UNIVERSITY OF CALIFORNIA, SAN DIEGO

Computer Vision Techniques for Underwater Navigation

A thesis submitted in partial satisfaction of the
requirements for the degree
Master of Science

in

Computer Science

by

Christopher M. Barngrover

Committee in charge:

Professor Ryan Kastner, Chair
Professor Serge Belongie
Professor Yoav Freund

2010

The thesis of Christopher M. Barngrover is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

Chair

University of California, San Diego

2010

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank San Diego iBotics for allowing me to be part of the team and for use of the Stingray vehicle. Specifically I would like to thank Thomas Denewiler, who was always willing to help with work on the Stingray even if it was only advantageous to my research. I would also like to thank Prof. Kastner who helped navigate the academic research environment. In addition, Prof. Belongie and Prof. Freund provided guidance in developing my classifying algorithms. Finally, I would like to thank Junguk Cho and Sunsern Cheamanunkul who's research in object detection and boosting algorithms helped jump start my own research.

ABSTRACT OF THE THESIS

Computer Vision Techniques for Underwater Navigation

by

Christopher M. Barngrover
Master of Science in Computer Science

University of California San Diego, 2010

Professor Ryan Kastner, Chair

In the world of autonomous underwater vehicles (AUV) the prominent form of sensing has been sonar due to cloudy water conditions and dispersion of light. Although underwater conditions are highly suitable for sonar, this does not mean that vision techniques should be completely ignored. There are situations where visibility is high, such is in calm waters, and where light dispersion is not an issue, such as shallow water or near the surface. In addition, even when visibility is low, once a certain proximity to an object exists, visibility can increase. The focus of this project is this gap in capability for AUVs, with an emphasis on computer-aided detection through machine learning and computer vision techniques. All experimentation utilizes the Stingray AUV, a small and unique vehicle designed by San Diego iBotics. The first experiment is detection of an anchored buoy, which mimics the real world application of mine detection for the Navy. The second experiment is detection of a pipe, which mimics pipes in bays and harbors. The current algorithm for this application uses boosting machine learning on hue, saturation, value (HSV) to create a classifier followed by post processing techniques to clean the resulting binary image. There are many further applications for computer-aided detection and classification of objects underwater, from environmental to military.

Chapter 1

Introduction

1.1 Background

Like in the animal kingdom, the world of unmanned systems has vehicles for land, sea, and air. The unmanned aerial vehicles (UAVs) have the largest piece of the pie when it comes to research dollars. The unmanned ground vehicles (UGVs) take the next largest piece, which leaves the vehicles of the sea, including unmanned surface vehicles (USVs) and unmanned underwater vehicles (UUVs), splitting the smallest piece of the research funds. As a result of this division of research, UAVs and UGVs are leading the way in terms of autonomous capabilities. This project intends to provide some progress in the realm of UUV autonomy with a particular task at its core.

The Center for Marine Biodiversity & Conservation (CMBC) is a part of the Scripps Institution of Oceanography. This organization does underwater research all over the world. Often the research involves the emplacement of underwater sensors in shallow water, which will collect data for years at a time. There are many problems with such long emplacement times. The most obvious problem is the lags in analysis and further progress while waiting on sensor data. Also, if a sensor breaks while emplaced, the researchers will not discover the problem until they finally retrieve the sensor and attempt to download the data.

This is just one of many examples of a real world application for a UUV capable of intelligent navigation. For this specific example, there is great potential

for a semi-autonomous UUV to navigate to the sensors and download the data. Depending on the breadth of the sensor field, the UUV could do these sweeps monthly, weekly, or potentially even daily. This will provide the researchers a steadier stream of data to work with and therefore allow for much more flexibility to alter aspects of the experiments.

1.2 Research Goal

The vehicle that I am using for this project is called the Stingray. This vehicle is owned and maintained by a non-profit organization called San Diego iBotics Student Engineering Society (SD iBotics). The vehicle was originally designed for use in the Association for Unmanned Vehicle Systems International (AUVSI) Autonomous Underwater Vehicle Competition and is well suited for such a project. While the yearly competition was the original cause for work on the vehicle, there are many opportunities for research projects with the Stingray at the center.

The goal of this research is to use the Stingray platform two investigate computer vision techniques for object detection and object classification as a basis for navigation. The focus is on developing robust classifiers for competition mandated objects as well as for real world applicative objects. The classifiers will often be optimized with the use of boosting algorithms for developing decision trees about individual pixels or the image as a whole. While the purpose of the object detection and classification is navigation in the context of the AUVSI competition, the main focus of this research will be in computer vision techniques for object detection and classification.

The objects that will be the targets for the Stingray are a buoy and a pipe. The buoy is anchored to the floor of the pool and is floating some distance off of the floor. The buoy will be sought via the front camera on the Stingray. The requirement is to find the buoy and use its location in the image to steer the vehicle towards it. This means that the algorithm should find the buoy and estimate its center. The pipe is resting on the floor of the pool and will be sought via the bottom camera on the Stingray. The requirement is to find the pipe and use it as

a path towards future obstacles. This means that the algorithm should find the pipe, estimate its center, and estimate its bearing so that the Stingray can center over the pipe and aim in the direction of the path.

1.3 Navigation Research

Through my research, it is clear that navigation techniques for autonomous underwater vehicles (AUVs) are still in a highly research-based, theoretic stage. There are many different techniques to consider for navigation. Some are more effective in certain scenarios, such as deep water, while others can be used in multiple environments. Some of these techniques are improved by working in collaboration with other techniques. And some can only be used on vehicles of a certain size or with a certain capability.

The paper titled *Autonomous Underwater Vehicle Navigation* seemed to give the best overview of this field. Although this paper is 10 years old, it does not appear to be outdated. More recent papers tend to corroborate the findings of this paper. The paper gives an overview of the different techniques available for AUV navigation. I will summarize each of the described techniques and emphasize any techniques that seem better suited for this project or techniques that would clearly not work for this project.

The first technique, called dead reckoning, is the most researched and supported of the ones available. Also, it happens to require the cheapest equipment and therefore has potential for poor position estimates. Generally a compass, a water speed sensor, a depth sensor, and maybe a pressure sensor are integrated to attempt to track velocity over time. The goal is to estimate location based on vehicle velocity over a given time period. Obviously there is great potential for error in high current environments, leaving room for improvement.

The inertial navigation system (INS) uses gyros and accelerometers to calculate acceleration and motion to on all three axes. This allows for more accurate tracking of vehicle location than dead reckoning. The drawbacks to an INS for a small AUV like the Stingray are power consumption and cost. Another problem,

which both dead reckoning and INS share, is that error accumulates with the distance traveled. The initial phases of this work will be such short distances, such that this problem may not yet be evident. However, it needs to be considered as the project expands.

The next technique is referred to as either the Doppler Velocity Sonar (DVS) or the Doppler Velocity Log (DVL). Both of these are based on the same science, which is essentially a sonar sensor that pings down towards the sea floor and uses the reflections over time to calculate velocity relative to the seabed. This technique is very useful in shallow water situations and therefore is an attractive option for this project. The paper *Advances in Doppler-Based Navigation of Underwater Robotic Vehicles* gives further insight into the benefits of the DVL technique. It describes specific algorithms and results for using low-cost bottom-lock doppler sonars.

Another technique that has shown promise is acoustic based navigation through either long baseline (LBL) or ultra-short baseline (USBL). Both of these involve arrays of external acoustic transducers placed, generally on buoys, in the area that the vehicle will be navigating. The concept is that the vehicle can triangulate its position based on the timing of acoustic signals from the transducers, which have known positions. LBL and USBL are different ways of accomplishing this technique. Both of these have two major drawbacks, especially for this project. First is the cost of deploying and configuring the array of transducers. Second, and probably more relevant to this project, is the problems with acoustics in shallow water due to reflection off of the sea floor.

The next technique is broadly known as geophysical navigation. This involves the use of an accurate *a priori* map of the environment combined with measurements of geophysical parameters, such as bathymetry, magnetic field, or gravitational anomalies. The main problems are cost of creating the *a priori* map and the computational complexity of finding a peak in the sensor data. The complexity can be reduced by limiting the types of sensor involved in the mapping, using lower resolution maps by only using this technique for disparate, important areas, restricting the vehicle orientation during sensor pulling, and finally by

combining with other techniques to limit valid search area.

Another important technique is vision-based navigation, which uses computer vision and classification to localize position and navigate accordingly. Vision based navigation can be used similarly to geophysical navigation if there is an a priori map with unique visual features. Also, computer vision could be used to navigate by following a path shown by a cable on the seabed. This solution as an example is described in the paper *Vision Based Autonomous Underwater Vehicle Navigation: Underwater Cable Tracking*. The paper describes the algorithms used to locate the cable in the image plane based on vision, acoustics, and Laplacian of Gaussian (LoG) filtering.

The paper *Positioning an Underwater Vehicle through Image Mosaicking* describes how apparent motion of images form a mosaic in by feature selection and matching, detection of dominant motion points, and homography computation. This type of computer vision could be highly useful in this project since the environment involves relatively shallow, clear waters.

An obvious technique for positioning, which is utilized by ground and air vehicles, is GPS. It would be possible to use GPS as way to rectify position estimations by surfacing. This would require an research to decide how often to surface and float while satellites come on line and a position is estimated. For this project, with the emphasis on the AUVSI competition, this technique is not practical.

1.4 AUVSI Competition

This competition is officially called AUVSI and ONR's 12th International Autonomous Underwater Vehicle Competition. The competition is held at the Space and Naval Warfare Systems Center Pacific (SSC PAC) TRANSDEC Facility in San Diego, CA. The TRANSDEC Facility is a large pool designed to mimic the acoustic characteristics of open water. Specifically, sonars will not get acoustic responses from the walls of this state of the art pool. An aerial view of this facility is shown in Figure 1.1.

The objective of the competition is to successfully navigate an underwater

Figure 1.1: Aerial view of TRANSDEC facility.

course and complete obstacles along the way. The first obstacle, the gate, is not even considered part of the course map. This obstacle is a very wide, white or black pipe that is at the surface and has two smaller pipes perpendicularly entering the water. The goal of the first obstacle is to drive straight through the gate, between the smaller pipes and below the surface pipe. Completing this task signifies starting the course.

The first true obstacle is an orange pipe that is on the bottom of the pool and as meant to show the path of the course. This means that the vehicle should use the pipe as a source of bearing to the next obstacle. The next obstacle is the flare, which is an orange buoy anchored to the bottom and floating in the pool. The goal for this obstacle is to hit the buoy with the vehicle. After the buoy, another pipe leads to the barb wire, which is two green pipes anchored to the bottom and floating in the pool. The goal for this obstacle is to dive below the fence and navigate under it. From here, there are two pipes leading to two different obstacles. To the left will be the bomb run, which is four boxes with different symbols in them.

The goal is to place a marble in the two boxes containing the specified symbols. The other path from the barb wire leads to the machine gun nest, which is a square anchored to the bottom. The goal is to fire a torpedo through the machine gun nest. The final obstacle is called the briefcase, and should be found via an acoustic pinger. Once the location is found, the vehicle surfaces inside of a ring at the surface. The diagram of the course is shown in Figure 1.2.

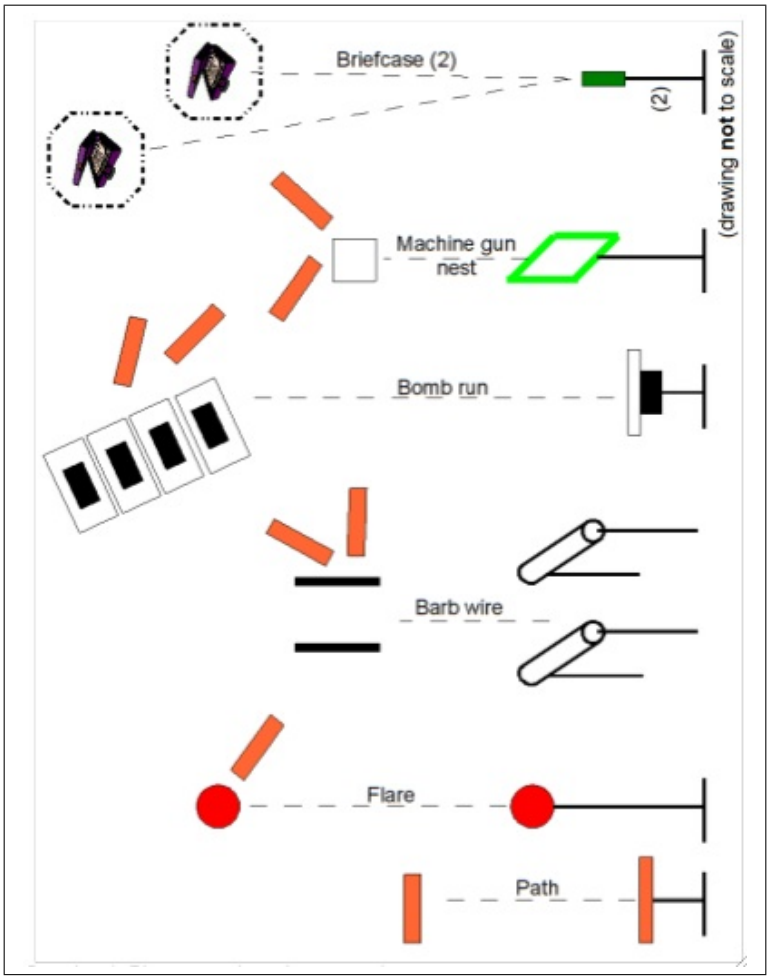

Figure 1.2: Competition (2009) course map.

1.5 The Stingray

The Stingray vehicle is a small semi-autonomous UUV, which is shown in Figure 1.3. The vehicle runs the Linux OS Ubuntu and starts multiple processes to handle different aspects of the vehicle. These processes are generally referred to as modules and they communicate via a standardized TCP protocol designed specifically for The Stingray. In order to understand the work for this project, it is important to understand the modules and the libraries the utilize.

Figure 1.3: The Stingray vehicle on its stand.

1.5.1 Vision Module

The vision module uses the two cameras, one in front and one underneath, to capture and process frames based on the current task. The task determines what algorithm the vision module uses on the image frames, and in effect determines what the vision module is looking for. This task element can be changed from other modules. This module is where nearly all of my work will be done, designing image processing algorithms.

1.5.2 Navigation Module

The navigation module controls the five propellers on the vehicle: one in the tail for pitch, two in the wings for roll, and two Voith propellers underneath for yaw. This module is constantly aiming for a target, which is defaulted to a flat and steady position. This means that if you push down on one of the wings or the tail, the Stingray will correct itself. Changing this target is what causes the vehicle to move, which can be done from other modules.

1.5.3 Planner Module

The planner module controls the autonomy of the vehicle. It tells the vision module what to look for through tasks and then creates targets for the navigation module based on the vision data and other sensor data. In addition, this module receives status data from the navigation module and from the labjack daemon, and serves this data out to interested modules.

1.5.4 Dock Control Station (DCS)

The Dock Control Station is the GUI portion of this software. It is generally run on a separate computer from The Stingray, and is used to control the vehicle for testing. This module provides statistics on the vehicle INS, pressure sensor, etc. It also allows a user to manually control the vehicle as the planner would. It can tell the planner module what task to run, which in turn controls the vision module, and it can tell the navigation module where to go.

1.5.5 Libraries

There are also, some other modules that are not considered main modules. One of these is the joystick module, which allows for control of the vehicle via gaming controller. Another important module is the labjack daemon, which monitors the battery levels on the vehicle and also captures the pressure sensor data. The labjack daemon used to be connected to navigation, but in order to minimize the

processing done in the navigation module, we have moved this to connect to the planner.

Chapter 2

Computer Vision

2.1 Labeling

Labeling is a very important element of boosting classifiers. The entire process of optimizing classifiers requires a large number of positive and negative examples for the goal object. With images, the labeling can be done with software and then the classifier examples at the specified locations can be extracted and formatted with scripts. The application Digital Notebook, which was developed at the Bio-Image Lab of the University of California Santa Barbara, is one such way to do visual labeling. A screen shot of the software is shown in Figure 2.1. This software allows for different shapes to be drawn on the image and given the property of valid or invalid. This means that a point or region defined by a shape can be labeled as positive or negative.

The output from labeling software such as Digital Notebook is in XML. After labeling many images, a script can use the XML information about coordinates and validity to pull from the actual image the classifier data for the regions in question. The classifier data depends on properties of the object, since some attributes will be more descriptive than others. The classifier data, along with its status as a positive or negative example, is used to create boosting input files in the format required by the boosting software.

Labeling is one of the most tedious and time consuming aspects of object detection and classification. Once the software packages are chosen and the scripts

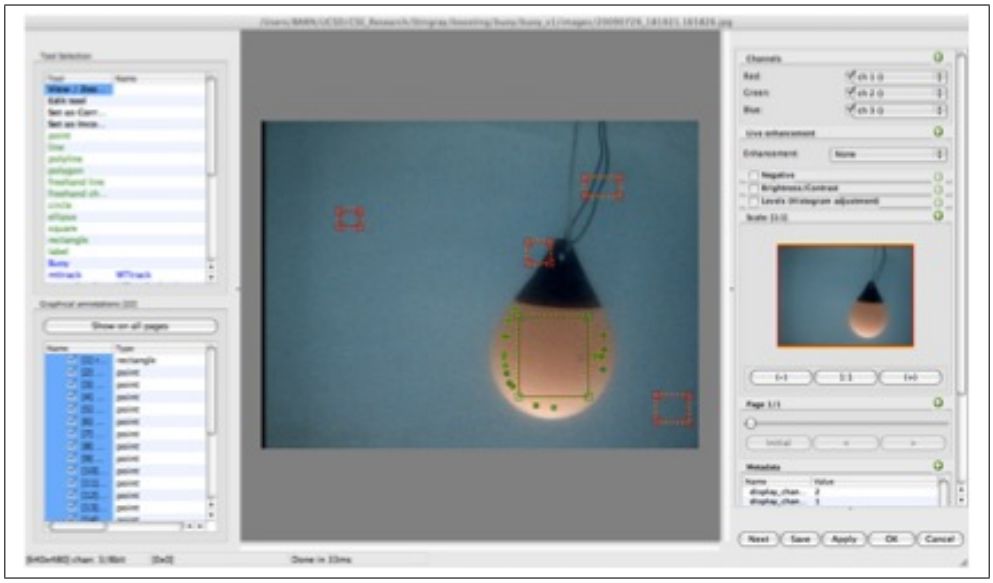

Figure 2.1: Screenshot of the Digital Notebook application used for labeling.

are designed and validated, creating a classifier for a new object is generally bottlenecked by labeling images. This is why a software package for visual labeling like Digital Notebook is so useful.

2.2 JBoost

The concept of *boosting* refers to systematically increasing the accuracy of a prediction rule through simple and often inaccurate base rules. The origin of boosting lies in Probably Approximately Correct (PAC) Learning Model, which is a framework for machine learning founded on learning by example. There are multiple variations of boosting algorithms, with AdaBoost being the first version with real potential. Since then there have been other variations such as GentleBoost, LPBoost, RankBoost, BrownBoost, LogLossBoost, and RobustBoost. Each has its own benefits and drawbacks, with no version being the clear winner in terms of results.

The boosting application JBoost is the premiere software for machine learning boosting. The software can be used in a plethora of ways with different boosting algorithm variations, such as AdaBoost, LogLossBoost, or RobustBoost. The ap-

plication expects the input files to be of a standard format providing the label and the classifier data. In addition the software expects and specification file to explain the format chosen for the boosting data files. JBoost can output the resulting decision tree visually as well as providing code in Java or C that will use the decision tree when given the classifier data types specified. An example of the first ten nodes of a decision tree is shown in Figure 2.2. For this research the C

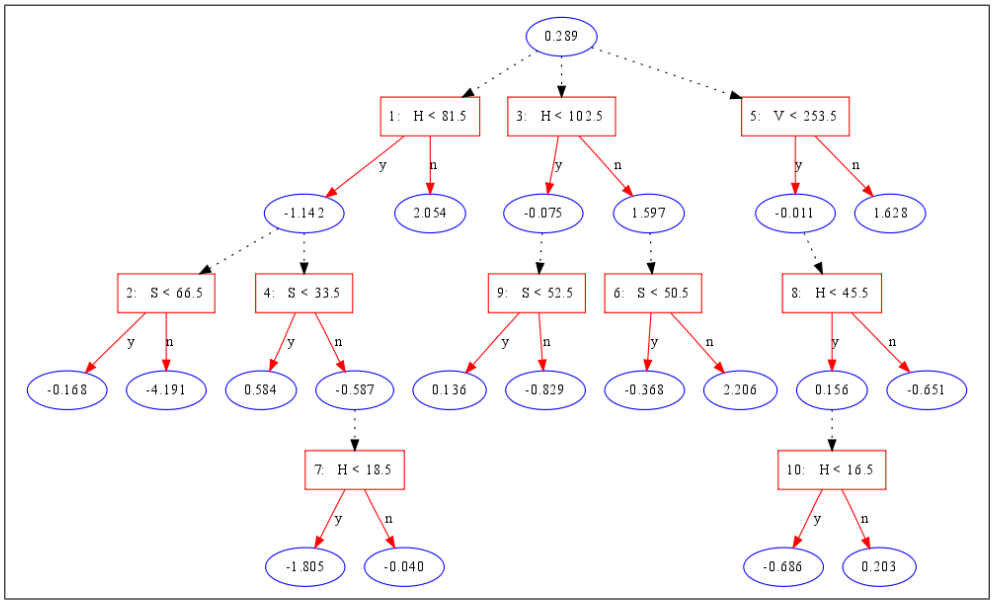

Figure 2.2: Example decision tree visualization.

code will be used to predict the label based on input. The output code in C needs to be altered slightly if the new `strlcpy` function is not in your `string` library. I developed a php script called `fix_c.php`, which will alter the C code so that it runs without needing the new library.

2.3 Color Classifiers

The most common color model is the Red-Green-Blue or RGB Color Model. The RGB color model is displayed graphically in Figure 2.3. This model is an additive model, which means it adds the wavelengths of light from the red, green, and blue in order to create the final color. When the absense of all three categories

occurs the resulting color is black. The final color is white when all the colors are at full intensity. Effectively, this model describes what kind of light needs to be emitted to produce a given color. Since the color is represented by how much of a given kind of light reflects off the object, it is often difficult to identify the same object under different lighting conditions based on color.

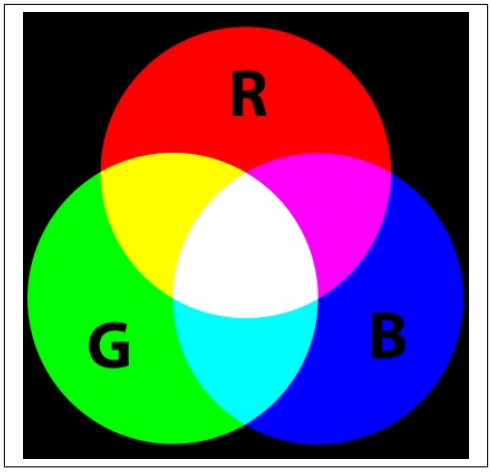

Figure 2.3: Additive colors of the Red-Green-Blue color scheme.

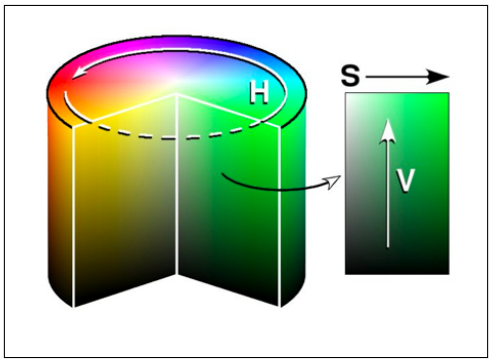

Figure 2.4: Cylinder representation of the Hue-Saturation-Value color scheme.

A second color model alleviates the problem of differing light conditions, to some degree. This model is called the Hue-Saturation-Value or HSV Color Model and is often referred to as Hue-Saturation-Brightness or HSB, since the value represents the brightness of the hue and saturation pair. This model can be represented in a cylinder, as shown in 2.4, where point on the circle is chosen

based on the hue and saturation and then the depth into the cylinder is the value or brightness. Because of this isolation of the brightness element of a color, a single object is more reliably detectable under different lighting conditions causing different reflectivity from the object.

2.4 OpenCV

Open Computer Vision or OpenCV was originally created by Intel in 1999 to advance CPU intensive applications. For nearly a decade, the software was gradually updated, focusing real time image processing. In 2008 the project received corporate support from Willow Garage and is currently under active development. The library has functions to support all aspects of image processing with the addition of some machine learning algorithms. Figure 2.5 gives a visual representation of the breadth of the library.

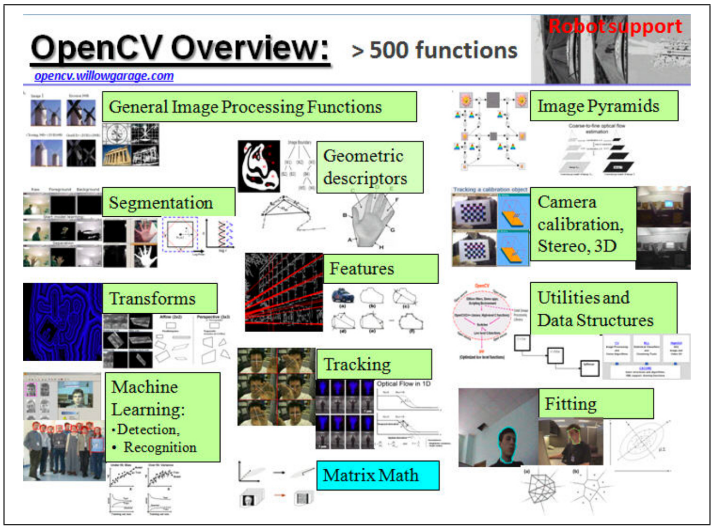

Figure 2.5: OpenCV map of capabilities.

My research only uses a fraction of the functions contained in the OpenCV library. Primarily, I use the library for post processing of the binary image created from the classifying decision tree. This includes smoothing, dilation, erosion, and convex hull techniques to name a few. The library also has Hough Transform and Canny edge detector algorithms, which are essential to this research.

Chapter 3

Buoy Detection

3.1 The Buoy

The buoy is one of the obstacles in the AUVSI competition of 2009 and is a relatively simple first object for detection. As can be seen in Figure 3.1, the buoy has distinct orange color and round shape. The data set of images of the

Figure 3.1: The orange buoy outside of water.

buoy underwater consists of images in the TRANSDEC tank as well as images in

a small test pool. The goal is to develop an algorithm that can accurately detect this buoy in both environments.

3.2 Baseline

The very simple algorithm used during the AUVSI competition creates a binary image from the color image. The algorithm considers each pixel and if its hue, saturation, and value (HSV) are within a predetermined range, then the pixel is considered positive and is made white. If the HSV is outside of this range, then the pixel is considered negative and is made black. An example of this baseline algorithm processing an image is shown in Figure 3.2. This algorithm is a good starting point, but is not very robust. It provides a good baseline for comparing iterations of for improvement via experimentation.

Figure 3.2: Baseline image of buoy and pipe.

Notice in this example that the buoy is not filled in because of the reflective part of the buoy. Also notice that the pipe lights up because the colors are so similar and both within the large range. Finally, look at the green circle in the color image, which represents the algorithms estimation of the center of the buoy. This simple algorithm places this green circle at the centroid of white pixels over the entire image. Also as seen in Figure 3.3, this algorithm is susceptible to noise because of the large HSV range.

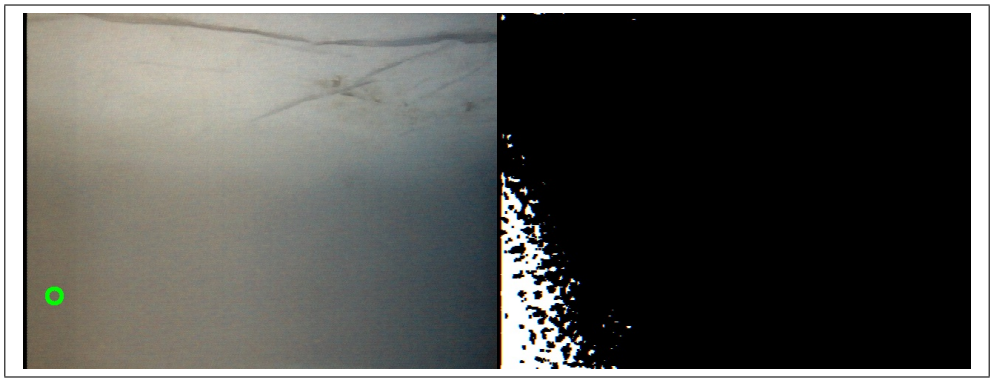

Figure 3.3: Baseline image with no buoy but noise causes a false positive.

When running this algorithm on a set of 100 images from the pool and 60 images from TRANSDEC we can see some metrics on the quality of the algorithm. These results can be seen in Table 3.1. This data shows us quite a few things about this algorithm. First of all, most of the false positives, caused by noise, and most of the false negatives, caused by not detecting anything when the buoy is present, occur on the pool images. The number of false negatives for the pool means that the algorithm is missing most of the buoy images. We can also draw conclusions about this algorithm from the quality of the center error, which is presented as a total error and on the individual axes. These center error values are in terms of pixels and naturally the lower the error the better the algorithm. It is important to note how high the standard deviations are for these errors. This means that the center error is quite a bit worse in some cases.

Table 3.1: Baseline metrics for buoy detection via AUVSI algorithm.

TRANSDEC		Pool	
False Positives	1	False Positives	8
False Negatives	22	False Negatives	63
Center Error	31.0640 ± 41.6213	Center Error	35.9370 ± 26.3397
X-Axis Error	9.7768 ± 24.9358	X-Axis Error	23.4375 ± 21.6101
Y-Axis Error	27.7946 ± 34.747	Y-Axis Error	24.0625 ± 19.7467
Time Per Frame	79.802868 ms	Time Per Frame	86.821631 ms

It is clear that this baseline algorithm leaves room for improvement, which is an excellent starting point for an introduction into computer vision techniques for classification using machine learning and boosting algorithms. The next section will follow the research and experiments as the algorithm is developed and improved.

3.3 Classification Techniques

This section describes different classification techniques and the experiments that either prove or disprove their usefulness for the improvement of the algorithm. Each technique considered involves one or more versions of the algorithm, whose results are compared to previous versions to determine the quality of the technique for improving classification of the buoy.

3.3.1 Boosting HSV

The first technique considered is the unique color of the buoy as a classifier. This is similar to the technique used by the baseline algorithm, except this is a much more sophisticated version. The first step for this technique is to label a group of training images using the Digital Notebook software. As described in Section 2.1, pixels can be labeled via regions, which in turn can be set as valid or invalid. In this case the script will only accept point and rectangle region types. Once a sufficient number of pixels are labeled positive and negative, these pixel values need to be placed into a `buoy.test` and `buoy.train` files for use by the JBoost application. Once these files are created and a `buoy.spec` file exists explaining the format of the files, the JBoost application uses machine learning to generate a decision tree. The resulting code called `buoy.c` is used in the Stingray vision code to predict whether a given pixel is positive or negative.

The first three versions of the classifying algorithm improve gradually by increased labeling between each round. The extra labeling is intended to clarify problem areas to the algorithm, such as false positives or false negatives. The images shown in Figures 3.4 - 3.6 show that the extra labeling removes noise by

improving the accuracy of the decision tree.

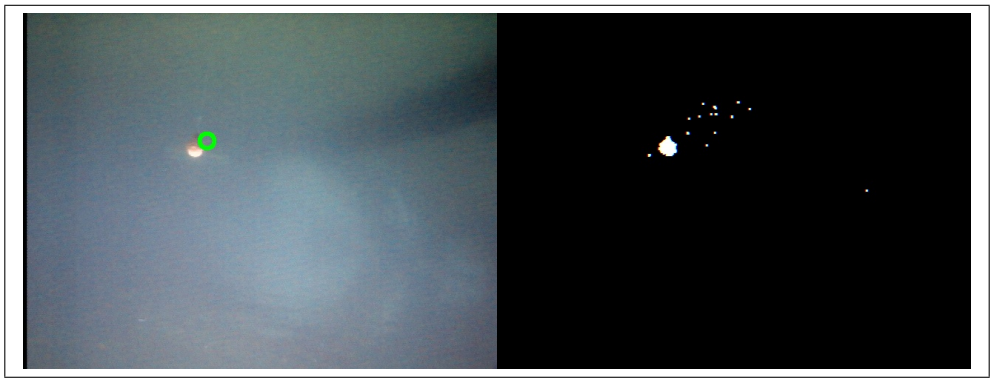

Figure 3.4: Example output from `buoy_v1`.

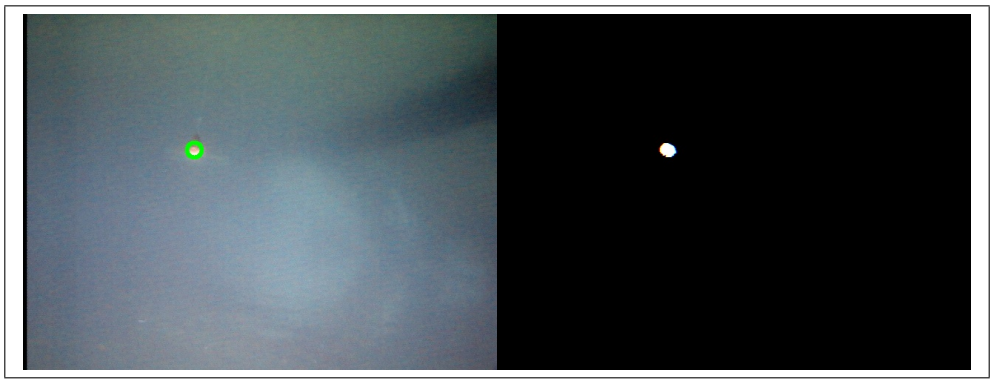

Figure 3.5: Example output from `buoy_v2`.

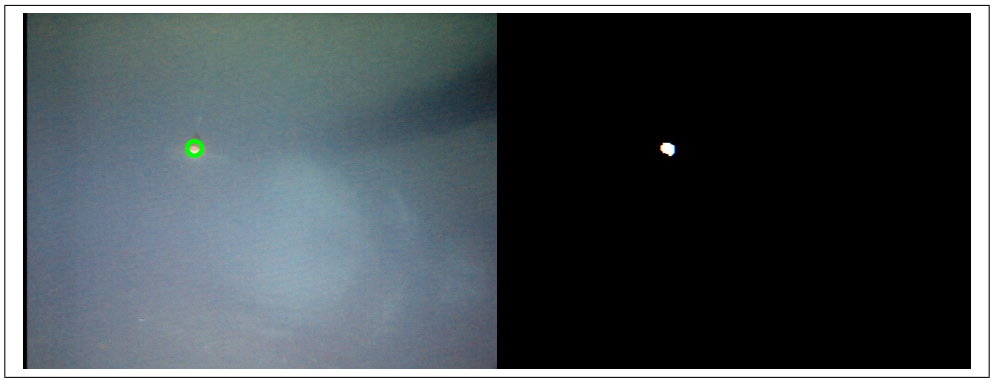

Figure 3.6: Example output from `buoy_v3`.

The boosting algorithm at this stage has some advantages and some disadvantages over the baseline algorithm. To understand the differences, Table 3.2 shows the metrics for algorithm `buoy_v3` on TRANSDEC and pool images respectively. The false positives are about the same, but there is an improvement in the number of false negatives for both sets of data. Also, this algorithm is much faster than baseline algorithm, taking 34.1 ms and 32.5 ms respectively. Notice however that when it comes to the numbers, this algorithm has a worse error for the center error in both environments. This means that some improvements are necessary overall, and especially for false negatives in the pool environment.

Table 3.2: Metrics via `buoy_v3` algorithm.

TRANSDEC		Pool	
False Positives	2	False Positives	8
False Negatives	0	False Negatives	48
Center Error	46.1943 ± 66.3879	Center Error	63.2915 ± 103.6714
X-Axis Error	13.8650 ± 48.8645	X-Axis Error	47.7097 ± 94.8096
Y-Axis Error	40.3250 ± 48.3238	Y-Axis Error	32.0000 ± 49.6439
Time Per Frame	34.118964 ms	Time Per Frame	32.526381 ms

3.3.2 Post Processing and Filters

With a fairly accurate HSV classifier now working, the next goal was to clean up the results with various filters and pre or post processing techniques. The techniques I tested, both before and after using the classifier, include smoothing, dilation, erosion, and contour approximation. The goal of these techniques is to clean up the resulting binary image for the best possible estimation of the location of the buoy.

The first such filter I tested was a smoothing algorithm, which attempts to smooth the edges of objects in the image. The OpenCV library provides a smoothing function, which can use different smoothing techniques such as Gaussian or Median blur. The Gaussian blur convolves each point with a Gaussian

kernel to produce the output array. The Median blur replaces each pixel by the median in a neighborhood of the pixel. The Median blur is used in this experimentation. Smoothing the image prior to using the boosting classifier causes extra positive pixels, which in turn causes more noise and increased size of connected components, or blobs. Doing the smoothing after the boosting classifier more accurately captures the location of valid objects. The resulting blobs, such as those seen in Figure 3.4, simply output with smooth edges as seen in Figure 3.7.

Figure 3.7: Example output from `buoy_v3` with smoothing.

The next filters I experimented with are known as dilation and erosion. Dilation is the act of calculating a local maximum over some region known as the kernel and replacing image pixels with the local minimum. The effect is that bright regions of the image grow. In a binary image, this means that any blob will expand under an iteration of the dilation algorithm. The opposite process is known as erosion, which uses the local minimum over some kernel to cause the bright regions of the image to shrink. Both dilation and erosion are provided functions with OpenCV.

These two algorithms can be used in sequence for interesting and useful tasks. Running the erosion algorithm followed by the dilation algorithm is known as opening. Any blob that is too small will erode away during the first step and any blob of decent size will simply return in approximately the same state. This means that the opening algorithm can be used to remove noise from the processed image. The reverse is called closing, which is dilation followed by erosion. This has the effect of filling in gaps and closing off edges to create more complete blobs.

The morphology functions of opening and closing are functions within the OpenCV library.

For the buoy processing, I experimented with both opening and closing in different iteration counts. If an iteration count is greater than one, then the first operation, such as erosion for opening, is done multiple times before the second operation, such as dilation for opening, is done the same number of times. In the case of opening this will remove larger blobs as noise as the iteration count increases. I first experimented with iterations of opening and found that due to the classifiers lack of false positives, only one iteration was necessary to remove noise. A second iteration removed the correct blob in some cases. Figure 3.8 shows the output from two versions of the algorithm combined into one image. The top half is the output from `buoy_v3` without any post processing, while the bottom half uses opening which removes the noise in the bottom left.

Figure 3.8: Example output from `buoy_v3` with opening.

Next I experimented with iterations of closing, starting with one iteration and increasing from there. I found that increasing the number if iterations was

beneficial at filling in gaps in blobs without any real negative. This is because the classifier is so strong that most blobs are a buoy or its reflection and rarely completely false. It is beneficial to close in order to get a full blob, so the result of the experiment is using a two iteration closing algorithm. Figure 3.9 shows combined outputs from `buoy_v3` without closing on top and with closing on bottom. Notice that the bottom binary image has a closed full blob for the buoy.

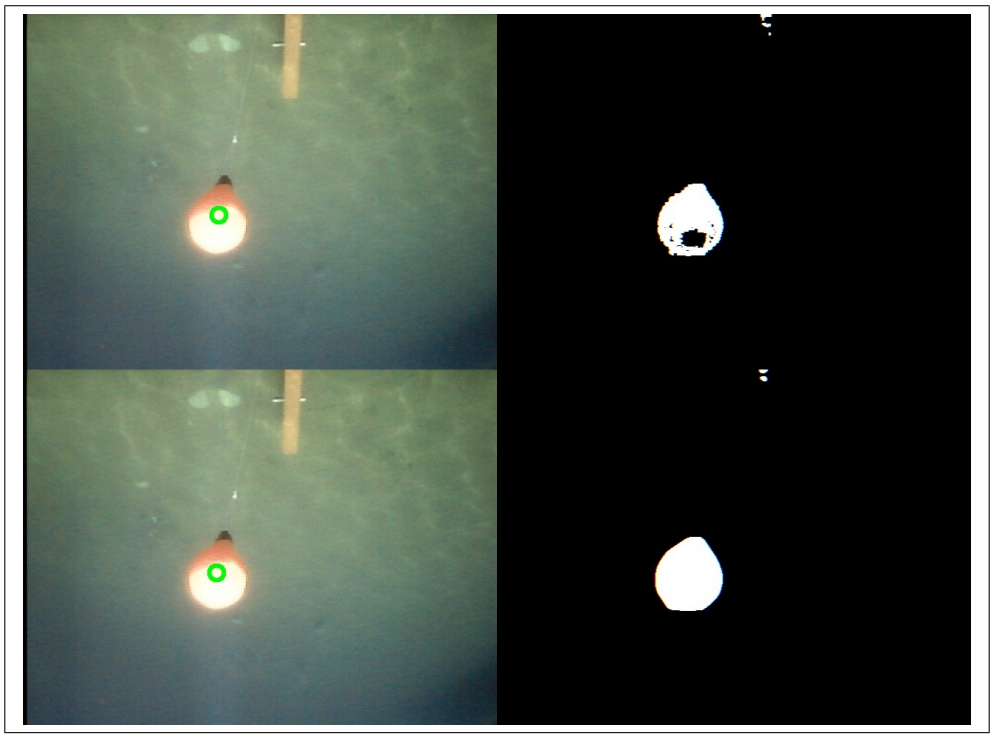

Figure 3.9: Example output from `buoy_v3` with closing.

The final technique I experimented with during the `buoy_v3` tests is called contour approximation. This algorithm is meant to use the information contained in a blob to estimate its actual shape. Similarly to closing, this is useful to create a full blob with a representative shape for better classification. There are two types of contour approximation algorithms with varying benefits and downsides. The first is known as polygon approximation, which gives a polygon with much fewer indices that closely matches the shape in the image. This is useful to have the same shape with a manageable footprint for analysis.

The other contour technique, called convex hull, creates a shape that con-

tains the actual object using only convex corners. This means that concave sections of the blob will be closed off, creating a larger image with convex defect information about those areas closed off. In the case of the buoy, the convex hull becomes useful when results have only a part of the buoy by closing off the shape into a rounded object. I use the convex hull algorithm based on the same reasoning that the classifier is strong and the goal is a solid buoy shape in the processed image. Figure 3.10 shows the output from `buoy_v3` without convex hull on top and with it on bottom. Notice that the bottom portion is closed off to create a solid shape. The closing algorithm would not be enough to create this full blob.

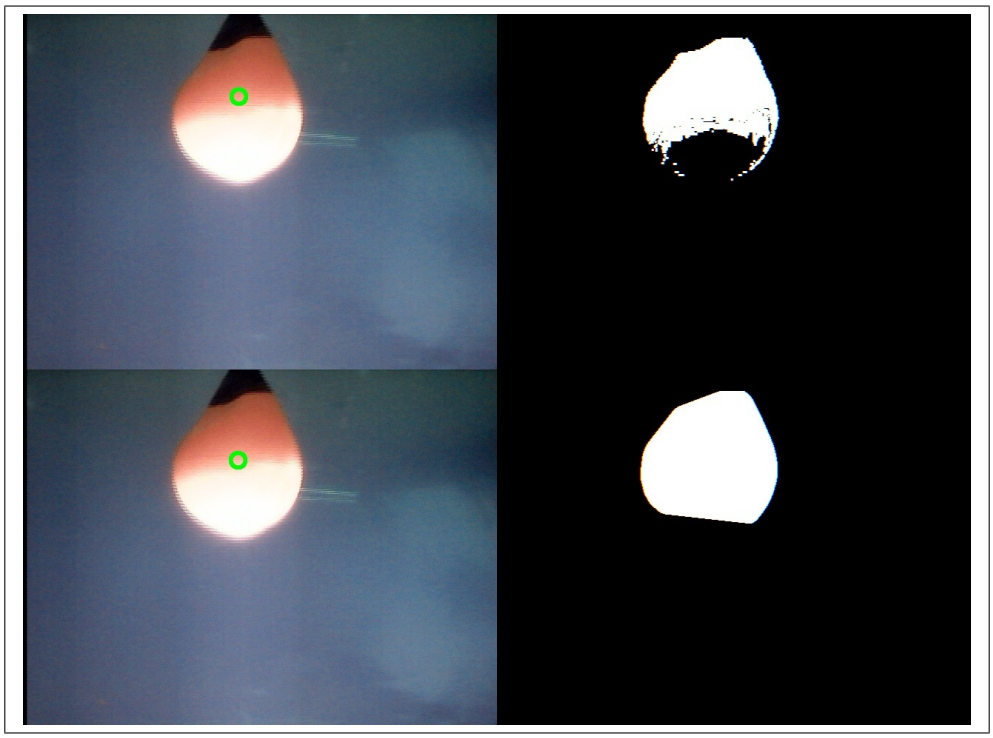

Figure 3.10: Example output from `buoy_v3` with convex hull.

Visually, these post processing improvements are very positive. Looking at the metrics shown in Table 3.3 we see that the results are mixed. The improvements in the shape and edges of the blobs allows for excellent center error results in both environments. However, these numbers are only calculated when the algorithm senses a buoy is present. The increase in false negatives for both environments means that the algorithm is completely missing the buoy in many images, which

helps bring the error average down. Also notice the large increase in time necessary to process an a frame with the inclusion of post processing. This still allows for five frames per second, which is acceptable for this experiment. Future iterations of the classification algorithm will need to reduce the false negatives while maintaining the quality of center estimation.

Table 3.3: Metrics via `buoy_v3` algorithm with post processing.

TRANSDEC		Pool	
False Positives	0	False Positives	8
False Negatives	15	False Negatives	57
Center Error	21.0287 ± 25.3694	Center Error	28.2274 ± 28.4738
X-Axis Error	5.2643 ± 6.4628	X-Axis Error	15.4545 ± 14.8622
Y-Axis Error	20.0571 ± 24.7799	Y-Axis Error	18.1818 ± 28.5874
Time Per Frame	169.103615 ms	Time Per Frame	168.808160 ms

3.3.3 Center Estimation

Once the algorithm has processed an image and output a binary image with some number of blobs, the next goal is to determine which blob is the most likely to be the buoy, and then find its center. In the baseline algorithm, the center estimation was simply the centroid of the binary image in terms of positive pixels. Figure 3.2 exemplifies this form of center estimation, since the green circle in the color image is estimated between the two blobs.

At this point, I thought about the best way to estimate the correct blob and therefore the correct center. Looking at images like Figure 3.2 show that shape is an obvious option for further classification. However, as a result of the quality of the HSV classifier, it is unnecessary to due such complicated analysis. It turns out that a high percentage of the time, the largest blob in terms of pixel count is the buoy. The only instances where this is not the case is when the buoy's reflection on the surface of the water is larger than the buoy itself in the image. At this juncture, this error is acceptable since in practice no reflection is expected due to

depth.

Using the algorithm `buoy_v3` with post processing, this new center estimation is shown in Figure 3.11. Notice that despite the small blob in the upper middle on the orange pipe, the algorithm places the green circle in the center of the buoy. Separately you may also note that the HSV classifier has ruled out much of the orange pipe as negative pixels.

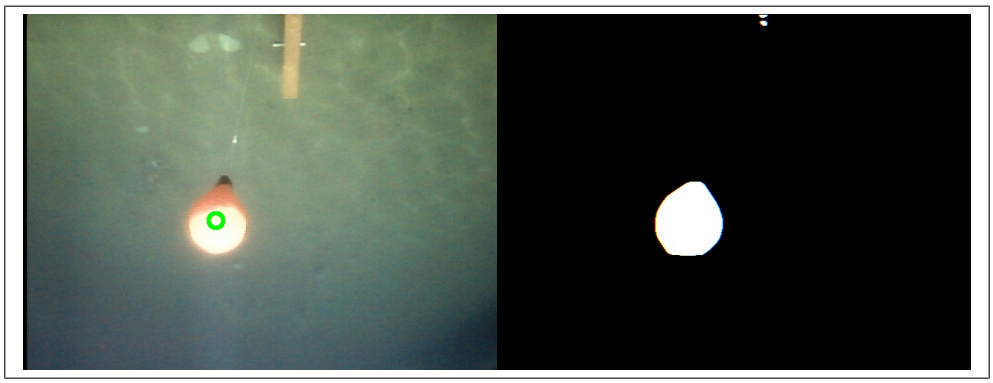

Figure 3.11: Example output using center estimation from area.

Table 3.4 shows the results when adding a more intelligent center estimation algorithm. The false positives and false negatives do not change for these results and the time per frame only increases slightly with the additional calculation of blob area. The important change for this experiment is the center error. Overall the center error is improved slightly for both environments, with the more substantial improvement in the pool. The interesting change is that the algorithm had opposite effects on the X and Y axis errors for the respective environments.

3.3.4 Edge Detection

This experiment is to determine the potential for edge detection to help with the detection and classification of the buoy. There are four different edge detection algorithms provided in OpenCV, which were each tested on a data set. The three that provided an output are shown through an example in Figure 3.12. The first algorithm is Sobel, which uses differentiation in either the x or y dimension. In

Table 3.4: Metrics via `buoy_v3` algorithm with center estimation.

TRANSDEC		Pool	
False Positives	0	False Positives	8
False Negatives	15	False Negatives	57
Center Error	17.2802 ± 27.3317	Center Error	25.5664 ± 47.7950
X-Axis Error	6.2643 ± 16.0965	X-Axis Error	6.3182 ± 4.4659
Y-Axis Error	15.4714 ± 22.5372	Y-Axis Error	24.2273 ± 47.8662
Time Per Frame	202.307322 ms	Time Per Frame	186.313001 ms

this example the Sobel operator is used to approximate a first derivative in the x-dimension and then in the y-dimension, with the results added to create a final image. The output shows a clear bright spot in the location of the buoy. This edge detection algorithm has potential use in underwater computer vision and which should be further investigated.

The next technique is called the Laplacian, which uses second derivatives in the mathematical Laplacian operator. This is very similar to the Sobel operator for second degree derivatives. The resulting image is very noisy and does not seem to work particularly well for underwater vision. Another technique, which is specifically known as an edge detection algorithm, is called Canny. This algorithm works similar to how my example uses the Sobel technique, combining first derivative outputs in the x and y dimension. The difference is that local maxima from the derivative outputs are considered when trying to assemble a contour to represent and edge. This means that the Canny technique has edges as goal in computation. The example shown in Figure 3.12 is very similar to the Sobel example, except with less noise and a clear contour edge. The Canny edge detector has the most potential for use in underwater computer vision.

3.3.5 Histogram Equalization

The intent of this experiment is to use histogram equalization to preprocess the images before boosting on the HSV. The thought is that the histogram equalization will create better contrast in the image and therefore make the buoy

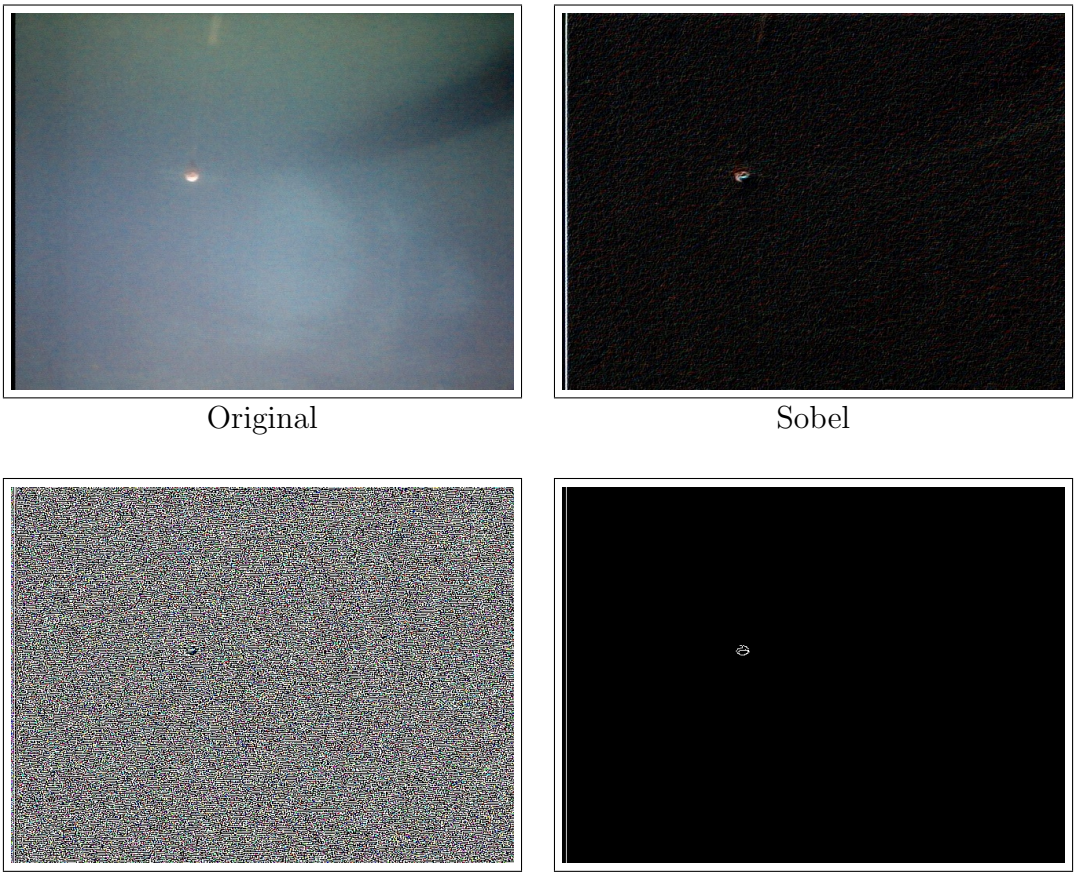

Figure 3.12: Edge detection algorithms on an example image.

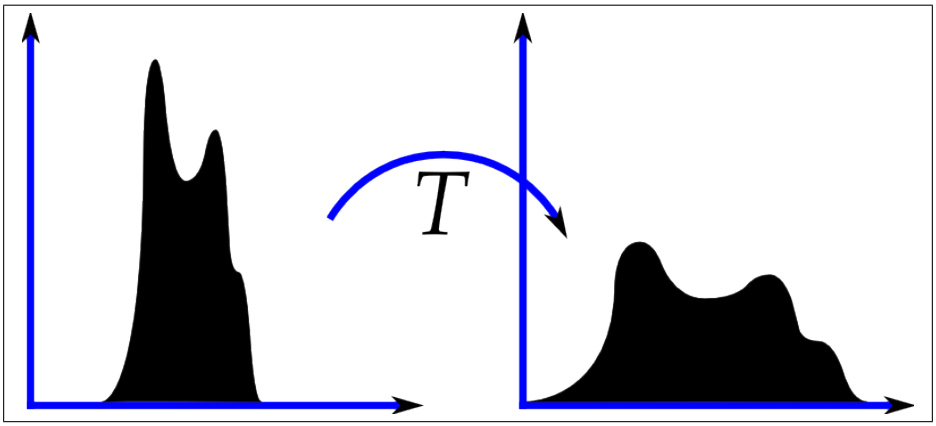

Figure 3.13: Histogram graphs before and after equalization.

more easily detectable. The concept of histogram equalization involves taking an images histogram, which is generally condensed to one portion of the full range, and spread the histogram across the range. Figure 3.13 visually shows how histogram equalization alters the histogram of an image. The implementation of this experiment is actually more complicated than simply calling a function in the OpenCV library. It turns out that the function provided in the OpenCV library is only capable of equalizing on a single channel. This means that each of the three channels needs to be equalized individually and then combined back together to create the equalized image.

After implementing the histogram equalization function for a three channel image, the experiment begins. The image set of equalized images is not noticeably different to the naked eye, which is not a good sign. This means that contrast did not improve as a result of the process, which likely means the histogram was

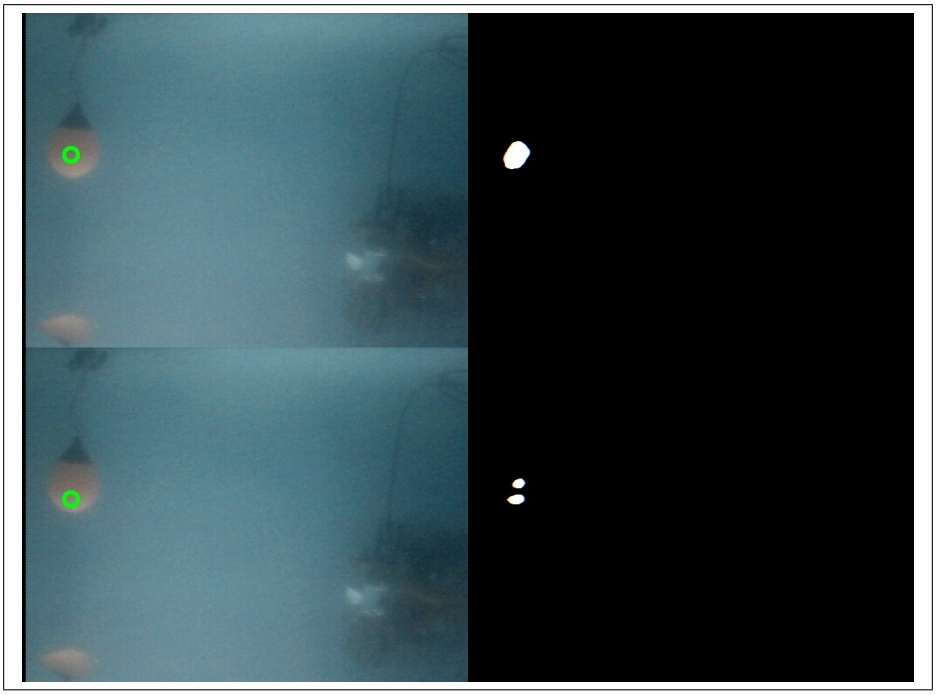

Figure 3.14: Example of negative impact of histogram equalization.

already well spread across the range. After boosting based on the equalized images the classifier created is called `buoy_v4`. Running the algorithm on the test images, it is clear that this process is not helpful for this set of images. Figure 3.14 shows

the comparison between buoy_v3 on top and buoy_v4 on bottom. Notice that the version with histogram equalization produces two blobs where as the original algorithm only produces one. This negative impact is a rare case, where most of the results are nearly the same.

3.3.6 Hybrid Boosting

The results from buoy_v3 is works well on the TRANSDEC images but not as well on the pool images and the experiments with edge detection and histogram equalization did not improve the results. The idea of hybrid boosting is to train on each environment separately and see if disparate decision trees can allow for better results. The labeled images for training in buoy_v3 are split into the two environments and handled individually.

First the TRANSDEC images are boosted creating the algorithm buoy_v5. The results are mixed, showing improvement in some cases, such as Figure 3.15, and showing a negative impact in other cases, such as in Figure 3.16. For each of these the top results are from buoy_v3, while the bottom is from buoy_v5.

Additional labeling to improve the results is necessary to make this algorithm for TRANSDEC optimal. There are two iterations of additional labeling, training, and boosting leading to buoy_v8. The intermediate algorithm buoy_v7 was not sufficient, and after some tests with the post processing, the next iteration began. The results at this point are excellent for this environment and the metrics are shown in Table 3.5. In addition, this algorithm was tested on the pool environment and did surprisingly well. There are less false negatives for TRANSDEC and less false positives for the pool. There are improvements in the center error as well as a speed increase over buoy_v3.

Next, the pool environment images are boosted to create algorithm buoy_v6. These results were substantially worse, with a lot of noise and errors. However, this is a misnomer, because the results show great potential for improvement over buoy_v3. Figure 3.17 shows a comparison between buoy_v3 on top and buoy_v6 on bottom. Notice that the buoy is found in the bottom binary, but that the excess noise drowns it out. This is a good sign, implying that with extra labeling, the

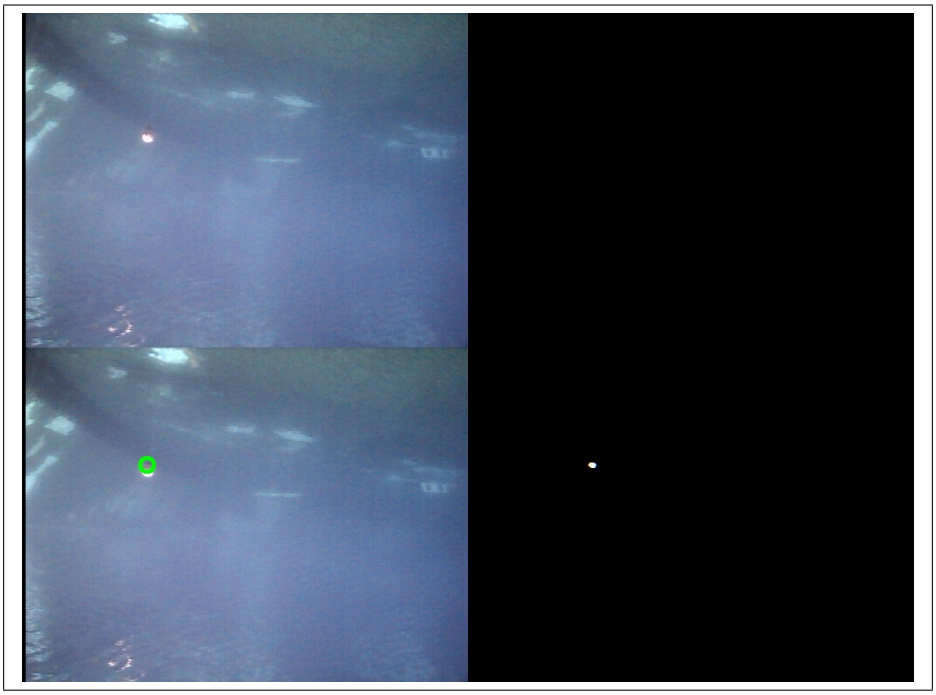

Figure 3.15: Example of positive impact via `buoy_v5`

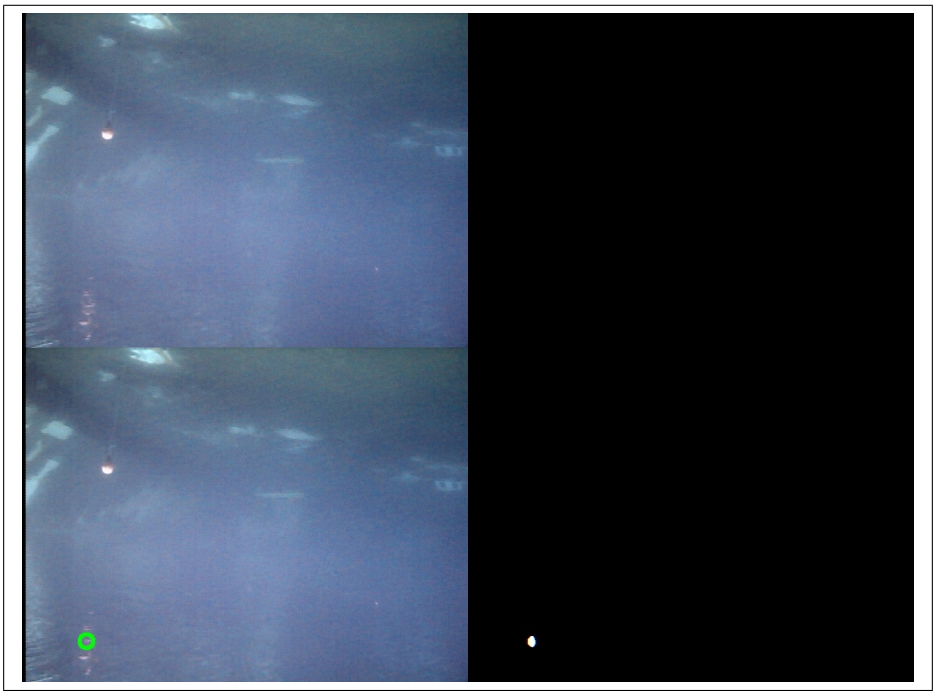

Figure 3.16: Example of negative impact via `buoy_v5`

Table 3.5: Metrics via `buoy_v8` algorithm.

TRANSDEC		Pool	
False Positives	0	False Positives	1
False Negatives	13	False Negatives	62
Center Error	15.1248 ± 23.5815	Center Error	26.5864 ± 53.6513
X-Axis Error	5.3311 ± 13.5135	X-Axis Error	4.1765 ± 4.2041
Y-Axis Error	13.5270 ± 19.7695	Y-Axis Error	25.9412 ± 53.6398
Time Per Frame	163.090370 ms	Time Per Frame	163.427548 ms

pool environment can make improvements.

Figure 3.17: Example of mixed impact via `buoy_v6`

Additional labeling through four iterations gives us algorithms `buoy_v9`, `buoy_v10`, `buoy_v11`, and `buoy_v14`. Each iteration shows improvement over the previous, narrowing in on the appropriate decision tree by removing noise and false negatives. Figure 3.18 shows the result via `buoy_v14` on the same image as in Figure 3.17. Notice that the buoy in the binary image is not as full as with `buoy_v6`, but that the noise is gone. The metrics for this final version of the algorithm is

shown in Table 3.6. Notice that the false negative and false positive numbers are much better for the pool, even though the average center error is worse.

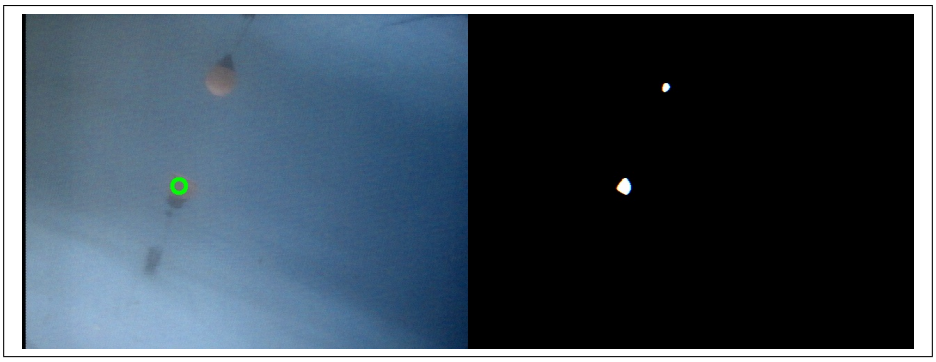

Figure 3.18: Example of improvement via `buoy_v14`

Table 3.6: Metrics via `buoy_v14` algorithm.

TRANSDEC		Pool	
False Positives	1	False Positives	2
False Negatives	8	False Negatives	32
Center Error	26.6376 ± 66.3759	Center Error	54.2160 ± 73.2491
X-Axis Error	6.2976 ± 21.6631	X-Axis Error	14.0851 ± 19.1665
Y-Axis Error	25.2738 ± 62.9889	Y-Axis Error	51.2766 ± 71.4827
Time Per Frame	158.809684 ms	Time Per Frame	160.468601 ms

3.3.7 RGB and HSV Boosting

This experiment tests whether boosting on both HSV and red, blue, green (RGB) classifiers per pixel increases the quality of the buoy classifier. This experiment involves an new function to create the training and testing files to include RGB with the HSV. Once the boosting files are created, JBoost is used to create a decision tree. This decision tree is then executed on the testing images to gauge the quality of the classifier. This experiment was done based on both of the environment specific boosting, with the algorithm for TRANSDEC called `buoy_v12` and the algorithm for the pool called `buoy_v13`.

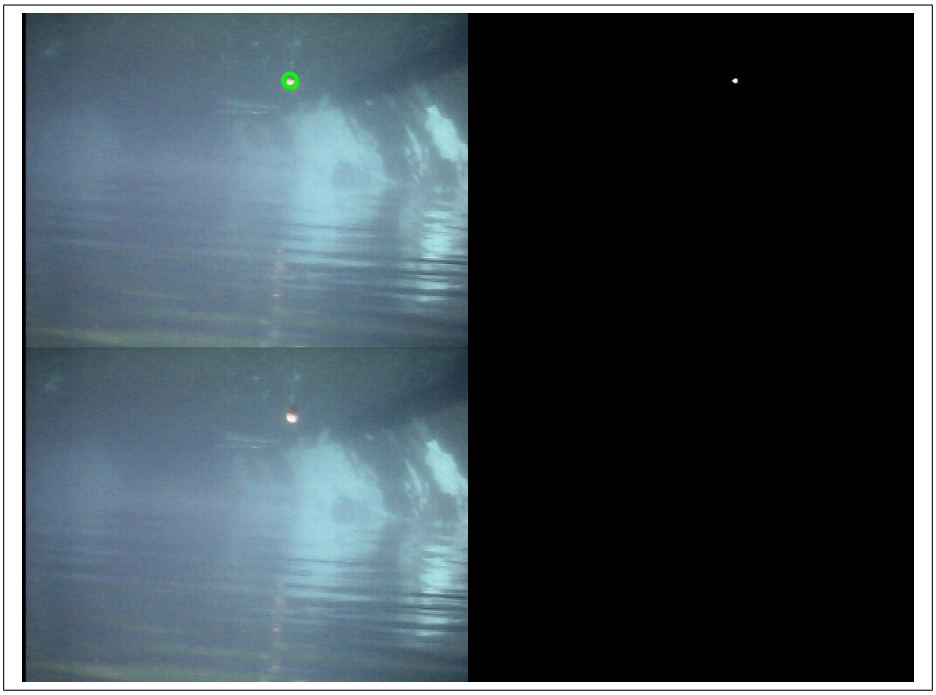

Figure 3.19: Example of negative impact of RGB and HSV classifiers.

For most of the test images, the result is very similar to the prior results, with a few exceptions. These few exceptions are cases where the result is worse than when using just HSV as a classifier. Figure 3.19 shows the comparison between `buoy_v8` on the top and `buoy_v12` on the bottom. Figure 3.20 shows the comparison between `buoy_v11` on the top and `buoy_v13` on the bottom. Notice in both cases that the new algorithm misses the buoy and therefore creates a false negative. This use of extra classifier information does not seem to be necessary, though it should not be thrown out forever. Though RGB and HSV are related by a formula, the machine learning of boosting could, in theory, create a better decision tree with this extra information. For now, this element will remain only an experiment and not part of the final algorithm.

3.3.8 Reflection Problem

The center estimation based on the area of the blob works most of the time due to the accuracy of the HSV classifier. However, there are some situations

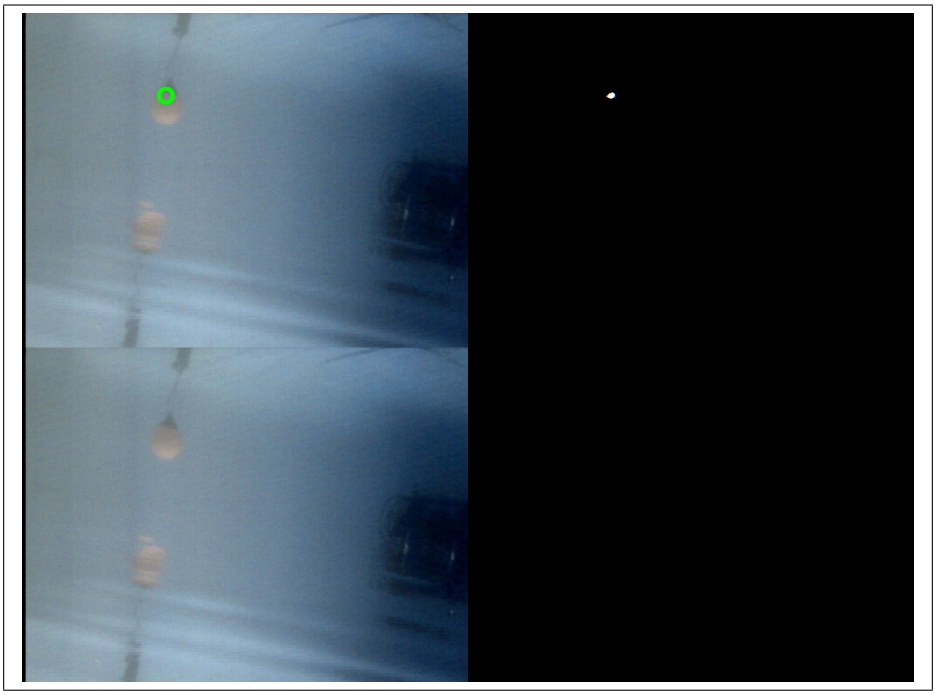

Figure 3.20: Pool example of negative impact of RGB and HSV classifiers.

where the largest blob is not the actual buoy but a reflection of the buoy on the surface of the water. Sometimes this reflection can becomes stretched and therefore larger. An improved center estimation algorithm compares the two largest blobs, and if the second largest is within a size range and above the largest, then the second largest is chosen instead. Figure 3.21 shows an example image comparing the center estimation algorithms. The top version simply chooses the largest blob, while the bottom version attempts to rectify the reflection problem.

3.3.9 Additive Boosting

The interesting part about the results from the hybrid boosting is that the environment specific algorithms worked relatively well on the alternate environment. This means that the two decision trees could be utilized additively to work well for both environments simultaneously. The next experiment is to use both decision trees on each pixel to determine if it is positive. This algorithm is `buoy_v15` and it is the combination of `buoy_v8` for the TRANSDEC and `buoy_v14` for the

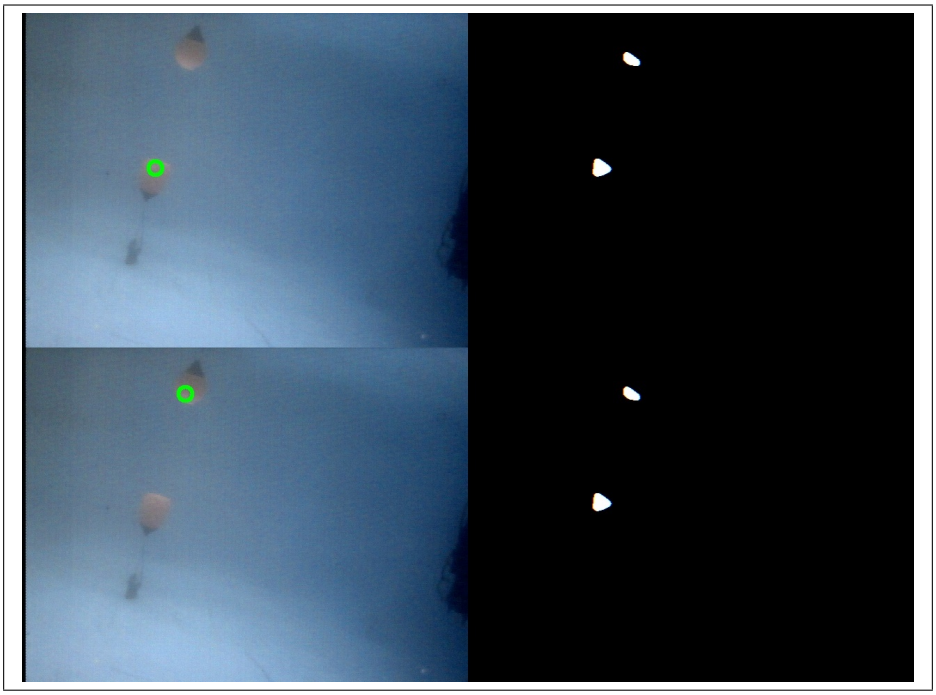

Figure 3.21: Example of the solution to reflection errors.

pool. The metrics are shown in Table 3.7. The false negatives are much better for TRANSDEC and the center error is quite good. The false negatives and false positives for the pool are much better than in `buoy_v3` and the center error is even a little better. The time per frame for both environments is also acceptable for this application.

Table 3.7: Metrics via `buoy_v15` algorithm.

TRANSDEC		Pool	
False Positives	1	False Positives	2
False Negatives	2	False Negatives	32
Center Error	20.8623 ± 61.8275	Center Error	24.4569 ± 52.5275
X-Axis Error	5.8125 ± 20.2303	X-Axis Error	7.7872 ± 13.3495
Y-Axis Error	19.6875 ± 58.5425	Y-Axis Error	21.9149 ± 51.3631
Time Per Frame	169.978417 ms	Time Per Frame	170.539260 ms

3.3.10 Final Algorithm

The final algorithm for buoy detection and classification is `buoy_v15`. This algorithm uses two decision trees, one trained on labeled images of the TRANSDEC environment and the other trained on labeled images from the pool environment. The first step of the algorithm is to use both decision trees to determine for each pixel whether it is positive or negative. This provides a binary image which can be appropriately post processed. The first post processing technique is the morphology called opening with one iteration, which is used to reduce noise. The next is the morphology called closing with two iterations, which is used to close the holes in blobs. Then the smoothing function is used to smooth the edges of the blobs before finding all the blobs as contours and using convex hull to recreate more full blob objects. The convex hull results are slightly smaller than before, so dilation of one iteration is used. Then the blobs are smoothed again, because convex hull sometimes creates harsh edges and corners. At this point the blobs are as good as they will get and the center estimation comes into play. The reflection technique is used, where the largest blob is chosen unless the second largest blob meets reflection criteria. The result is the coordinate estimation for the center of the buoy.

Chapter 4

Pipe Detection

4.1 The Pipe

The pipe is another one of the obstacles in the AUVSI competition of 2009 and is a slightly more difficult object for detection. The difficulty is increased because the pipe is meant to be used as a navigation tool, meaning when the pipe is found, it provides a direction of travel. Furthermore, there can be two pipes side by side suggesting two options for direction of travel. As can be seen in Figure 4.1, the pipe has distinct orange color, similar to the buoy, and a long rectangular shape. The data set of images of the pipe underwater consists of images in the TRANSDEC tank only. The goal is to develop an algorithm that can accurately detect this pipe and determine the heading of the pipe to be used for vehicle navigation.

4.2 Baseline

The very simple algorithm used during the AUVSI competition creates a binary image from the color image, just like with the buoy. Again, the algorithm checks if the HSV for a pixel are within a predetermined range, and if so, then the pixel is considered positive and is made white. If the HSV is outside of this range, then the pixel is considered negative and is made black. An example of this baseline algorithm processing an image is shown in Figure 3.2. This algorithm does

Figure 4.1: The orange pipe outside of water.

a good job for many of the images, but it struggles on others. Also, this algorithm attempts to determine the heading of the pipe, which is not always accurate. There is a threshold on number pixels before calculating the center and heading of the pipe, which is so large that many images are skipped. The blue circle and teal line in Figure 4.3 shows what the center and heading estimation look like on the original image.

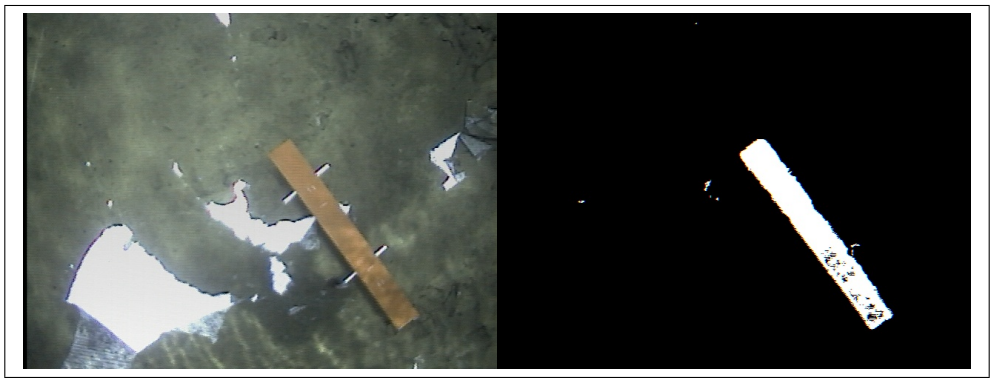

Figure 4.2: Baseline image of buoy and pipe.

Notice in this example that the pipe is not filled in and that there is some noise where the pipe is not. Also as seen in Figure 4.3, this algorithm is susceptible

to noise because of the large HSV range. In this example the blue circle in the color image represents the algorithms estimation of the center of the buoy, while the teal line represents the estimation of the bearing of the pipe. This simple algorithm places this blue circle at the centroid of white pixels over the entire image and uses a custom algorithm to estimate bearing.

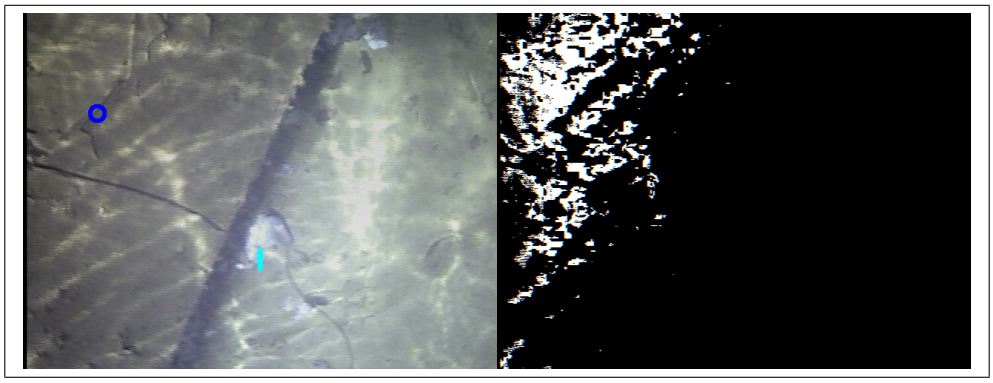

Figure 4.3: Baseline image with no buoy but noise causes a false positive.

In order to accurately gauge the quality of one algorithm versus another the 73 test images are only used for testing an algorithm. Of the 73 images only 16 of them do not have any pipe present, while 8 images have two pipes present. This means that 49 of them have one single pipe present. For all techniques to improve classification of the pipe, there is a separate training set of images. To calculate metrics on how an algorithm performs, the starting point is to label the test images with what is considered correct. Since this is prone to human error, there is a certain degree of noise and therefore no expectation of perfectly correct outputs. For each test image that is processed through the algorithm, the center estimation and bearing estimation for each pipe is logged to a text file. The metrics provided are the average center error in pixels, including overall, X-Axis, and Y-Axis, and the average bearing error in degrees. False positive and false negative results do not compare the center or bearing, but are presented in the metrics table.

Notice in the baseline metrics, shown in Table 4.1, that there are a high number of false negatives and that the bearing error is more than 9° with a standard

devation causing the error to sometimes exceed 23°. Also, it is important to notice how fast this simple algorithm is, only taking around 68 ms per frame which translates to close to 15 fps.

Table 4.1: Baseline metrics for pipe detection via AUVSI algorithm.

Pipe Test Images	
False Positives	4
False Negatives	14
Center Error	27.8029 ± 40.8258 pixels
X-Axis Error	17.2353 ± 33.0511 pixels
Y-Axis Error	17.6863 ± 27.1571 pixels
Bearing Error	9.0153 ± 14.5512 degrees
Time Per Frame	68.312311 ms

It is clear that this baseline algorithm leaves room for improvement, which is an opportunity to apply the lessons of the buoy detection algorithm, while experimenting with new computer vision techniques. The next section will follow the research and experiments as the algorithm is developed and improved.

4.3 Classification Techniques

This section describes different classification techniques and the experiments that either prove or disprove their usefulness for the improvement of the algorithm. Each technique considered involves one or more versions of the algorithm, whose results are compared to previous versions to determine the quality of the technique for improving classification of the pipe and determination of bearing.

4.3.1 Boosting HSV

Similar to the boosting approach for the buoy, this technique uses the unique color of the pipe via the HSV classifier to create a binary image with the pipe displayed in white and the background displayed in black. As described in Section

2.1, pixels are labeled as points or rectangles and defined to be positive or negative examples. Again, a script pulls the HSV values for the labeled pixels and creates `pipe.test` and `pipe.train` files and a `pipe.spec` file describes the format of the other files. The files are all input into the JBoost application, which uses machine learning to generate a decision tree that quantifies a confidence level of positivity for a given pixel's HSV information. The decision tree is encapsulated in software in `pipe.c`, which is used in the Stingray vision code to predict whether a given pixel is positive or negative. When every pixel is considered, a binary image results.

It takes three versions of the boosting, each with an increased number of labeled pixels, in order to converge on an optimum decision tree for this classifier. Each time more labeling occurs, the labeled spots are where errors the previous version was wrong in its output. Generally where it made pixels positive when they should have been negative. This extra labeling improves the quality of the algorithm. The images shown in Figures 4.4 - 4.6 show that the extra labeling removes noise by improving the accuracy of the decision tree.

In this particular case, the improvements in `pipe_v2` came with some unwanted negative side effects. It turned out that some of the extra labeling caused a deterioration in the quality of actual pipes. So for when relabeling for `pipe_v3` some of the trouble spots were ignored for images that contained abnormal angles for normal operation. The result is that the noise is still reduced as shown in Figure 4.6, but the side effects are fixed, as shown in Figures 4.7 - 4.9. This lesson shows that more labeling is not always best and that where you label in an image is important.

The algorithm `pipe_v3` is the final iteration of the changes to the boosting portion of the algorithm. The resulting binary images capture the location of the pipe with impeccable precision. The results are much better than with the buoy, most likely due to the maximum distance from the pipe provided by the relatively small depth of the pool. No metrics were taken at this juncture, since the goal is to find the center and the bearing of the pipe and compare this to the original algorithm. This stage could have been tested using the new algorithm to create the binary image and the old algorithm for finding the heading from a binary image,

yet this step was skipped for the to consider a more advanced technique for finding the heading.

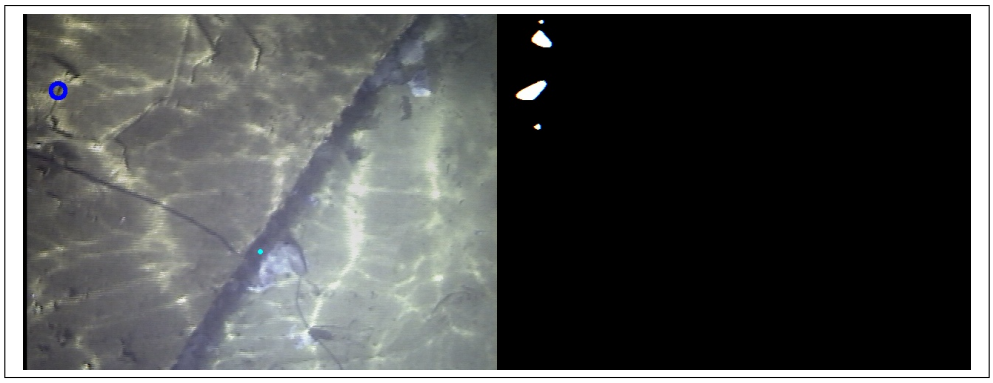

Figure 4.4: Noise example output from `pipe_v1`.

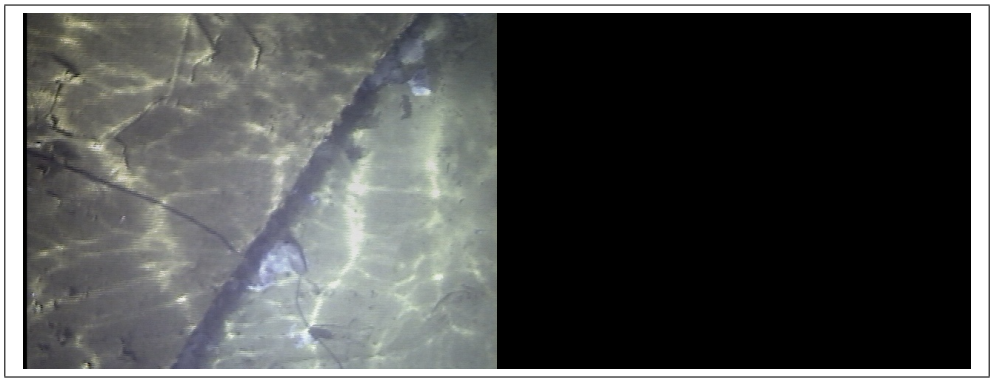

Figure 4.5: Noise example output from `pipe_v2`.

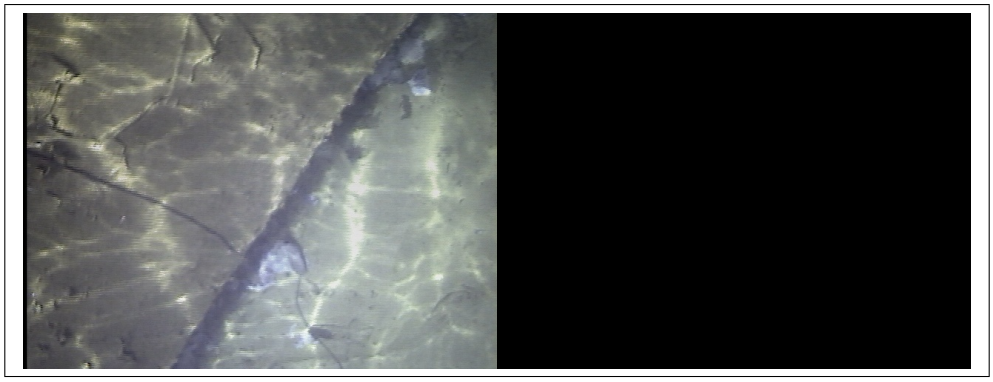

Figure 4.6: Noise example output from `pipe_v3`.

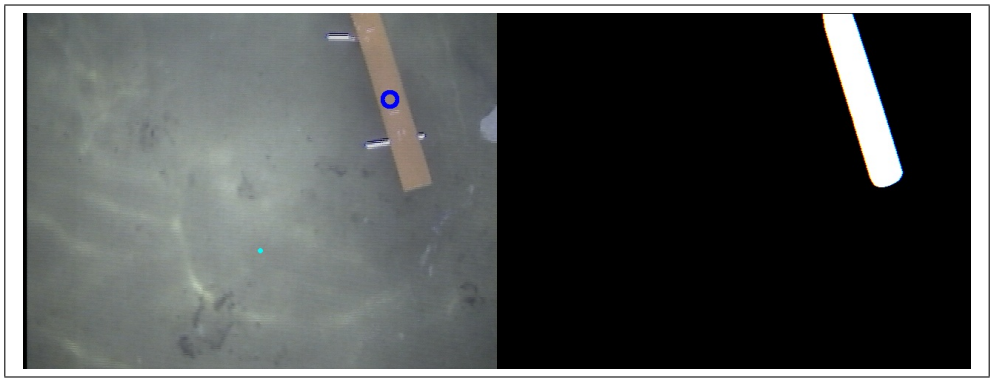

Figure 4.7: Side effect example output from `pipe_v1`.

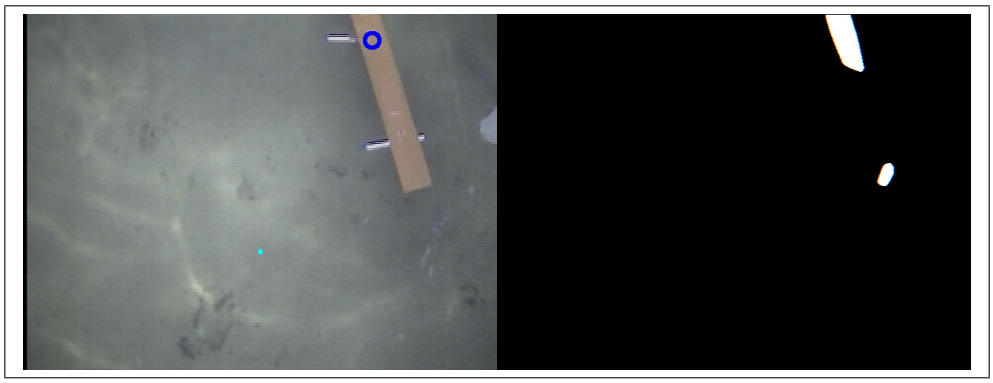

Figure 4.8: Side effect example output from `pipe_v2`.

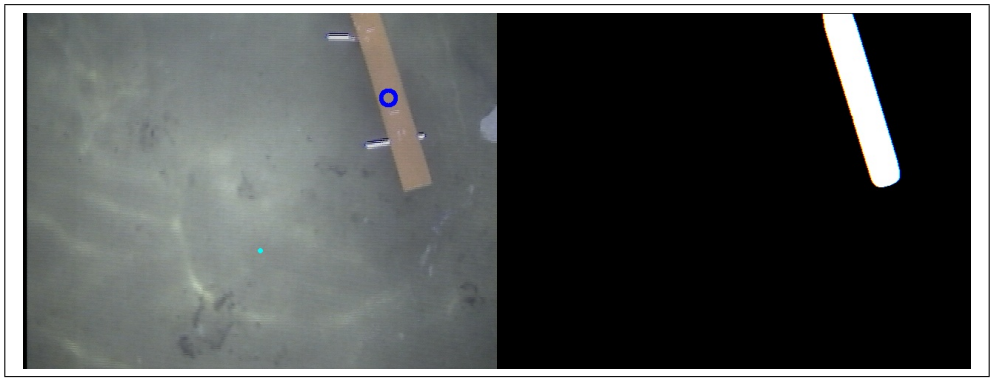

Figure 4.9: Side effect example output from `pipe_v3`.

4.3.2 Post Processing and Filters

The quality post processing techniques that were verified during the buoy detection stage were implemented starting with `pipe_v1`. The first post processing

is the smoothing of shapes created from the decision tree. Next is opening, which consists of a single erosion followed by a single dilation. This is done to remove extra noise in the image. Following is the closing method, which in this case consists of two dilations followed by two erosions. This is done to fill holes in shapes. An additional method to close of shapes is called the convex hull which fills the remaining holes in connected components of the binary image. Examples of all of these post processing techniques can be found in Section 3.3.2

4.3.3 Edge Detection

The overall goal of the pipe detection is to determine the heading of the pipe for navigation purposes. The next step towards this goal following the binary image creation is to determine the edge of the blob. The first attempt uses the blob object to find its perimeter. Visually this works correctly, however the result is not in the proper format for the Hough Transform, which is the next step. The result is that the Hough Transform algorithm fails when using the perimeter of the blob. The solution is to use the Canny edge detector on the blob. The result is identical to the perimeter of the blob, except that it is in the correct format. The similarities between perimeter and canny edge can be seen in Figure 4.10. The top is the perimeter of the blob and the bottom is the canny edge detection output.

4.3.4 Hough Transform

The next two versions of the algorithm add a step utilizing the Hough Transform algorithm, which is designed to find straight lines in an image. The inputs to this algorithm are important in deciding how conservative or liberal the algorithm should be in its search for straight lines. The first two inputs to the algorithm are Rho and Theta, which represent the granularity of pixels and the angle for scanning the image. With a low Rho and Theta, the search will consider more possibilities in its search, finding more of the lines present, but taking more processing. The next value is Threshold, which is how long in pixels a straight line needs to be in order to return as a line. In version `pipe_v6` the standard version of

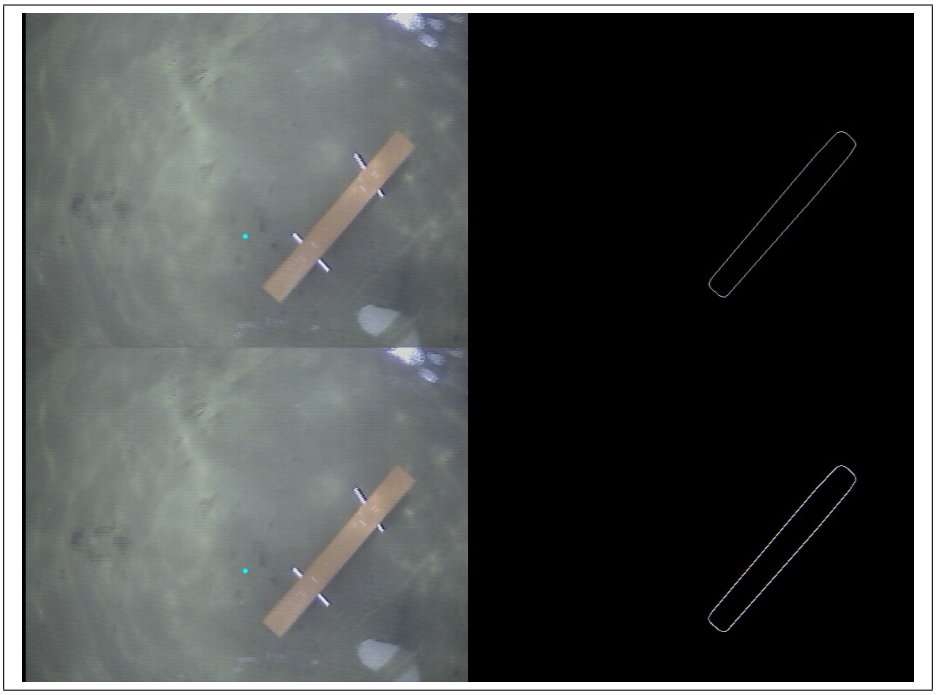

Figure 4.10: Perimeter of the blob compared to Canny edge detection.

the Hough Transform (SHT) is used with different inputs, attempting to narrow in on the optimal version. The inputs decided upon for `pipe_v6` are a Rho of 1 pixel, a Theta of $\frac{\pi}{30}$ radians or 6 degrees, and a Threshold of 30 pixels. An example of the results is shown in Figure 4.11. The two characteristics to note about these results are the multiple lines per edge of the pipe, due to the estimated edge not being perfect, and the fact that the lines extend infinitely.

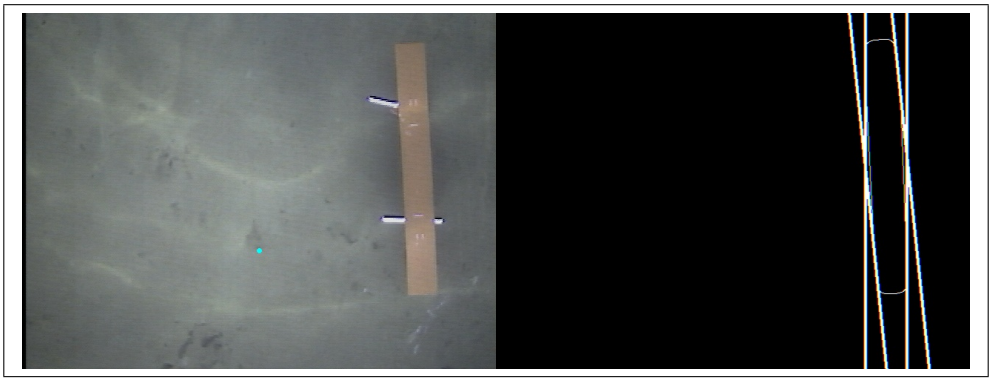

Figure 4.11: Standard Hough Transform example results.

In the next stage of the algorithm, `pipe_v7` the Probabilistic Hough Transform (PHT) is used for its ability to find the line segment in an image. Again, experimentation is used to determine what inputs to provide. The Rho remains at 1 pixel, while the Theta is reduced to $\frac{\pi}{120}$ radians or 1.5 degrees, and the Threshold remains at 30 pixels. The probabilistic version of the algorithm has two extra inputs for length of line segment and gap in pixels between line segments for merging into one result, for which we use 20 pixels for both. The results are very similar, but in the form of line segments. This allows more flexibility for our next stage, which is dubbed Hough Pruning and involves merging semi-collinear lines and removing lines without parallel matches. The output from PHT on the same image as shown for the SHT is shown in Figure 4.12. It is hard to see but there are actually two line segments in bold on the left of the pipe and two on the right of the pipe.

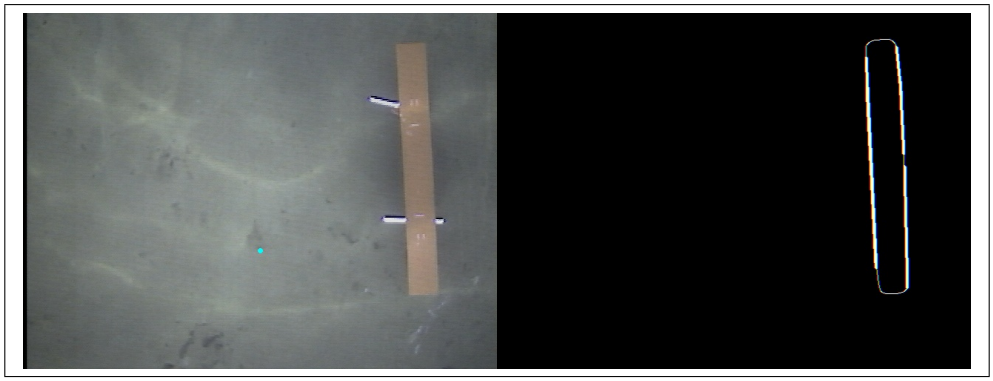

Figure 4.12: Probabilistic Hough Transform example results.

4.3.5 Hough Pruning

The concept of Hough Pruning is the most complicated and math intensive part of this algorithm. It consists of two steps, the first of which is merging all line segments that are close to being collinear. The second step is search for parallelism between lines, since the pipe will have parallel sides. These two steps together prune the results down to the lines that are most likely to be the sides of the pipe.

The fist step of merging semi-collinear lines is experimented with in `pipe_v8`. This part of the algorithm considers pairs of line segments by running a Best Fit Line algorithm on the four endpoints of the two lines and then calculating the average error in distance for each point from the best fit line. If this average error is within a threshold, in our case 3.0 pixels, then the line segments are merged. Merging of line segments involves finding the best fit line for all semi-collinear lines and using the best fit line as the merged line. The result of this stage is the minimum number of collinear line segments in vector form. The same example image from before the Hough Pruning is shown in Figure 4.13 with the semi-collinear algorithm. Also, note that the estimated heading is shown as a blue line out from the blue center of the pipe in the color image. In this example, the semi-collinear step is enough to provide useful results. We will see that there are some examples that need further pruning.

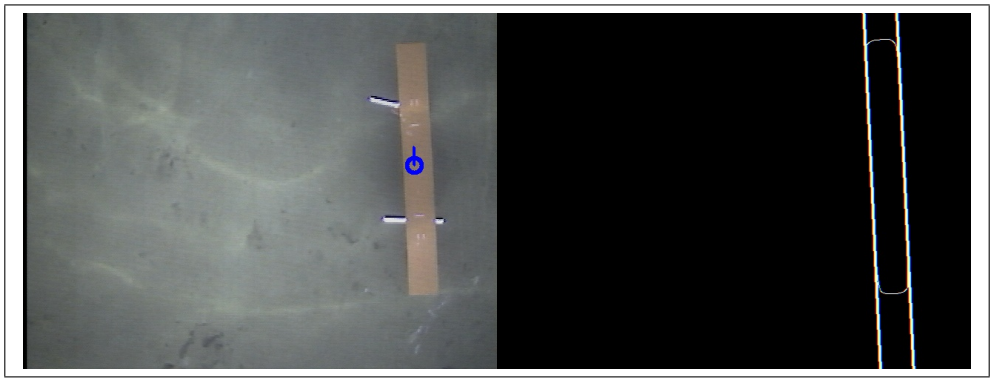

Figure 4.13: The bearing results with semi-collinear Hough Pruning.

The next step of using parallelism to further prune the results is done in `pipe_v9`. In some cases the results will have conflicting parallel lines or a line without a matching parallel line. The first part of this step is converting all vector lines into degrees and making sure that the degrees are between 0 and 179, which means pointing up. This allows us to compare lines in terms of degrees in order to determine the level of parallelism for pairs of lines. If the lines are within ten degrees of one another, then they are considered semi-parallel. Any line that does not have a parallel match is pruned from the list of lines. An example of this result is shown in Figure 4.14. This image is from the front camera for the sake of more

challenging images, so it is unlikely to see the buoy in the bottom camera. However, notice that the bottom version removes the extraneous line from consideration and fixes the resulting bearing estimation.

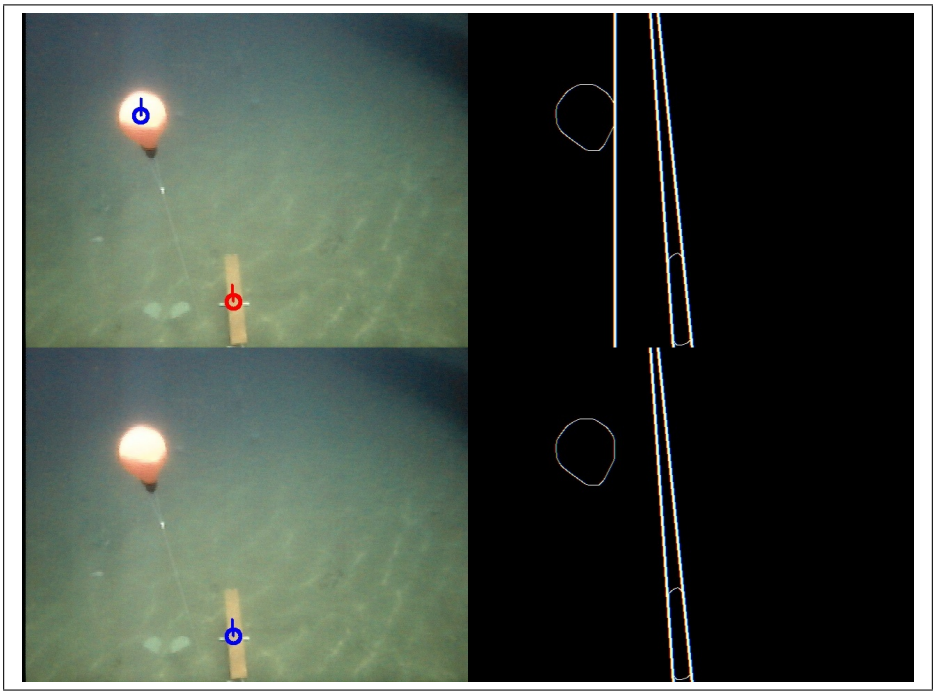

Figure 4.14: Example of removing an extraneous line based on parallel pairs.

Whenever two lines are considered parallel, there are a few follow up steps to determine if this pair should remain. The first step is to see if the semi-parallel lines surround the center of a pipe. This means that one line is to the left or below the center and one line is to the right or above the center. If the pair does not surround a center of a pipe, then the pair is removed. There are also two possible special cases to consider. First there is the scenario of two pipes, where a set of semi-parallel lines could pertain to either centroid. So any pair of semi-parallel lines is matched to the centroid to which they are closest in proximity. An example of finding two pipes is shown in Figure 4.15. The other possible mistake is when their are multiple pairs of semi-parallel lines that surround the same pipe center. In this case, more than likely, all four sides of the pipe have been found and therefore the pair that represents the long sides of the pipe will be closest to the center. This means that the best pair for a centroid is considered correct and

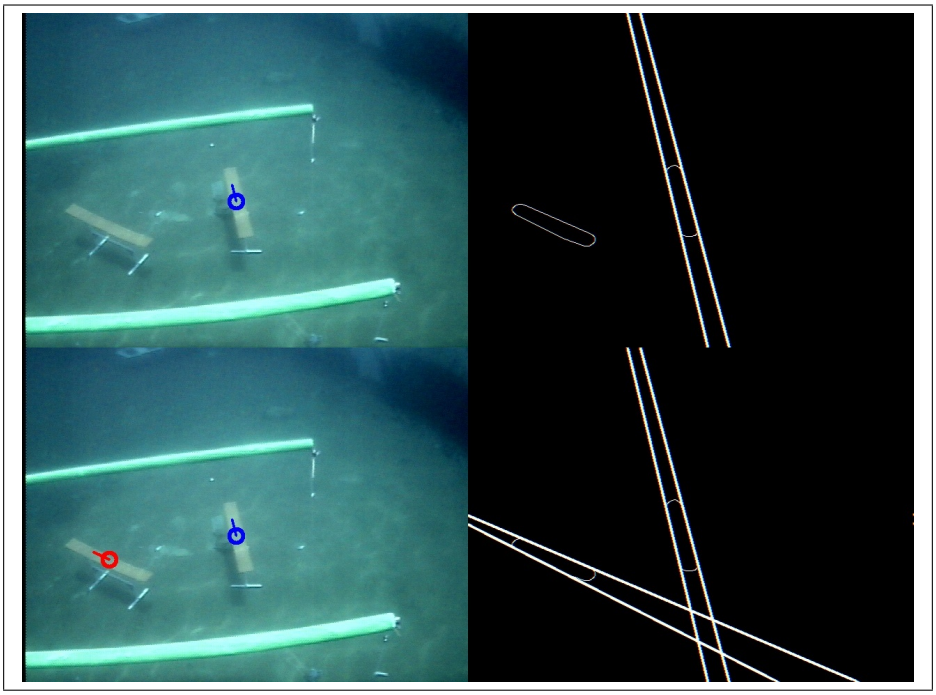

Figure 4.15: Two pipes found and matched to the respective pipe centers.

the other pairs are removed. An example of this scenario is shown in Figure 4.16. The top one is from the previous version and the bearing is incorrect due to the extra valid pair of semi-parallel lines, while the bottom one removes the incorrect lines and estimates the correct bearing.

4.3.6 Final Algorithm

The final algorithm is a tight consideration between `pipe_v9` and `pipe_v10`, which are only slightly different in terms of the threshold for finding lines using the Probabilistic Hough Transform. The threshold for `pipe_v9` is 30 pixels, while the threshold for `pipe_v10` is 20 pixels. This means that `pipe_v10` is slightly more liberal and has a tendency to find more line segments. The metrics for these two versions are very important in determining which is our final algorithm for the pipe. The tables of metrics are shown in Tables 4.2 and 4.3 respectively. First, let's compare the results from these two versions to each other before comparing them to our baseline metrics. All of the error results for the center and bearing

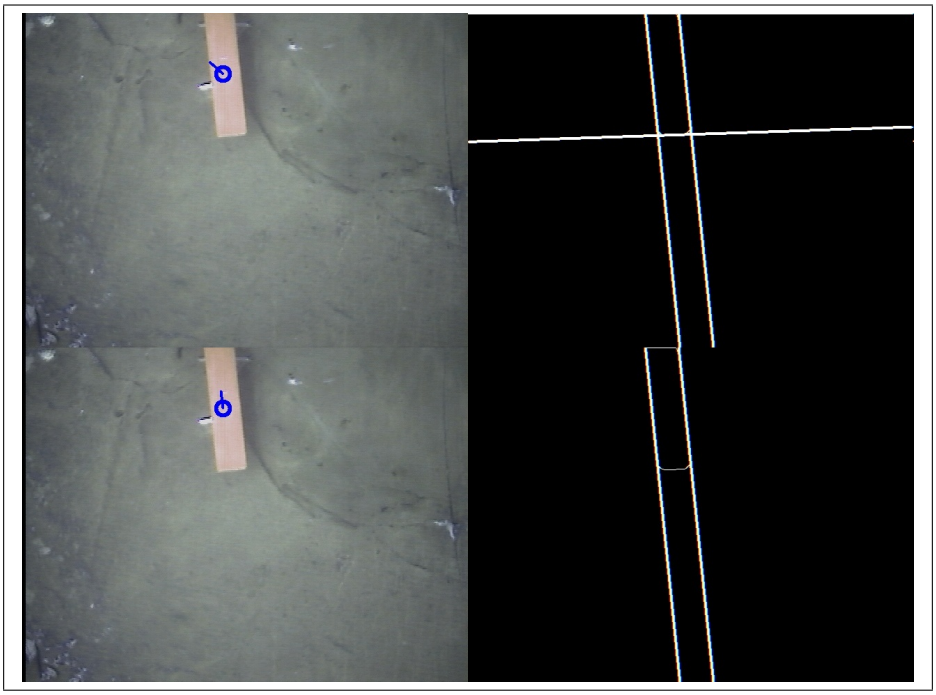

Figure 4.16: Conflicting parallel lines are pruned to find the correct bearing.

of the pipe are so similar that the difference is negligible. Similarly, the change in Time Per Frame is not substantial enough to be necessarily a result of the change. This makes sense, since the portion of the algorithm that determines center and bearing has not changed. The part that changed is how many line segments are found by the PHT algorithm. The important difference, therefore, is that `pipe_v10` has only four False Negatives, while `pipe_v4` has six False Negatives. The point is that the more liberal PHT algorithm helps to find more of the pipe edges and the corresponding bearing.

Now we should compare these metrics to that of the baseline AUVSI algorithm shown in Table 4.1. The False Positive and Negative rates are substantially better in `pipe_v10`, compared to the four False Positives and 14 False Negatives from the baseline algorithm. Also, the center error has improved substantially from the 28 pixel error in the baseline algorithm. The most important improvement however is in the estimation of the bearing. The baseline algorithm had a horrible 9° of error with a standard deviation of 14.5°. The Bearing Error in `pipe_v10` is so low that it is hard to say if that error is the fault of the algorithm or

Table 4.2: Metrics for pipe detection version 9 algorithm.

Pipe Test Images	
False Positives	0
False Negatives	6
Center Error	16.0584 ± 36.0307 pixels
X-Axis Error	10.5254 ± 35.2321 pixels
Y-Axis Error	10.2034 ± 9.9945 pixels
Bearing Error	0.5927 ± 0.8426 degrees
Time Per Frame	196.280539 ms

Table 4.3: Metrics for pipe detection version 10 algorithm.

Pipe Test Images	
False Positives	0
False Negatives	4
Center Error	15.7518 ± 35.4742 pixels
X-Axis Error	10.3279 ± 34.6674 pixels
Y-Axis Error	10.0164 ± 9.8854 pixels
Bearing Error	0.6584 ± 0.8374 degrees
Time Per Frame	197.113293 ms

the labeling of the edges of the pipe for the correct heading. Finally, the Time Per Frame has increase quite a lot from the 68 ms in the baseline algorithm. However this increase in time is acceptable for the much improved results, especially since this still provides better than 5 fps.

The final algorithm uses one decision tree created from JBoost using labeled pixel examples to create a binary image with the pipe blobs in white. The post processing techniques that were proven in the Buoy Detection experiments are used to clean the binary image. Next the Canny edge detector finds the edges of all pipe blobs in the binary image. Then the Probabilistic Hough Transform searches for line segments among the edges, return many line segments per pipe outline. The Hough Pruning algorithm merges semi-collinear lines and removes lines without parallel pairs and pairs that do not surround a center of a pipe.

Once all extraneous lines are removed and the parallel pairs have been matched to the one or two pipe centers, the average bearing is calculated and returned. The result is the coordinate estimation of the center of the pipe and the bearing, always point forward, that the pipe is pointing.

Chapter 5

Future Work

5.1 Overview

The concepts used this research for creating robust classifiers to be used by algorithms for object detection and classification in underwater imagery has many applications. The possibilities for continued research with regard to underwater applications is the focus of this section, though it is important to note that the process outlined in this research could be applied to imagery from any autonomous vehicle or static sensor. Furthermore, the concepts could be applied to imaging sonars and used in parallel to improve detection in classification of more complex objects in less ideal environments.

5.2 Fish Detection

The organization Reef Check attempts to monitor the status of our oceans reefs in terms on growth of coral and number of various species of fish present in the reef. Currently, this monitoring is done by volunteer divers, who dive to the reef and take measurements and visually count fish. This is not only time consuming, but requires a large number of qualified and certified divers who are willing to volunteer. The goal would be to automate this monitoring into an AUV with algorithms designed for fish detection and classification as well as saving image data. The vehicle would be able to detect and classify the different species

of fish present and count the number present. Figure 5.1 shows an example of the output from a detection algorithm using haar classifiers to look for a Skythe Butterfly fish. There would be different classifiers for different fish and in theory

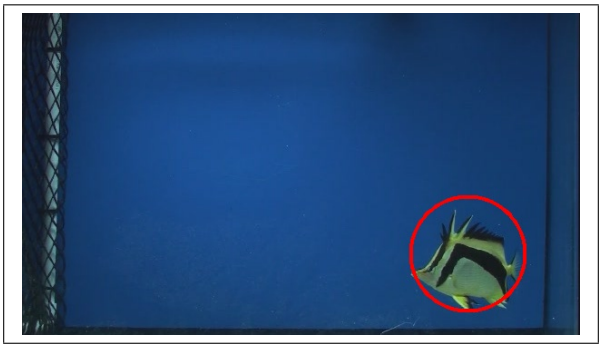

Figure 5.1: Scythe Butterfly detection using HAAR classifier

the vehicle could be deployed from shore and sent to a certain location, or multiple locations, in order to count fish. A working system of this type would save marine scientist time and money when studying underwater environments such as reefs.

5.3 Quagga Mussels

The Quagga Mussel is a very small freshwater animal that has become a nuisance in many places in the United States. Their numbers grow quickly until they are covering the beaches and filling every pipe available. Figure 5.2 shows a close up of the Quagga Mussel shells, a beach that is entirely covered by the shells, and a pipe that is split open revealing the mussels. This is especially a problem

Figure 5.2: Quagga mussels covering a beach and clogging a pipe.

in the reservoirs that are our public water sources. In California, these pests have only just started to appear, so there is time to catch them before they can build up. The problem we are facing is that humans are not permitted to enter our reservoirs, so the quick solution of diver investigation is off limits. The goal is to use computer vision techniques on optical and acoustic sensors to detect and classify the mussels in reservoirs. This would allow us to determine the precise locations where infestation may become an issue so that the proper precautions can be taken.

5.4 Naval Mines

The military still runs searches for mines attached to the hulls of ships or anchored in harbors. There are multiple methods, including the use of mammals and remote operated vehicles (ROVs) or AUVs with limited capabilities. In many instances the mammals are excellent mine detectors, but their are high costs associated with training and caring for a marine mammal. The problems with the vehicles currently is the high cost in time to run a mission. The goal of this research would be to develop algorithms to systematically search for mines using various object detection and classification algorithms and both optical and acoustic sensors.

5.4.1 Hull Inspection

The hull inspection area of this research involves ships that are in harbor and therefore in a contained environment. The vehicle can relatively easily do a simple pass-by flight around the entire hull of the ship looking for limpet mines such as the one shown in Figure 5.3. During the this pass-by flight a high-level search algorithm would look for classifiers implying the likelihood of a mine. If an area of interest is found a secondary search mode look closer at an area with different sensors and algorithms to classify if the object is in fact a mine.

Figure 5.3: Example of a limpet mine that would attach to a hull.

5.4.2 Anchored Mines

Anchored mines, such as the one shown in Figure 5.4 are still present out at see and searching contained areas is a common task. This research would be similar to the hull inspection, except with a more open preliminary search algorithm in open water. This algorithm would look for classifiers suggesting locations as areas of interest needing further investigation. Again secondary searches could take a better look at an area of interest. The challenge here, outside the scope of the vision research, is vehicle self-localization in order to systematically search out an area.

Figure 5.4: Example of an anchored mine in open ocean.

Bibliography

[BTL+97] B.A.A.P. Balasuriya, M. Takai, W.C. Lam, T. Ura, and Y. Kuroda. Vision based autonomous underwater vehicle navigation: underwater-cable tracking. *MTS/IEEE Conference Proceedings*, 1997.

[DGHW98] T. Darrell, G. Gordon, M. Harville, and J. Woodfill. Integrated Person Tracking Using Stereo, Color, and Pattern Detection. *In Proceedings of IEEE Computer Vision and Pattern Recognition*, 1998.

[FM99] Yoav Freund and Llew Mason. The Alternating Decision Tree Learning Algorithm. *In Machine Learning: Proceedings of the Sixteenth International Conference*, 1999.

[FS99] Yoav Freund and Robert Schapire. A Short Introduction to Boosing. *Journal of Japanese Society for Artificial Intelligence*, 1999.

[GBCA01] R. Garcia, J. Batlle, X. Cufi, and J. Amat. Positioning an underwater vehicle through image mosaicking. *Proceedings IEEE International Conference on Robotics and Automation*, 2001.

[LBS+98] John J. Leonard, Andrew A. Bennett, Christopher M. Smith, Hans Jacob, and S. Feder. Autonomous Underwater Vehicle Navigation. *MIT Marine Robotics Laboratory Technical Memorandum*, 1998.

[VJ01] Paul Viola and Michael Jones. Robust Real-time Object Detection. *International Workshop on Statistical and Computational Theories of Vision - Modeling, Learning, Computing, and Sampling*, 2001.