_j = \frac{\boldsymbol{e}.\boldsymbol{center}.y - \boldsymbol{d}.\boldsymbol{center}.y}{\boldsymbol{e}.\boldsymbol{center}.x - \boldsymbol{d}.\boldsymbol{center}.x}$$

Equation 3.34

The positions of blobs are not exact projections of actual buoys so a buffer angle is subtracted from the average tracking pair angle to create the threshold bound.

$$c_{(\boldsymbol{\theta},minSlope,t)} = tan\left(atan\left(m_{(\boldsymbol{\theta},avg,t)}\right) - pad\right)$$

Equation 3.35

Finally, the system uses the threshold to constrain blobs with the formula

$$v_{(\boldsymbol{j},\boldsymbol{\theta},minSlope,t)} \leftarrow \left(m_j > c_{(\boldsymbol{\theta},minSlope,t)}\right)$$

Equation 3.36

Figure 3.22 shows an example of firing the constraint. In the example blob pairs created with the white blob whose connecting line has an angle less than pad radians below the average tracking pair angle are rejected.

(a) (b) (c) (d)

Figure 3.22: The Slope Greater Than Constraint. (a) The Original Frame. The White Solid Line Represents the Line Used to Determine the Threshold. The White Dotted Line Denotes the Pair under Consideration. (b) The Extracted Blob Set with the Specified Blob in White. (c) The Blobs Allowed to Be in a Pair with the Specified Blob by the Constraint. (d) The Blobs Rejected by the Constraint.

3.6.1.3.7 Slope Less Than

The Slope Less Than constraint is the other side of the Slope Greater Than constraint. The system uses the constraint to reject blobs whose pair angle is too far above the average pair slope using the formula

$$v_{(\boldsymbol{j},\boldsymbol{\theta},maxSlope,t)} \leftarrow m_j < c_{(\boldsymbol{\theta},maxSlope,t)}$$

Equation 3.37

The threshold value of $c_{(\boldsymbol{\theta},maxSlope,t)}$ comes from adding pad radians to the average tracking pair angle.

$$c_{(\boldsymbol{\theta},maxSlope,t)} = tan\left(atan\left(m_{(\boldsymbol{\theta},avg,t)}\right) + pad\right)$$

Equation 3.38

Figure 3.23 shows an example of the constraint. In the example the blob pairs rejected are the blobs whose angles are too far above the average tracking pair angle.

(a) (b) (c) (d)

Figure 3.23: The Result of Applying the Pair Slope Greater Than Constraint. (a) The Original Frame with the Tracking Pair in White Boxes and a Solid White Line. (b) Original Blob Set with the Blob Tested in All Pairs in White. (c) The Blobs Allowed to Be in Pairs with the Specified Blob under the Constraint. (d) The Blobs Invalidated by the Constraint.

3.6.1.3.8 One Per Lane

Since all buoys are near collinear with one of two parallel lines, the system can expect that one blob should be on one line, while the other blob is located on the other line. In other terms, blobs of an image buoy pair will not exist in the same lane. This constraint is enforced by checking the blobs position and determining which lane the blob lies in. If both blobs lie in one lane the pair is invalidated.

$$v(\boldsymbol{j}, \boldsymbol{\varepsilon}, inLane, t) \leftarrow inlane(\boldsymbol{d}.\boldsymbol{center}, \boldsymbol{l_l}, t) \wedge inlane(\boldsymbol{e}.\boldsymbol{center}, \boldsymbol{l_r}, t)$$

Equation 3.39

In the equation, $\mathbf{l_l}$ signafies the left lane and $\mathbf{l_r}$ denotes the right lane. Figure 3.24 shows an example of applying the constraint to blob pairs constructed with the blob in white from Figure 3.24 (b). The blobs rejected are the blobs whose centers do not lie in the opposing lane.

(a) (b) (c) (d)

Figure 3.24: The Result of the One Per Lane Constraint. (a) The Original Frame with Tracking Image Buoys in White Boxes and White Lines Drawn in Between. (b) The Original Blob Set with the Specific Blob in White. (c) The Blobs Allowed To Be in a Pair with the Specified Blob. (d) The Invalid Blobs under the Constraint.

3.6.2 Scoring Functions

After the constraint functions have removed invalid blobs and blob pairs, the system uses scoring functions to determine which blob maps to which buoy. A scoring function decides which blob is the best fit for a buoy by measuring the blob against some metric. Individually the metrics utilized by the system are weak at classifying the strength of a blob to buoy mapping. When multiple metrics are combined, however, the scoring functions consistently label blobs correctly. In this process either a pair of blobs **j**, **k**, or **m** is mapped to a pair of buoys **ε, θ, σ** or **λ** or an individual blob **d**, **e**, or **f** is mapped to an individual virtual buoy **α**, **β**, **γ** or **δ**. In a detection or tracking function multiple scoring metrics are used to create an overall blob to buoy score $s_{(\boldsymbol{d},\boldsymbol{\alpha},t)}$ or blob pair to buoy pair score $s_{(\boldsymbol{j},\boldsymbol{\varepsilon},t)}$. The overall score is created by adding weighted metric based scoring results. The sum of scores formula is

$$s_{(\boldsymbol{d},\boldsymbol{\alpha},t)} = w_{type_u} * s_{(\boldsymbol{d},\boldsymbol{\alpha},type_u,t)} + w_{type_v} * s_{(\boldsymbol{d},\boldsymbol{\alpha},type_v,t)} + \cdots$$

Equation 3.40

The two types of scoring functions presented can be logically split into two groups, order independent and order dependent. The following two sections discuss the scoring function used.

3.6.2.1 Order Independent

An order independent scoring function can fire on any blob pair at any time. Unlike the order dependent scoring functions, the order independent functions do not require that assigned blobs be removed from the set of usable blobs. The order independent scoring functions also do not require that the identification module iterate through buoys in a nearest to furthest manner.

3.6.2.1.1 Similar Slope

As shown in Figure 3.25 the slope of a line passing through a pair of image buoys is about the same as the slope of a line passing through other pairs of image buoys.

Figure 3.25: Pair Lines Drawn between Image Buoy Pairs.

An intuitive score criteria based this observation is the following: If blobs are in a pair, then the slope of the line joining the pair should be the same as the slopes of lines joining other pairs of image buoys. Formally this type of scoring is expressed as

$$s_{(\boldsymbol{j},\boldsymbol{\varepsilon},slope,t)} = \frac{min(m_{(avg,t)}, m_{\boldsymbol{j}})}{max(m_{(avg,t)}, m_{\boldsymbol{j}})}$$

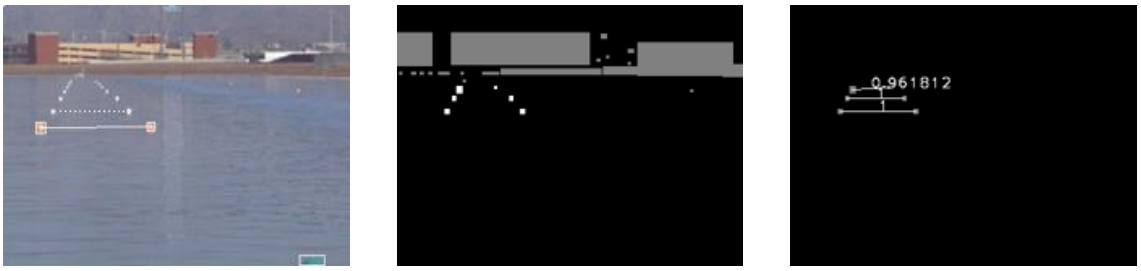
Equation 3.41

where $m_{(avg,t)}$ is the average slope of all tracking pairs at time t and $m_{\boldsymbol{j}}$ is the slope between the blobs in pair **j**. The min and max functions only ensure that the result is between zero and one. If no pairs of virtual buoys are in tracking states, then $m_{(avg,t)}$ is set to zero since the camera orientation assumptions state that the camera is upright**.** After scores are computed for every blob pair, the computed scores are normalized by comparing each individual score to the maximum score produced by the metric. This operation is done with

$$s_{(\boldsymbol{j},\boldsymbol{\varepsilon},slope,t)} = \frac{s_{(\boldsymbol{j},\boldsymbol{\varepsilon},slope,t)}}{max\left(s_{(\boldsymbol{k},\boldsymbol{\varepsilon},slope,t)}\right)}$$

Equation 3.42

where **j** represents the blob pair whose score is being normalized, and **k** represents any blob pair. Figure 3.26 visually illustrates this type of scoring. Figure 3.26 (a) is the original frame where the white line and boxes denote the tracking pair that $m_{(avg,t)}$ comes from. Figure 3.26 (b) displays the valid blob set to be scored in white and the unusable blobs in gray. Figure 3.26 (c) shows the scores of valid blob pairs.



(a) (b) (c)

Figure 3.26: The Output of the Similar Slope Scoring Function. (a) The Original Image with a Solid White Line Denoting the Average Slope Line. (b) The Blobs To Be Scored in White and Unusable Blobs in Gray. (c) The Result of Scoring Valid Blob Pairs with the Metric

3.6.2.1.2 Similar Area

Actual buoys of the same pair should be at about the same distance from the image sensor. Since actual buoys have the same dimensions, an intuitive scoring metric based off of this observation is: If blobs are representatives of image buoys from the same pair, then the blobs should have similar areas. The statement is turned into a scoring function with

$$s_{(\boldsymbol{j},\boldsymbol{\varepsilon},sim\_area)} = \frac{min(\boldsymbol{d}.area, \boldsymbol{e}.area)}{max(\boldsymbol{d}.area, \boldsymbol{e}.area)}$$

Equation 3.43

where it is assumed that the predicate $pair(\boldsymbol{d}, \boldsymbol{e})$ is true and blob pair $\boldsymbol{j} = \{\boldsymbol{d}, \boldsymbol{e}\}$. The score for a blob pair **j** then is normalized against all other pairs by using a method similar to Equation 3.42. Figure 3.27 shows the result of this type of scoring metric. The blob pairs the receive the best scores are the pairs whose elements are closest to the same size.

(a) (b) (c)

Figure 3.27: The Output of the Similar Area Scoring Function. (a) The Original Frame. (b) The Extracted Blob Set with Usable Blobs in White and Unusable Blobs in Gray. (c) The Scores Assigned to Valid Blob Pairs Using the Metric.

3.6.2.1.3 Similar Center

The pair scoring schema can be further used to track buoys through frame sequences by comparing a blob in the current frame against a tracking virtual buoy's blob from the previous frame. If it is assumed that **d** is the tracking virtual buoy's previously assigned blob, $\boldsymbol{d} = \boldsymbol{\alpha}.\boldsymbol{blob}(t-1)$ then the score computed for cross-temporal pair **j** = {**d**, **e**} is

$$s_{(\boldsymbol{j},\boldsymbol{\alpha},sim\_center,t)} = \frac{1}{1 + dist(\boldsymbol{d},\boldsymbol{e})}$$

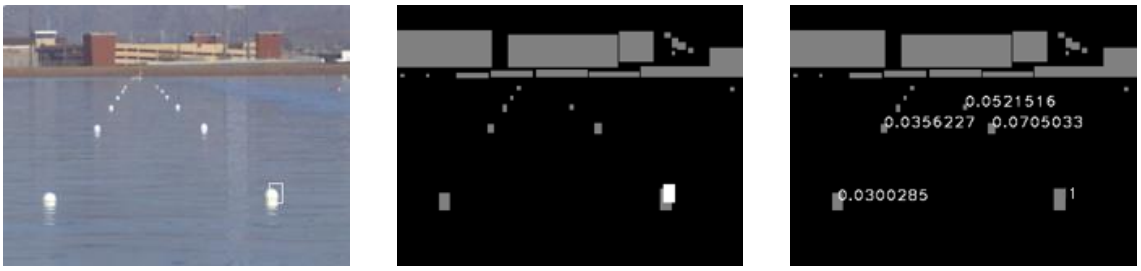
Equation 3.44

The score is normalized against all cross-temporal blob pairs using the function

$$s_{(\boldsymbol{j},\boldsymbol{\alpha},sim\_center,t)} = \frac{min\left(s_{(\boldsymbol{k},\boldsymbol{\alpha},sim\_center,t)}\right)}{s_{(\boldsymbol{j},\boldsymbol{\alpha},sim\_center,t)}}$$

Equation 3.45

where **k** is any blob pair and **j** is the pair being normalized. Note that the score applies to individual buoys instead of pairs of buoys. How to combine individual scores with pair scores is discussed in the Track Pair algorithm section. Figure 3.28 displays the results of scoring an individual blob from the previous frame against blobs in the current frame.



(a) (b) (c)

Figure 3.28: Result of the Similar Center Scoring Metric. (a) The Original Frame with a White Box Representing the Blob in the Previous Frame. (b) The Extracted Blobs from the Original Frame in Grey with the Previous Frame Blob in White. (c) The Scores for Some of the Blobs Using the Metric.

3.6.2.2 Order Dependent

The Order Dependent scoring functions require that the module map buoy pairs in the order of nearest pair to furthest pair from the camera. They also require that the module remove an assigned pair directly after selection. The order dependent scoring functions use perspective projection and the fact that the nearest buoy pair is under examination to generate the best mapping scores. Under the condition that buoy pairs are scored from nearest to furthest, a pair of buoys has the following properties when compared to other valid pairs: (1) its elements are the furthest apart, (2) its elements are the lowest in the frame, and (3) its elements are the largest in the frame.

3.6.2.2.1 Widest Pair

The pair of image buoys closest to the camera, and therefore the corresponding blobs, should be the widest pair due to perspective projection. The score based on the widest pair metric is generated for each blob pair **j** using

$$s_{(\boldsymbol{j},\varepsilon,widest,t)} = dist(\boldsymbol{j})$$

Equation 3.46

where $dist(\boldsymbol{j})$ is the distance formula. The scores produced are normalized against the maximum score in a manner similar to that of Equation 3.42. Figure 3.29 illustrates the result of this type of scoring metric. Figure 3.29 (a) is the original frame with a pair of tracking buoys in white boxes with a white line drawn in between. Figure 3.29 (b) is the usable blob set for the image buoy pair following the tracking pair. The white blobs are the usable blobs while the grey blobs are the unusable blobs. Figure 3.29 (c) is the result of scoring each of the valid blob pairs using the widest pair metric.

(a) (b) (c)

Figure 3.29: The Output of the Widest Pair Scoring Function. (a) The Original Frame. (b) The Usable Blobs in White and Unusable Blobs in Gray (c) The Scores Produced Using the Metric.

3.6.2.2.2 Lowest Pair

If the camera is upright and facing forward, then the nearest pair of actual buoys should appear as the lowest pair of image buoys in the frame. A scoring metric based on this observation gives pairs of blobs with lower positions in the frame a higher score. Assuming the predicate $pair(\boldsymbol{d}, \boldsymbol{e})$ is true and blob pair $\boldsymbol{j} = \{\boldsymbol{d}, \boldsymbol{e}\}$, the score for pair **j** is computed as

$$s_{(j,\varepsilon,lowest,t)} = \boldsymbol{d}.\boldsymbol{center}.y + \boldsymbol{e}.\boldsymbol{center}.y$$

Equation 3.47

and normalized against all other pairs with a method similar to Equation 3.42. Figure 3.30 (a) shows the original frame where the scoring function is being applied to the pair of buoys following the tracking image buoy pairs. Figure 3.30 (b) shows the usable blobs in white and the unusable blobs in gray. Figure 3.30 (c) displays the scoring of valid blob pairs using the lowest pair scoring metric.

(a) (b) (c)

Figure 3.30: The Output of the Lowest Pair Scoring Metric. (a) The Original Frame. (b) The Extracted Blobs with Usable Blobs in White and Unusable Blobs in Gray. (c) The Scores Assigned to Valid Blob Pairs Using the Metric.

3.6.2.2.3 Largest Pair

The pair of actual buoys closest to the optical sensor should appear as the largest pair of image buoys due to perspective projection and the assumption that all buoys are nearly the same size. This observation is utilized as a scoring metric by use of the function

$$s_{(\boldsymbol{j},\boldsymbol{\varepsilon},largest,t)} = \boldsymbol{d}.area + \boldsymbol{e}.area$$

Equation 3.48

The score $s_{(\boldsymbol{j},\boldsymbol{\varepsilon},largest,t)}$ is normalized using a method similar to Equation 3.42. Figure 3.31 illustrates the scoring metric. Figure 3.31 (a) is the original frame where the white boxes and lines represent tracking pairs. Figure 3.31 (b) shows the usable blob set for the

image buoy following the furthest tracking pair. Figure 3.31 (c) displays the scores assigned to valid blob pairs using the largest pair metric.

(a) (b) (c)

Figure 3.31: The Output of the Largest Pair Scoring Function. (a) The Original Frame . (b) The Blobs Extracted from the Original Frame with Valid Blobs in White and Invalid Blobs in Grey. (c) The Scores Produced with the Metric.

3.6.3 Updating A Usable Blob Set

The system would not work correctly if a single blob could be assigned to multiple buoys. If this were the case, then the proposed constraint and scoring process would map the same highest scoring blob to every buoy. To avoid this scenario, a blob assigned to a buoy is removed from the set of usable blobs before another buoy is processed. Blob set updating occurs after blob assignment and after the virtual buoy's state is updated using the freshly assigned blob. The blob is removed only if the buoy is in a tracking, partially occluded, or search state. Formally the rule for blob set updating is

$$\boldsymbol{\lambda}.\boldsymbol{blobset}(t) = \left(\boldsymbol{\sigma}.\boldsymbol{blobset}(t) - \boldsymbol{\alpha}.\boldsymbol{blob}(t)\right) \leftarrow$$

$$(\boldsymbol{\alpha}.state(t) = track) \vee (\boldsymbol{\alpha}.state(t) = search) \vee (\boldsymbol{\alpha}.state(t) = partiallyOccluded)$$

Equation 3.49

where $\boldsymbol{\lambda}$ is the next pair of virtual buoys to be processed and pair $\boldsymbol{\sigma}$ is the virtual buoy pair just processed. This function usually fires twice, once for each virtual buoy in the

pair **σ**. Some of the tracking and detection algorithms work one buoy at a time instead of buoy pair at a time. In this case the blob set updating function is

$$\boldsymbol{\beta}.\boldsymbol{blobset}(t) = \left(\boldsymbol{\alpha}.\boldsymbol{blobset}(t) - \boldsymbol{\alpha}.\boldsymbol{blob}(t)\right) \leftarrow$$

$$(\boldsymbol{\alpha}.state(t) = track) \vee (\boldsymbol{\alpha}.state(t) = search) \vee (\boldsymbol{\alpha}.state(t) = partiallyOccluded)$$

Equation 3.50

where β is the next buoy to be examined. Again, the updating procedure occurs after blob assignment and buoy state updating .

3.6.4 Detection And Tracking Algorithms

The system uses two detection algorithms and two tracking algorithms as a means of labeling blobs. Two versions of each type of algorithm exist to satisfy the necessity for a single buoy and buoy pair version. All of the four algorithms are similar. The differences between them are related to the system's need to respond to special cases of buoy detection and tracking. In all situations, the algorithm to be executed is selected specifically to incorporate the maxim amount of available information into blob labeling. The remainder of this section discusses the four algorithms and the conditions that prompt their execution.

3.6.4.1 Detect Pair

The system uses the detect pair procedure when it is processing a pair of virtual buoys that both exist in the search state. The pair detection algorithm is the most straightforward of the four detection and tracking methods. Its simplicity stems from the fact that virtual buoys only transition out of the search state in pairs. Due to this transition, the pair detection algorithm's only goal is to find and report the highest ranking blob pair. The pseudocode for the algorithm is the following:

detectPair(ε, t)

1. *[α, β] = ε*
2. *blobSet0 = α.blobSet(t)*
3. *blobSet1 = β.blobSet(t)*
4. *validSingles0 = constrainSingles(blobSet0)*
5. *validSingles1 = constrainSingles (blobSet1)*
6. *pairs = permutePairs(validSingles0, validSingles1)*
7. *validPairs = constrainPairs(pairs)*
8. *pairScores = scorePairs(validPairs)*
9. *winningPair = getHighestRank(pairScores, validPairs)*

In the pseudocode, the constrainSingles( ) function from lines 4 and 5 uses constraints mentioned in the Single Blob Constraints section. The function permutePairs( ) on line 6 constructs every possible blob pair under the restriction that the first element in the pair is from the first list of valid blobs, and the second element in the pair is from the second list of valid blobs. This method of pair generation is necessary to accommodate pair building with two unique blob sets.

The function constrainPairs( ) constrains blob pairs with constraints mentioned in the Blob Pair Constraints section. The blobs are scored based on the scoring functions mentioned in the Scoring Functions section. The getHighestRank( ) function returns the maximum scoring blob pair. If multiple blob pairs tie for the highest score, then one is selected at random. If no valid blob pairs are present for scoring, meaning all pairs were removed by the constrainPairs( ) function, then the null blob pair is returned by the detection function. The detect pair function's placement in relation to the data structures, state updating, and blob set updating functions is shown in Figure 3.32.

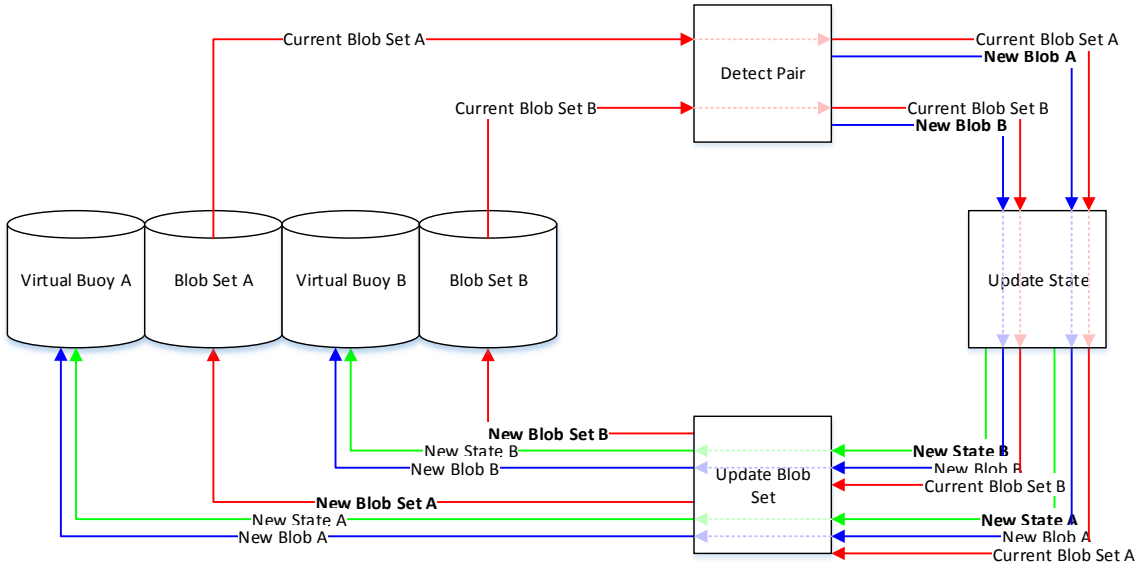


Figure 3.32: The Detect Pair Function's Placement in Relation to Other Functions.

3.6.4.2 Track Pair

The track pair function is executed if both elements of a virtual buoy pair are in the track state. The track pair procedure is very similar to the detect pair function. The difference between the track pair function and the detect pair function is the addition of scoring and constraints based on virtual buoy's previous blobs. Furthermore, there is extra logic to deal with all possible contingencies.

The basic idea behind the function is to first constrain and score blobs individually based on comparisons to the virtual buoy's previously assigned blob. If both sets of blobs are reduced to the empty set, then the function returns the null blob pair. If, after individual blob set constraints, one of the blob sets has elements while the other is empty, only the highest scoring blob from the populated list is returned. Finally, if both blob lists are populated, then the blob pairs are built and scored. The pseudocode for the track pair function is

```
trackPair(ε, t)
1.  [α, β] = ε
2.  blobSet0 = α.blobSet(t)
3.  blobSet1 = β.blobSet(t)
4.  validSingles0 = constrainSingles (blobSet0)   //only use locked constraints
5.  validSingles1 = constrainSingles (blobSet1)
6.  scores0 = scoreSingles(validSingles0)          //only use scoring functions meant to test blobs
7.  scores1 = scoreSingles(validSingles1)          //against virtual buoy blobs from the last frame
8.  if(validSingles0. isEmpty && validSingles1. isEmpty)
9.      winningPair = [nullSingle, nullSingle]
10. else if(!validSingles0. isEmpty && validSingles1. isEmpty)
11.     winningSingle = getHighestRank(scores0, validSingles0)
12.     winningPair = [winningSingle,nullSingle]
13. else if(validSingles0.isEmpty && !validSingles1.isEmpty)
14.     winningSingle = getHighestRank(scores1, validSingles1)
15.     winningPair = [nullSingle, winningSingle]
16. else          //both lists of valid blobs have at least one element.
17.     pairs = permutePairs(validSingles0, validSingles1)
18.     validPairs = constrainPairs(pairs)
19.     if(validPairs.isEmpty)     //if no valid pairs exist, return the highest ranking individual
20.             allSingleScores = merge(scores0,scores1)
21.             allValidSingles = merge(validSingles0, validSingles1)
22.             winningSingle = getHighestRank(allSingleScores,allValidSingles)
23.             winningPair = makePair(winningSingle.side, winningSinle)
24.     else
25.             pairScores = scorePairs(validPairs)
26.             extendedPairScores = addSingleScores(pairScores,leftScores,rightScores)
```

27. *winningPair = getHighestRank(extendedPairScores)*

In the pseudocode, the function addSingleScores( ) adds the individual blob scores to the blob pairs containing the individual blobs. The pseudocode for this function is

addSingleScores (pairScores, scores0, scores1)

1. *for i = 1:pairScores.size()*

2. *[single0,single1] = pairScores.blobs[i]*

3. *singleScore0 = getScore(scores0,single0)*

4. *singelSccore1 = getScore(scores1,single1)*

5. *pairScore = pairScores[i]*

6. *newScore = pairScore* singleScore0*singelSccore1*

7. *extendedScores[i] = newScore*

The track pair function is executed relative to other updating functions at a time similar to the execution time of the detect pair function. Figure 3.33 shows the track pair functions placement with respect to buoy state updating functions and blob set updating functions.

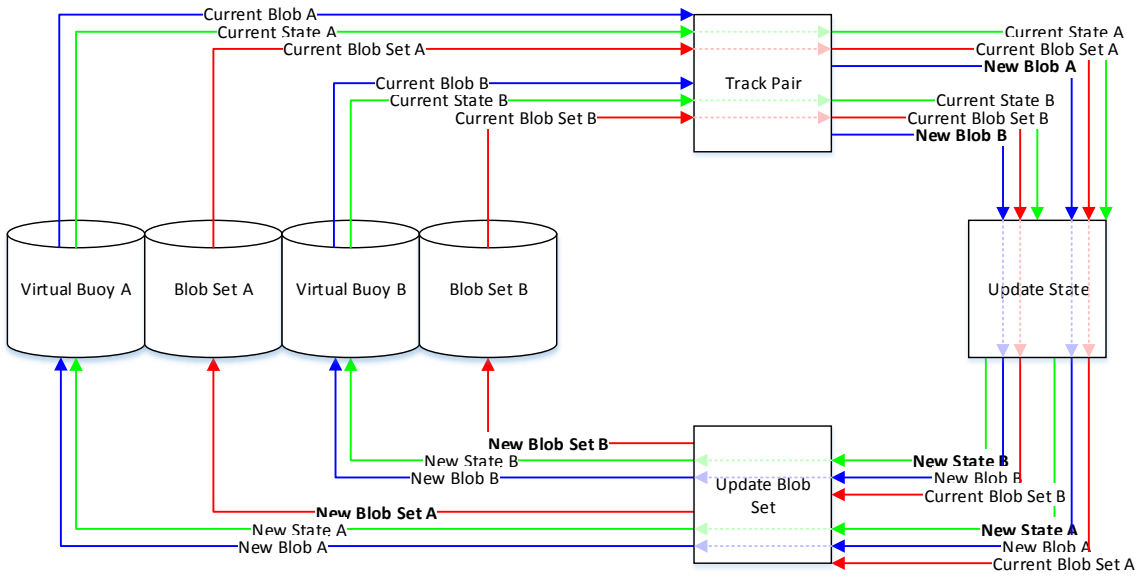


Figure 3.33: The Placement of the Track Pair Function in Relation to Other Functions and Data Structures.

3.6.4.3 Detect Single

The system executes the Detect Single buoy function when a pair of virtual buoys exists in the occluded and tracking state. In this case, the function uses as much information from the tracking buoy as possible when searching for the occluded buoy. First, it checks the state of the partner virtual buoy. If the other buoy is in a tracking state, then pairs are generated where at least one of the blobs is the tracking blob representing the tracking buoy. These pairs are then constrained and scored normally using pair constraints and pair scoring functions. The scores are ordered and the winning blob pair contains the tracking buoy's blob and the correct representation of the occluded blob. The pseudocode for the function is

*detectSingle(**ε**, currentSide , t)*

1. *[α, β] = ε*
2. *if(currentSide == port)*
3. *γ = α , δ = β // γ is the virtual buoy the function is running on*
4. *else*
5. *γ = β, δ = α*
6. *blobSet0 = γ.blobSet(t)*
7. *blobSet1 = δ.blob(t) // blob set is one element, the partner buoy's selected blob*
8. *if (tracking(γ,t))*
9. *validBlobs0 = constrainSingles(blobSet0)*
10. *validBlobs1 = constrainSingles(blobSet1)*
11. *pairs = permutePairs(validBlobs0,validBlobs1)*
12. *validPairs = constrainPairs(pairs)*
13. *pairScores = scorePairs(validPairs)*
14. *winningPair = getHighesRank(pairScores, validPairs)*

15. winningSingle = getSingleFromSide(winningPair,currentSide)

16. else //the other virtual buoy is not in a tracking state, nothing can be done

17. winningSingle = nullSingle

The Placement of the detect single function in relation to other components of the system is shown in Figure 3.34. The function will always be executed after the track single function has been executed due to its reliance on a partner buoy's current information.

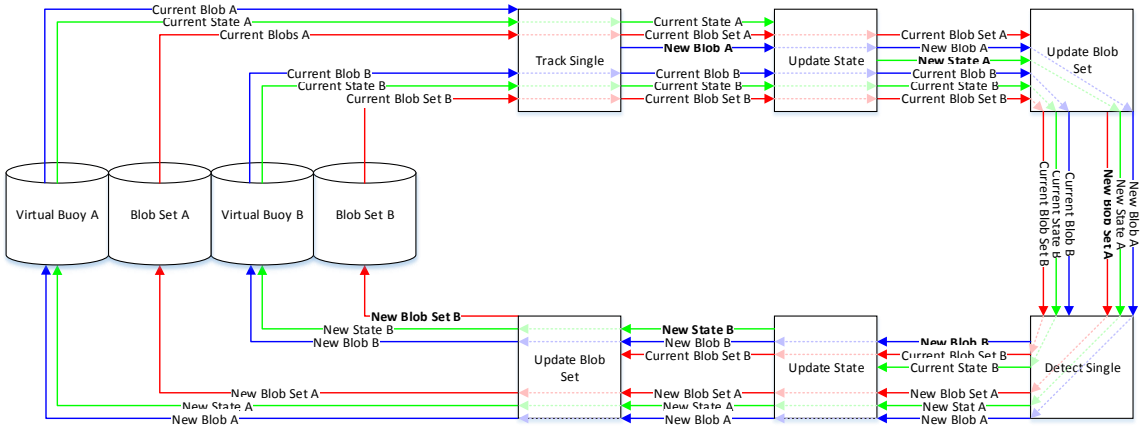


Figure 3.34 The Detect Single Function's Placement in Relation to other Functions When Tracking Then Detecting.

3.6.4.4 Track Single

The system uses the Track Single function when one of two conditions are met. First, if one virtual buoy in a pair has been marked as occluded, from the previous frame, and the other is tracking, then the Track Single function is executed in the hopes that the tracking buoy can still be found in the current frame. Second, if one virtual buoy in the pair has become partially occluded, then the system uses the Track Single function twice in a row instead of the Track Pair function. This approach is used in order to handle the case in which the partially occluded buoy has become occluded. The pseudocode for the function is

trackSingle(α, t)

1. *blobSet0 = α.blobSet(t)*
2. *blobSet1 = α.blob(t-1) //the other blob set is the previous frame blob.*
3. *validBlobs0 = constrainSingles(blobSet0)*
4. *validBlobs1 = constrainSingles(blobSet1)*
5. *pairs = permutePairs(validBlobs0,validBlobs1)*
6. *pairScores = scorePairs(validPairs) //only cross-temporal scoring functions are used.*
7. *winningPair = getHighesRank(pairScores, validPairs)*
8. *winningSingle = winningPair[0]*

The placement of the track single function in the first aforementioned case is shown in Figure 3.34. The placement for the second case is shown in Figure 3.35.

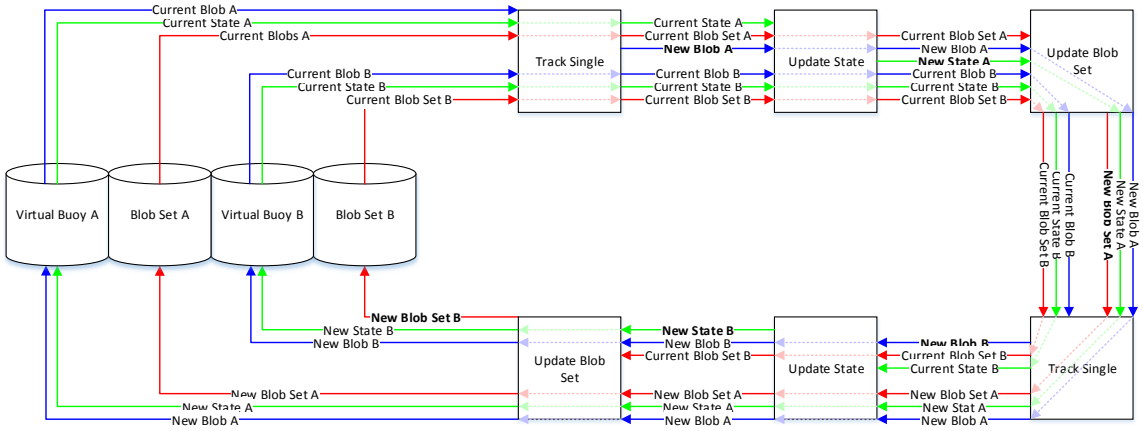


Figure 3.35: The Track Single Function's Placement When Tracking Two Buoys Individually.

3.7 Validation of the labeling system

Validation of the identification module can be found in the Control chapter of the thesis. In the Control chapter, a full system experiment is presented in which the entire system operates as intended. Real system operation can be seen as evidence that the ideas presented in the Identification chapter correctly fit blobs to buoys.

3.8 Future Research

The generate and constrain method presented uses many intuitive and mathematically simple constraint and scoring metrics. Improvement to the module can be made by using more complex constraint and scoring metrics. An example of such a constraint is one that uses a predicted pose estimate, most likely from a Kalman filter, to estimate the position of buoys in images.

Outside of what is available to the current system, it is also important to consider augmenting the system with other sensors such as a GNSS receiver, accelerometer, gyroscope, or digital compass. If the accelerometer, gyroscope and digital compass were calibrated with the system's camera such that the camera's orientation and height above water could be determined from non-camera sensor data, and the planar position of the camera was found via a GNSS, then it would be possible to estimate the position of buoys in incoming frames. The integration of more sensors could improve the robustness and possibly the processing time requirement of the identification module.

4 POSE ESTIMATION

4.1 Introduction

The two modules presented so far will turn a frame into a set of blobs and a set of correspondences between those blobs and buoys in the world. The task now is to use the blob-buoy mapping to estimate the position and orientation of the camera.

4.1.1 Formal Statement of the Problem

The pose estimation module is given three pieces of information from which it needs to find camera pose: (1) a set of image coordinates extracted from the blobs produced by the identification module, (2) a set of world coordinates, and (3) a mapping between image coordinates and world coordinates. Using these three pieces of information, the pose estimation module needs to estimate the position and orientation, in reference to the world, of the camera when the original frame was captured. The generic version of this type of problem is often called the Perspective-n-Point (PnP) problem [39].

The pose estimation module's problem can be viewed as a degenerate case of the PnP problem. The problem is degenerate due to its extra constraints on both the configuration of object coordinates as well as the allowable camera positions and orientations. In terms of the object coordinate configuration, the presented problem has four extra constraints. First, all object coordinates are coplanar. Second, all object coordinates lie on one of two parallel lines. Third every object coordinate **a** on line **l1** has a corresponding "pair" coordinate **b** on **l2**. **b** is located at the intersection of **l2** and a line that perpendicularly intersects **l1** at **a**. Finally, the distance between two points on the

same line, $|\overrightarrow{\boldsymbol{ca}}|$, is significantly larger than the distance between pair points on opposing lines, $\left|\overrightarrow{\boldsymbol{ab}}\right|$. Figure 4.1 shows an example of this type of configuration.

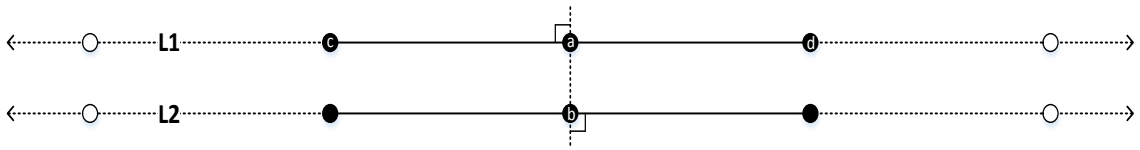


Figure 4.1: A Valid Point Configuration for the Pose Estimation Module.

It is important to note that the object coordinate configuration constraints do not include the constraint $\overrightarrow{\boldsymbol{ca}} = \overrightarrow{\boldsymbol{ad}}$.

The camera orientation and position constraints for the degenerate version of the problem are similar to those that have been expressed in other chapters of this thesis. These constraints are the following: (1) the camera is located between line **l1** and **l2**, (2) the camera's optical axis is near parallel to **l1** and **l2**, and (3) the distance between the camera and the object coordinate plane is nonzero and less than the distance $\left|\overrightarrow{\boldsymbol{ab}}\right|$.

Beyond the goal of finding a pose estimate for the degenerate PnP problem presented above, the pose estimate module must also make its computations in a minimal amount of time and produce robust and consistent results. The time requirement is necessary to meet real-time constraints and to allow other modules more processing time. The robustness expectation is necessary so that post processing methods that detect invalid pose estimates can be avoided.

4.1.2 Significance of the Research

Pose estimation is a fundamental problem in computer vision. Many applications require some form of camera pose estimation in their processing pipeline so that the systems can interact with the world. This chapter contributes to the breadth of knowledge

associated with pose estimation in two ways. First, it introduces two new methods designed specifically for the degenerate PnP problem presented. Second, it provides a quantitative, problem specific analysis of both preexisting methods and proposed methods.

4.1.3 Notation

Before presenting pose estimation methods, the common notation used throughout the rest of the chapter must be presented. The ideas and important concepts behind object, camera, and image space, as well as their relationships will also be discussed. In general, a lowercase bold letter such as "**v**" represents a column vector. A scalar element within vector **v** is expressed with subscripts such as "$\boldsymbol{v}_1$". An uppercase bold letter such as "**A**" represents a matrix. A matrix is also indexed with subscripts such as "$\boldsymbol{A}_{a,b}$" denoting the a'th row and b'th column of the matrix. Any superscript on a scalar, vector, or matrix usually represents the space in which the value resides. For example, "$\boldsymbol{p}^i$" represents vector element **p** which is a member of image space. Superscripts are also sometimes concatenated with other letters to denote special elements within the space. For example, "$\boldsymbol{p}^{\infty i}$" may represent a vanishing image coordinate. The superscript "T" such as that in "$\boldsymbol{A}^T$" denotes the transpose operation. Any other special points, or spaces will be introduced prior to their use.

4.1.3.1 Object Space and Camera Space

In all of the following pose estimation methods, there are two 3D coordinate systems. These are the camera coordinate system and the object coordinate system. Elements of these coordinate systems are sometimes referred to as elements of "object space" or "camera space." Sometime the term "world" is used to describe object space.

The interchangeability of these two words stems from the fact that the goal of the pose estimation module is to calculate the pose relation between the camera and a slalom course object, an object that is anchored to the world. In general, the object system has its origin at some user or method defined coordinate and the camera coordinate system has its origin at the camera's center of projection $\boldsymbol{o}^c$.

Any coordinate in the object coordinate system is denoted by $\boldsymbol{w}_a^o$. The superscript "o" denotes that the coordinate is an object space coordinate and the subscript "a" indexes the coordinate. The object coordinate $\boldsymbol{w}_a^o$ has three scalar components x_a^o, y_a^o, and z_a^o. Any coordinate in the camera coordinate system is denoted as $\boldsymbol{w}_a^c$. Synonymous with object coordinates, the superscript "c" denotes that a coordinate is a camera space coordinate and the subscript "a" indexes the coordinate. $\boldsymbol{w}_a^c$ has 3D components x_a^c, y_a^c, and z_a^c.

4.1.3.2 Image Space

Camera space an object space represent 3D coordinates in the world. Image coordinate are best expressed as either 2D coordinates or 3D homogeneous coordinates. All image coordinates are expressed with a superscript "i" which represent membership to image space, and a subscript "a" which indexes the element. In general, any image coordinate $\boldsymbol{i}_a^i$ has two $\boldsymbol{i}_a^i = \left[x_a^i, y_a^i\right]^T$, or in the homogeneous case three, $\boldsymbol{i}_a^i = \left[sx_a^i, sy_a^i, s\right]^T$, components.

One important image coordinate is the principal point of the image plane. This is the point at which the optical axis of the camera lens intersects the image plane. This

point is referenced as $\boldsymbol{i}^{pi} = \left[x^{pi}, y^{pi}\right]^T$. Another important image space term that is not an image coordinate but a value expressed in image space units is the focal length f^i.

4.1.4 Fundamental Conversions

With some notation now established, the relationship between object, camera, and image space will be discussed. Transformations from one space to another will be presented and any needed elements involved with the transformations will be defined.

4.1.4.1 The Relationship between Object Space and Camera Space

The relationship between the camera coordinate system and the object coordinate system can be described with equations that model rigid transformations. A rigid transformation maps one space to another and ensures that the distances between elements are preserved. A rigid transform in 3D requires three rotation and three translation elements.

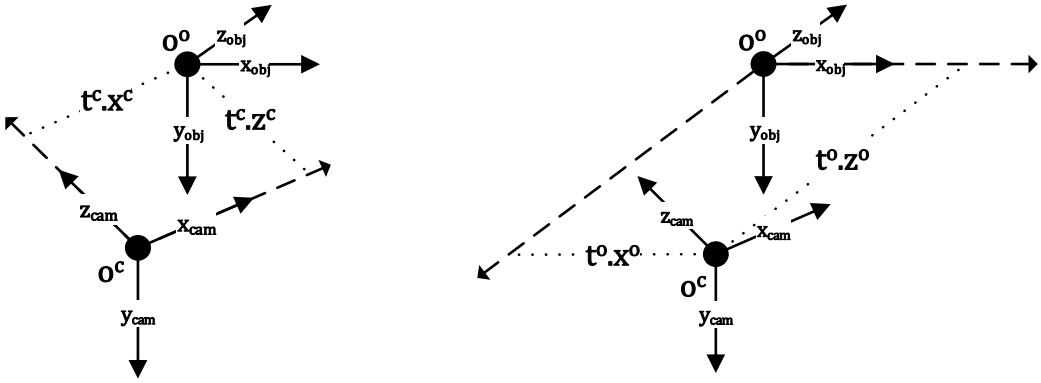
Rotation is often designated by the rotation matrix **R**. The rotation matrix **R** is equivalent to the $3x3$ matrix

$$\boldsymbol{R} = \begin{bmatrix} i_{X^o} & i_{Y^o} & i_{z^o} \\ j_{X^o} & j_{Y^o} & j_{Z^o} \\ k_{X^o} & k_{Y^o} & k_{Z^o} \end{bmatrix}$$

Equation 4.1

where each row of **R** equivalent to the unit vectors $\hat{\boldsymbol{\imath}}$, $\hat{\boldsymbol{\jmath}}$, and $\hat{\boldsymbol{k}}$. The row vectors in **R** are orthonormal vectors that express the direction of the camera coordinate system's x-, y-, and z-axis in object space. If the desired that rotation be in terms of Euler angles rather than unit vectors, methods such as [40] can be used to convert between the two representations.

Translation is denoted by the three element vector **t**. The pose estimation methods presented further in this thesis will produce two types of translation vectors. The first is the translation vector $\boldsymbol{t}^{o}$. It represents the translation of a camera expressed in the object coordinate system. It has object space components x^{o}, y^{o}, and z^{o}. The second type of translation vector is $\boldsymbol{t}^{c}$. It signifies the translation of the object in the camera coordinate system. The vector $\boldsymbol{t}^{c}$ has camera space components x^{c}, y^{c}, and z^{c}. A illustration of each type of translation vector is depicted in Figure 4.2.



(a) (b)

Figure 4.2: An Illustration of the Components That Form the Two Types of Translation Vectors from Two 3D Axis which Differ by an X-Z Translation and Rotation about the Y-Axis. (a) The $\mathbf{t}^{c}$ Vector Components. (b) The $\mathbf{t}^{o}$ Vector Components.

The rigid transformation equations that use each type of translation vector are the following:

$$\boldsymbol{w}^{c} = \boldsymbol{R} * \boldsymbol{w}^{o} + \boldsymbol{t}^{c}$$

Equation 4.2

$$\boldsymbol{w}^{c} = \boldsymbol{R} * (\boldsymbol{w}^{o} - \boldsymbol{t}^{o})$$

Equation 4.3

Both types of translation vectors are useful, however, the type used in Equation 4.2, $\boldsymbol{t}^{c}$, is not desirable for a system that requires the position of the camera expressed in world coordinates. The two types of translation vectors $\boldsymbol{t}^{c}$ and $\boldsymbol{t}^{o}$ are related by the rotation matrix **R**. The relation between $\boldsymbol{t}^{c}$ and $\boldsymbol{t}^{o}$ is

$$\boldsymbol{t}^{o} = -\boldsymbol{R}^{-1} * \boldsymbol{t}^{c}$$

Equation 4.4

4.1.4.2 The Relationship between Camera Space and Image Space

The classic pinhole camera model is used to model the system's camera. For all of the following pose estimation methods, it is assumed that the focal length is known and the camera is intrinsically calibrated. With this assumption, the relationship between camera space and image space can be described with perspective projection or

$$x_a^i = \frac{f}{z_a^c} * x_a^c \qquad\qquad y_a^i = \frac{f}{z_a^c} * y_a^c$$

Equation 4.5

The pinhole model of the camera can be seen in Figure 4.3.

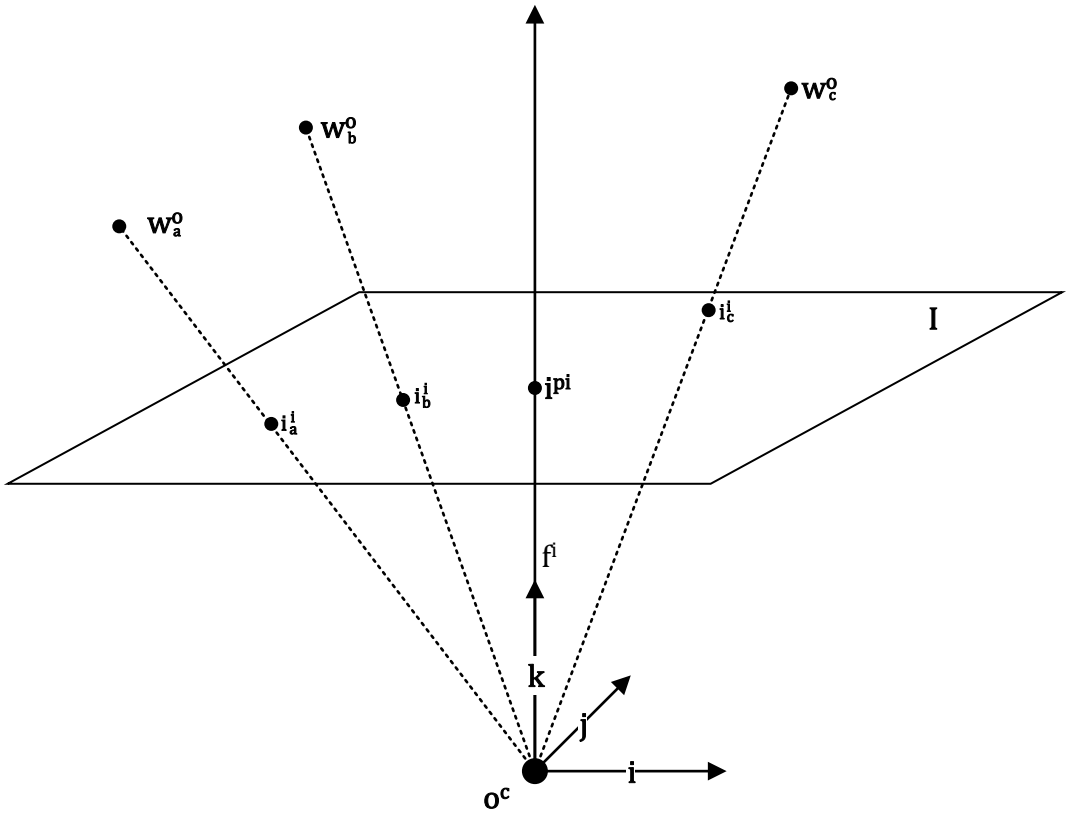


Figure 4.3: Pinhole Camera Model. Object Coordinates $\boldsymbol{w}_a^o$, $\boldsymbol{w}_b^o$ and $\boldsymbol{w}_c^o$ Project onto the Image Plane **I** by Perspective Projection to Points $\boldsymbol{i}_a^i$, $\boldsymbol{i}_b^i$ and $\boldsymbol{i}_c^i$ Respectively.

4.2 Review of Related Literature

The machine vision community has dealt with the problem of calculating camera pose numerous times. Some methods are for the generic case [41,42,43] and some methods are specially built for degenerate cases of the problem. These include methods for coplanar points [44], methods that rely on vanishing geometry[45,46,47], and methods which look for a specified number of point correspondences [48,49].With that being said, the problem faced by the system does not appear to have any specifically tailored methods associated with it. In the subsequent text, tested methods, or methods that significantly contribute to the two proposed methods are reviewed in depth.

4.2.1 Iterative Methods

4.2.1.1 Posit Coplanar

The posit algorithm [41,44] is an iterative algorithm that initially estimates object pose by making assumptions about the depth of object coordinates along the optical axis of the camera. The pose of the object is estimated with assumed values, and the resulting estimate provides improved assumption values. The algorithm repeats and the pose estimate parameters often converge on their true values.

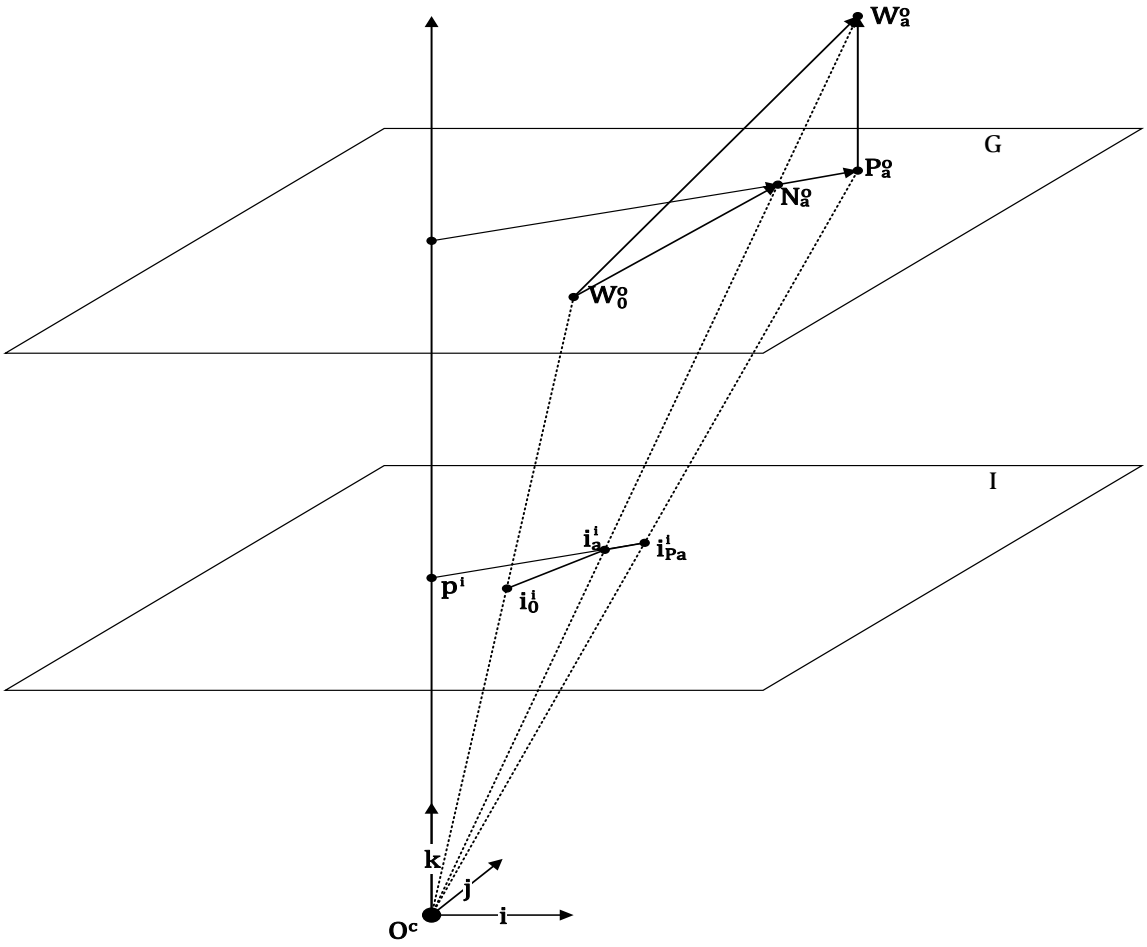


Figure 4.4: Posit Diagram. The World Coordinates $\boldsymbol{w}_0^o$ and $\boldsymbol{w}_a^o$, Their Image Coordinates $\boldsymbol{i}_0^i$ and $\boldsymbol{i}_a^i$, the Scaled Orthographic Projection $\boldsymbol{p}_a^o$ of $\boldsymbol{w}_a^o$, and Its Corresponding Image Point $\boldsymbol{i}_{Pa}^i$.

A pose estimation problem interpreted by the posit algorithm is described as follows: An object has a set of n known object coordinates $\{\boldsymbol{w}_0^o, \boldsymbol{w}_1^o, ..., \boldsymbol{w}_n^o\}$ that correspond to camera coordinates $\{\boldsymbol{w}_0^c, \boldsymbol{w}_1^c, ..., \boldsymbol{w}_n^c\}$ and image coordinates $\{\boldsymbol{i}_0^i, \boldsymbol{i}_1^i, ..., \boldsymbol{i}_n^i\}$. The object coordinate $\boldsymbol{w}_0^o$ is designated as the "reference point" of the object coordinate system. The translation vector computed by the algorithm, $\boldsymbol{t}^c$, is the camera space coordinates of the reference point. The algorithm begins by assuming that an image coordinate $\boldsymbol{i}_a^i$ is in fact the scaled orthographic projection image point $\boldsymbol{i}_{Pa}^i$. As seen in Figure 4.4, the point $\boldsymbol{i}_{Pa}^i$ is the perspective projection of point $\boldsymbol{p}_a^o$ onto the image plane **I**. $\boldsymbol{p}_a^o$ is the orthographic projection of $\boldsymbol{w}_a^o$ onto the plane G. The plane G is a plane that lies at the same optical depth, z_0^c, of reference point $\boldsymbol{w}_0^c$ and is parallel to the image plane **I**. Any orthographically projected point $\boldsymbol{p}_a^c$ in camera coordinates is $[x_a^c, y_a^c, z_0^c]^T$.

The authors of the posit algorithm develop two functions for generating the images, $\boldsymbol{i}_{Pa}^i$, of the orthographic points, $\boldsymbol{p}_a^c$. The two equations correspond to the left and right sides of the equations in Equation 4.6.

$$\overrightarrow{\boldsymbol{w}_0^o\boldsymbol{w}_a^o} \circ \left(\frac{f^i}{z_0^c}\right)\hat{\boldsymbol{\imath}} = x_a(1+\varepsilon_a) - x_0 \qquad \overrightarrow{\boldsymbol{w}_0^o\boldsymbol{w}_a^o} \circ \left(\frac{f^i}{z_0^c}\right)\hat{\boldsymbol{\jmath}} = y_a(1+\varepsilon_a) - x_0$$

Equation 4.6

The term ε_a in these equations is

$$\varepsilon_a = \left(\frac{1}{z_0^c}\right)\overrightarrow{\boldsymbol{w}_0^o\boldsymbol{w}_a^o} \circ \hat{\boldsymbol{k}}$$

Equation 4.7

In Equation 4.7 the "∘" operator expresses the dot product between the two 3D vectors. For an in depth proof and derivation of these equations reference [41,44]. The equations

are manipulated into a form that allows all point correspondences to create a system of, usually overdetermined, linear equations.

$$\boldsymbol{I} = \left(\frac{f}{z_0^c}\right)\hat{\boldsymbol{\imath}} \qquad\qquad \boldsymbol{J} = \left(\frac{f}{z_0^c}\right)\hat{\boldsymbol{\jmath}}$$

Equation 4.8

$$\overrightarrow{\boldsymbol{w_0^o w_a^o}} \circ \boldsymbol{I} = x_a^i(1+\varepsilon_a) - x_0^i \qquad\qquad \overrightarrow{\boldsymbol{w_0^o w_a^o}} \circ \boldsymbol{J} = y_a^i(1+\varepsilon_a) - y_0^i$$

Equation 4.9

$$\overrightarrow{\boldsymbol{w_0^o w_a^o}} \circ \boldsymbol{I} = \xi_a \qquad\qquad \overrightarrow{\boldsymbol{w_0^o w_a^o}} \circ \boldsymbol{J} = \eta_a$$

Equation 4.10

$$\boldsymbol{AI} = \xi_i \qquad\qquad \boldsymbol{AJ} = \eta_i$$

Equation 4.11

The vectors **I** and **J** are solved for by finding the pseudoinverse of **A**. The author recommends that the pseudoinverse is found with singular value decomposition [50].

$$\boldsymbol{I} = \boldsymbol{A}^{+}\xi_i \qquad\qquad \boldsymbol{J} = \boldsymbol{A}^{+}\eta_i$$

Equation 4.12

Once the components of vectors **I** and **J** have been found, the rotation components $\hat{\boldsymbol{\imath}}$, and $\hat{\boldsymbol{\jmath}}$ can be extracted by dividing the solved **I** and **J** vectors by the scaling factor $\left(\frac{f}{z_0^c}\right)$. This operation is equivalent to taking the norm of **I** or **J**. The orthonormal vector $\widehat{\boldsymbol{k}}$ is found with the cross product as $\widehat{\boldsymbol{k}} = \hat{\boldsymbol{\imath}} \times \hat{\boldsymbol{\jmath}}$. The newly equated $\widehat{\boldsymbol{k}}$ and z_0^c can be used to update the value of ε_a in Equation 4.7. Running the algorithm again with a new value of ε_a yields improved estimates.

Once the algorithm has converged, the rotation matrix components will already be present in the orthonormal vectors $\hat{\boldsymbol{\imath}}$, $\hat{\boldsymbol{\jmath}}$, and $\hat{\boldsymbol{k}}$. The values of the translation vector $\boldsymbol{t}^c$, equivalent to $[x_o^c, y_o^c, z_o^c]^T$, come from Equation 4.13, Equation 4.14, and the scaling factor $\left(\frac{f}{z_0^c}\right)$.

$$x_0^c = x_0^i\left(\frac{z_0^c}{f}\right) \qquad\qquad y_0^c = y_0^i\left(\frac{z_0^c}{f}\right)$$

Equation 4.13

$$z_0^c = \frac{f}{\|\boldsymbol{I}\|} = \frac{f}{\|\boldsymbol{J}\|}$$

Equation 4.14

The explanation presented so far is for the version of POSIT that computes a pose estimate with non-coplanar object coordinates. Extra steps must be taken in order to compute pose with coplanar coordinates. It is first necessary to notice that the vectors **I** and **J** computed in the POSIT algorithm with coplanar points lead to vectors that, if placed such that their tails lie on $\boldsymbol{w_0^o}$, point to some point **Q** lying within the same plane as the object points. It is also necessary to note that the true values of **I** and **J** are any vectors whose projection onto the object plane is similar to $\overrightarrow{\boldsymbol{w_0^o Q}}$.

In order to find the correct **I** and **J**, the vectors are first rewritten as

$$\dot{\boldsymbol{I}} = \boldsymbol{I} + \lambda\boldsymbol{u} \qquad\qquad \dot{\boldsymbol{J}} = \boldsymbol{J} + \mu\boldsymbol{u}$$

Equation 4.15

where $\dot{\boldsymbol{I}}$ and $\dot{\boldsymbol{J}}$ are the true vectors. Using the new equations, the following two constraints are imposed: (1) the lengths of $\dot{\boldsymbol{I}}$ and $\dot{\boldsymbol{J}}$ must be the same, (2) $\dot{\boldsymbol{I}}$ and $\dot{\boldsymbol{J}}$ must be perpendicular. These constraints lead to the two equations

$$\lambda\mu = -\boldsymbol{I} \circ \boldsymbol{J}$$

Equation 4.16

$$\lambda^2 - \mu^2 = \|\boldsymbol{J}\|^2 - \|\boldsymbol{I}\|^2$$

Equation 4.17

Using these two equations, it is possible to find the correct values of $\dot{\boldsymbol{I}}$ and $\dot{\boldsymbol{J}}$ using Equation 4.18.

$$(\lambda + i\mu)^2 = \lambda^2 - \mu^2 + 2i\lambda\mu$$

Equation 4.18

Terms in Equation 4.18 can be substituted with the terms of Equation 4.16 and Equation 4.17 so that Equation 4.18 becomes

$$(\lambda + i\mu)^2 = \|\boldsymbol{J}\|^2 - \|\boldsymbol{I}\|^2 + 2i(-\boldsymbol{I} \circ \boldsymbol{J})$$

Equation 4.19

Since $\boldsymbol{I}$ and $\boldsymbol{J}$ are known, $\dot{\boldsymbol{I}}$ and $\dot{\boldsymbol{J}}$ are found as the real and imaginary parts of the roots of Equation 4.19. For the complete method to derive the roots as well as a proof, reference [44].

4.2.1.2 Levenberg-Marquardt Optimization

The problem of estimating camera pose can be interpreted as a nonlinear least squares optimization problem. In this type of interpretation, the extrinsic parameter configuration that yields minimum reprojection errors provides the ideal pose estimate. Before revealing how a pose estimation problem is translated into a nonlinear least squares problem, the basics of nonlinear least squares problems will be examined. Nonlinear least squares problems have the form

$$e(\boldsymbol{p};\boldsymbol{G}) = \frac{1}{2}\sum_{a=1}^{m}\left(r_a(\boldsymbol{p};\boldsymbol{G}_a)\right)^2$$

Equation 4.20

and the goal of the optimization operation is to minimize the scalar value $e(\boldsymbol{p};\boldsymbol{G})$. In Equation 4.20, each function $r_a(\boldsymbol{p};\boldsymbol{G_a})$ is known as a residual function and maps adjustable model parameters $\boldsymbol{p} \boldsymbol{\epsilon} \mathbb{R}^n$ and given values $\boldsymbol{G}_a$ to a scalar error value. In most cases, $\boldsymbol{G}_a$ contains static parameters and observation pairs for which the adjustable parameters are being tuned to fit. A residual function $r_a(\boldsymbol{p};\boldsymbol{G}_a)$ takes the form

$$r_a(\boldsymbol{c};\boldsymbol{G}_a) = \boldsymbol{G}_{a,y} - f(\boldsymbol{p};\boldsymbol{G}_{a,x})$$

Equation 4.21

where $\boldsymbol{G}_{a,x}$ and $\boldsymbol{G}_{a,y}$ are components of the observed pair and $f(\boldsymbol{p};\boldsymbol{G}_{a,x})$ is a function that uses adjustable model parameters, static parameters, and one component of the observed pair to generate a parameter dependent estimate of the other component in the observed pair. The difference between the observed value and the parameter dependent value represents how well the adjustable model parameters fit the observed pair.

Methods for minimizing Equation 4.20 most often depend on some derived components of the original nonlinear least squares problem. One of the derived component necessary is the gradient of a function. A gradient of a function is a matrix of partial derivatives of the function with respect to parameter components of that function. It applies to functions that map $\mathbb{R}^n \rightarrow \mathbb{R}$. The gradient of a residual function $r_a(\boldsymbol{p},\boldsymbol{G}_a)$ is

$$\nabla r_a(\boldsymbol{c},\boldsymbol{G}_a) = \left[\frac{\partial r_a(\boldsymbol{p};\boldsymbol{G}_a)}{\partial p_1} \quad \cdots \quad \frac{\partial r_a(\boldsymbol{p};\boldsymbol{G}_a)}{\partial p_n}\right]$$

Equation 4.22

Like the gradient, the hessian of a function applies to functions that map $\mathbb{R}^n \rightarrow \mathbb{R}$ and is a matrix of second order partial derivatives. The hessian of the error function $e(\boldsymbol{p}; \boldsymbol{G})$ would be something like

$$\nabla^2 \boldsymbol{e}(\boldsymbol{p}; \boldsymbol{G}_a) = \begin{bmatrix} \frac{\partial^2 e(\boldsymbol{p}; \boldsymbol{G}_a)}{\partial p_1 \partial p_1} & \cdots & \frac{\partial^2 e(\boldsymbol{p}; \boldsymbol{G}_a)}{\partial p_1 \partial p_n} \\ \vdots & & \vdots \\ \frac{\partial^2 e(\boldsymbol{p}; \boldsymbol{G}_a)}{\partial p_n \partial p_1} & \cdots & \frac{\partial^2 e(\boldsymbol{p}; \boldsymbol{G}_a)}{\partial p_n \partial p_n} \end{bmatrix}$$

Equation 4.23

Building on the gradient and hessian, the last derived component needed for nonlinear optimization is the Jacobian of a vector of functions. A Jacobian is like a gradient, but applies to functions that map $\mathbb{R}^n \rightarrow \mathbb{R}^m$. If the residual functions of the original nonlinear least squares problem are combined into one vector **r**

$$\boldsymbol{r}(\boldsymbol{p}; \boldsymbol{G}) = \begin{bmatrix} r_a(\boldsymbol{p}, \boldsymbol{G}_a) \\ \vdots \\ r_m(\boldsymbol{p}, \boldsymbol{G}_m) \end{bmatrix}$$

Equation 4.24

then the Jacobian of that matrix is the matrix of partials of each residual function.

$$\boldsymbol{J}(\boldsymbol{c}; \boldsymbol{G}) = \begin{bmatrix} \frac{\partial r_1(\boldsymbol{p}; \boldsymbol{G}_1)}{\partial p_1} & \cdots & \frac{\partial r_1(\boldsymbol{p}; \boldsymbol{G}_1)}{\partial p_n} \\ \vdots & & \vdots \\ \frac{\partial r_m(\boldsymbol{p}; \boldsymbol{G}_m)}{\partial p_1} & \cdots & \frac{\partial r_m(\boldsymbol{p}; \boldsymbol{G}_m)}{\partial p_n} \end{bmatrix}$$

Equation 4.25

With a basis of nonlinear least squares problems and knowledge of some of the derived components, the Levenberg-Marquardt algorithm can be discussed. The Levenberg-Marquardt optimization algorithm is a means to find the parameter setting **p** that minimizes the scalar error $e(\boldsymbol{p}; \boldsymbol{G})$. It is composed of a mixture of gradient descent

and Gauss-Newton optimization. The Levenberg-Marquardt algorithm uses the update equation

$$\boldsymbol{p}_{b+1} = \boldsymbol{p}_b - \left(\boldsymbol{\nabla^2}(\boldsymbol{p}_b; \boldsymbol{G}) + \lambda diag\left(\boldsymbol{\nabla^2}(\boldsymbol{p}_b; \boldsymbol{G})\right)\right)^{-1} \boldsymbol{\nabla e}(\boldsymbol{p}_b; \boldsymbol{G})$$

Equation 4.26

where "b" indexes the iteration number and the function $diag(\boldsymbol{A})$ represents a zeroed matrix whose dimensions are the same as **A**, and whose diagonal elements are the same as the diagonal elements of **A**. The parameter λ of the update equation controls the behavior of the update method. If λ is large, then the update equation acts like gradient descent, if λ is small then the update equation acts like Gauss-Newton Iteration [51,52]. The value of λ is dynamically controlled by the algorithm with the following pseudocode:

1. $\boldsymbol{p}_{b+1} = \boldsymbol{p}_b - \left(\boldsymbol{\nabla^2}(\boldsymbol{p}_b; \boldsymbol{G}) + \lambda diag\left(\boldsymbol{\nabla^2}(\boldsymbol{p}_b; \boldsymbol{G})\right)\right)^{-1} \boldsymbol{\nabla e}(\boldsymbol{p}_b; \boldsymbol{G})$;
2. $e_{b+1} = e(\boldsymbol{p}_{b+1}; \boldsymbol{G})$;
3. $if(e_{b+1} > e_b)$
4. $\quad \lambda = 10\lambda;\ goto\ 1$;
5. $else$
6. $\quad \lambda = \frac{\lambda}{10};\ e_b = e_{b+1};\ \boldsymbol{p}_b = \boldsymbol{p}_{b+1};\ goto\ 1.$

One detail when thinking about computational cost of the algorithm is the cost evaluating the hessian of a function. Like the Gauss-Newton method, the hessian is estimated with the Jacobian as $\boldsymbol{J}(\boldsymbol{p}_b; \boldsymbol{G})^{\boldsymbol{T}}\boldsymbol{J}(\boldsymbol{p}_b; \boldsymbol{G})$ producing the optimized update equation:

$$\boldsymbol{p}_{b+1} = \boldsymbol{p}_b - \left(\boldsymbol{J}(\boldsymbol{p}_b; \boldsymbol{G})^{\boldsymbol{T}}\boldsymbol{J}(\boldsymbol{p}_b; \boldsymbol{G}) + \lambda diag\left(\boldsymbol{J}(\boldsymbol{p}_b; \boldsymbol{G})^{\boldsymbol{T}}\boldsymbol{J}(\boldsymbol{p}_b; \boldsymbol{G})\right)\right)^{-1} \boldsymbol{J}(\boldsymbol{p}_b; \boldsymbol{G})^{\boldsymbol{T}}\boldsymbol{r}(\boldsymbol{p}_b; \boldsymbol{G})$$

Equation 4.27

A pose estimation problem can be translated into a nonlinear least squares problem if the value $e(\boldsymbol{p}; \boldsymbol{G})$ is set to represent reprojection errors. In such an interpretation, the residual functions would take the form

$$r_{a,x}(\boldsymbol{\theta}, \boldsymbol{t};\ f^i, \boldsymbol{w_a^0}) = x_{a,obs}^i - rigidProj_x(\boldsymbol{\theta}, \boldsymbol{t};\ f^i, \boldsymbol{w}_a^0)$$

Equation 4.28

$$r_{a,y}(\boldsymbol{\theta}, \boldsymbol{t};\ f^i, \boldsymbol{w_a^0}) = y_{a,obs}^i - rigidProj_y(\boldsymbol{\theta}, \boldsymbol{t};\ f^i, \boldsymbol{w}_a^0)$$

Equation 4.29

where $x_{a,obs}^i$ and $y_{a,obs}^i$ are the components of the observed image coordinates. The function $rigidProj\left(\boldsymbol{\theta}, \boldsymbol{t};\ f^i, \boldsymbol{w}_a^0\right)$ would look like

$$\begin{bmatrix} x_{a,est}^i \\ y_{a,est}^i \end{bmatrix} = \begin{bmatrix} \dfrac{f^i * (x^c + \boldsymbol{R}_{11} * x_a^0 + \boldsymbol{R}_{12} * y_a^0 + \boldsymbol{R}_{13} * z_a^0)}{z^c + \boldsymbol{R}_{31} * x_a^0 + \boldsymbol{R}_{32} * y_a^0 + \boldsymbol{R}_{33} * z_a^0} \\ \dfrac{f^i * (y^c + \boldsymbol{R}_{21} * x_a^0 + \boldsymbol{R}_{22} * y_a^0 + \boldsymbol{R}_{23} * z_a^0)}{z^c + \boldsymbol{R}_{31} * x_a^0 + \boldsymbol{R}_{32} * y_a^0 + \boldsymbol{R}_{33} * z_a^0} \end{bmatrix}$$

Equation 4.30

which provides image coordinates corresponding to a rigid transformation and perspective projection of known coordinate $\boldsymbol{w}_a^0$. This is only an example of how the LM algorithm works. Another example can be found in [53] and in the end, the OpenCV 2.4.8 implementation of the algorithm for pose estimation is used.

4.2.2 Linear Complexity

4.2.2.1 Efficient Perspective-n-Point Camera Pose Estimation

One of the goals for the overall system is real-time performance. In pursuit of this goal it is necessary to examine pose estimation algorithms that have bounds on their complexity. One such method is the EPnP algorithm [42], which has an O(n) bound. In

the subsequent sections, it is discussed in detail as a potential solution to the pose estimation problem.

The first step in the EPnP algorithm involves generating coordinates, in object space, for four virtual control points. The locations of the control points are selected such that one point is positioned at the centroid of the incoming object coordinate set, and the other three points, in combination with the centroid point, form a basis for 3D object space that aligns with the object system axis. Once the positions of the four control points have been generated, the weighting factors that express each object coordinate in terms of the control points are selected such that Equation 4.31 and Equation 4.32 are satisfied.

$$\boldsymbol{w}_a^o = \sum_{b=1}^{4} \alpha_{ba} \boldsymbol{c}_b^o$$

Equation 4.31

$$\sum_{b=1}^{4} \alpha_{ba} = 1$$

Equation 4.32

Equation 4.31 and Equation 4.32 state that all object coordinates can be expressed as a mixture of the four control points. One important detail to notice about the mixing weights, α_{ba}, in Equation 4.31 and Equation 4.32 is that the same mixing weights that express object coordinates can also express camera coordinates, but only if the camera space locations of the virtual control points are known. This is expressed in Equation 4.33.

$$\boldsymbol{w}_a^c = \sum_{b=1}^{4} \alpha_{ba} \boldsymbol{c}_b^c$$

Equation 4.33

Again, determining the position in camera space of object coordinates is only possible if the camera coordinates of the four control points, $\boldsymbol{c}_b^c$, are known. From this point, it can be seen that the pose estimation problem now lies in determining the location of the control points in camera space. Once this information is known, 3D fitting methods such as [54,55,56] can recover the rotation and translation values necessary for a complete camera pose estimate.

The authors find that known image coordinates, object coordinate weights, and the unknown location of the control points are related by Equation 4.34 and Equation 4.35.

$$\sum_{b=1}^{4} \alpha_{ba} f_x x_b^c + \alpha_{ba}(x^{pi} - x_a^i) z_b^c = 0$$

Equation 4.34

$$\sum_{b=1}^{4} \alpha_{ba} f_y y_b^c + \alpha_{ba}(y^{pi} - y_a^i) z_b^c = 0$$

Equation 4.35

In these two equations each α_{ba} is the known mixing weight determined in Equation 4.31, f_u and f_v are the known focal lengths, x^{pi} and y^{pi} are the known image coordinates of the principal point, and x_a^i and y_a^i are the known image coordinates corresponding to object coordinate $\boldsymbol{w}_a^o$. With some manipulation, the two equations can be reformed into a linear system which looks like Equation 4.36.

$$\boldsymbol{Mx} = \mathbf{0}$$

Equation 4.36

In more detail, Equation 4.36's terms can be expanded to those in Equation 4.37.

$$\begin{bmatrix} \alpha_{11}f_x & 0 & \alpha_{11}(x^{pi} - x_1^i) & \cdots & \alpha_{41}f_x & 0 & \alpha_{41}(x^{pi} - x_1^i) \\ 0 & \alpha_{11}f_y & \alpha_{11}(y^{pi} - y_1^i) & \cdots & 0 & \alpha_{41}f_y & \alpha_{41}(y^{pi} - y_1^i) \\ & & & \vdots & & & \\ \alpha_{1n}f_x & 0 & \alpha_{11}(x^{pi} - x_n^i) & \cdots & \alpha_{4n}f_x & 0 & \alpha_{4n}(x^{pi} - x_n^i) \\ 0 & \alpha_{1n}f_y & \alpha_{11}(y^{pi} - y_n^i) & \cdots & 0 & \alpha_{4n}f_y & \alpha_{4n}(y^{pi} - y_n^i) \end{bmatrix} \begin{bmatrix} x_1^c \\ y_1^c \\ z_1^c \\ \vdots \\ x_4^c \\ y_4^c \\ z_4^c \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Equation 4.37

From Equation 4.34 to Equation 4.37 it can be seen that the vector **x** is a 12x1 vector equivalent to $[\boldsymbol{c}_1^c;\ \boldsymbol{c}_2^c;\ \boldsymbol{c}_3^c;\ \boldsymbol{c}_4^c]$, where the semicolon denotes row concatenation. The true value of the camera space positions of the control points is therefore somewhere in the null space of **M**. The null space of **M** can be efficiently found as the span of the eigenvectors corresponding to zeroed eigenvalues of the matrix $\boldsymbol{M}^T\boldsymbol{M}$.

In practice there may be anywhere from one to four null eigenvectors of $\boldsymbol{M}^T\boldsymbol{M}$. The correct value of **x** is therefore some linear combination of the null eigenvectors as in Equation 4.38.

$$\boldsymbol{x} = \sum_{d=1}^{N_e} \beta_d \boldsymbol{v}_d$$

Equation 4.38

In Equation 4.38, N_e is the number of null eigenvectors of $\boldsymbol{M}^T\boldsymbol{M}$, $\boldsymbol{v}_d$ is the d'th null eigenvector, and the values of β_d are the constants that generate the correct linear combination needed to find **x**.

As said before, there are anywhere from one to four vectors that represent the null space of **M**. The authors of EPnP examine each of the four cases. They derive methods of finding the correct values of each β_d in each of the four cases. The simple case is that in which the number of null eigenvectors is just one. In this case the vector **x** is found as

$$\boldsymbol{x} = \beta_1 \boldsymbol{v}_1$$

Equation 4.39

The correct value of β_1 is found by ensuring that the distances between virtual control points in object space is equivalent to the distances between control points in camera space, or

$$\left\|\beta_1 \boldsymbol{v}_{a,1}^c - \beta_1 \boldsymbol{v}_{b,1}^c\right\|^2 = \|\boldsymbol{c}_a^o - \boldsymbol{c}_b^o\|^2$$

Equation 4.40

where $\boldsymbol{v}_{a,1}^c$ denotes the 3x1 subvector corresponding to virtual control point "a" from the single vector $\boldsymbol{v}_1^c$. In this simple case, the value of β_1 can be found directly. In the case that the null space is the span of 2 vectors, a similar distance constraint, as seen in Equation 4.41, is used.

$$\left\|\left(\beta_1 \boldsymbol{v}_{a,1}^c + \beta_2 \boldsymbol{v}_{a,2}^c\right) - \left(\beta_1 \boldsymbol{v}_{b,1}^c + \beta_2 \boldsymbol{v}_{b,2}^c\right)\right\|^2 = \|\boldsymbol{c}_a^o - \boldsymbol{c}_b^o\|^2$$

Equation 4.41

$$\beta_1^2\left(\boldsymbol{v}_{a,1}^c - \boldsymbol{v}_{b,1}^c\right)^T\left(\boldsymbol{v}_{a,1}^c - \boldsymbol{v}_{b,1}^c\right) + 2\beta_1\beta_2\left(\boldsymbol{v}_{a,1}^c - \boldsymbol{v}_{b,1}^c\right)^T\left(\boldsymbol{v}_{a,2}^c - \boldsymbol{v}_{b,2}^c\right) + \beta_2^2\left(\boldsymbol{v}_{a,2}^c - \boldsymbol{v}_{b,2}^c\right)^T\left(\boldsymbol{v}_{a,2}^c - \boldsymbol{v}_{b,2}^c\right)$$
$$= (\boldsymbol{c}_a^o - \boldsymbol{c}_b^o)^T(\boldsymbol{c}_a^o - \boldsymbol{c}_b^o)$$

Equation 4.42

In this case each equation of the form of Equation 4.41 can be rewritten into the form of Equation 4.42 and it can be seen that finding the values of β_1 and β_2 becomes a problem of solving a system of 6 quadratic equations.

To extract the correct values of β_1 and β_2 a technique called linearization [43] is used. The linearization technique takes the terms β_1^2, $\beta_1\beta_2$, and β_2^2 and replaces them with linear terms β_{11}, β_{12} , and β_{22} so that the system can be easily solved as a linear system. The linear system takes the form.

$$L\,\beta = \rho$$

Equation 4.43

which expanded looks like

$$\boldsymbol{L}\begin{bmatrix}\beta_{11}\\ \beta_{12}\\ \beta_{22}\end{bmatrix} = \begin{bmatrix}\|\boldsymbol{c}_1^o - \boldsymbol{c}_2^o\|^2\\ \vdots \\ \|\boldsymbol{c}_3^o - \boldsymbol{c}_4^o\|^2\end{bmatrix}$$

Equation 4.44

In Equation 4.44, $\boldsymbol{L}\epsilon\mathbb{R}^{6x3}$ and is composed of the components of each of the two null eigenvectors as seen in Equation 4.42. The system is solved with the pseudoinverse of $\boldsymbol{L}$ and the correct sign of β_1 and β_2 are selected such that the control points in camera space are positioned in front of the camera. When the number of vectors that span the null space of **M** is three, the solution to β_1, β_2, and β_3 are found again using linearization. In this case, the inverse of $\boldsymbol{L}$, which is now a 6x6, is used to solve the linear system.

When there are four null eigenvectors of $\boldsymbol{M}^T\boldsymbol{M}$, the distance constraints looks like Equation 4.45.

$$\left\|\left(\beta_1 \boldsymbol{v}_{a,1}^c + \beta_2 \boldsymbol{v}_{a,2}^c + \beta_3 \boldsymbol{v}_{a,3}^c + \beta_4 \boldsymbol{v}_{a,4}^c\right) - \left(\beta_1 \boldsymbol{v}_b^c + \beta_2 \boldsymbol{v}_b^c + \beta_3 \boldsymbol{v}_b^c + \beta_4 \boldsymbol{v}_b^c\right)\right\|^2 = \|\boldsymbol{c}_a^o - \boldsymbol{c}_b^o\|^2$$

Equation 4.45

In this case, the distance constraints produce a system of six equations with ten 2nd degree polynomial terms. To find the correct values of β_1, β_2, β_3 and β_4, a technique called relinearization [57] is applied. The relinearization technique is similar to the linearization technique. First, all 2nd degree terms are replaced with linear terms. This leaves a underdetermined system of six linear equations with ten terms. The method resolves the underdetermined linear system by converting it into a parametric system with four new variables. The result is that each of the 2nd degree terms has the form of Equation 4.46

$$\beta_{12} = c_1 z_1 + c_2 z_2 + c_3 z_3 + c_4 z_4$$

Equation 4.46

where c_i is some constant and z_i is a term introduced in the parameterization process. The relinearization technique then uses the constraints such as:

$$\beta_{12}\beta_{34} = \beta_{13}\beta_{24} = \beta_{14}\beta_{23}$$

Equation 4.47

to generate enough quadratic equations such that the linearization technique can be applied.

4.2.3 Vanishing Geometry

4.2.3.1 Four Parallel Lines

In [46] the author attempts to find the pose of a UAV-mounted camera relative to four equally space parallel lines on the ground plane. The world coordinate system is

configured such that its x-axis is parallel with the ground lines and the author attempts to estimate all but one of the parameters of pose. The neglected parameter, translation along the x-axis, is deemed uncomputable due to the lack of reference points along its direction. Nevertheless, the author derives a method that retrieves all rotation parameters and the two computable translation parameters.

The method presented in [46] makes the assumption that the x-axis of the world coordinate system is parallel with the ground lines, the positive values of the z-axis indicate height above the ground plane, and by right-hand convention, the y-axis is left when looking in the direction of the positive x-axis. The method presented in the paper is for this specific configuration, but is easily adaptable to any coordinate system axis configuration.

The vanishing point, $\boldsymbol{v}^{\infty i}$, of the world system x-axis is calculated as the intersection of the images of the parallel ground lines. From the image coordinate, the vector indicating the orientation of the world system x-axis in the camera space is found with

$$\boldsymbol{r}_{x^c} = \frac{\boldsymbol{K}^{-1}\boldsymbol{v}^{\infty i}}{\|\boldsymbol{K}^{-1}\boldsymbol{v}^{\infty i}\|}$$

Equation 4.48

where **K** is the 3x3 intrinsic camera parameter matrix.

The vanishing line of the ground plane, an x-y plane, is calculated with the DLT [58] and the equation for a projective line map presented in [59]. Two important steps before calculating the vanishing line are: (1) normalizing the image line vectors such that

they change from $[a_a, b_a, c_a]^T$ to $[x_a, y_a, 1]^T$, and (2) creating the points $\boldsymbol{x}_a^T$ as $[a, l]$. The components of the line map matrix **A** are found with the linear system:

$$\begin{bmatrix} \mathbf{0} & -\boldsymbol{x}_a^T & y_a\boldsymbol{x}_a^T \\ \boldsymbol{x}_a^T & \mathbf{0} & -x_a\boldsymbol{x}_a^T \end{bmatrix} \begin{bmatrix} \boldsymbol{a}_1^T \\ \boldsymbol{a}_2^T \\ \boldsymbol{a}_3^T \end{bmatrix} = \mathbf{0}$$

Equation 4.49

where $\boldsymbol{a}_1^T$ is the 2x1 column vector corresponding to the first row of **A**. As described in [59], the first column **A** corresponds the vanishing line of the x-y ground plane, $\boldsymbol{l}^{\infty xyi}$, whose components, $[a, b, c]^T$, correspond to the coefficients of a line in general form, $ax + by + c = 0$. Once the vanishing line is found, the orientation of the world coordinate system axis normal to the ground plane, the z-axis in this case, can be found as

$$\boldsymbol{r}_{z^c} = \pm \frac{\boldsymbol{K}^T \boldsymbol{l}^{\infty xyi}}{\|\boldsymbol{K}^T \boldsymbol{l}^{\infty xyi}\|}$$

Equation 4.50

The third orientation vector, $\boldsymbol{r}_{y^c}$ is found with the cross product using right hand convention, $\boldsymbol{r}_{y^c} = \boldsymbol{r}_{z^c} \times \boldsymbol{r}_{x^c}$. With all rotation vectors computed, the rotation matrix **R**, which expresses the rotation of the camera coordinate system axis in its rows, is $\boldsymbol{R} = [\boldsymbol{r}_{x^c}, \boldsymbol{r}_{y^c}, \boldsymbol{r}_{z^c}]$.

Before continuing to translation recovery, it is important to point out two unexplained yet trivial issue in the algorithm. First, if any image line passes through the origin of the image coordinate system, then its third component becomes zero, or $\boldsymbol{l}^i = [a, b, 0]^T$. If the third component of the line is zero, then the vector can never be scaled to form $[a, b, 1]^T$ and the method produces unstable results. A simple workaround

for this type of problem is to use the three line method presented in [59] for computing the line at infinity. The three line method computes the line at infinity as

$$\boldsymbol{l}_{\infty}^{i} = det([\boldsymbol{l}_1^i\ \boldsymbol{l}_2^i\ \boldsymbol{l}_d^i])\boldsymbol{l}_3^i - det([\boldsymbol{l}_d^i\ \boldsymbol{l}_2^i\ \boldsymbol{l}_3^i])\boldsymbol{l}_1^i$$

Equation 4.51

where $\boldsymbol{l}_d^i$ is any line that makes the determinants non-zero. The second issue is the sign of the ground plane normal axis, $\boldsymbol{r}_{z^c}$. It is ambiguous but can be trivially found if an assumption is made about the orientation of the camera. A normal assumption is that the camera is upright and the sign of $\boldsymbol{r}_{z^c}$ is selected such that $\boldsymbol{r}_{z^c}$'s 3rd component is positive.

Now that the rotation matrix has been compute, the translation vector can be found. Again the DLT is used, but only one equation is generated from each line correspondence.

$$\begin{bmatrix}(\boldsymbol{K}^T\boldsymbol{l}_a^i)^T\boldsymbol{r}_{y^c} & (\boldsymbol{K}^T\boldsymbol{l}_a^i)^T\boldsymbol{r}_{z^c}\end{bmatrix}\begin{bmatrix}\boldsymbol{t}_y^o\\ \boldsymbol{t}_z^o\end{bmatrix} = -w_a^o(\boldsymbol{K}^T\boldsymbol{l}_a^i)^T\boldsymbol{r}_{y^c}$$

Equation 4.52

In the equation, the scalar w_a is the third component of the object space line corresponding to the image line $\boldsymbol{l}_a^i$. In this case, w_a^o is the positions on the y-axis of the ground line. It should be noted that the translation parameters found are the type that work with rigid transformation expressed by Equation 4.3.

4.2.3.2 Three or More Parallel Lines

In [47], the problem of pose estimation from parallel lines is resolved through the use of geometric clues about the problem. The authors find that a ground line, its projection on the image plane, and the center of projection are all coplanar. Because a line and a point in 3D space contain enough information to define a plane, the planes

corresponding to each line, interpretation planes, can be estimated in camera space from images of the lines and their relationship to the center of projection. Figure 4.5 shows this type of construction.

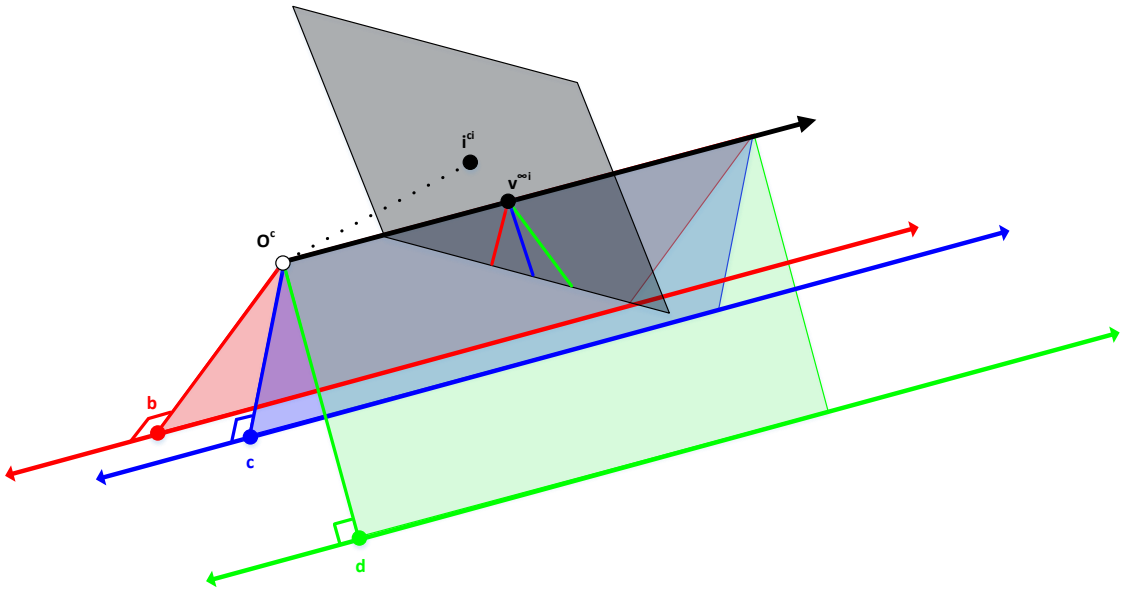


Figure 4.5: Constructing Interpretation Planes from Image Lines.

In Figure 4.5 $\boldsymbol{v}^{\infty i}$ is the vanishing point on the image plane, $\boldsymbol{o}^{c}$ is the center of projection, and $\boldsymbol{i}^{ci}$ is the origin of the image coordinate system. Once equations of the interpretation planes have been estimated, the dihedral angles between the planes can be estimated. The dihedral angles correspond to the angles between lines from the center of projection to ground lines as seen in Figure 4.5 and Figure 4.6 as $\overline{\boldsymbol{o}^{c}\boldsymbol{b}}$, $\overline{\boldsymbol{o}^{c}\boldsymbol{c}}$, and $\overline{\boldsymbol{o}^{c}\boldsymbol{d}}$.

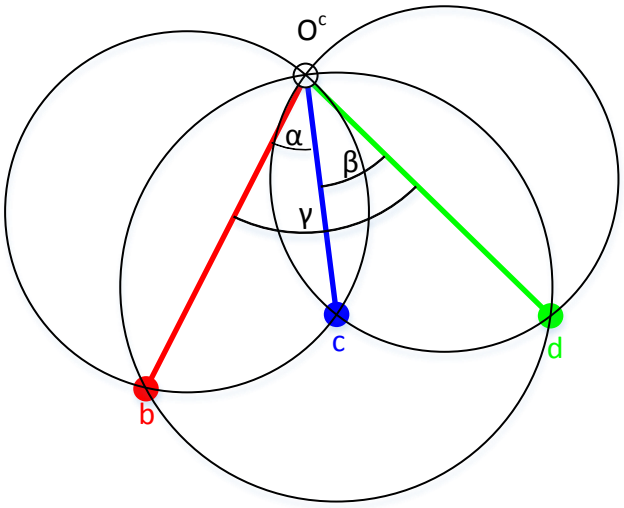


Figure 4.6: The Geometry Used to Calculate Camera Translation.

The object space distances between lines **b**, **c**, and **d** are known, and by both law of cosines and inscribed angles, the position of the center of projection, in object space, can be estimated.

Rotation parameters are found using the normal vectors of the interpretation planes. The normal vectors in camera space, $\hat{\boldsymbol{n}}_a^c$, are computed from the plane equations found when calculating translation parameters. The normal vectors in object space, $\hat{\boldsymbol{n}}_a^o$, are found using the now known translation $\boldsymbol{o}^c$. From $\boldsymbol{o}^c$the vectors $\overrightarrow{\boldsymbol{o}^c\boldsymbol{b}}$, $\overrightarrow{\boldsymbol{o}^c\boldsymbol{c}}$, $\overrightarrow{\boldsymbol{o}^c\boldsymbol{d}}$ as well as their unit vector forms $\hat{\boldsymbol{k}}_b^o$, $\hat{\boldsymbol{k}}_c^o$, $\hat{\boldsymbol{k}}_d^o$ are easily computable. The object space normal vectors to the interpretation planes can be found as

$$\hat{\boldsymbol{n}}_a^o = \hat{\boldsymbol{m}}_a^o \times \hat{\boldsymbol{k}}_a^o$$

Equation 4.53

where $\widehat{\boldsymbol{m}}_a^o$ is a unit vector in the same direction as the parallel lines. Once the normal vectors in both object and camera space are known, the rotation can be found using 3D fitting methods such as [54,55].

4.3 Proposed Methods

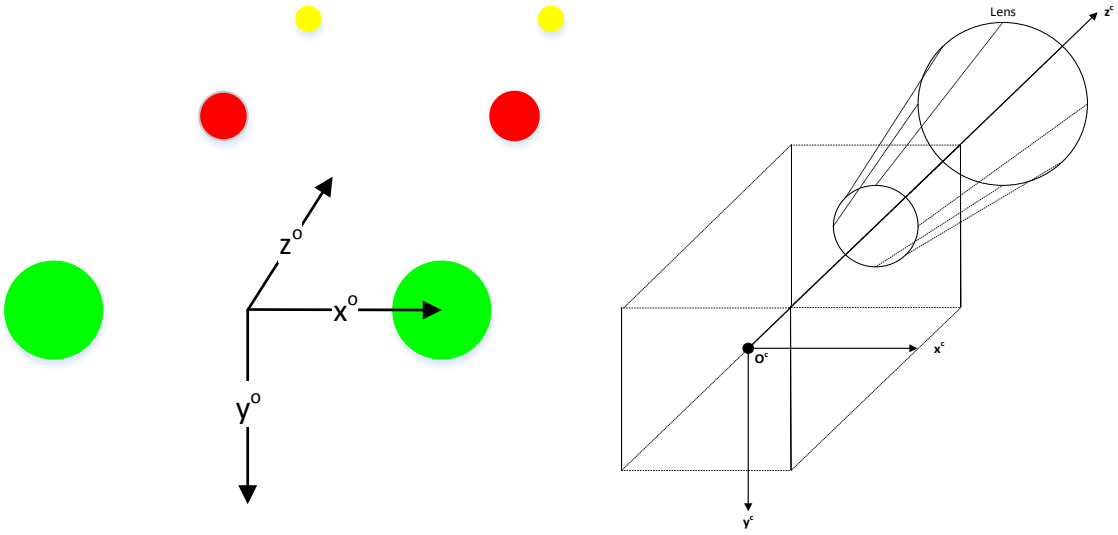
In the following two sections, two methods of finding camera pose for the degenerate PnP problem presented at the start of this chapter will be presented. One method makes use of linear systems and is called "complete" due to its mathematical correctness. The other method, named "simple," is not mathematically valid yet produces accurate pose estimates using only basic geometry. Before introducing the methods, the formulation of the problem will be presented.

4.3.1.1 Problem Formulation

The camera coordinate system z-axis is parallel with the optical axis and in the same direction as the vector which has its tail at the center of projection and head at the principal point of the image plane. The camera system's x-axis aligns with the rows of the image sensor and is considered positive in the right direction if looking in the positive z-axis direction. The y-axis aligns with the columns of the image sensor and is positive in the downward direction. The origin of the system is taken to be the center of projection. This configuration is shown in Figure 4.7 (b) and is selected so that the image coordinate system axis align with the camera coordinate system x-axis and y-axis in orientation.

The z-axis of the world coordinate system lies in the same direction as the two given parallel lines. The y-axis corresponds to the height above the ground plane and is positive as one moves underground or underwater. The x-axis, by right hand convention, is oriented right when looking in the direction of the z-axis. The origin of the world

coordinate system is selected to be the midpoint between the first object coordinate pair. This configuration is shown in Figure 4.7 (a) and is selected so that it matches the camera coordinate system when the system is operating under normal conditions.



(a) (b)

Figure 4.7: Coordinate System Orientation. (a) The Object Coordinate System Placement. (b) The Camera Coordinate System Placement.

4.3.1.2 Complete Method

The method in [46] is extended to fit the aforementioned coordinate system configurations and to provide a complete pose estimate. One significant difference between the proposed method and the method in [46] is that the latter depends on four parallel ground lines whereas the former requires two parallel ground lines and known coordinates along those lines.

Suppose that there are two parallel lines, $\boldsymbol{l}_1^o$, $\boldsymbol{l}_2^o$, in world space that align with the z-axis of the world coordinate system and lie on the x-z world plane. Each of the lines are represented in vector form as $[1, 0, -x_d^o]^T$ where "d" indexes the line number and each

component corresponds to the parts of a line in general form, $ax + bz + c = 0$. Along these lines are reference points, $\boldsymbol{w}_{d,a}^{o}$, where "a" indexes coordinate number. The reference points have the form $[x_d^o, 0, z_a^o]^T$. For each reference coordinate on $\boldsymbol{l}_1^o$, there is a corresponding coordinate on $\boldsymbol{l}_2^o$ with the same z component. The parallel lines project onto the image plane as $\boldsymbol{l}_d^i$ and the references project to the image plane as $\boldsymbol{w}_{d,a}^{i}$.

The orientation of the object system's z-axis is found in a similar manner to [46] with

$$\boldsymbol{r}_{z^c} = \frac{\boldsymbol{K}^{-1}\boldsymbol{z}^{\infty i}}{\|\boldsymbol{K}^{-1}\boldsymbol{z}^{\infty i}\|}$$

Equation 4.54

In [46], the presence of three or more parallel lines allowed for vanishing line computation from just the lines. In this version of the problem, the references that lie on the ground lines provide the information needed to compute the x-z ground plane vanishing line, $\boldsymbol{l}^{\infty xzi}$. As stated previously, a pair of coordinates is the duplet of coordinates that share a common z component in their corresponding object coordinates. A pair line is the line between these coordinates. In the case that there are two pair lines available in a frame, the x-axis vanishing coordinate is computed as the intersection of the pair lines. The vanishing line is the line between the z-axis and x-axis vanishing coordinates. In the case that there are three or more pair non-coincident pair lines, the x-axis vanishing coordinate is found with the linear system

$$[a^i \quad b^i \quad c^i]\begin{bmatrix} sx^{\infty i} \\ sy^{\infty i} \\ s \end{bmatrix} = \mathbf{0}$$

Equation 4.55

where $[a_a^i, b_a^i, c_a^i]$ are the coefficients of the a'th pair line's image line. If the pair lines are parallel, or if only one exists, the vanishing line is found as a line that passes through the z-axis vanishing coordinate, $\boldsymbol{z}^{\infty i}$,with a slope similar to the pair line(s). The orientation of the object system y-axis in camera space is found using a method similar to that presented in [46].

$$\boldsymbol{r}_{y^c} = \pm \frac{\boldsymbol{K}^T \boldsymbol{l}^{\infty xzi}}{\|\boldsymbol{K}^T \boldsymbol{l}^{\infty xzi}\|}$$

Equation 4.56

It is assumed that the camera is upright and the sign ambiguity of the y-axis orientation can be resolved by ensuring that the orientation of $\boldsymbol{r}_{y^c}$ is similar to the camera's y-axis orientation. The x-axis is computed with the cross product, $\boldsymbol{r}_{x^c} = \boldsymbol{r}_{y^c} \times \boldsymbol{r}_{z^c}$.

The translation parameters orthogonal to the parallel lines are found using the method presented [46] with different axis components.

$$\begin{bmatrix} (\boldsymbol{K}^T \boldsymbol{l}_a^i)^T \boldsymbol{r}_{x^c} & (\boldsymbol{K}^T \boldsymbol{l}_a^i)^T \boldsymbol{r}_{y^c} \end{bmatrix} \begin{bmatrix} \boldsymbol{t}_x^o \\ \boldsymbol{t}_y^o \end{bmatrix} = -w_a (\boldsymbol{K}^T \boldsymbol{l}_a^i)^T \boldsymbol{r}_{x^c}$$

Equation 4.57

In Equation 4.57, w_a is the common x coordinate of all points on line $\boldsymbol{l}_a^o$.

In [46] the translation down the parallel lines is deemed uncomputable. In the problem presented, the known object coordinates along the parallel lines allow for the computation of the z component of translation. In the assumptions, it was stated that the object-image correspondences provided to the module contained at least one pair. The z-axis translation is found by modifying Equation 4.57 to use lines oriented in the x-axis

direction and to include the previously found height component $\boldsymbol{t_y^o}$. The modified form of Equation 4.57 that accounts for x-axis direction ground lines is

$$\left[(\boldsymbol{K}^T\boldsymbol{l}_a^i)^T\boldsymbol{r}_{y^c} \quad (\boldsymbol{K}^T\boldsymbol{l}_a^i)^T\boldsymbol{r}_{z^c}\right]\begin{bmatrix}\boldsymbol{t_y^o}\\ \boldsymbol{t_z^o}\end{bmatrix} = -w_a(\boldsymbol{K}^T\boldsymbol{l}_a^i)^T\boldsymbol{r}_{y^c}$$

Equation 4.58

Since $\boldsymbol{t_y^o}$ is known, the linear system can be rewritten as

$$[(\boldsymbol{K}^T\boldsymbol{l}_a^i)^T\boldsymbol{r}_{z^c}][\boldsymbol{t_z^o}] = -w_a(\boldsymbol{K}^T\boldsymbol{l}_a^i)^T\boldsymbol{r}_{y^c} - \boldsymbol{t_y^o}(\boldsymbol{K}^T\boldsymbol{l}_a^i)^T\boldsymbol{r}_{y^c}$$

Equation 4.59

4.3.1.3 Simple Method

The Complete Method requires knowledge of linear algebra, the DLT, and projective space. The Simple Method estimates pose using basic trigonometry and statistics. The problem formulation is identical to that of the Complete Method, however, the approach is different.

First, the optical axis rotation of the camera is retrieved using the average slope of all pair lines.

$$\theta_z = tan^{-1}\left(\left(\frac{1}{n}\right)\sum_{a=1}^{n} m_a^i\right)$$

Equation 4.60

In Equation 4.60, "n" is the number of available image coordinate pairs in the frame and m_a^i is the slope of the line that traverses both elements of the pair.

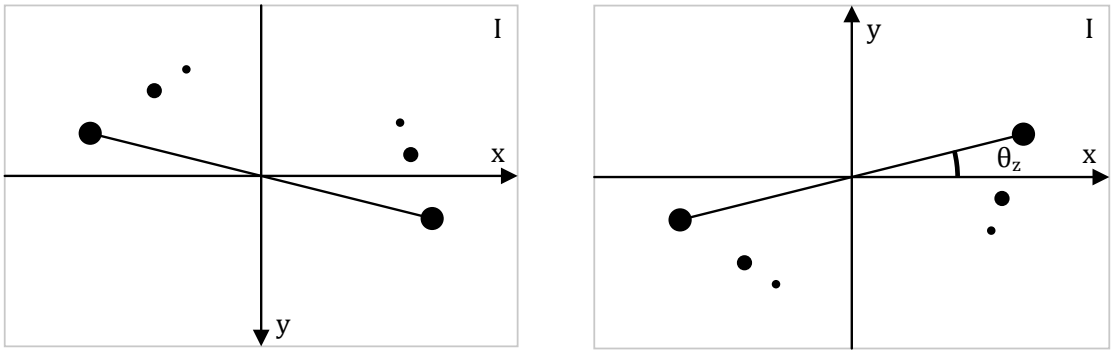


Figure 4.8: A Pair Line in the Image Plane. (a) As viewed in a Frame. (b) Angle $\boldsymbol{\theta}_z$.

Knowing optical axis rotation allows for incoming image coordinates to be corrected of their z-axis rotation. A corrected image coordinate, $\left[\dot{x}_a^i, \dot{y}_a^i\right]^T$, is found by counter-rotating the original coordinates about the image system origin with a 2D rotation transformation

$$\begin{bmatrix} \dot{x}_a^i \\ \dot{y}_a^i \end{bmatrix} = \begin{bmatrix} cos\,(-\boldsymbol{\theta}_z) & -sin\,(-\boldsymbol{\theta}_z) \\ sin\,(-\boldsymbol{\theta}_z) & cos\,(-\boldsymbol{\theta}_z) \end{bmatrix} \begin{bmatrix} x_a^i \\ y_a^i \end{bmatrix}$$

Equation 4.61

The problem now is more degenerate than what was previously presented because the camera z-axis rotation is known to be zero. Using the corrected image coordinates, the z-axis vanishing coordinate may be recalculated as $\left[\dot{x}^{\infty i}, \dot{y}^{\infty i}\right]^T$ and used to find yaw and pitch angles. Equation 4.62 and Equation 4.63 derive the angles from the roll corrected vanishing coordinate. Figure 4.9 illustrates the geometry of the equations.

$$\boldsymbol{\theta}_x = tan^{-1}\left(\frac{\dot{x}^{\infty i}}{f^i}\right)$$

Equation 4.62

$$\boldsymbol{\theta}_y = tan^{-1}\left(\frac{-\dot{y}^{\infty i}}{f^i}\right)$$

Equation 4.63

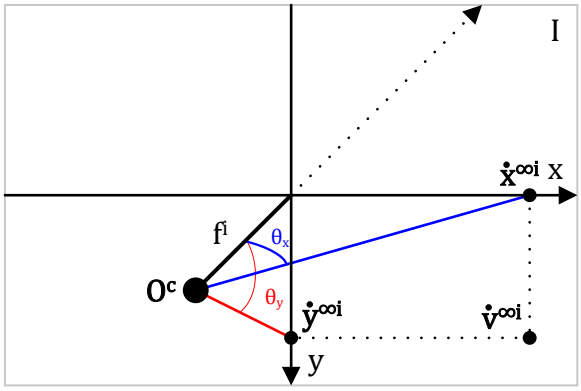


Figure 4.9: A Diagram of the Trigonometry Used to Calculate Yaw and Pitch.

The z-axis rotation corrected points are fitted with lines, and the resulting lines have slopes $\dot{m}_1^i$ and $\dot{m}_2^i$. The x-axis and z-axis object space translations are computed from the slopes as

$$t_x^o = \frac{-\dot{m}_1^i - \dot{m}_2^i}{\dot{m}_1^i - \dot{m}_2^i} * x_d$$

Equation 4.64

$$\boldsymbol{t}_{y1}^o = (x_d + t_x^o) * \dot{m}_1^i \qquad\qquad \boldsymbol{t}_{y2}^o = (-x_d + t_x^o) * \dot{m}_2^i$$

Equation 4.65

$$t_y = \frac{\left(\boldsymbol{t}_{y1}^o + \boldsymbol{t}_{y2}^o\right)}{2}$$

Equation 4.66

The pitch of the camera is filtered from the coordinates in the same manner as the roll was filtered. Filtering pitch amounts to aligning the vanishing coordinate with the row of

the principal point, and ensuring that all angles between image coordinates are conserved. Equation 4.67 is the equation needed to compute the row component of pitch corrected points. Figure 4.10 illustrates how the angles between image points are conserved in the transformation.

$$\ddot{y}_a^i = tan\left(atan\left(\frac{\dot{y}_a^i}{f^i}\right) - \boldsymbol{\theta}_y\right) * f^i$$

Equation 4.67

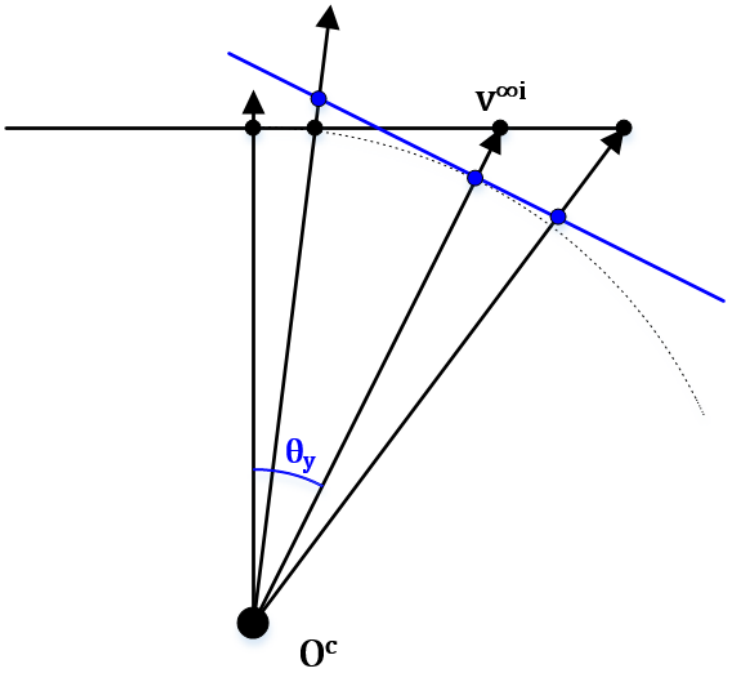


Figure 4.10: Rotation Operation Preserves the Angles between Points.

The z-axis translation is found using roll and pitch corrected image coordinates simply as

$$t_z^o = \left(\frac{1}{n}\right)\sum_{a=1}^{n}\frac{f^i}{\ddot{y}_a^i} * \boldsymbol{t}_y^o$$

Equation 4.68

There are many shortcuts one can apply to the Simple Method, however, the shortcuts yield less intuitive methods. The method is not mathematically valid, however, it does produce good results.

4.4 Experiment Design

Each pose estimation method is tested with a procedure that aims to mimic the conditions present in real world situations. First, Gaussian noise is added to a model of the slalom course. The course model consists of 3D coordinates that represent buoys in the slalom course and adding noise to the coordinates is synonymous with moving a buoy. The IWSF rulebook defines a minimum and maximum allowable tolerance for every buoy in every dimension. The tolerances used in the test are given in Table 4.1. The tolerances, in conjunction with a dependent, test varied parameter supply a standard deviation to the normal distribution used to generate noisy object coordinates. The process for generating noise in the x dimension of coordinate **p** is defined in the following three equations.

$$\mu_x = \boldsymbol{p}_x$$

Equation 4.69

$$\sigma_x = k * max(\boldsymbol{p}_x - T_{x,min}, T_{x,max} - \boldsymbol{p}_x)$$

Equation 4.70

$$p'_x = \eta(\mu_x, \sigma_x)$$

Equation 4.71

In the equation set, k is the "Noise Standard Deviation Multiplier". When k is zero, the distribution does not add noise to the coordinates. When k is one the distribution has a

standard deviation equivalent to the maximum tolerance. A noisy coordinate is only accepted if it falls within its corresponding tolerances. In the third equation, $\eta(\mu, \sigma)$ generates a random point from a distribution whose mean is μ and standard deviation is σ.

The noisy course model and a randomly generated ground truth pose, which is comes from a uniform distribution, are combined to produce image coordinates using an ideal pinhole camera model. The image coordinates are inspected to ensure that they conform to the assumptions that: (1) at least one image coordinate pair exists, and (2) at least two points exist on each line. Figure 4.11 shows some valid and invalid configurations.

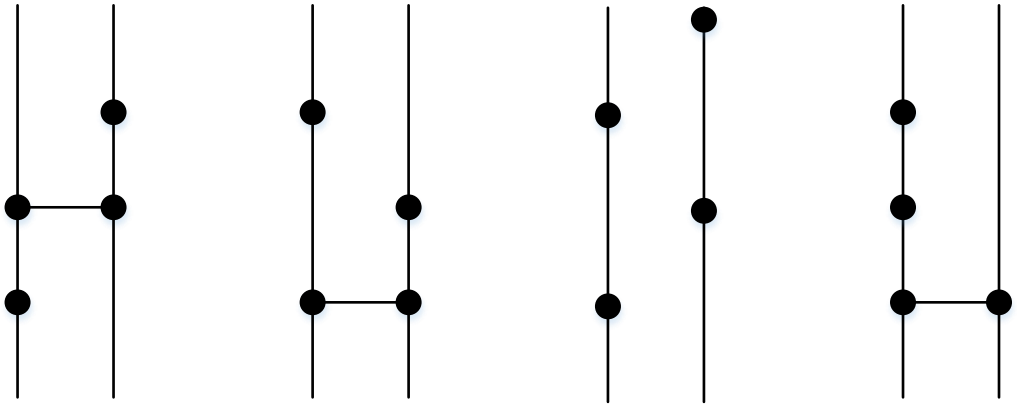


Figure 4.11: Valid and Invalid Configurations. (a)(b) Valid Test Configurations. (c)(d) Invalid Test Configurations. (c) No Pairs Present. (d) A line Fails to Have at Least Two Reference Points.

If the image coordinates pass all constraints, then they are further inspected to conform with the "maximum points per line" test parameter. If the pose-course combination produces more image coordinates per line than the maximum, then the furthest are removed. If the image coordinates pass all constraints and fit the test

parameters, then the original model, noisy image coordinates, and ground truth pose are passed into on to a pose estimation method.

	Pre-Gate (F, C)			Gate (E, A)			Guide (F, B)		
X(width)	1.035	1.15	1.265	1.188	1.25	1.313	1.035	1.15	1.265
Y(height)	.075	.1125	.15	.05	.0575	.085	.075	.1125	.15
Z(depth)	54.725	55	55.275	26.865	27	27.135	40.795	41	41.205

Table 4.1: Tolerances for Buoy Position.

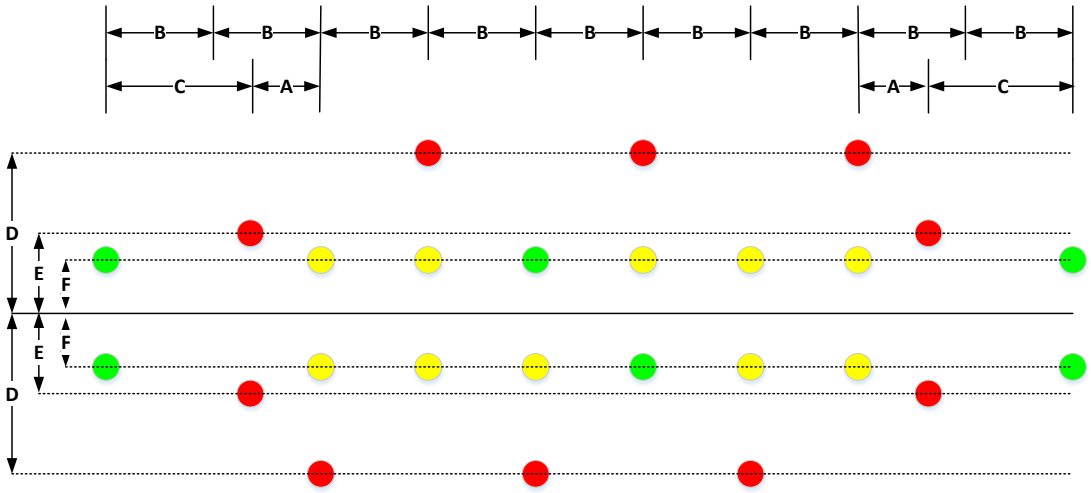


Figure 4.12: Segments Corresponding to Table 4.1.

4.4.1 Data Collection and Treatment

The generate and test process is repeated 1000 times for every combination of pose estimation method and test parameter shown in Table 4.2.

Parameter Name	Value(s)
Minimum Points Per Line	2
Minimum Pairs	1
Maximum Points Per Line	2, 4, 6
Noise Standard Deviation Multiplier	0, .005, .01, ... , .990, .995, 1

Table 4.2: Varied Parameters in the Pose Estimation Experiment.

In some cases, the methods produce pose estimates that are obviously incorrect. Incorrect pose estimates are detected by examining the average reprojection error. If the average reprojection error is greater than a threshold, 9 pixels for all tests, the pose estimate does not contribute to the calculated statistics. It only affects the outlier count for the method.

4.4.2 Source Code

All algorithms are implemented in C++. The source code for Posit Coplanar comes from the author's website. Upon testing Posit Coplanar, it was found that one of the code's functions, pseudoinverse, had a memory issue. The issue was resolved by replacing pseudoinverse function calls with calls to the OpenCV equivalent function. Two implementations of EPNP were found. One is provided by the authors, and one is included with OpenCV 2.4.8. A preliminary test of the two implementations showed that the OpenCV version was more robust than the author provided version. For this reason, all tests reflect the performance of the OpenCV version of the algorithm. The Levenberg-Marquardt optimization algorithm is also from OpenCV 2.4.8. The two proposed algorithms are encoded by the author of the thesis.

4.5 Results

It is believed that the most important pose parameter for a system attempting to drive the boat in a straight line is the deviation about the world coordinate system x-axis(side to side motion). While the algorithms tested produce all pose parameters, only statistics about the x-axis deviation are presented.

In the following graphs, 'Noise Standard Deviation Multiplier' is the parameter discussed in the design section that controls the standard deviation of the normal distribution used to generate noisy coordinates. The simulated units are in meters and the unit of the y-axis for each of the first six graphs is in meters. Each graph represents a separate "maximum points per line" parameter.

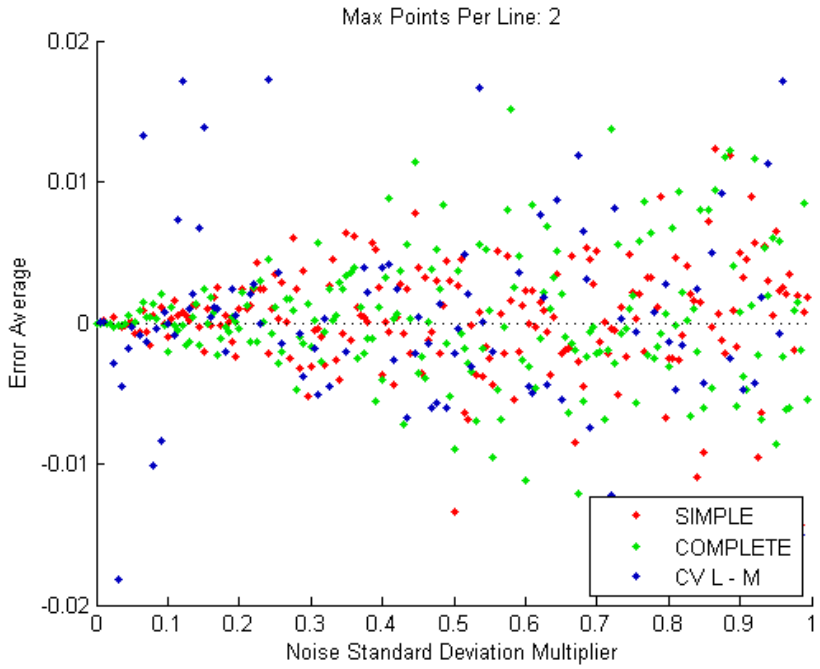


Figure 4.13: Average Error of the X-Axis Translation over 1000 Tests for Various Amounts of Noise When the Maximum Coordinates per Line Is Two.

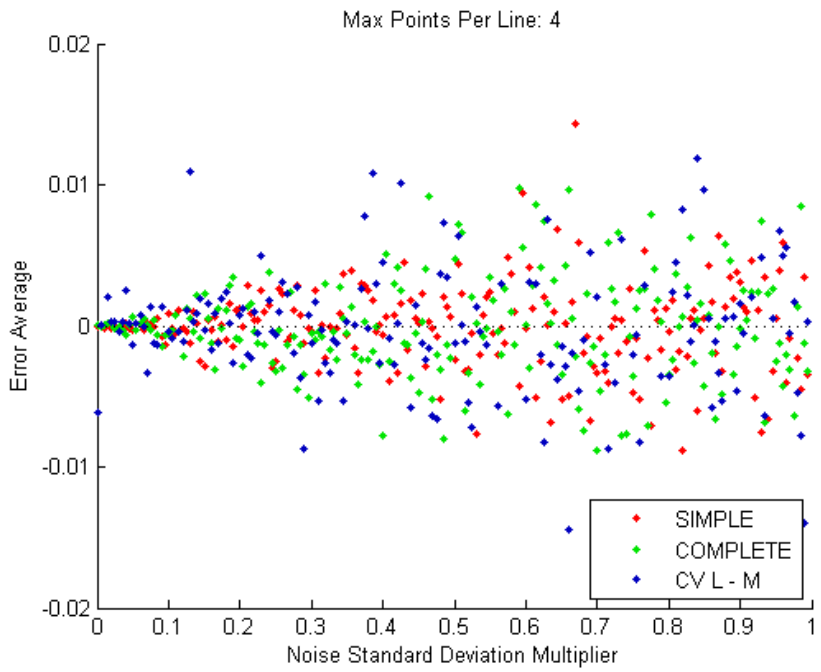


Figure 4.14: Average Error of the X-Axis Translation over 1000 Tests for Various Amounts of Noise When the Maximum Coordinates per Line Is Four.

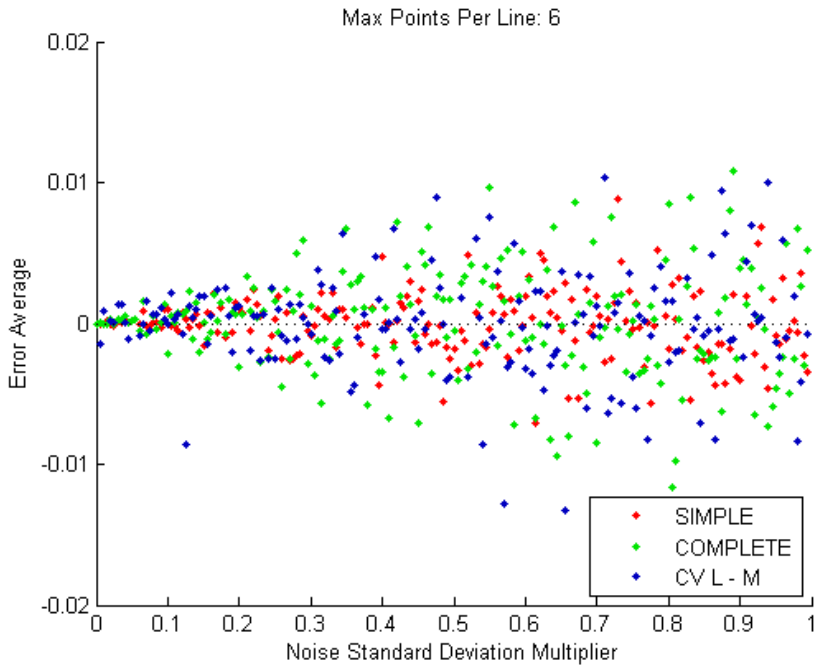


Figure 4.15: Average Error of the X-Axis Translation over 1000 Tests for Various Amounts of Noise When the Maximum Coordinates per Line Is Six.

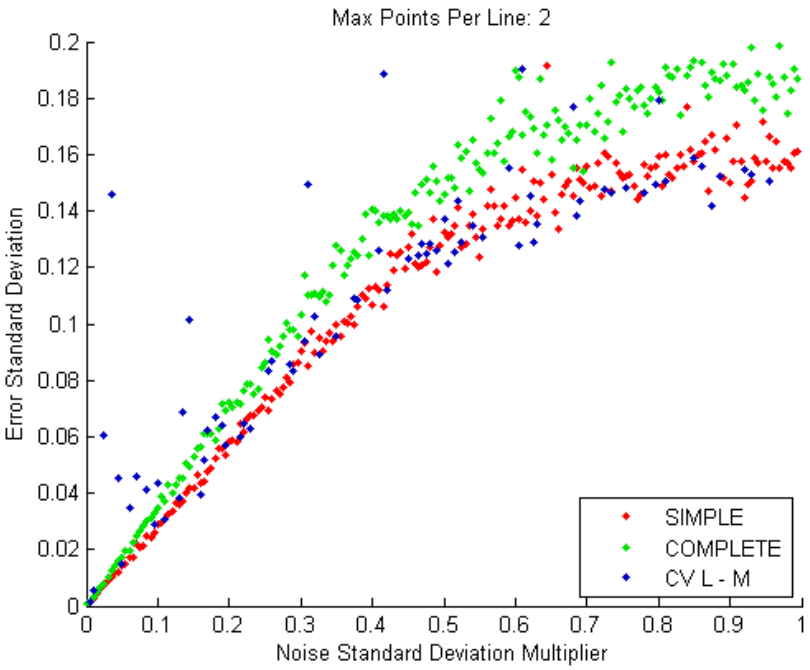


Figure 4.16: Error Standard Deviation of the X-Axis Translation over 1000 Tests for Various Amounts of Noise When the Maximum Coordinates per Line Is Two.

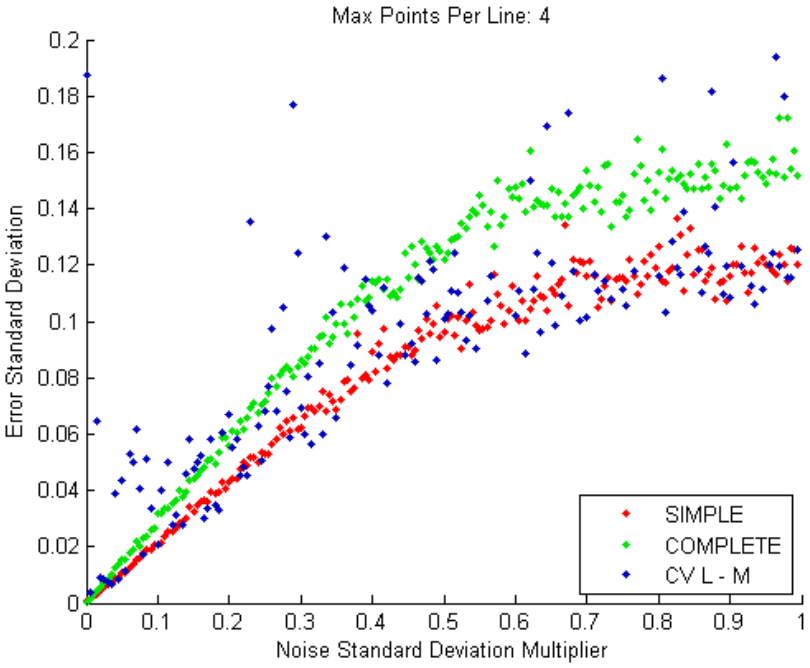


Figure 4.17: Error Standard Deviation of the X-Axis Translation over 1000 Tests for Various Amounts of Noise When the Maximum Coordinates per Line Is Four.

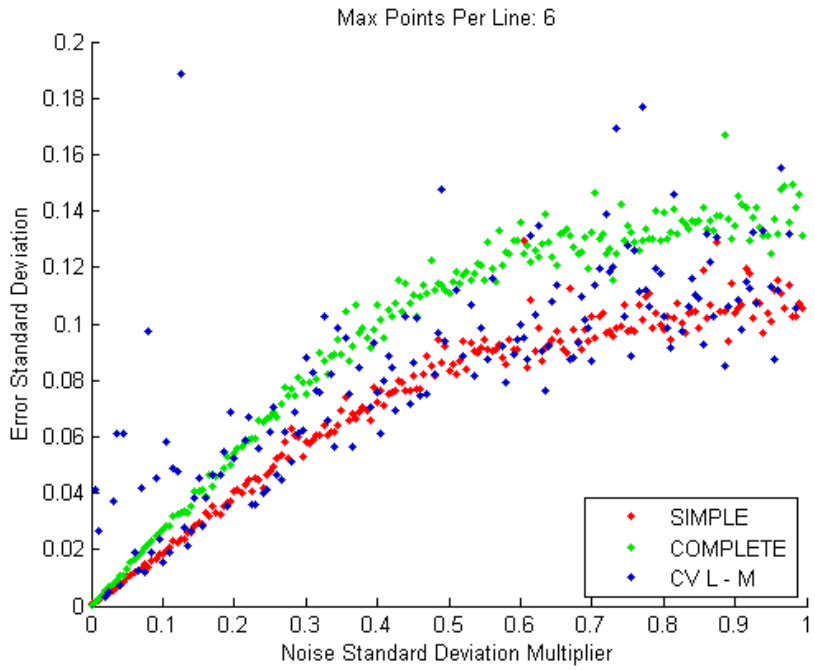


Figure 4.18: Error Standard Deviation of the X-Axis Translation over 1000 Tests for Various Amounts of Noise When the Maximum Coordinates per Line Is Six.

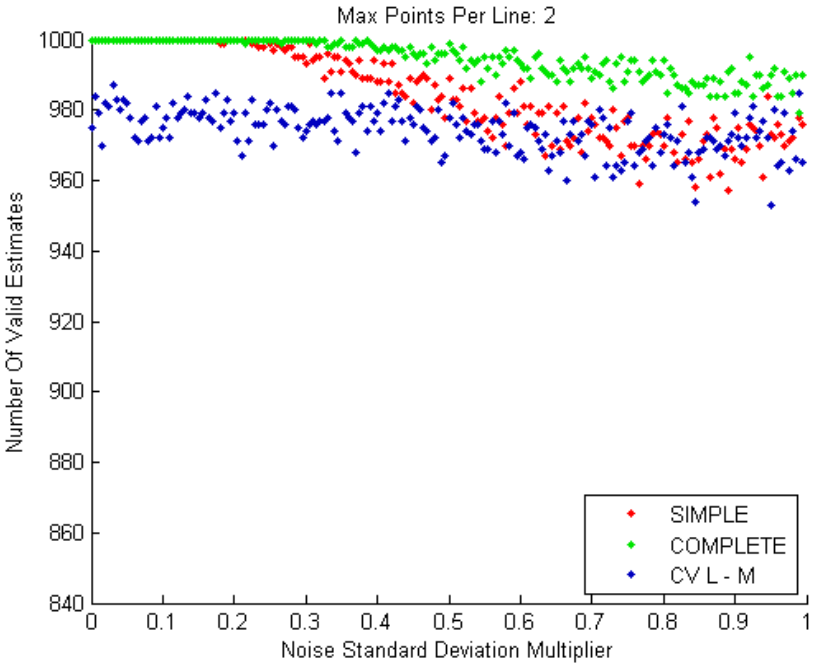


Figure 4.19: Number of Valid Pose Estimates out of 1000 When the Maximum Points per Line Is Two.

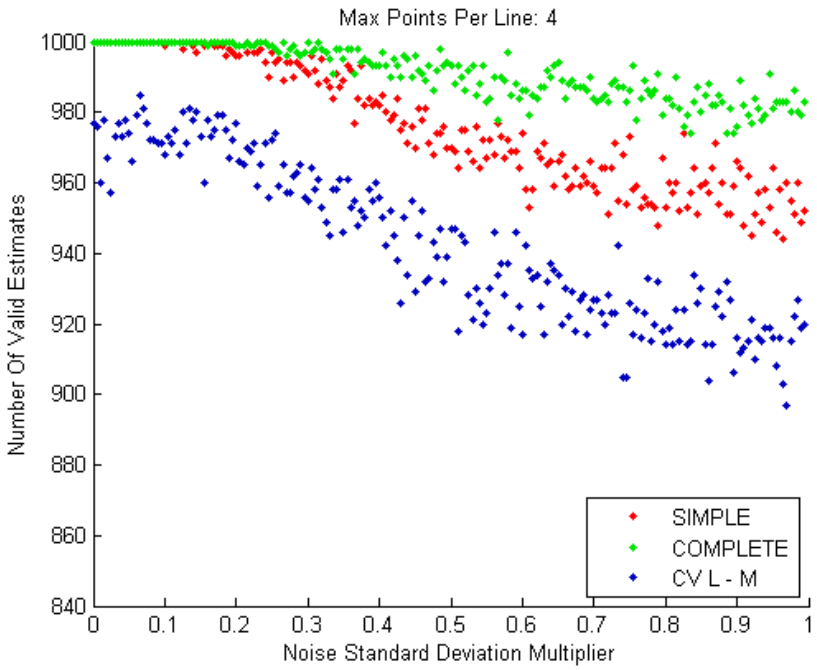


Figure 4.20: Number of Valid Pose Estimates out of 1000 When the Maximum Points per Line Is Four.

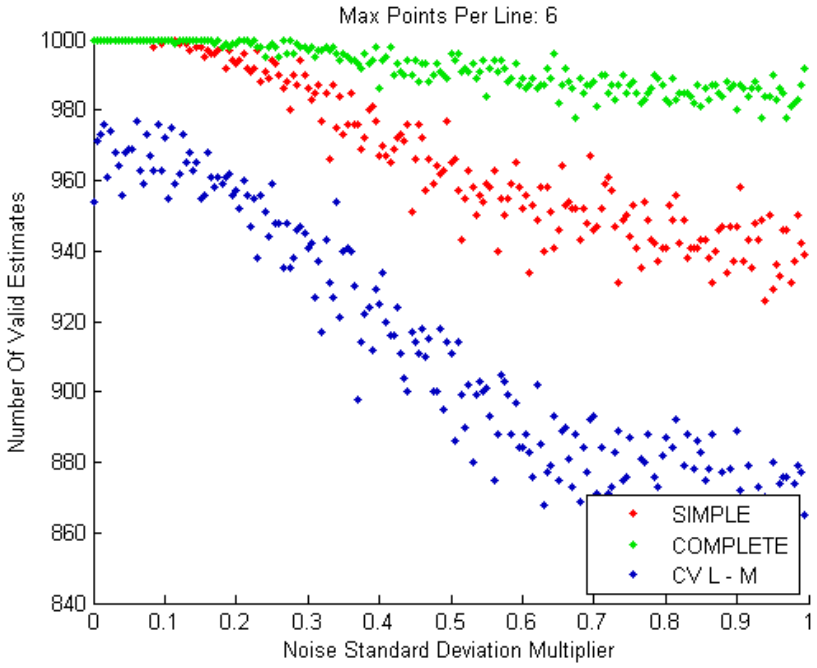


Figure 4.21: Number of Valid Pose Estimates out of 1000 When the Maximum Points per Line Is Six.

4.6 Discussion

The above graphs do not include the results of the EPNP and Posit Coplanar algorithms. The exclusion is due to the poor, incomparable performance of the algorithms on this type of object configuration. The poor performance of these two algorithms was not expected. As a precaution, the algorithms were tested with noiseless grid type object configurations to ensure that the errors were not implementation dependent. It was found that both algorithms performed as expected with the grid patterns. The poor performance of these algorithms is most likely due to the slalom course object configuration, the introduction of noise to the object coordinates, and the camera configuration (large focal length).

As for the three tested algorithms, the first three graphs presented, average error graphs, indicate that the algorithms are not predisposed to favor error in any certain direction. The shape of the data, narrow when the standard deviation multiplier is zero, and wide then it is one, indicates that in noiseless conditions, the algorithms have average errors near zero, and with more noise, the offsets become greater. This type of behavior is to be expected.

The graphs of the error standard deviation show that LM optimization procedure can outperform both proposed methods, however, its performance gain is not universal. The LM method's data shows that the algorithm is hit or miss while the proposed methods have predictable error bounds. This behavior is explainable by the mechanics of LM algorithm itself. The method may become trapped in a local minima while searching for optimum parameter settings. It is somewhat unexpected that the simple method outperforms the complete method in terms of error standard deviation. This may be due

to the fact that the graphs only display the x-axis translational error statistics. In terms of the other parameters, especially the z-axis translational error, the complete method generally outperforms the simple method.

The number of outliers, as shown in the last three graphs, also supports the idea that the LM optimization method is less stable than the two proposed methods. While the two proposed methods can generate less accurate parameter estimates, they generally are more robust than the LM algorithm. This behavior again is explainable by the fact that LM method can get stuck in a local minima.

The result of these tests indicate that a universally optimum pose estimation method may be obtained by first generating a pose estimate with one of the two proposed methods, and then using the LM algorithm to improve the accuracy of the estimate. This type of algorithm would give positive gains both in terms of stability and accuracy.

4.6.1 Limitations and Delimitations

The experiment mentioned has limitations. First, each method is only tested on synthetically generated data rather than real data. The synthetic data does account for noisy conditions, but does so in a manner that assumes an ideal pinhole camera model. This limitation is due to the fact that testing with real data would require an alternative method of estimating camera pose, and consequently the position and orientation of a boat on a lake, to provide ground truth data. Second, the experiment is designed to test ideal, high probability, slalom course configurations. Some configurations, such as a configuration in which a slalom course is out of tolerance, are not tested.

4.7 Conclusion

Five algorithms have been presented in depth and tested with a model tailored to simulate real slalom course conditions. It was found that two of the tested algorithms, EPNP and Posit Coplanar, struggle with the conditions, object coordinates, noise introduction, and camera intrinsic parameters, that mimic the complete system. The other three algorithms are comparable in terms of both the average and variance of error. Of the three acceptable algorithms, one is an iterative method, and the other two are vanishing geometry based methods constructed specifically for the problem of pose from points on two parallel lines.

One of the proposed methods is built such that linear systems allow all data points to contribute to the best possible result. The other method is constructed for maximum simplicity and only relies on basic trigonometry and basic statistics. The performance of the complete method, as displayed in the graphs in the results section, appears to be slightly less than the performance of the simple method. This is due to the fact that only x-axis translational error is examined. Both the two proposed methods have performance comparable with the LM optimization method and as a result, an algorithm which makes its initial guess with a geometry based method and optimizes with the LM method is proposed.

5 CONTROL

The prior three modules can generate a noisy pose estimates in real-time. The task now is converting noisy pose estimates into mechanical motion within the boat's steering system. A few additional steps are added in order to accomplish this task. First, pose estimates from the pose estimation module are filtered. Next, the filtered estimates are converted into stepper motor step positions. Finally, the step positions are transmitted to the stepper motor. A high level overview of each component necessary for steering control is given. Each control specific component is also discussed.

5.1 Overall Design

The additional logic required to convert pose estimates into mechanical motion is shown in Figure 5.1 as the Command Calculation, Instruction Translation, Communication, Physical Motion, and Control Instruction blocks.

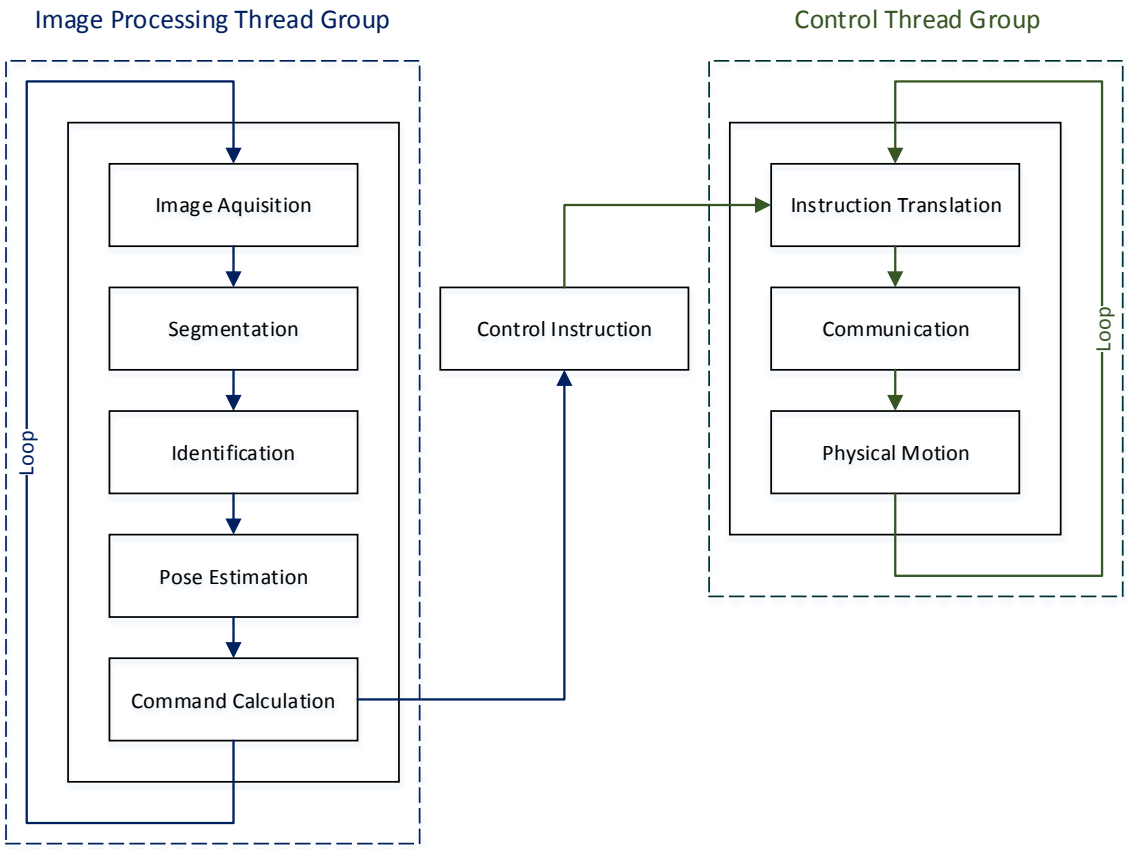


Figure 5.1: Additional Logic for Steering Control.

In terms of the entire system, determining and communicating the desired stepper motor step position can be divided into a two logical groups. The division is made based on thread synchronization. In Figure 5.1, the two main parts of the system are illustrated as the Image Processing Thread Group and Control Thread Group. The next section explains the need for two asynchronous thread groups.

5.2 Thread Interaction

Image acquisition, Segmentation, Identification, Pose estimation, and Instruction Calculation form one processing pipeline that utilizes one or more synchronized threads. The other processing blocks, Instruction Translation, Communication, and Physical

Motion form a separate processing pipeline. The two asynchronous pipelines exist due to the difference in execution time of one iteration. The image processing group is expected to iterate every 33 milliseconds or faster. The control group iterates at about five Hz. If the image processing pipeline included the elements in the control pipeline, the system would never meet real-time demands. The system would spend all of its time waiting for stepper motor commands to complete.

The two thread groups interact with each other via the Control Instruction block. The Control Instruction block is an allocated memory block accessible by both thread groups. It has no synchronization mechanisms associated with it. The image processing pipeline always writes the Control Instruction block with the a desired step position and the Control Thread group only reads the block when it is ready to execute a new command.

5.3 Command Calculation

Pose estimates are generated by the pose estimation module. The estimates are then converted into stepper motor step positions in the Command Calculation block. The computational time required by the block is insignificant. For this reason, and to keep the actual implementation as simple as possible, the Command Calculation block is part of the Image Processing Thread Group. Calculating a desired step position is a two part process. First, pose estimates are filtered. Then, they are converted into step positions.

5.3.1 Pose Estimate Filtering

There is existing work on modeling the dynamics of a surface vehicle and integrating sensor data into the model [60,61]. Due to the fact that this thesis is aimed at the vision components of the proposed control system rather than the control

components, a quick and easy implementation is used. A recursive averaging filter is used to filter all six components of the current pose estimate. The filter is simply

$$\boldsymbol{p}_t^f = \alpha * \boldsymbol{p}_t + (1 - \alpha) * \boldsymbol{p}_{t-1}^f$$

Equation 5.1

In the equation, $\boldsymbol{p}_t$ is the current six dimensional pose estimate and $\boldsymbol{p}_t^f$ is the filtered pose estimate at time t. α is the smoothing factor that controls which frequencies pass through the filter. Typically a value of $\alpha = .05$ allows the filter to act as a good low-pass filter.

5.3.2 Path Tracking

The smoothed data is the input to a controller that is based off the pure pursuit method [62, 63]. The pure pursuit method can be applied to curved paths, however, the desired path for the boat, a straight line collinear to the slalom course centerline, is a simple shape that allows for a simplified method. Calculating the step position is done with

$$s_t = k\left(atan\left(\frac{x_t}{L}\right) - \theta_y\right)$$

Equation 5.2

where s_t represents the desired stepper motor step position at time t, k is the proportional gain, x_t is x-axis displacement found from the filtered pose estimate, L is the lookahead distance, and θ_y is the current yaw angle. Figure 5.2 illustrates a typical situation and the source of the angles used in Equation 5.2. In the figure the term $\text{atan}\left(\frac{x_t}{L}\right) = \phi$.

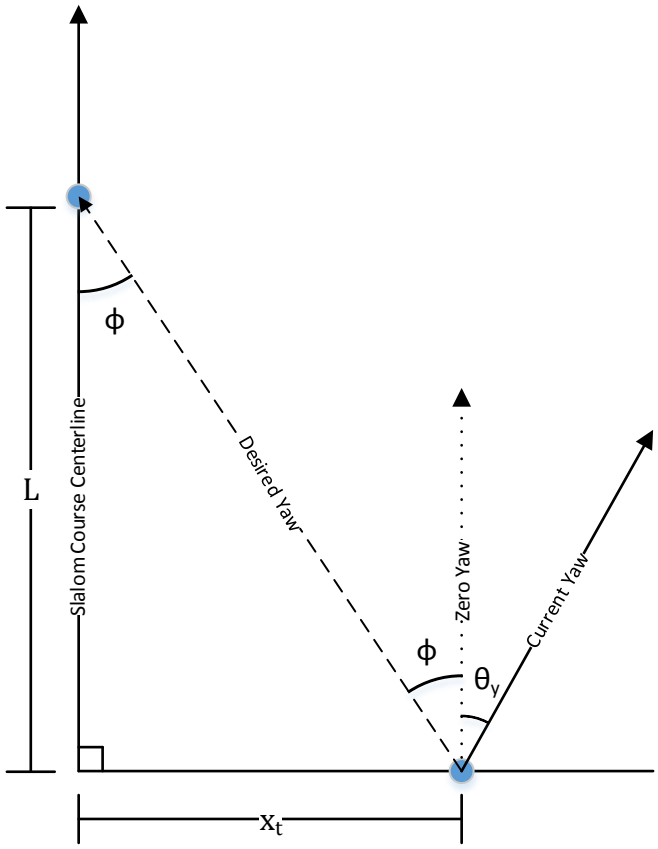


Figure 5.2: The Yaw Angles Used in Step Position Calculation.

5.4 Instruction Translation and Communication

The Stepper motor and computer interact via serial connection. Setting the stepper motor to a desired step position requires that the computer following the stepper motor's communication protocol. For simplicity, only the broad details of the communication protocol are provided. In the current implementation, the protocol includes querying, setting, and acknowledging values associated with the direction of motion, and number of steps. The computer also has to issue a "go" command and wait until the stepper motor finishes executing the loaded instruction.

5.5 Mechanical Motion

The stepper motor's connection to the boat's steering system is illustrated in Figure 5.3 and Figure 5.4. All components in the illustrations undergo three types of motion: non-moving, rotating, and linear. In terms of non-moving parts, the rack housing, shaft bearings, and stepper motor are all rigidly connected to the boat. In terms of rotational motion, the shaft pulley, steering shaft, pinion, and steering wheel all rotate together and ride on the shaft bearings. When the pinion spins, the rack moves, one-dimensionally, from side to side. This motion is rigidly transferred to the steering cable and eventually the boat's rudder.

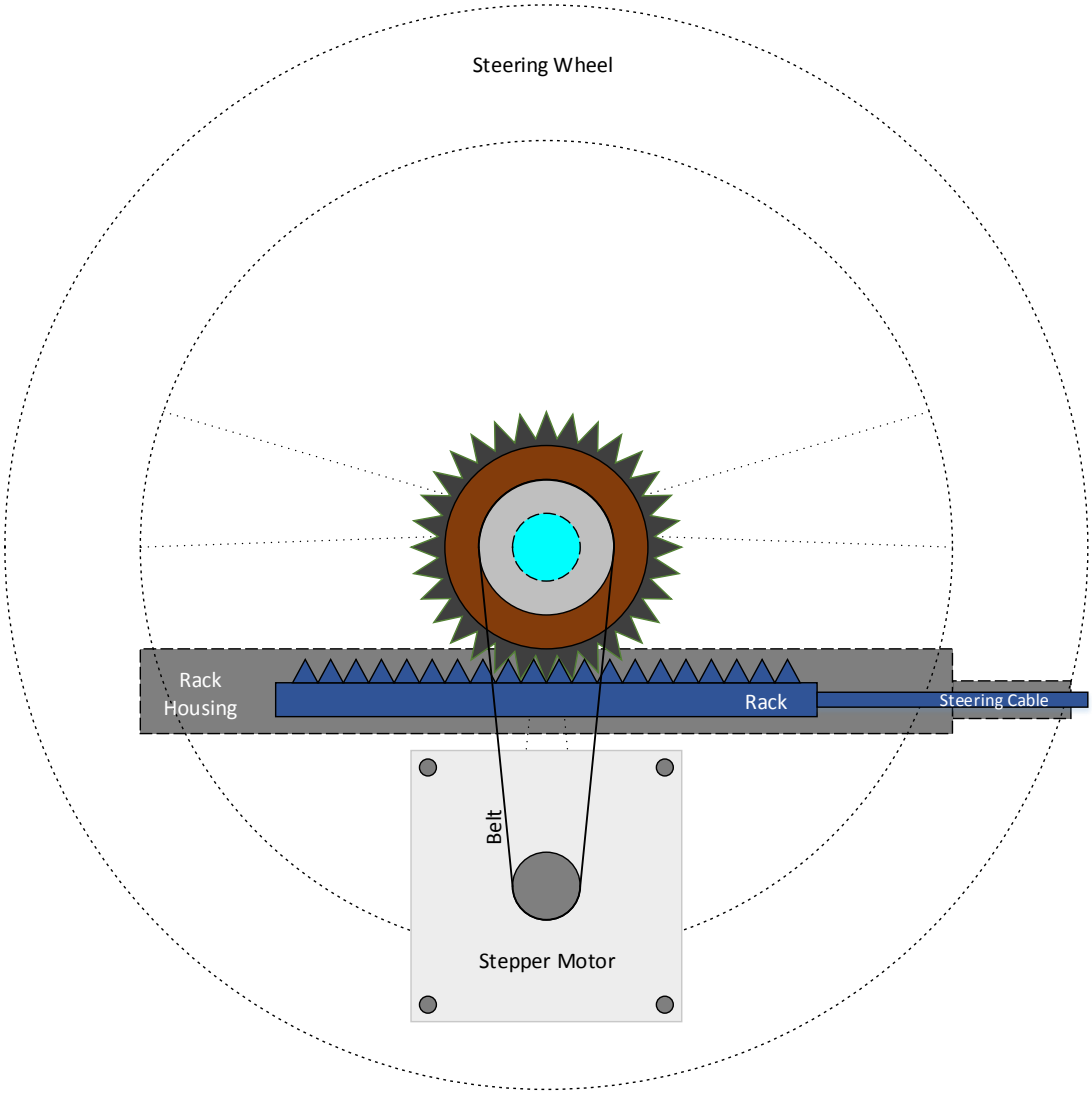


Figure 5.3: Stepper Motor Mechanical Connection Front View.

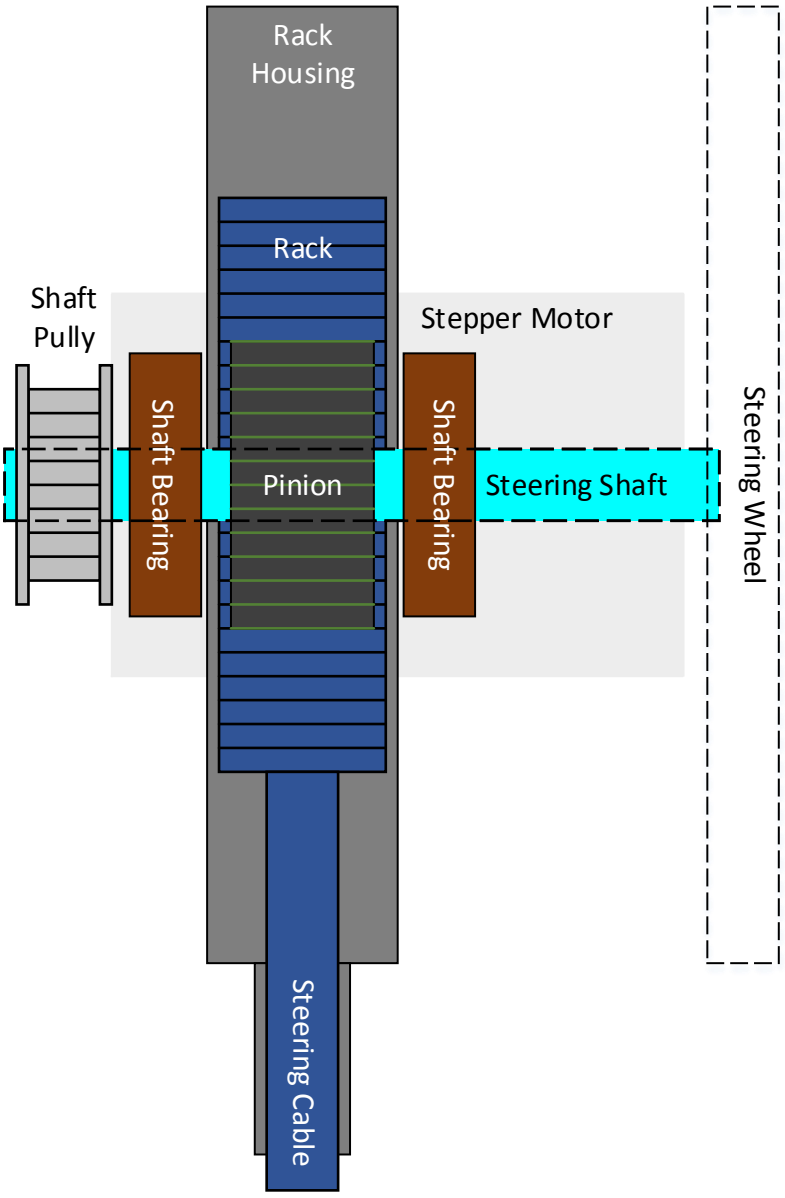


Figure 5.4: Stepper Motor Mechanical Connection Top View.

5.6 Overall Experiment and Discussion

The system was tested on multiple occasions. The tests took place around noon on sunny days. This time frame was selected because it offered the best lighting conditions. In the tests, the system successfully locked onto the slalom course and engaged stepper motor control at a distance of 35 meters from the first set of pre-gates. The system

remained in control until the boat was about a 25 meters from the last pair of gate buoys. While in control, the system appeared to slowly respond to lateral deviation. As a result, the boat seemed to oscillate from side to side as it traveled the length of the slalom course. Despite the oscillations, the system's camera was always able to view buoys and the boat never deviated out of the slalom course. It is estimated that the lateral offset of the boat from slalom course centerline never exceeded 30 cm. This estimate, however, is a human estimate, and a method of generating ground truth pose estimates for comparison is an area of further research. During the tests, the average and slowest frame rates were recorded as a metric of system throughput. It is found that the average frame rate of the system was 45 frames per second, and that the longest frame processing time was equivalent to a frame rate of 20 fps.

6 CONCLUSION

The thesis set out to develop a real-time boat steering control system for use in a waterski slalom course. The desire to build the system came from the need to resolve issues pertaining to boat driving in waterski competition and training. A design consisting of a segmentation, identification, pose estimation, and control module has been presented. The specific inputs, outputs, and goals of each module have also been defined. Finally, methods that appear to be capable of accomplishing module goals, as well as experiments that validate method performance, have been presented. A brief summary of findings, implications, and new questions are given in subsequent text.

An attempt to find a robust segmentation algorithm was made using background subtraction methods, color based methods, and a difference of Gaussians method. Experiments showed that the Visual Background Extractor provided stronger performance in comparison to other tested methods. In the end, the ViBe algorithm was selected as the segmentation algorithm of the system due to its simplicity and low computational complexity. The identification module chapter outlined a design, specific to slalom course configurations, that used intuitive to complex classifiers to find the optimum blob-to-buoy mapping. The ability of the identification module was validated by the successful real world test of the system presented in the control chapter. The pose estimation chapter presented and tested three publically available and two proposed pose estimation methods. In the end it was found that the two proposed algorithms had comparable performance to the iterative Levenberg-Marquardt optimization scheme. It was also determined that an even stronger pose estimation method could be created using a proposed method in conjunction with the Levenberg-Marquardt scheme.

The findings of the segmentation chapter align with the findings of [3] even though the authors of [3] used video sequences captured by infrared cameras. The ViBe algorithm appears to be a suitable segmentation algorithm in applications where camera's motion is primarily in the direction of the camera's optical axis and the camera views a uniformly colored and textured surface. The identification module is custom built for slalom courses but has the potential to operate in similar applications such as that of identifying markers along the edges of an airport runway or identifying lane markers on straight roadways. The proposed pose estimation algorithms are also custom tailored for a slalom course but are applicable to problems where object configuration can be described as points along two parallel lines. Interestingly, it is found that the publicly available implementation of Posit Coplanar and EPNP struggled to generate valid estimates when tested with slalom course object coordinate configurations.

There are a wide variety of questions opened by the thesis. As with any algorithmic research, any valid future research effort is one that aims to gain improvements in terms of robustness, simplicity, and time complexity. Specific to this thesis, however, there are questions that have more importance than others. What modifications are necessary so that the segmentation module handle images of low contrast such as those of the cloudy video sequences? Is background subtraction a good enough segmentation method or could other types of image segmentation methods provide better performance? In terms of the identification module, is there a simpler design that offers the same robustness? Finally, can other ideas such as a Kalman filter or inertial sensor integration improve position estimates? These are the questions that, if answered, would provide the bases of a much stronger overall machine.

A real-time vision based control system has been designed, defined and tested in the hopes of resolving problems in waterskiing. This thesis can serve as a guide for others who face problems that align with either the overall goal of the system or the chapter specific problems.

REFERENCES

[1] Hillel, Aharon Bar, et al. "Recent progress in road and lane detection: a survey."*Machine vision and applications* 25.3 (2014): 727-745.

[2] Corson, Robert K., ed. "International Waterski and Wakeboard Federation 2013 Tournament Water Ski Rules." (2013).

[3] Alexander, Borghgraef, et al. "An evaluation of pixel-based methods for the detection of floating objects on the sea surface." *EURASIP Journal on Advances in Signal Processing* 2010 (2010).

[4] Suzukawa Jr, Henry H., and Morton S. Farber. "Long-range airborne detection of small floating objects." *SPIE's 1995 Symposium on OE/Aerospace Sensing and Dual Use Photonics*. International Society for Optics and Photonics, 1995.

[5] Wei, Zhao-Yi, et al. "Motion projection for floating object detection." *Advances in Visual Computing* (2007): 152-161.

[6] Tall, M. H., et al. "Visual-Based Navigation of an Autonomous Surface Vehicle."*Marine Technology Society Journal* 44.2 (2010): 37-45.

[7] Dunbabin, Matthew, Alistair Grinham, and James Udy. "An autonomous surface vehicle for water quality monitoring." *Australasian Conference on Robotics and Automation (ACRA)*. 2009.

[8] Huh, Sungsik, and David Hyunchul Shim. "A vision-based automatic landing method for fixed-wing uavs." *Journal of Intelligent and Robotic Systems* 57.1-4 (2010): 217-231.

[9] Sharp, Courtney S., Omid Shakernia, and S. Shankar Sastry. "A vision system for landing an unmanned aerial vehicle." *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*. Vol. 2. IEEE, 2001.

[10] Olivares-Méndez, Miguel A., et al. "Fuzzy controller for uav-landing task using 3d-position visual estimation." *Fuzzy Systems (FUZZ), 2010 IEEE International Conference on*. Ieee, 2010.

[11] Miller, Andrew, Mubarak Shah, and Don Harper. "Landing a UAV on a runway using image registration." *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*. IEEE, 2008.

[12] Liu, Xinhua, and Yunfeng Cao. "Research on the application of vision-based autonomous navigation to the landing of the UAV." *Fifth International Symposium on Instrumentation and Control Technology*. International Society for Optics and Photonics, 2003.

[13] Wang, Xiaobing, Baokui Li, and Qingbo Geng. "Runway detection and tracking for unmanned aerial vehicle based on an improved canny edge detection algorithm." *Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2012 4th International Conference on*. Vol. 2. IEEE, 2012.

[14] Piccardi, Massimo. "Background subtraction techniques: a review." *Systems, man and cybernetics, 2004 IEEE international conference on*. Vol. 4. IEEE, 2004.

[15] Wren, Christopher Richard, et al. "Pfinder: Real-time tracking of the human body." *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 19.7 (1997): 780-785.

[16] Intille, Stephen S., James W. Davis, and Aaron F. Bobick. "Real-time closed-world tracking." *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*. IEEE, 1997.

[17] Stauffer, Chris, and W. Eric L. Grimson. "Adaptive background mixture models for real-time tracking." *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*. Vol. 2. IEEE, 1999.

[18] Zivkovic, Zoran. "Improved adaptive Gaussian mixture model for background subtraction." *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*. Vol. 2. IEEE, 2004.

[19] Zivkovic, Zoran, and Ferdinand van der Heijden. "Efficient adaptive density estimation per image pixel for the task of background subtraction." *Pattern recognition letters* 27.7 (2006): 773-780.

[20] Barnich, Olivier, and Marc Van Droogenbroeck. "ViBe: a powerful random technique to estimate the background in video sequences." *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*. IEEE, 2009.

[21] Barnich, Olivier, and Marc Van Droogenbroeck. "ViBe: A universal background subtraction algorithm for video sequences." *Image Processing, IEEE Transactions on* 20.6 (2011): 1709-1724.

[22] Jodoin, P., Janusz Konrad, and Venkatesh Saligrama. "Modeling background activity for behavior subtraction." *Distributed Smart Cameras, 2008. ICDSC 2008. Second ACM/IEEE International Conference on*. IEEE, 2008.

[23] Cheung, Sen-Ching S., and Chandrika Kamath. "Robust techniques for background subtraction in urban traffic video." *Proceedings of SPIE*. Vol. 5308. No. 1. 2004.

[24] Parks, Donovan H., and Sidney S. Fels. "Evaluation of background subtraction algorithms with post-processing." *Advanced Video and Signal Based Surveillance, 2008. AVSS'08. IEEE Fifth International Conference on*. IEEE, 2008.

[25] Walther, Dirk, Duane R. Edgington, and Christof Koch. "Detection and tracking of objects in underwater video." *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*. Vol. 1. IEEE, 2004.

[26] McFarlane, Nigel JB, and C. Paddy Schofield. "Segmentation and tracking of piglets in images." *Machine Vision and Applications* 8.3 (1995): 187-193.

[27] Remagnino, Paolo, et al. "An Integrated Traffic and Pedestrian Model-Based Vision System." *BMVC*. 1997.

[28] Jung, Young-Kee, and Yo-Sung Ho. "Traffic parameter extraction using video-based vehicle tracking." *Intelligent Transportation Systems, 1999. Proceedings. 1999 IEEE/IEEJ/JSAI International Conference on*. IEEE, 1999.

[29] Winnemöller, Holger, Jan Eric Kyprianidis, and Sven C. Olsen. "XDoG: an extended difference-of-Gaussians compendium including advanced image stylization." *Computers & Graphics* 36.6 (2012): 740-753.

[30] Rahman, Zia-ur, Daniel J. Jobson, and Glenn A. Woodell. "Multiscale retinex for color rendition and dynamic range compression." *SPIE's 1996 International Symposium on Optical Science, Engineering, and Instrumentation*. International Society for Optics and Photonics, 1996.

[31] Song, Keng Yew, Josef Kittler, and Maria Petrou. "Defect detection in random colour textures." *Image and Vision Computing* 14.9 (1996): 667-683.

[32] Alvarez, José M., A. López, and Ramon Baldrich. "Illuminant-invariant model-based road segmentation." *Intelligent Vehicles Symposium, 2008 IEEE*. IEEE, 2008.

[33] Moreno-Noguer, Francesc, Alberto Sanfeliu, and Dimitris Samaras. "A target dependent colorspace for robust tracking." *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*. Vol. 3. IEEE, 2006.

[34] Moreno-Noguer, Francesc, and Alberto Sanfeliu. "Integration of shape and a multihypotheses fisher color model for figure-ground segmentation in non-stationary environments." *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*. Vol. 4. IEEE, 2004.

[35] Chong, Hamilton Y., Steven J. Gortler, and Todd Zickler. "A perception-based color space for illumination-invariant image processing." *ACM Transactions on Graphics (TOG)* 27.3 (2008): 61.

[36] Yilmaz, Alper, Omar Javed, and Mubarak Shah. "Object tracking: A survey."*Acm computing surveys (CSUR)* 38.4 (2006): 13.

[37] Lorigo, Liana M., Rodney A. Brooks, and W. E. L. Grimsou. "Visually-guided obstacle avoidance in unstructured environments." *Intelligent Robots and Systems, 1997. IROS'97., Proceedings of the 1997 IEEE/RSJ International Conference on*. Vol. 1. IEEE, 1997.

[38] Katramados, Ioannis, Steve Crumpler, and Toby P. Breckon. "Real-time traversable surface detection by colour space fusion and temporal analysis."*Computer Vision Systems* (2009): 265-274.

[39] Fischler, Martin A., and Robert C. Bolles. "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography." *Communications of the ACM* 24.6 (1981): 381-395.

[40] Slabaugh, Gregory G. "Computing Euler angles from a rotation matrix."*Retrieved on August* 6 (1999): 2000.

[41] Dementhon, Daniel F., and Larry S. Davis. "Model-based object pose in 25 lines of code." *International journal of computer vision* 15.1-2 (1995): 123-141.

[42] Lepetit, Vincent, Francesc Moreno-Noguer, and Pascal Fua. "Epnp: An accurate o (n) solution to the pnp problem." *International journal of computer vision* 81.2 (2009): 155-166.

[43] Ansar, Adnan, and Konstantinos Daniilidis. "Linear pose estimation from points or lines." *Pattern Analysis and Machine Intelligence, IEEE Transactions on*25.5 (2003): 578-589.

[44] Oberkampf, Denis, Daniel F. DeMenthon, and Larry S. Davis. "Iterative pose estimation using coplanar feature points." *Computer Vision and Image Understanding* 63.3 (1996): 495-511.

[45] Orghidan, Radu, et al. "Camera calibration using two or three vanishing points."*Computer Science and Information Systems (FedCSIS), 2012 Federated Conference on*. IEEE, 2012.

[46] Wang, Yuxiang. "An Efficient Algorithm for UAV Indoor Pose Estimation Using Vanishing Geometry." *MVA* (2011): 361-364.

[47] Ying, Xianghua, and Hongbin Zha. "Camera pose determination from a single view of parallel lines." *Image Processing, 2005. ICIP 2005. IEEE International Conference on*. Vol. 3. IEEE, 2005.

[48] Zhi, Lihong, and Jianliang Tang. "A complete linear 4-point algorithm for camera pose determination." *AMSS, Academia Sinica* 21.239-249 (2002): 18.

[49] Quan, Long, and Zhongdan Lan. "Linear n-point camera pose determination."*Pattern Analysis and Machine Intelligence, IEEE Transactions on* 21.8 (1999): 774-780.

[50] Baker, Kirk. "Singular value decomposition tutorial." *The Ohio State University*(2005).

[51] Roweis, Sam. "Levenberg-marquardt optimization." *Notes, University Of Toronto*(1996).

[52] Ranganathan, Ananth. "The levenberg-marquardt algorithm." *Tutoral on LM Algorithm* (2004): 1-5.

[53] Mittrapiyanuruk, Pradit. "A memo on how to use the levenberg-marquardt algorithm for refining camera calibration parameters." *Website, http://cobweb. ecn. purdue. edu/∼ kak/courses-i-teach/ECE661/HW5 LM handout. pdf* (2006).

[54] Arun, K. Somani, Thomas S. Huang, and Steven D. Blostein. "Least-squares fitting of two 3-D point sets." *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 5 (1987): 698-700.

[55] Horn, Berthold KP, Hugh M. Hilden, and Shahriar Negahdaripour. "Closed-form solution of absolute orientation using orthonormal matrices." *JOSA A* 5.7 (1988): 1127-1135.

[56] Umeyama, Shinji. "Least-squares estimation of transformation parameters between two point patterns." *IEEE Transactions on pattern analysis and machine intelligence* 13.4 (1991): 376-380.

[57] Kipnis, Aviad, and Adi Shamir. "Cryptanalysis of the HFE public key cryptosystem by relinearization." *Advances in cryptology—CRYPTO'99*. Springer Berlin Heidelberg, 1999.

[58] Abdel-Aziz, Y. I. "Direct linear transformation from comparator coordinates in close-range photogrammetry." *ASP Symposium on Close-Range Photogrammetry in Illinois, 1971*. 1971.

[59] Schaffalitzky, Frederik, and Andrew Zisserman. "Planar grouping for automatic detection of vanishing lines and points." *Image and Vision Computing* 18.9 (2000): 647-658.

[60] Sonnenburg, Christian, et al. "Control-oriented planar motion modeling of unmanned surface vehicles." *OCEANS 2010* (2010): 1-10.

[61] Riggins, Jamie N. "Location Estimation of Obstacles for an Autonomous Surface Vehicle." (2006).

[62] Snider, Jarrod M. "Automatic steering methods for autonomous automobile path tracking." *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RITR-09-08*(2009).

[63] Amidi, Omead, and Chuck E. Thorpe. "Integrated mobile robot control." *Fibers' 91, Boston, MA* (1991): 504-523.