Perception and Motion Planning for Autonomous Surface Vehicles in Aquaculture

by

Jerry Zhang

S.B. Computer Science and Engineering, Massachusets Institute of Technology (2022)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2022

Author . . . .
Department of Electrical Engineering and Computer Science
May 6, 2022

Certified by. . . . .
John J. Leonard
Professor
Thesis Supervisor

Certified by. . . . .
Andrew Bennett
Lecturer
Thesis Supervisor

Accepted by . . . .
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Perception and Motion Planning for Autonomous Surface Vehicles in Aquaculture

by

Jerry Zhang

Submitted to the Department of Electrical Engineering and Computer Science
on May 6, 2022, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The "Oystermaran" USV was designed and developed by students at MIT SeaGrant to resolve the oyster basket flipping bottleneck that slows down oyster farming at Ward Aquafarms. The state of the USV requires remote operation within close distance of the vessel. In this thesis, we present an automated solution that will enable the Oystermaran to autonomously depart from its parked location, navigate to its destination, execute the flipping tasks, and return to a designated location with little to no human intervention. The details explored in this project and discussed in the thesis focus on the perception and motion planning aspects of the proposed autonomous system. Our results show a capable basket detection algorithm based on our collected dataset. The system's path planning approach is also proven sufficient in simulation. Additional data collection with further testing may be required to fully realize the system on board the Oystermaran.

Thesis Supervisor: John J. Leonard
Title: Professor

Thesis Supervisor: Andrew Bennett
Title: Lecturer

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

1.1.1 History of Oyster Farming

Dating back to the early 1700s, oyster farming in New England was initially to harvest limestone from the oysters' shells [7]. Now, it is part of the growing trend of aquaculture's contribution to seafood production, shown in Figure 1-1. The growth of aquaculture is made in part due to the increase in demand for seafood, but also the efficiency of new technology for aquaculture.

Facilities such as Ward Aquafarm LLC are vertically integrated farms that produces multiple different seafood varieties while also acting as a center for research and development in improving aquaculture [19]. Founded by MIT Alum Dr. Dan Ward in 2012, the farm has expanded from an initial 2.6 acres to more than 10 acres, as seen in 1-2. A major contributor to Ward's growth is the farm's willingness to experiment with new technologies and a commitment to engaging with the scientific and aquaculture communities to make the farming process more efficient. The details regarding aquaculture and oyster farming will be in the context of this particular farm. Discussion of techniques and systems will be first developed for the specifics with respect to Ward, but attempts to generalize will be discussed.

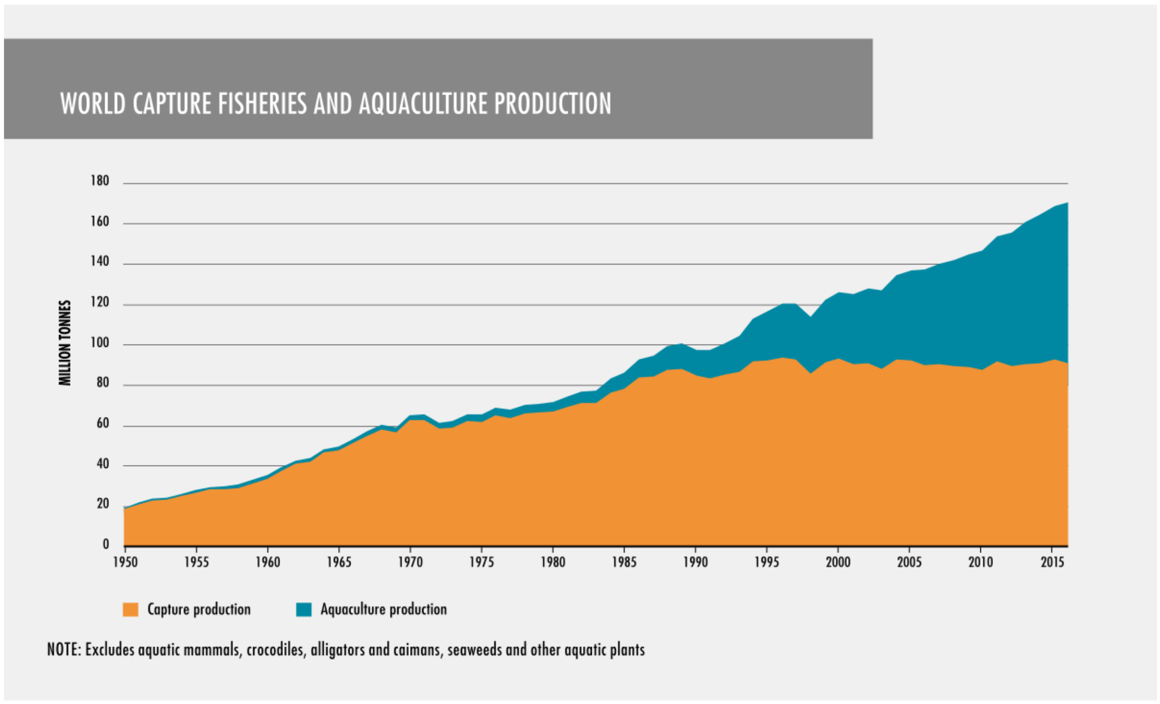


Figure 1-1: A graph showing the production of seafood from both capturing and aquaculture between 1950s-2016. [5]

1.1.2 Project Background

A major bottleneck in the process of oyster farming is the maintenance of the oyster farm arrangement. While the structure and technique for oyster farming varies around the world, the oyster farm present at Ward is designed for the shore-side and near shore oyster farming. As in Figure 1-3, Ward Farms uses baskets made of sturdy plastic mesh to contain the oysters. These baskets are partially submerged to to allow for the oyster to extract food and oxygen from the water while also allowing part of the basket to be surfaced for biofilm removal. Each row of baskets are connected length-wise, and the entire floating array of 30×17 bags is anchored within a 60 feet by 63 feet area. Within this area, they are free to drift.

These arrays are in the ocean during the warmer part of the year. Because of this, the arrays have to be maintained to prevent the bags from suffering from the Cape's environmental conditions. The main consequence is the growth and accumulation of algae and biofilm on the submerged surface of the oyster bags. These growths consume oxygen, thus impeding oxygen from flowing through the basket's mesh and

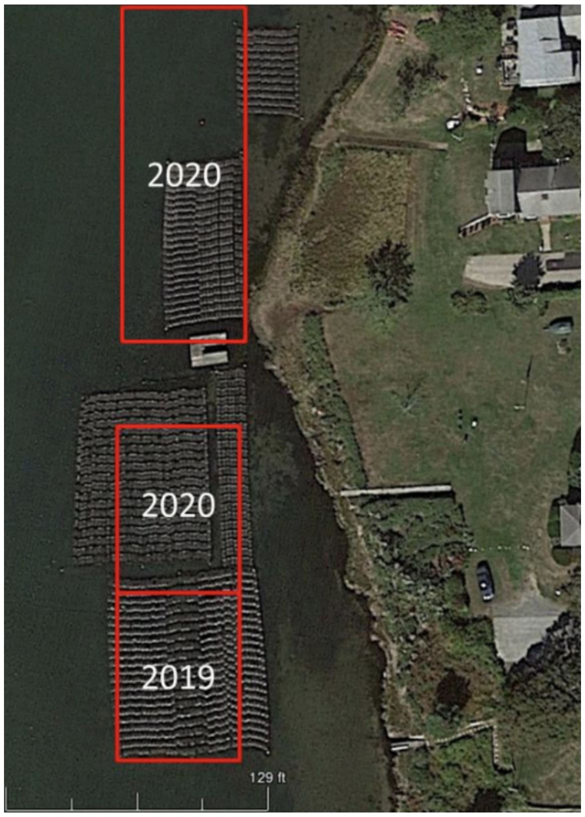


Figure 1-2: Satellite view of Ward Aquafarm's East Falmouth farm. The bounded areas indicate the expansion of the farm throughout the years.

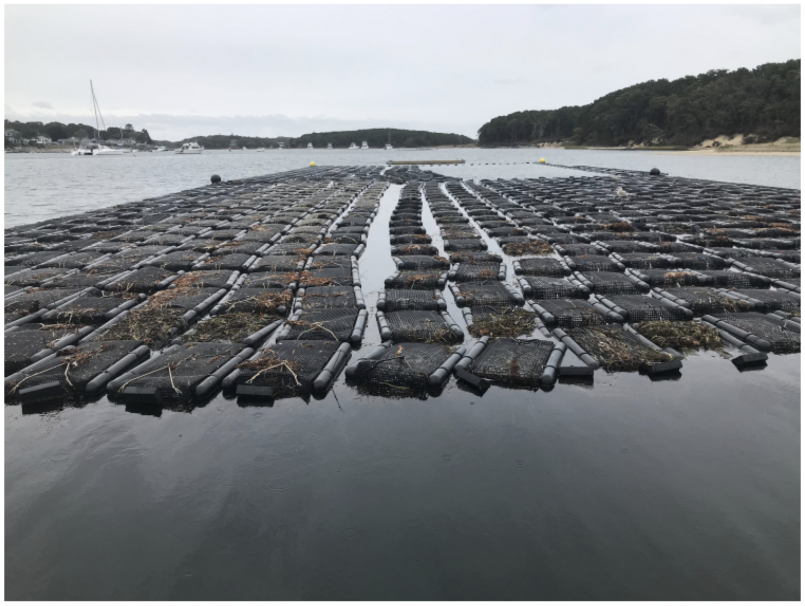


Figure 1-3: Ward Farm's open ocean oyster basket array

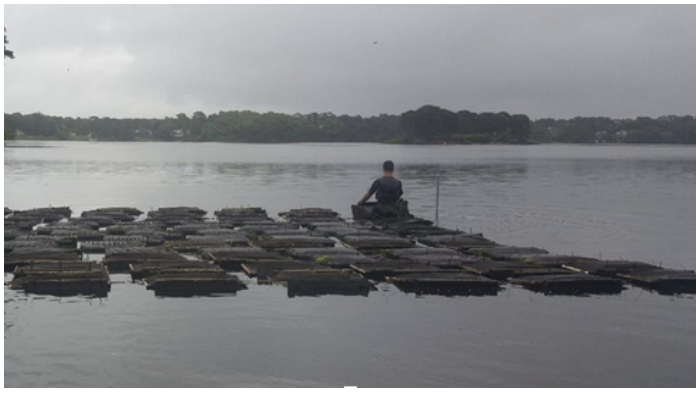


Figure 1-4: Worker in a kayak manually flipping the oyster bags at Ward Aquafarms

reaching the oysters. The common solution is to flip the oyster basket, exposing the submerged side to air and sunlight, which kills the algae and biofilm buildup. This is currently accomplished by having workers of the farm to navigate through each row of the array in a kayak and manually flip the baskets, shown in Figure 1-4.

With multiple arrays, each consisting of hundreds of bags, and each bag weighing up to 25 kilograms when near harvesting season, this process is tedious, and even potentially dangerous under certain conditions. Depending on the season and algae activity, these bag flipping sessions occur at least once a week and requires multiple workers an entire day to go through all of the baskets.

In cooperation with MIT SeaGrant Lab, this project was created to introduce robotics manipulation as a powerful tool into the field of aquaculture, and tasked to create a system that is able to alleviate the works of the tedious and dangerous task of basket flipping [17]. In conjunction with the MIT Class 2.017, Design of Electromechanical Robotic Systems, the Oystermaran, an unmanned surface vehicle (USV), was constructed for this purpose [2]. The design of the vessel is a catamaran with a flipping mechanism between the pontoons. As seen in the design rendering 1-5 and a field test 1-6, the vessel traverses the rows of the baskets, and when it aligns with a basket, it can flip it with the custom built flipping mechanism. The goals of the Oystermaran's design was to reduce the strain on physical laborers and significantly reduce the danger factor of oyster farming. The Oystermaran was designed to be economical and efficient, and it to be able to replace the need for human workers

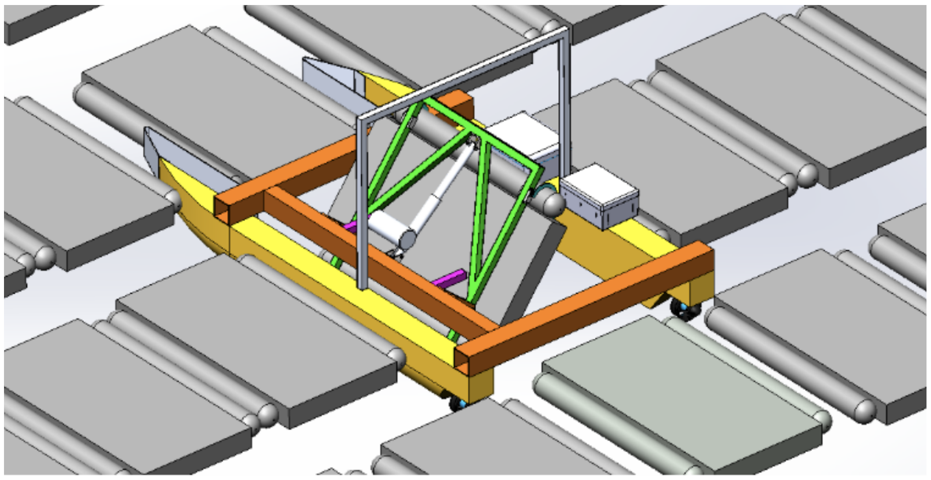


Figure 1-5: Design of Oystermaran MK. I. Design depicts the catamaran traversing through a line of oyster baskets

through remote operation. An updated design is in the works as of the writing of this thesis, seen in Figure 1-7. The new vessel will be equipped with a forward facing RGBD camera and a downward facing, optically flat RGB camera. The on-board computer is capable of processing both video feeds, and uses the results from the video to determine the appropriate control signals to navigate the vessel and flip the target baskets.

1.2 Thesis Scope and Organization

The goal of this thesis is to develop an efficient, robust, and modular autonomous system for the Oystermaran. While the current USV accomplishes the goal of removing physical laborer from the flipping process of oyster farming, the actual vessel still requires human guidance and input in order to operate effectively when flipping baskets. To provide a step to full autonomy, the goal of the autonomous system design presented within this paper is to provide a solution for the vessel to accomplish a vaguely predefined task with minimal human intervention.

This thesis is organized top-down; discussion of each section will begin at the high-level of the problem before exploring specific design and implementation choices that have been taken for this project. Because of this organization and the breadth

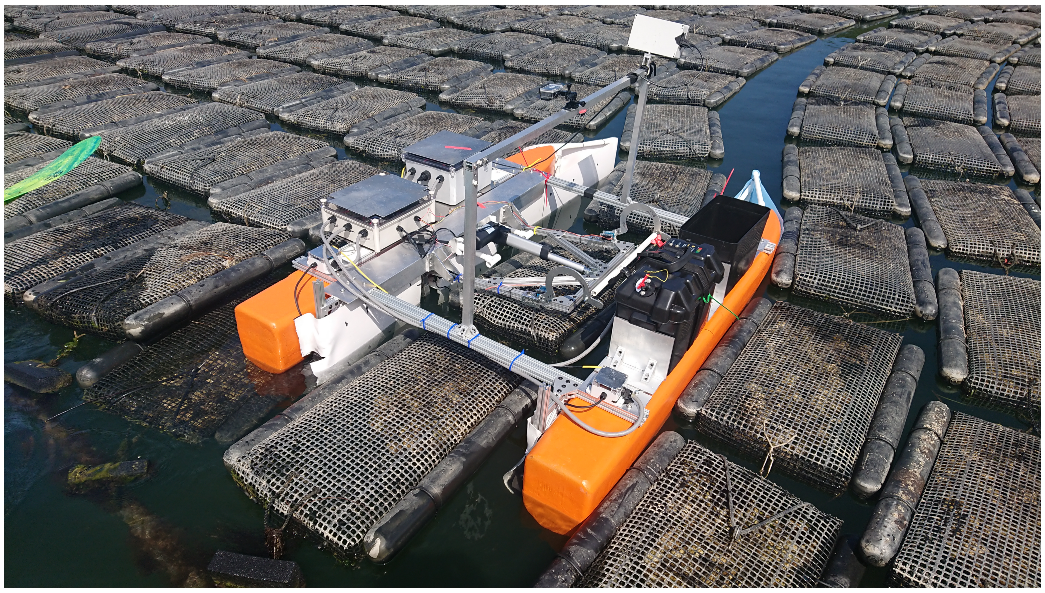


Figure 1-6: The Oystermaran Mk. II field test at Ward Aquafarms

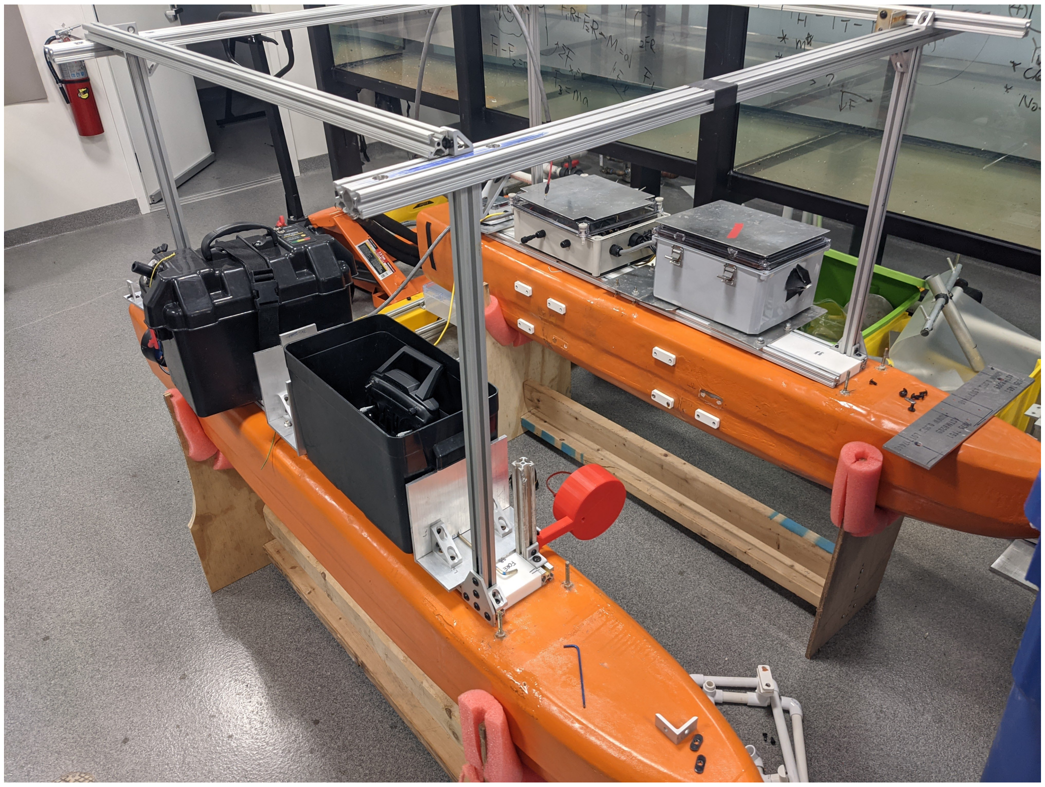


Figure 1-7: The Oystermaran Mk. III in development

of the project, the discussion of previous and related works will be distributed within the relevant chapters of this thesis.

Chapter 2

Autonomous System Design

2.1 Oystermaran Design Constraints

The autonomy system is constrained by the specifications and original goals of the Oystermaran: speed, cost, and power efficiency. Furthermore, the hardware specifications of both the on board computer and physical mechanisms are constraints the system needs to work with. The specifications relevant to the autonomous system are below:

1. Computer Specifications
   (a) Nvidia Jetson Xaviar NX with 8 GB 128-bit LPDDR4x and 32 GB eMMC 5.1
   (b) Arduino with Adafruit 9-DOF Absolute Orientation IMU Fusion Breakout and Adafruit Ultimate GPS Breakout V3
2. Vessel Specifications
   (a) Dimensions: 1 meter wide, 1.5 meters long, 0.82 meters high; the camera center sits 0.71m above the water.
   (b) 370 Wh battery capacity
   (c) GPS and WiFi connectivity for navigation and shoreside communication.

(d) Blue Robotics T200 Thrusters providing 5.25 / 4.1 kg f at 16V

(e) Sufficiently saltwater proof [2]

Given the constraints to the Oystermaran, multiple iterations of the system's design were considered before the current design approach to motion planning.

2.2 Design Problem: Motion Planning

The autonomy system this thesis is designing is in the domain of motion planning. Motion planning is the computational problem that autonomous systems solve to determine the discrete actions necessary to accomplish a predefined task. The problem can be broken down into three separate optimization problems: the mission, the behavior, and the local environment. Each optimization is concerned with different aspects of the environmental input, thus the complexity of the problem is governed by the degrees of freedom of the autonomous system and within the environment [6].

The motion planning challenge for land and aeronautical autonomy have environments that are either highly constrained or minimally constrained, respectively; they provide boundaries for the possible considerations for paths the system may take. The challenge of marine autonomy lies in the high degree of environmental freedom with a large number of movement constraint, requiring the search of a larger space in order to find an optimal path. These two factors combine to create a computationally complex pathing problem where the agent is limited in its movement's control [21].

Our motion planning problem is confined in a relatively constrained environment; our AUV will be operating within the context of aquaculture. This simplifies the problem by providing a more predictable environmental condition for as well as a simplified motion planning problem for the vessel.

For our autonomous system, the general processing flowchart is similar to that shown in Figure 2-1. It can be generalized into three distinct parts which will work together to complete its tasked mission:

1. Mission planning

2. Real-time trajectory planning for the "last mile"

3. Basket alignment and task execution

To design a system that accomplishes these steps, multiple software control frameworks were considered. The main points of consideration are modularity, robustness, and the ability to integrate custom algorithms. In the field, two main frameworks were considered: MOOS-IvP and ROS.

2.2.1 MOOS-IvP

The most prominent mission planning framework for autonomous surface vehicles is the Mission Oriented Operating Suite - Interval Programming, or MOOS-IvP. The primary motivation for using MOOS-IvP is its modular infrastructure approach to autonomy design. MOOS-IvP uses three architecture schemes to accomplish robust autonomy: the backseat driver paradigm, publish and subscribe autonomy middleware, and behavior based autonomy [3]. This software design allows for separation of concerns between autonomy subsystems and is structured to accommodate the integration of external systems. The libraries that exist for the middleware also allow for efficient programming of the system without repeating tasks.

Publisher-Subscriber

MOOS-IvP implements the publisher-subscriber model of communication through the MOOSDB, shown in Figure 2-2. MOOS apps publish and subscribe their data to the MOOSDB, which acts as an intermediate communication data structure [3]. Each MOOS app is a piece of software that ingests data from the MOOSDB and publishes computed information back to the MOOSDB. When an app first connects to the DB and subscribes to a variable, they see the most recent value that is stored for that variable. This model allows for complete modularity when adding functionality. This enables parallel development of the autonomy system of independent functionality.

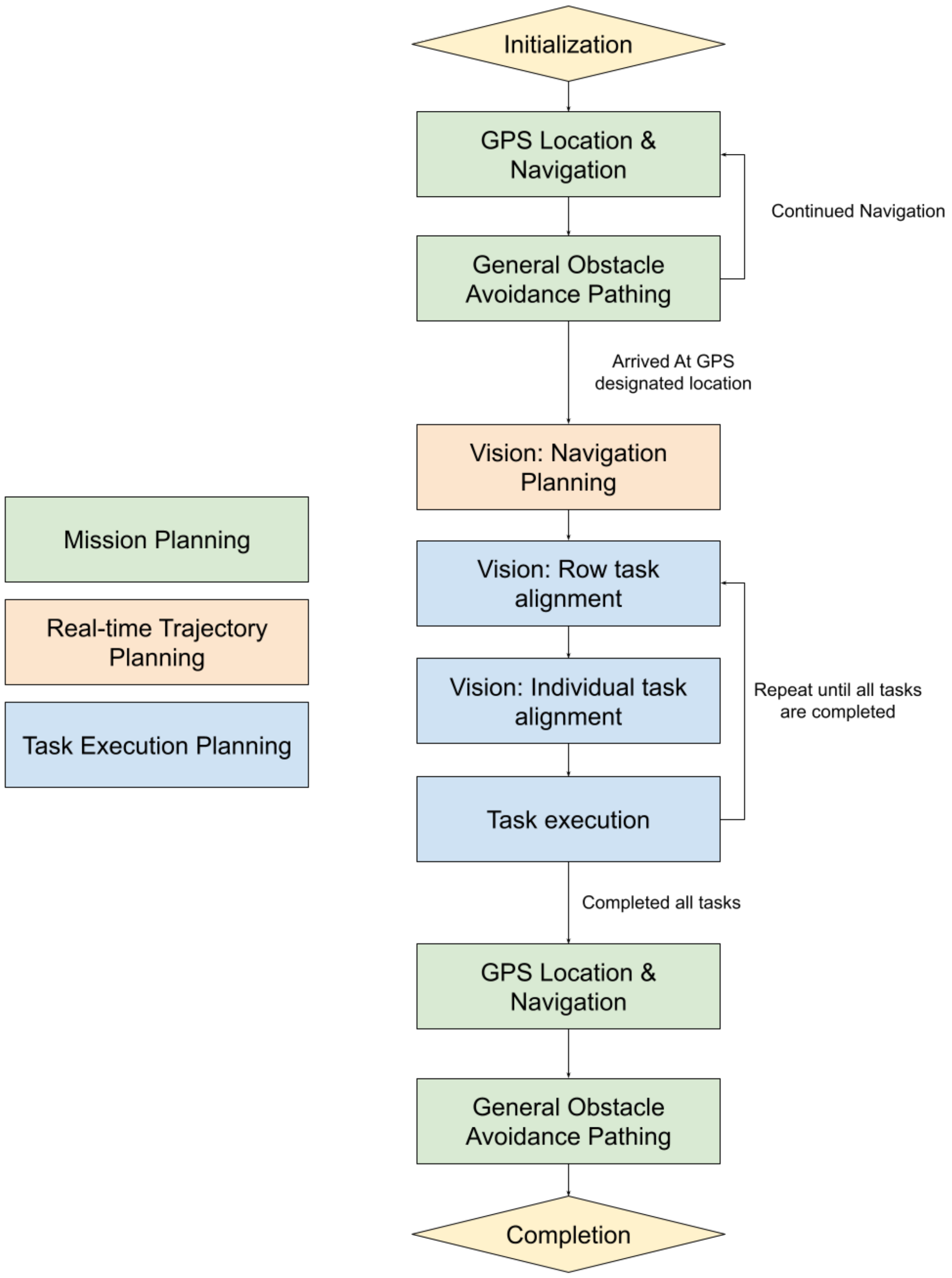

Initialization
GPS Location & Navigation
Continued Navigation
General Obstacle Avoidance Pathing
Arrived At GPS designated location
Vision: Navigation Planning
Mission Planning
Real-time Trajectory Planning
Task Execution Planning
Vision: Row task alignment
Vision: Individual task alignment
Repeat until all tasks are completed
Task execution
Completed all tasks
GPS Location & Navigation
General Obstacle Avoidance Pathing
Completion

Figure 2-1: Motion planning flowchart

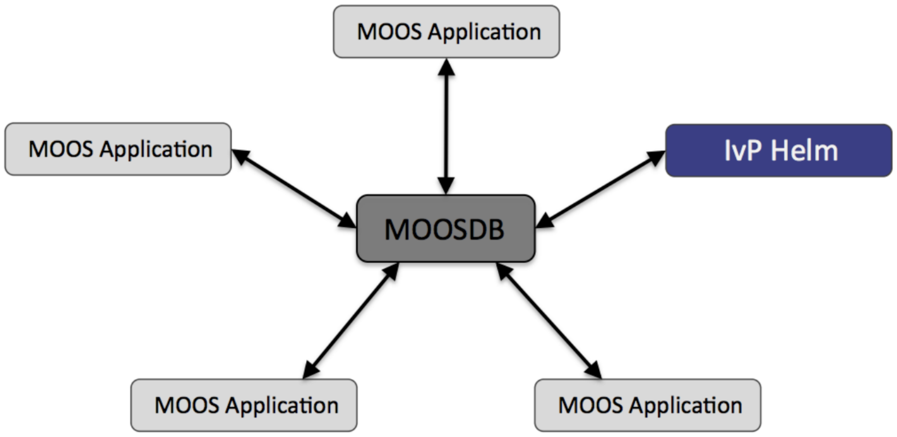


Figure 2-2: Helm as a MOOS app [3]

Helm

The behavior based decision making within MOOS-IvP is computed by a MOOS app called Helm, shown in the MOOS architecture in Figure 2-2. Helm determines a heading and speed decision by considering multiple different behaviors that a AUV should follow, seen in the flow chart in Figure 2-4. Some of these behaviors may include collision avoidance, obstacle avoidance, waypointing, and custom user defined behaviors. Each behavior takes in subscribed data and computes the unique objective function for that behavior, as seen in Figure 2-3. Helm then takes each objective function, and creates an cumulative objective function with a weighted priority of all of the behaviors according to solver Equation 2.1.

The result from this solver is then used to determine the final heading and speed which is subsequently sent to the MOOSDB to be injected by other apps.

$$\vec{x}^* = \underset{\vec{x}}{\text{argmax}} \sum_{i=0}^{k-1} w_i f_i(\vec{x}) \tag{2.1}$$

2.2.2 ROS

The Robot Operating System (ROS) is a widely adopted robotics software library that can be used for general robotics. Part of its popularity is due to the large, open

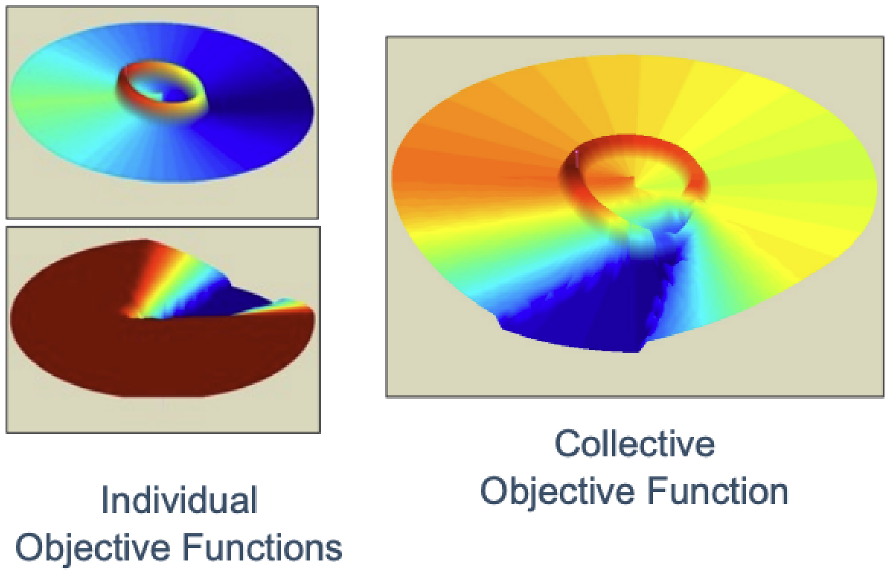


Figure 2-3: Each behavior produces a IvP function that is then used for solving for the optimal point or region [3]

source library of common robotics needs such as localization, pathing, and calibration methods. Additionally, it follows the publisher-subscriber model to allow for modular development of its system, allowing for separation of concerns for each node.

Publisher-Subscriber

ROS also communicates through the publisher-subscriber model. However, the implementation is different than what exists in MOOS. Information is communicated through topics to which ROS nodes subscribe to for updates. When a node broadcasts to a topic, all nodes that are actively subscribed and listening to the topic will be able to get the broadcast. Unlike the MOOSDB, new nodes which subscribe to the topic will not be able to receive data from previous events. Implementations such as ROS bags can be used to store such messages, but the implementation is still different than that in MOOS. This distinction will be discussed later in this section.

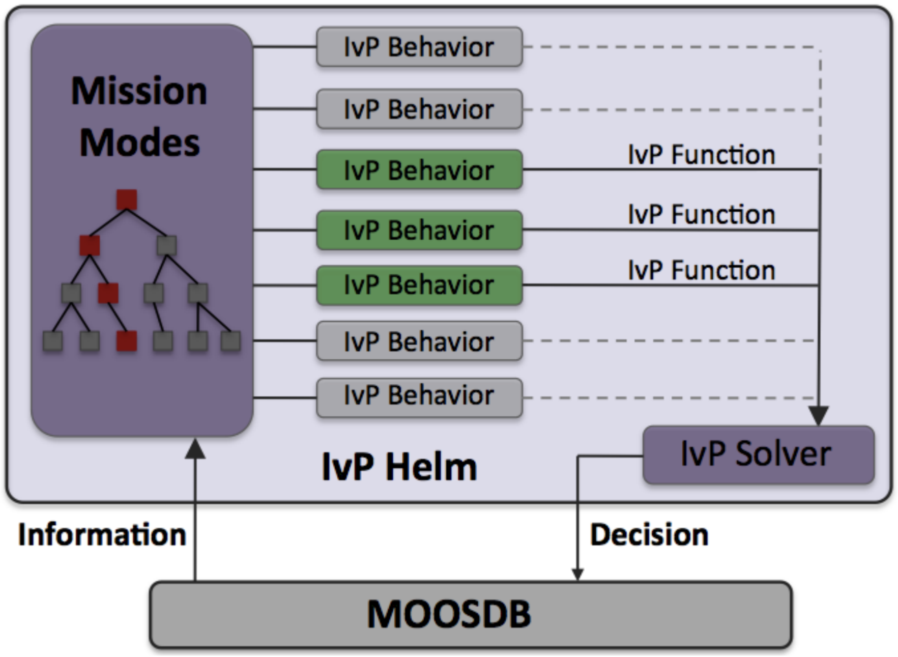


Figure 2-4: IvP Helm employs behavior based decision making. New data is read in the MOOS function and applied to the local buffer. Then the helm mode and set of running behaviors are determined. Each behavior executes, publishing data and an IvP function. Competing behaviors are resolved with the IvP solver by computing the combined decision landscape of all relevant IvP functions. The helm decision and any behavior postings are published to the MOOSDB. [3]

Hardware Nodes

The benefit of using ROS is its widespread adoption and integration. Because of this, many different external equipment have drivers or nodes that can be quickly set up and connect to ROS. Major nodes that will be important to the system include camera nodes to obtain and transfer the data as well as GPS for positional arrangement. We can utilize pre-established driver and libraries in order to streamline the communication between nodes as well as focus efforts on the problem in question.

Pathing algorithms

ROS libraries also contain existing frameworks for path planning and pathing algorithms. Traditional algorithms such as A* and grid paths are directly available as well as lateral pathing controls such as pure pursuit. These require a model of the vessel to implement, but can play well with the type of sensor information that we gather.

2.2.3 Communication Between MOOS-IvP and ROS

In the case of using both system frameworks for the autonomy of the Oystermaran, a communication bridge can be used. Developed by MIT SeaGrant for the Philos project [4], the communication bridge uses the NMEA communication message protocol to communicate between the two frameworks. The reason for splitting the autonomy system between two frameworks is due to computational speed. MOOS-IvP generally runs at a slower fixed cycle rate compared to ROS, and thus it is sometimes not fit to run larger and more intense processes without blocking and delaying the reoccurring MOOS processes. Thus, MOOS apps are not fit for more complex or time consuming computation that might be needed; in the case of this autonomy system, the image processing pipeline and path planning overwhelm and delay the process. ROS does not have this limitation, and along with its compatibility with Python and C++, the nodes can be implemented to be separate threads to allow for parallel execution.

The communication bridge between MOOS-IvP and ROS consists of a MOOS app that receives and broadcasts NMEA messages to update and read from the MOOSDB, respectively. On the ROS side, a node listens for these messages to then publishes to their respective topics. To ensure that there is no packet loss, the communication and buffer sizes need to be synchronized to ensure that the topic subscription on the ROS side is sending the all of the information through the bridge to the MOOSDB in a timely manner.

2.3 Software System Design

As each software control middleware has its strengths and weaknesses, the final system for the Oystermaran is to leverage the optimal aspects of each framework for the final autonomy system. To achieve the correct balance and separation, each task of the autonomous system is considered for each framework.

2.3.1 Overall Software Design Considerations

When choosing which framework to use for each aspect of the system, the following overall design considerations were considered. These considerations are used to analyze the mission, trajectory, and task execution planning of the autonomous system, seen in Figure 2-1.

1. It is favorable to have a system that is dominantly one framework. Communication between the two frameworks is not trivial, so minimizing the need for it will keep the system concise.

2. Minimizing processing time is critical: the system should respond in a reasonable time to allow for accurate motion planning and action execution responses.

3. Built in libraries and algorithms: we want to utilize what exists in order to minimize time spent on re-implementation.

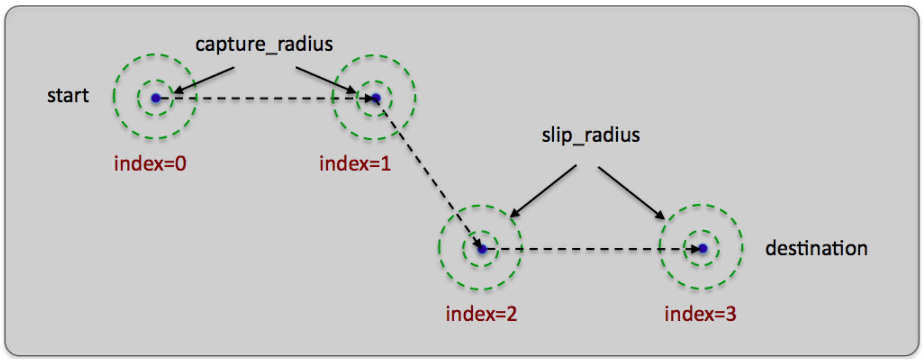


Figure 2-5: Helm's waypoint behavior [3]

4. Maintain modular design: if more than one framework needs to be used, have it designed to only need one framework to be functional. This makes hot swapping and debugging easier due to separation of concerns.

2.3.2 Design Considerations - Mission Planning

Mission planning requires the understanding of the global pathing problem as it entails the planning and trajectory of the vessel on a large scale. The actual path is less important than simply getting to the destination safely, which is vetted and predefined by the user prior to the mission. Thus, a passive pathing approach is sufficient; the vessel simply needs navigate to the destination while insuring that it is avoiding obstacles. During this time, communication with a shoreside computer is not strictly necessary and would mostly be used for position monitoring and GPS tracking.

Given these points, MOOS-IvP is a more suited framework for the system. While the parts of the navigation such as GPS planning and obstacle avoidance can be implemented using ROS, MOOS-IvP provides existing behaviors for GPS waypointing (Figure 2-5) and collision avoidance (Figure 2-6) [3].

2.3.3 Design Considerations - Short Range/Last Mile Trajectory Planning

The state of the vessel's processing changes to the last mile trajectory planning when it nears a field of baskets. Given that the specific positions of the baskets within

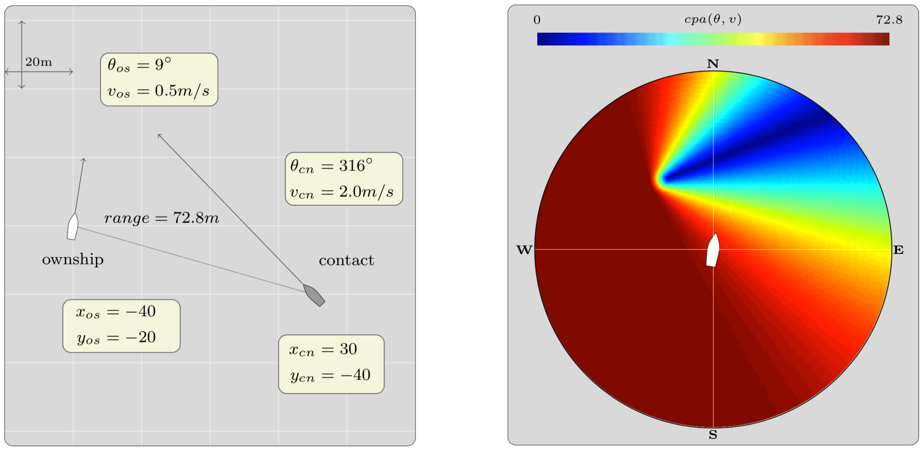


Figure 2-6: Helm's collision and obstacle avoidance behavior calculating the closest point of approach (cpa). Left shows the situation and right shows the visualized output from the objective function [3]

the anchored field area may change due to waves or tides, the Oystermaran will need a more specific navigation prompts and rely on a vision system to navigate the environment. The processing of the vision during this state would require more computation time as a result.

Additionally, a finer grain control system needs to be implemented, as the heading, momentum, and path of the Oystermaran is more important than it was in the mission planning aspect of the task. In addition, the alignment of the vessel with each row is a precise task that requires more frequent and high fidelity feedback when navigating.

Due to the need to faster, more precise, and higher fidelity feedback loops, ROS is the better system to use in this situation. The framework allows the Oystermaran to connect to ROS enabled vision and sensor peripherals as well as better control over the thrusters of the Oystermaran.

2.3.4 Design Considerations - Task Execution Planning

When the Oystermaran travels down each row, both the forward facing vision and the downward facing vision systems must be engaged for alignment with the baskets. The fine maneuvering necessary in trajectory planning is still required. For this system, ROS was used as the main middleware for communication and processing.

2.3.5 Final System Design

Using the considerations and conclusions from the previous subsections, the final system diagram is shown in Figure 2-7. This is the system that the project moved forward with. Key technologies and files for each framework and section are outlined, with further details discussed in later chapters.

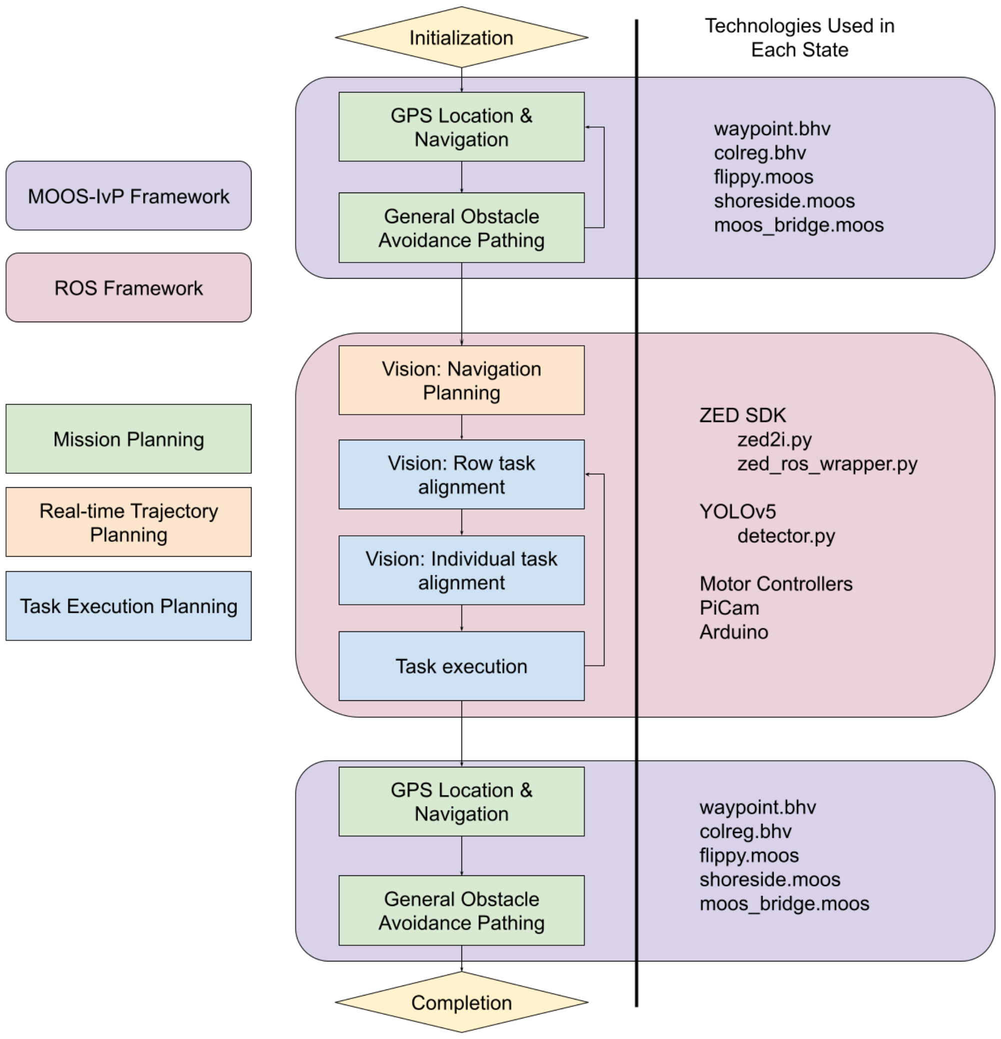


Figure 2-7: The system diagram that the project continues forward with. The right indicates the key technologies and files used for each of the sections of the system flow state.

Chapter 3

Perception

The main focus of this chapter will be within the realm of computer vision. Other perception methods were considered in the original design of the Oystermaran, but to minimize cost and power consumption, computer vision is designed to be the source of perception for the ASV.

3.1 Oystermaran's Specifications

The main perception on board will be computer vision based. There are two distinct vision systems on board of the Oystermaran:

1. Forward facing vision
   (a) Purpose: To guide the Oystermaran on precise, short range ($<$30 meters), path planning.
   (b) Hardware: ZED 2i RGBD camera with 2.5mm focal length
2. Downward facing vision
   (a) Purpose: To help with alignment of an oyster basket once it is between the Oystermaran.
   (b) Hardware: Rasberry Pi PiCam Module

Component	Power Draw
Nvidia Jetson Xaviar NX	10W-30W
ZED 2i Camera	2W
Arduino	0.5 W - 2W
Raspberry Pi	2.7W-5.1W
PiCam	1.5W

Table 3.1: Power consumption of computation components

The on-board computer is capable of processing both video feeds, but the constraints that are also important to note are processor and graphics capabilities of the computer and the power budget for to the vision system computation.

3.1.1 Power Consumption Constraint

Based on the specification of the Oystermaran white paper [2], the allotted power capacity for controls and computation is 68Wh and for a runtime of up to 3 hours. This approximately gives a constraint of 23W of continuous power draw per hour for the systems presented in this thesis. The following are the max rated power consumption from the on board computation and controls devices:

The total max power consumption of all the relevant equipment is 40W. Thus, some power efficiency metric is necessary to maintain within the power consumption of the specifications of the Oystermaran.

Since the Raspberry Pi and Arduino are operating only as a source of sensor collection and do no active data collection, it is possible to run them at at close to their minimum power consumption. This means that we have approximately 16W of power reserved to run the on board computer.

The main source of power consumption during this time will be the processing and decision making of the perception systems. To measure this metric, each vision model will be set to run on a pre-defined vision input for an hour, and the power consumption was recorded.

Nominally, the idle power consumption of the Nvidia Jetson Xaviar is 10W. Running a few idle trials and measuring power usage, we get that the average idle power

Trial	Power consumed in a hour
1	10.243W
2	11.197W
3	10.788W
4	10.143W
5	10.312W
Average	10.537W

Table 3.2: Power consumption data from five trials of the onboard computer in an idle state

consumption is a bit higher at 10.537W, as seen in the results in Table 3.2.

3.2 Vision System Techniques

There are multiple approaches to the vision system for the autonomous system. The criteria that we want to consider for each are as follows.

1. Speed; real time or near real time responses are optimal to get an accurate decision for the path. To achieve this, we would need to use an efficient algorithm. The rate we want to be at is above 40 frames per second for our system to operate smoothly.

2. Power consumption; given the power budget of 16W for the Jetson system. This will require a computation/resource efficiency.

3. Accuracy; the solution must be able to correctly and confidently identify baskets with tight bounding boxes in order for the Oystermaran to travel as precisely and accurately as possible.

With these metrics, the following methods were candidates, and were compared using the aforementioned criteria.

1. Traditional Computer Vision

2. Deep Learning Models

    (a) Single-Shot Detector (SSD)

    (b) YOLOv5

3.2.1 Traditional Computer Vision

Traditional computer vision (CV) systems focus on the use of intrinsic image data to determine information from the input. Relevant information involve edge detection, convolutions, and temporal/frequency domain analysis to obtain information from sequential image inputs. These can be combined and expanded onto achieve class-agnostic feature and object detection/classification systems, and often result in efficient deployment, both in the areas of implementing such a solution as well as not needing comprehensive datasets for an accurate model to be created [12].

This project uses OpenCV, an open source computer vision processing library, for most of its image and vision processing tasks. Specific implementation and uses will be covered in detail in its relevant sections.

3.2.2 Deep Learning

These traditional vision solutions often become less efficient to develop when compared to learning based systems, this is especially the case as the complexity of the problem increases and less invariants exist within the system. Deep Learning (DL) approaches to vision are often more data dependent in order to achieve accurate results, but can often outshine traditional CV by requiring less feature engineering by the implementation [12].

Deep learning is popular in other domains of autonomy due to the often non-convex natures of the problem that makes designing for all the features intractable [16]. Furthermore, the ability to gather large amounts of training data in such environments makes it more of a feasible solution than in other vision systems.

This thesis explores two DL models for multiple object detection: the Single-Shot Detector (SSD) and the YOLOv5 model.

Single-Shot Detector

The Single-Shot Detector, or SSD, uses one pass and two main networks to be able to do multibox object detection, a base network and a SSD head. This can be seen in Figure 3-1. The SSD head subdivides the image into grids and determines if there are objects within each grid cell. Then it applies a series of convolutional layers to collect the information within the grid to obtain an understanding of the objects in the scene. The base network is any compatible classifier network (VGG19, Faster R-CNN, etc.) that handles the classification of the object [11]. This network is often cited to be more accurate than competing models, with a small tradoff in speed.

YOLOv5

You Only Look Once, or YOLO, is a model that most notably uses a single network to do both the detection and classification of multiple object classes within a frame. Implemented to be fast, it is able to accomplish its labelling in real time (45+ FPS). The network accomplishes this through convolutional layers in a similar manner to that of SSD; however, the final layers use two fully connected layers to generate an output, as seen in Figure 3-1. The network's architecture is inspired by GoogleNet, but does not depend on its design. While significantly faster, YOLO is generally less accurate than SSD [13].

YOLO has developed multiple newer models since [13]. In YOLOv2, the number of classes the base detection algorithm has been expanded on and the classification can now recognize many more classes of objects. It also solves an issue of not being able to detect swarms of objects in the background (i.e. field of oyster baskets from a distance [14]. The architecture was moved to a Darknet-19 and made compatible with cuda. YOLOv3 further advances the accuracy and speed of the detector by updating the architecture to a deeper Darknet-53 [15].

For this project, the version of YOLO we will use is version 5. This version is an open-source project that focuses on different scales of YOLO detection models, seen in Figure 3-2. The different models in YOLOv5 are used for varying scales and computational power. It is maintained by Ultralytics which is part of the organizations initiative for open-source research for the future of computer vision [18].

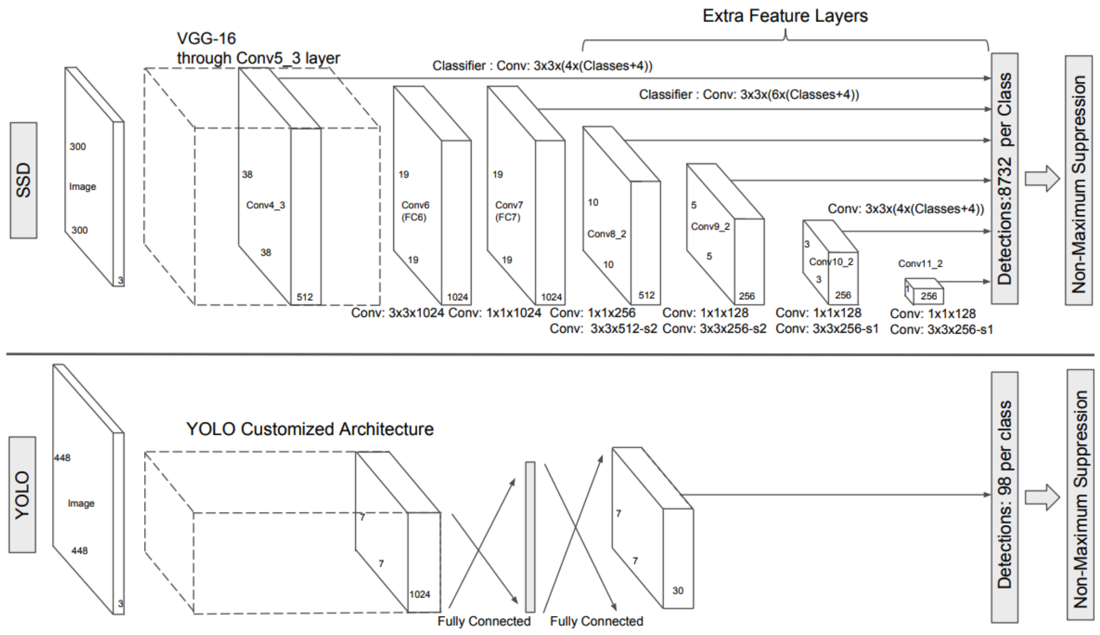


Figure 3-1: Figure from [11] comparing the network architecture of SSD and YOLOv1.

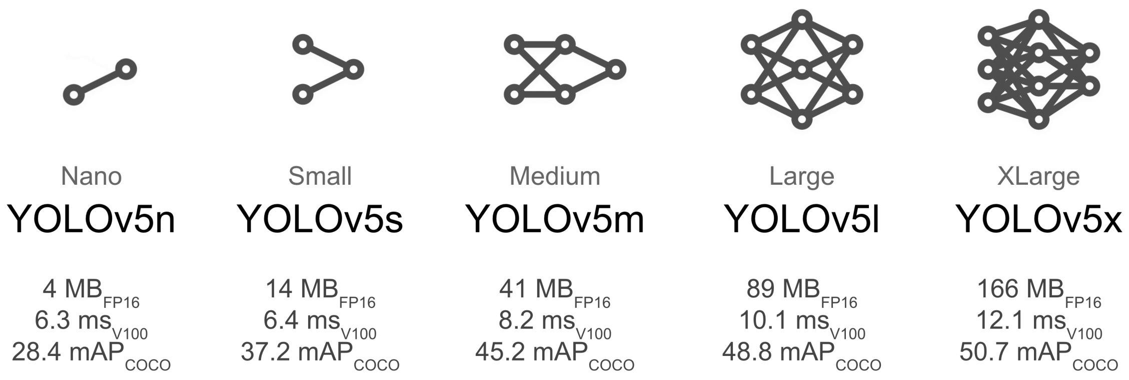


Figure 3-2: The different versions of YOLOv5 that are provided by Ultralytics with their pre-trained performance on the COCO dataset [10]

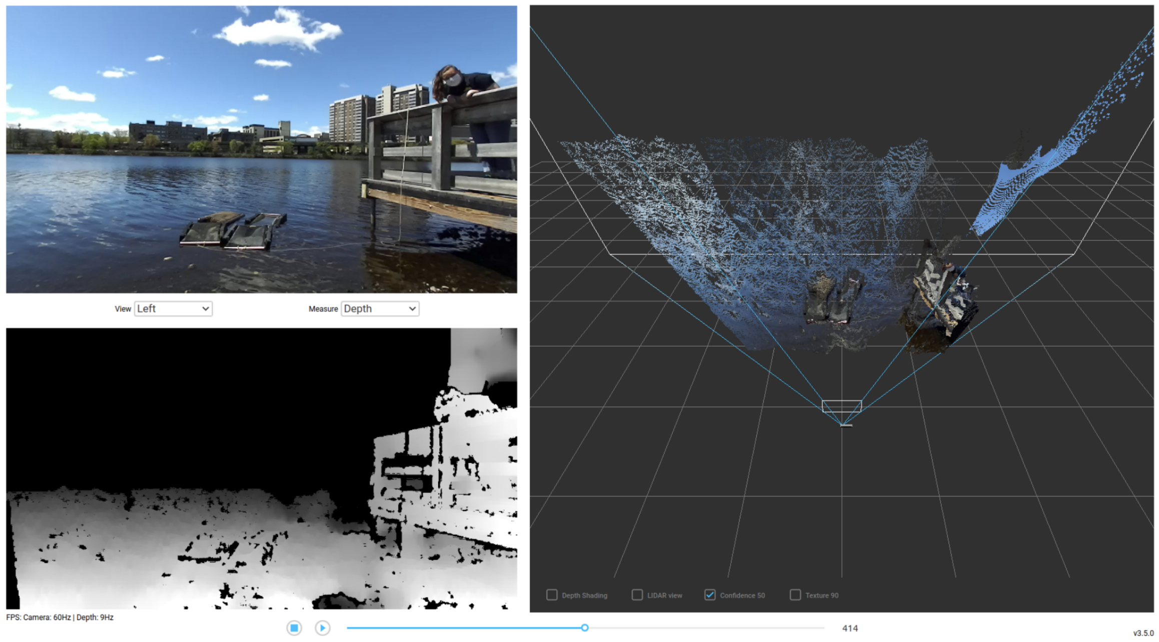


Figure 3-3: Top left: The RGB input seen from ZED camera. Bottom left: The depth sensor data from the same scene. Right: The generated point cloud from the RGB and depth sensors.

3.3 Forward Facing Vision

3.3.1 Objective

The goal of the forward vision system is for two main problems in our autonomy system.

1. Navigation and guidance when switching from GPS MOOS navigation to vision based navigation.

2. Front alignment when travelling through a row of baskets.

3.3.2 Hardware Specifications

The camera that is used for this is the ZED 2i RGBD camera. The ZED is able to provide 720p 60FPS video, which is the configuration most suitable for computer vision tasks. The depth sensor is also able to create a point cloud for 3D representation, as seen in Figure 3-3. With this, the camera can provide SLAM for the vessel.

Additionally, the depth sensor is capable of generating a 3D bounding box of an object from a 2D bounding box of our detector algorithm. This will provide better orientation and alignment and assist with the Oystermaran's navigation through the rows of baskets.

3.3.3 Dataset

The training data consists of 112 images of baskets, similar to the one in Figure 3-6. They were also annotated with the bounding boxes and labels for each of the oyster baskets. Most of the images of the oyster baskets are from the height and perspective of the vessel. Some of the images are more robust and are from different angles and elevations. A general overview of the dataset can be seen in the correlogram in Figure 3-4.

An important note is that the dataset consists of mostly single baskets or cluster of baskets images. The number of basket field training images are limited, so there may be more image data collection needed in order to achieve a greater accuracy in the field.

3.3.4 Discussion of Techniques

The project explores traditional computer vision techniques along with different scales of SSD and YOLOv5 detectors to determine the model that is best fit for our constraints.

Traditional Computer Vision

To be able to recognize baskets, we want to determine the features of the baskets from the surrounding. To do this, we use OpenCV for the next few image transformations. First, we convert the image to HSV color representation. Then, we take the value and present a grey-scale image. This process can be seen in Figure 3-7. This will make it easier to determine where a basket exists by narrowing down the search patterns we need to match. To make them further distinguishable, we applied red zip ties to

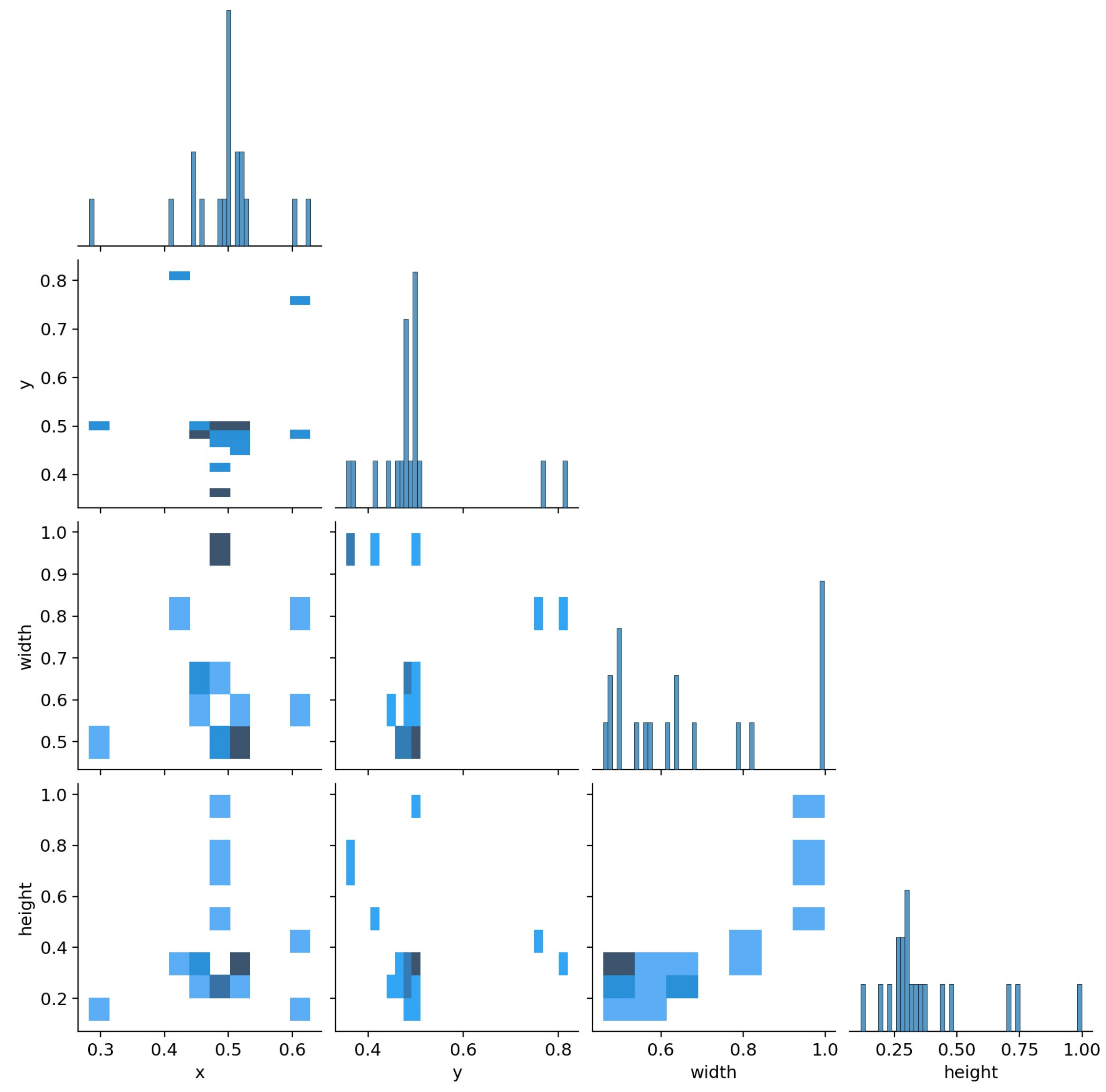


Figure 3-4: The correlogram shows the distribution of image parameters within the dataset and their correlation to each other. The width and height refer to resulting bounding box after the crop of the data and xy refer to the center of the bounding box location of the basket within the original image. In the context of the vision system, this figure describes the different views and angles the vision system may see within the dataset. The more likely a particular view or angle, the higher the correlation and darker the shading. The self histograms are a count of the occurrence of the parameters in the dataset. The sparse distribution seen in this figure is due to the small amount of data within our dataset. This is likely an indication that we do not have enough data to achieve robustness, and is a issue that may be resolved in the future.

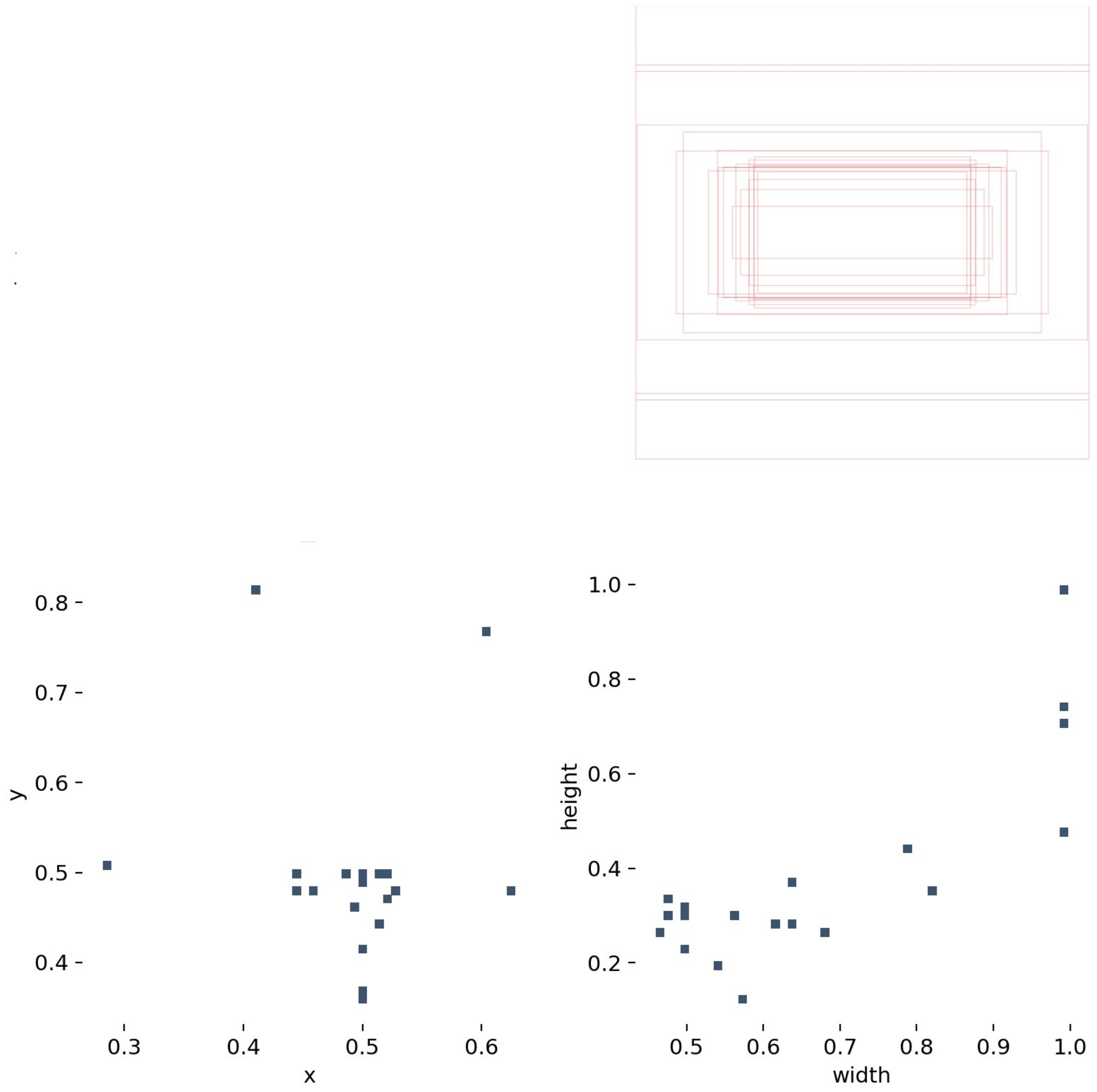


Figure 3-5: Top: All bounding box locations and sizes across all images within the dataset overlayed in one area. Bottom Left: The xy relation of the center of the bounding box location for each image in the dataset. Bottom Right: The width-height relation of the bounding boxes for each image in the dataset.

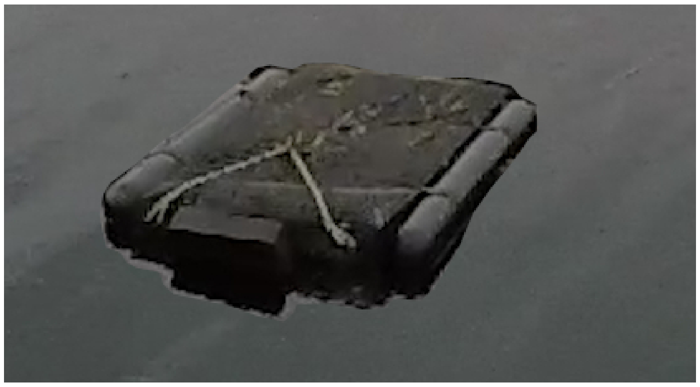


Figure 3-6: An example image from the training dataset

the edges of the baskets, seen in Figure 3-8. These zip ties stand out from the darker background and body and allow for easier detection by the camera.

To determine whether an image has baskets or not, we use line and angle detection on the grayscale image. Given the quite "unnatural" grid like structure of the basket's mesh, we are able use it as a feature to detect. The detector will then create a bounding box around the object that it determines to contain these features. This is cross checked with the depth image to determine if image pattern matches the depth pattern of a oyster basket.

Additionally, based on the angles the mesh makes, we can determine the relative difference between the heading and orientation of the basket. This can be used to determine whether the angle of attack of the vessel is aligned with the row of baskets. An example of this is seen in 3-9. Since the camera is placed in the center of the Oystermaran, when the basket is aligned, the angles of the grid at the center of the basket basket achieves a near 90°.

The grid-detection technique is the most accurate and consistent iteration of the traditional computer vision approach that we have achieved for this project. However, the results that it produces is still not reliable enough to be implement and used as a basket detection algorithm.

In this case, and in future discussions in this thesis, accuracy is determined by the Jaccard index, which is also known as Intersection over Union (IoU) metric, seen in equation 3.1.

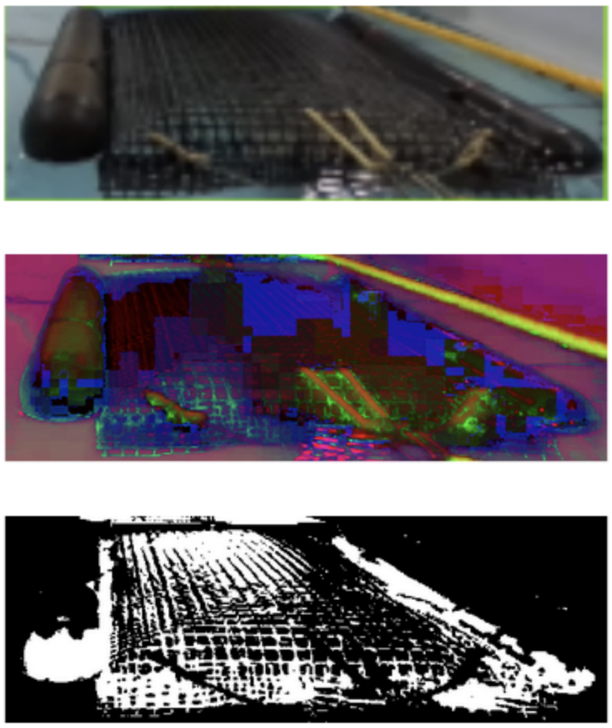


Figure 3-7: The image preprocessing pipeline to make the existence of oyster baskets more apparent for traditional computer vision techniques. Top: Raw image; Middle: HSV representation; Bottom: A grayscale image from the value channel

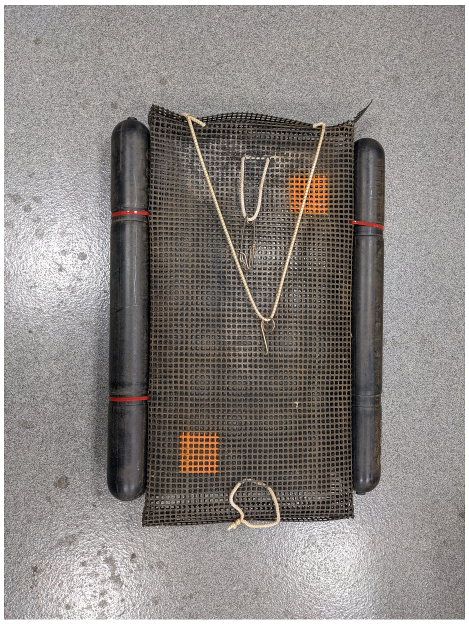


Figure 3-8: Example of an oyster basket with red zip ties for greater vision accuracy

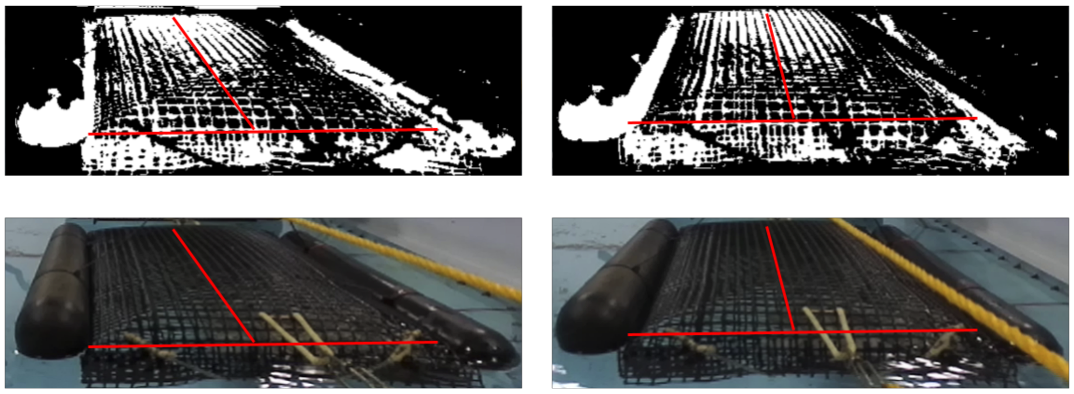


Figure 3-9: Top: Images of the basket when the processing pipeline from Figure 3-7 is applied. The line and angle recognition algorithm computes the lines and angles within the image. NOTE: the algorithm does not produce the red lines; those were post processed to demonstrate what the algorithm is doing. Bottom: The raw camera images of the oyster baskets with the post processed lines displayed to show alignment. Left: The angle of the line intersections are too acute for the camera to be lined up with the basket. Right: The camera is in a direction such that the wire mesh is closer to right angles, which means it is aligned with the camera.

Image Type	Average IoU Score
Vessel Level	0.24
High Angle	0.78

Table 3.3: Table showing the accuracy on different image types when using traditional computer vision detection

Trial	Power consumed in a hour
1	12.64 W
2	11.89 W
3	12.48 W
4	12.12 W
5	12.59 W
Average	12.34 W

Table 3.4: Power consumption data from five trials of running traditional computer vision

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{3.1}$$

A and B represent the areas of the two bounding regions. This metric is used to compare the predicted bounding region with the ground truth bounding region. The greater the IoU, the more accurate the predicted bounding region is. The metric ranges from zero to one. Through testing within datasets and ROS bags, a consistent IoU of above 80% is sufficient.

The results of testing the grid-detection forward facing algorithm on the dataset are shown in Table 3.3. A surprising note is how well the algorithm works for higher angled shots. This is likely due to the case that there is better clarity of the features that our algorithm looks for. Since the downward facing camera will have such a vantage point, this result and technique is further discussed in the downward facing section.

The power consumption of the process was promising, as the pipeline does not use the GPU heavily. This can be seen in Table 3.4

Overall, the traditional computer vision approach does not seem to be robust enough to be able to act as our primary source of forward facing vision. However, it has potential for the downward basket alignment vision.

Single Shot Detector

As discussed in an earlier section, SSD networks have a base network and a SSD head. The SSD head is mostly unchanged between networks for our purpose, so the main comparisons between SSD were the base networks and the size of the input image. The network models that we work with are contributed from the TensorFlow Model Zoo which are a set of models pretrained on the COCO dataset [1]. The following four models were trained and tested:

1. SSD ResNet50 V1 FPN 1024×1024 (RN1024)
2. SSD ResNet50 V1 FPN 640×640 (RN640)
3. SSD MobileNet V2 FPNLite 320×320 (MN320)
4. SSD MobileNet V2 FPNLite 640×640 (MN640)

These SSD architectures use the MobileNet and ResNet50 bases, respectively. Each of these networks are popular base networks for SSD.

The MobileNet network was created for the use on mobile devices or devices with limited computational resources [9]. It utilizes a shallower network with deep convolutional layers to achieve moderate accuracy with more efficiency and speed. It is also optimized for CPU or low-core GPU performance.

The ResNet50 is a residual network with 50 layers. Due to the deeper nature of the network, it is noticeably more accurate within a short amount of time compared to MobileNet. The residual connections allow for efficient deeper networks by reducing the vanishing gradient problem that is introduced by naturally adding more layers [8].

Both of these models use a structure known as a Feature Pyramid Network (FPN) which detects which parts of the input image may be of interest. A robust FPN will produce a greater accuracy.

As these are both pretrained networks, we can apply transfer learning to allow for our network to recognize classes that we want, in this case the oyster baskets.

Because it is already trained to recognize pre-trained classes already, the training to transfer oyster basket recognition will be shorter than to train an untrained network to recognize the baskets [20].

We pre-process the dataset to fit into the dimensions of the expected input for the network. Additional data augmentations were also made to the images to increase the robustness of the data and thus the learning. The following augmentations were applied.

1. Center crop of the image to size. This allows for a cleaner background of the image.

2. Added 16% random noise to the image. This is to simulate potential poor image quality when on the water

3. Up to 22° degree skew to simulate the possible lens distortion and basket orientations.

Each network was set to train for 1000 epochs with a patience of 150 epochs (if there is no progress in terms of accuracy after this threshold, then the training is stopped prematurely and the best weight is recorded). The total training time was around 120 hours. The accuracy (Figure 3-10), frames per second (Table 3.5, and power consumption (Figure 3-11) from the result of the training and testing are shown below. Other training efficiency metrics such as total training time (Figure A-1) and time per epoch (Figure A-2) are also available.

Based on Figure 3-10, the best model is the larger and deeper model of the ResNet50 with better performance at the larger image input. MobileNet performance requires more time to train and results in an average accuracy of around 80%. This is to be expected as there are more parameters and layers in ResNet to learn the features. However, the larger networks take more time to both train and run, as seen in both Figure A-1 and 3.5. While pre-processing and training times are not as strict, the in field run times have a greater impact on the performance of the vehicle. This means that the larger networks will likely not be considered. Additionally, the

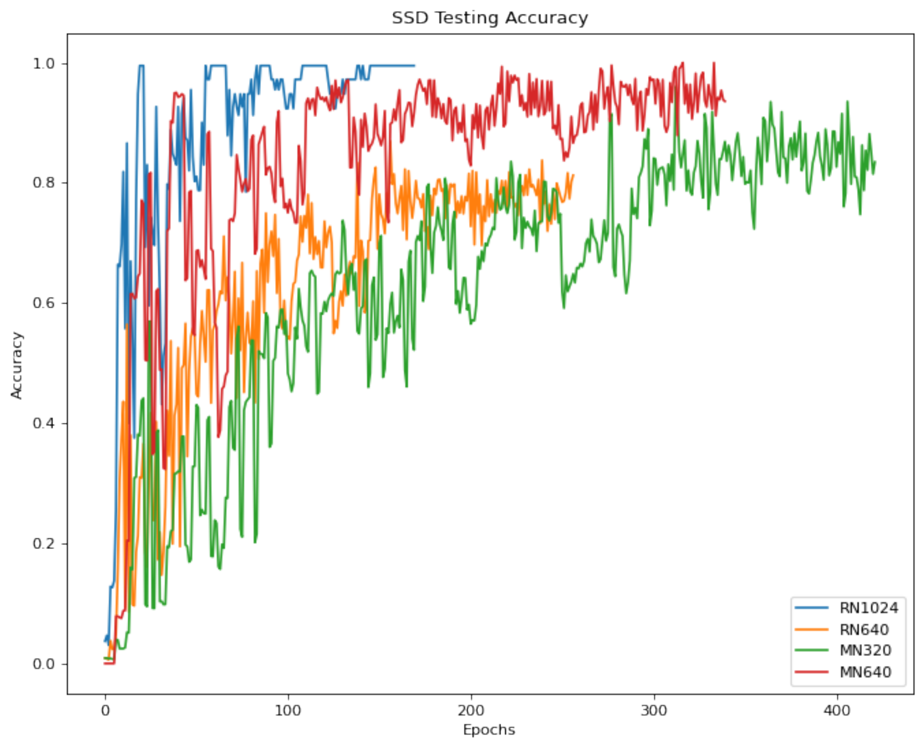


Figure 3-10: Graph showing the accuracy during training of the SSD models

Model	Frames-Per-Second
RN1024	8
RN640	10
MN320	33
MN640	29

Table 3.5: The average frames-per-second of the video processing

power consumption of ResNet exceed the allotted power budget, with the MobileNet base models using most of the 16W we are allowing.

Of the SSD networks, the most likely candidate will be the Mobilenet base that takes in 640×640 inputs, as it has the greatest balance of accuracy and efficiency. If power becomes an issue, the 320×320 variant of the network may also be considered.

YOLOv5

The YOLOv5 package that we are using contains 5 different sizes of models of the YOLO architecture, as seen in Figure 3-2.

Based on Figure 3-12 from the YOLOv5 publication, the potential best models

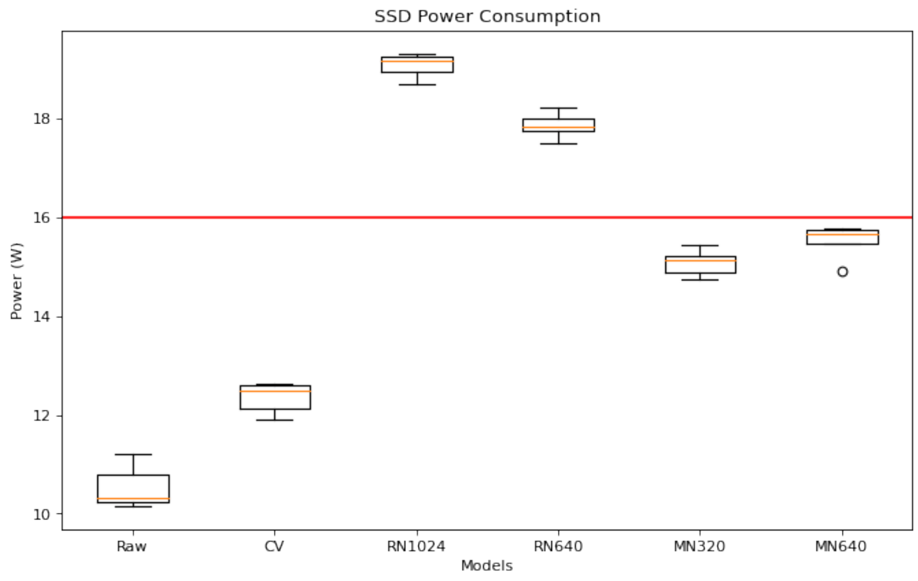


Figure 3-11: Graph showing the power consumption of the SSD models, with the power budget marked as the red horizontal line.

for our application, and the ones that we explored in the project were:

1. YOLOv5n
2. YOLOv5s
3. YOLOv5m
4. YOLOv5l

The same pre-processing procedure as present in SSD training were used, and all the networks use input images of size 416× 416.

For each network, we provided 1000 epoches of training with a patience of 150 epoches. The total resulting training time was approximately 20 hours. The drastic difference in time (Figure A-3 and Figure A-4) is due to both the YOLOv5 network's single pass architecture and also the newer models compared to the ResNet and MobileNet architectures.

From the results, all of the networks produce a model that achieve an accuracy that is sufficient for detecting the baskets (Figure 3-13. From a power consumption

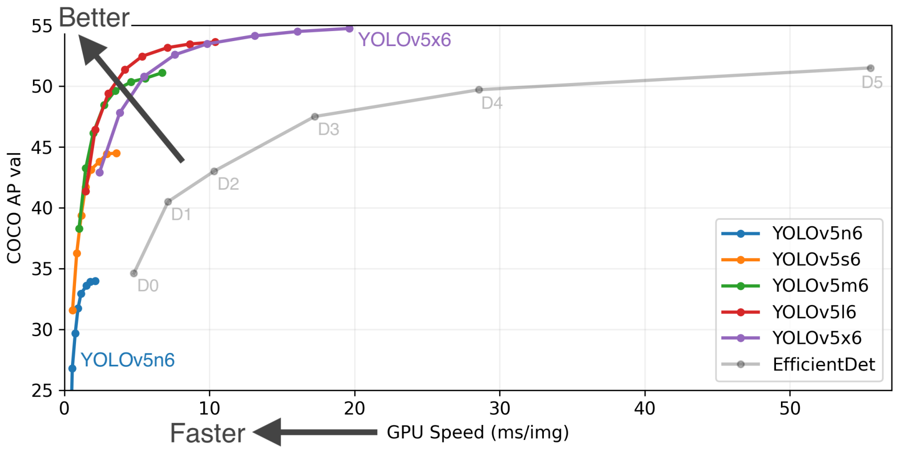


Figure 3-12: The graph from [10] indicating the theoretical speed and accuracy of each network size

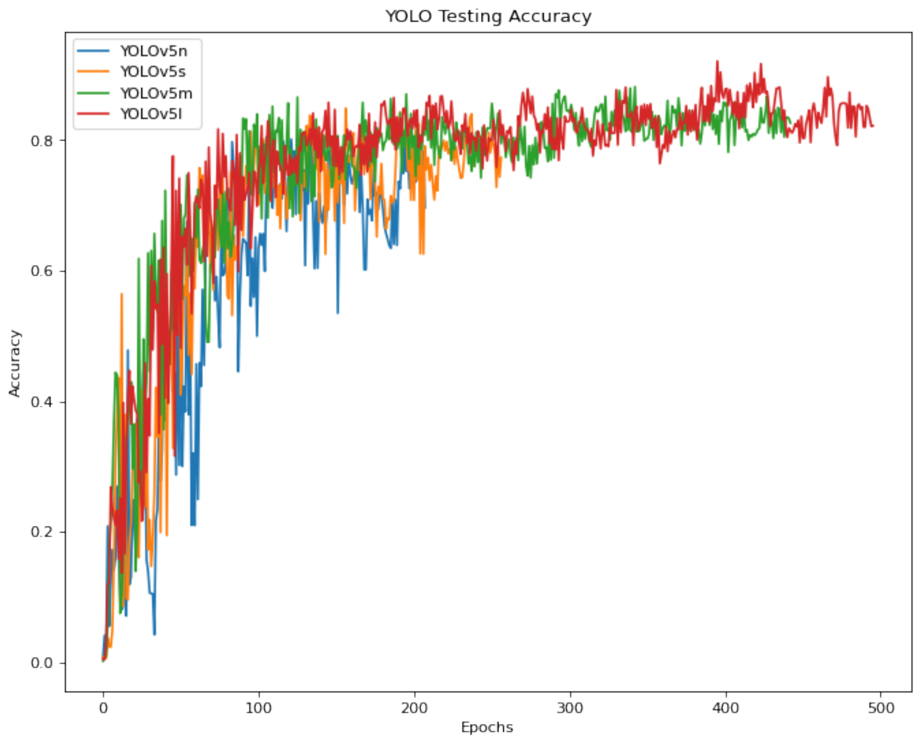


Figure 3-13: Graph showing the accuracy of the YOLOv5 models during training

Model	Frames-Per-Second
YOLOv5n	60*
YOLOv5s	54
YOLOv5m	37
YOLOv5l	19

Table 3.6: The average frames-per-second of the video processing for YOLOv5 networks. *Note: A frame rate cap was implemented during the test.

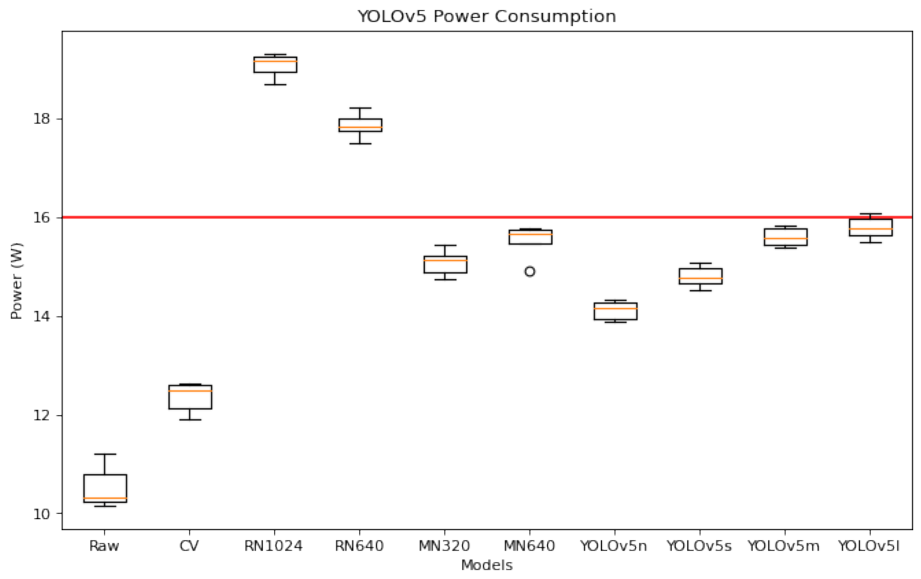


Figure 3-14: Graph showing the power consumption of the YOLOv5 models, with the power budget marked as the red horizontal line.

perspective, the YOLOv5n and YOLOv5s are safely below the power threshold. Because YOLOv5s is more accurate overall, and the training efficiencies are not much worse(Figure A-3 and A-4), we choose the YOLOv5s to be the best YOLOv5 model for our use.

3.3.5 Comparisons of vision processes

Of the object detection models that were presented, the system which is accurate enough, fast enough, and power efficient enough is the YOLOv5s network. The model achieves above an 80% accuracy score during testing while also being well within the energy budget, the network is able to achieve real time performance while minimizing power consumption. Detection from the YOLOv5s model results in images like Figure 3-15.

This is also definitive result that we will be using the YOLOv5s model for object detection for the Oystermaran.

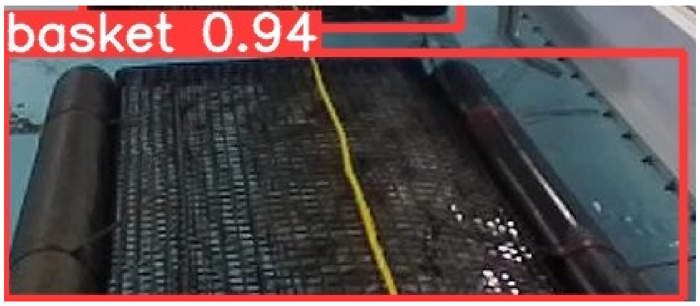


Figure 3-15: A sample image that is output by the YOLOv5s model

3.3.6 Integration

To integrate the detector into the vision system, we make use of the ZED SDK. The ZED2i has a built in system for object recognition, and a callback function for custom detectors. The software the camera comes with also has a ROS node wrapper, allowing for a seamless integration for a ROS node. A custom script was written to use this callback.

1. A frame of the video is grabbed from the camera and published to the raw image ROS topic

2. The script's node receives the image, and it pre-processes the image to fit the size of the detector's input

3. The image is run through the YOLOv5s network to produce a bounding box image

4. The bounding boxes of the detected oyster baskets are then passed to the callback function.

The callback uses the image bounding box and depth information to determine the 3D point cloud location of the basket. The object's orientation can then be determined and the heading calculated.

Once the desired heading is determined, it is published to the appropriate topic and consumed by the path planning ROS node.

3.4 Downward Facing Vision

3.4.1 Objective

The other half of the vision system of the Oystermaran is the basket alignment task. This is used when the basket sits between the hulls of the catamaran, and flipping mechanism needs to be aligned with the basket. Alignment is defined by two criteria:

1. Translational alignment: the basket needs to be within the area of the flipping mechanism
2. Rotational alignment: the sides of the basket should be parallel with the hull

Due to the design of the flipping mechanism, it is possible that a basket may fail to be flipped. In these cases, the camera will detect it and attempt to flip the basket again. The Oystermaran will move to the next basket when it determines that the basket has been flipped successfully.

3.4.2 Hardware Specifications

The camera that will be used for this system is a Raspberry PiCam. The camera feed can achieve 720p 60FPS video. The video stream is delivered through a Raspberry Pi, but will ultimately be processed by a node running on the Nvidia Jetson.

The camera will be mounted in the middle of the catamaran and be downward facing. The field of view of the camera is able to capture the entire area in between the hulls.

3.4.3 Dataset

The dataset of downward facing images consists of 23 images. Due to the smaller number of images and a different objective, deep learning models were not used for this purpose. Additionally, as determined in the previous section, using traditional image manipulation and masking techniques, it is possible to effectively determine basket alignment.

The dataset consists of images like 3-8, and with the markers on the flotation device, we are able to determine alignment, orientation, and the state of whether the basket has been flipped or not.

3.4.4 Basket Alignment

The image processing pipeline discussed in the previous section is used again here. Additionally, the zip ties are used for positional, perspective, and state determination.

Since the camera is aligned to be parallel to the hull, the red zip ties which are on the same flotation device are used to determine if the basket is positioned correctly. The line is used to determine if basket is far enough in between the Oystermaran in order for the flipper to be able to grab onto.

The basket's rotation is determined by the angle at which the three zip ties are aligned; the closer to 82° the more aligned it is. The angle is determine from the lens curvature and the cameras relative positioning to the basket.

Finally, the state of the basket is recorded before each flip. This is determined by creating a "triangle" of the three zip ties. If the flip was successful, then the "triangle" direction changes. If the change is not detected, then the flipping system tries again.

3.4.5 Power Consumption

The hardware power consumption is discussed previously and is already within the budget. Since the forward facing vision system is not active during this time, the only main source of computational power draw is with the traditional computer vision system. This, as determined in the previous section, is well within the power budget for the system. No direct power consumption tests were recorded for the system, but the estimate is similar to the system discussed in the forward facing vision system: at most about 13W per hour.

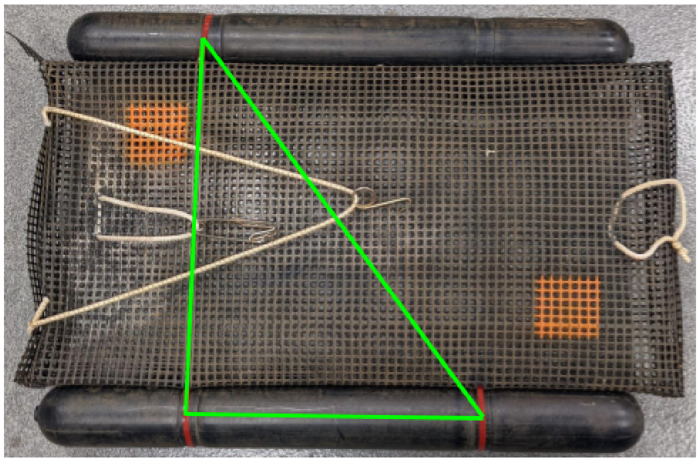


Figure 3-16: The red zip ties are used to determine the orientation and alignment of the basket. They are also used to determine the state of the basket since the placement is asymmetric. The green triangle is drawn in post to represent the state processing.

3.4.6 Integration

The system is implemented with its own ROS node, and listens to the topic for when task execution needs to run. Then, it starts processing the downward vision from the camera. The process mainly communicates to the topics regarding heading and orientation as well as the triggering the flipping mechanism. The former is to help align the Oystermaran to the correct position, and the latter is to trigger the flipping system when the vessel is aligned.

Chapter 4

Motion Planning

4.1 Overview

The motion planning for the Oystermaran considers the two major states the vessel is in. The first is to arrive at the designated location, and then it is to navigate the space while completing its tasks. As outlined in the design section of the thesis, the motion planning systems used for each state partially dictates the type of system used.

During the initial navigation to the field, the system will utilize behavior based motion planning from the Helm IvP app in the MOOS-IvP controls suite. During this state, the path is not as important as the destination; a reactionary system is sufficient to ensure the safe arrival of the vessel. The parameter for which we would like to optimize is power consumption; the speed and heading of the vessel should be such that we use the least amount of the battery needed to arrive at the designated location.

During the navigation to the baskets, among the baskets, and between rows of baskets, the size and need for lateral translation of the Oystermaran becomes an important consideration. During this time, the priority is to get the correct heading and orientation when approaching a row of baskets.

4.2 Behavior Based Motion Planning

During the first state, Helm-IvP contains many available behaviors that will allow for passive and reactive motion planning. Some that were described earlier, and the ones that we will be using are the waypointing, obstacle avoidance, and collision avoidance.

Waypointing will follow a tracking algorithm and maintain straight line travel between consecutive waypoints. These can be pre-defined such that it navigates from the shoreside dock to the appropriate field, illustrated in Figure 4-1. Certain waypoints can be placed in the fields to avoid known obstacles; however, when there are other vessels or objects in the water, reactive behaviors will be needed.

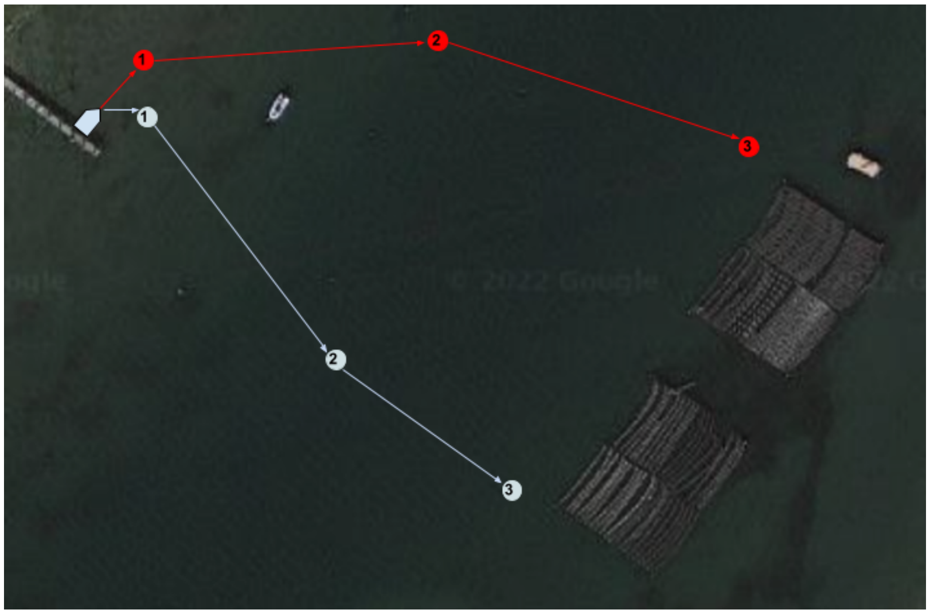


Figure 4-1: An illustration of waypoints for a vessel travelling from the dock to different fields at Ward Aquafarms' East Falmouth location.

In addition to waypoints, the system will implement international marine standard collision avoidance (COLREGs) and obstacle avoidance. These will rely on the forward facing vision, as described in the previous chapter. The ZED2i generates a depth point cloud and detects when the vessel may be encountering an obstacle. Passing the point cloud detection into the Helm behavior will allow for a reactive avoidance behavior to safely navigate the vessel.

Using the behavior based protocols will allow for safe navigation of the waters between the oyster basket fields and the dock or station in which the Oystermaran will reside.

4.3 Tracking Algorithms: Pure Pursuit

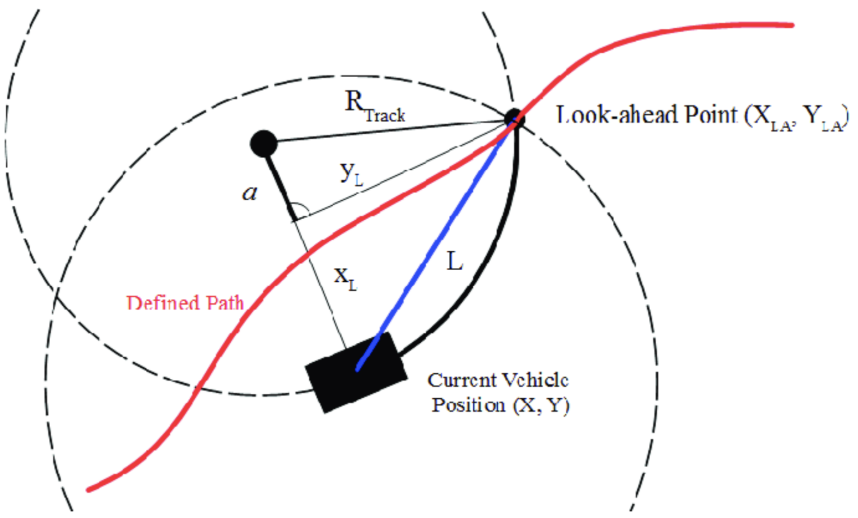
Once the Oystermaran is within vision distance of the fields, the forward facing vision system starts the detection process outlined in the previous Chapter. The system will need to direct the path planning in three different scenarios:

1. Navigating from <20 meters to the first row of baskets.
2. Navigating from one row of baskets to the next row
3. Navigating down a row of baskets

"To field" pathing

Since the heading of the vessel is of importance, we need to focus on pathing that will allow the vessel to enter the row of baskets in line. A naive solution is to do point turns in the water until the vessel is aligned to a row of baskets. However, this is very inefficient for both power consumption and time and will place extra lateral stress on the vessel.

Instead, we will consider algorithms that factor in lateral control. One of these popular lateral path tracking algorithms known as pure pursuit. The algorithm takes the tangents of planned path and uses a look ahead distance to calculate a desired velocity in order to get to the location, as shown in Figure 4-2. This algorithm will allow for the vessel to follow the necessary curvature to arrive at the row with the correct heading. This orientation is determined based on the results from perception. Thus, the nominal path could be projected with straight lines, as long as the lead distance to travel into the row of baskets is sufficiently long, the pure pursuit algorithm will naturally generate the necessary headings that will enter the baskets while following a smooth trajectory.



R_{Track}
Look-ahead Point (X_{LA}, Y_{LA})
y_L
a
x_L
L
Defined Path
Current Vehicle Position (X, Y)

Figure 4-2: A diagram of the pure pursuit path tracking algorithm. The algorithm uses the current location of the vessel and the next point in the path to determine the heading and speed that the vessel needs to take in order to take a smooth trajectory to that location.

Between row pathing

When the Oystermaran has completed a row, it will need to travel to the next one. The path it is programmed to take is to exit the row with some clearance, and then a Manhattan distance to the next row. When processed by the pure pursuit tracking, it will result in a "U-turn" to path to the next row.

Each row is delimited so the vision system is able to determine which row. A software counter and flag determines how many rows it has traversed. When the Oystermaran has completed the last row, then it will trigger the return waypoint and use behavior based motion planning back to the dock.

Within-row pathing

When the Oystermaran is pathing down a row and executing the task for each basket, the main guidance is the location of the next basket in front as well as the water guides to the sides of the current row that is being traversed. Based on the downward facing and the forward facing camera, micro adjustments by the thrusters may be used to ensure that the hulls of the catamaran are travelling down the two water guides.

4.3.1 Implementation

MOOS-IvP Implementation

The behavior based path planning was created in the file `flippy.moos`. The on board MOOS file serves two main purposes:

1. To communicate between the vessel and a shoreside computer that is running `shoreside.moos` which can remotely trigger and monitor the location of the vessel

2. Communicate between the MOOS and ROS components of the autonomy system. Data such as GPS, video stream, heading, and speed commands.

The behaviors that the system uses are defined within the file `flippy.moos`. Subscriptions to the variables that Helm and other MOOS apps may need are also defined in this file. Since the system is used in conjunction with ROS, the variables carrying the vessel's navigational and operational states along with the input from GPS are constantly updated within the MOOSDB.

Within Helm, we define the behaviors that we want to execute and their priorities. The parameters for each behaviors are stored in mission specific configuration files. For example, the waypoints and the clearance distance are unique for each different field of oyster baskets, and these differences are defined prior to the launch of the vessel.

Once the Oystermaran has reached the destination, it transitions to the trajectory planning state. During this state, the app `moos_bridge.moos` reads the MOOSDB variables that are necessary for the ROS system and communicates them to the appropriate ROS topics. These are ingested by the nodes that are coordinating pathing. Mainly a pure pursuit node ingests the location, heading, and speed states of the vessel to determine the next set of heading and speed that should be published. If the vessel is entering task execution, the vessel is stationed and flips the basket.

Once all the tasks are complete, control is handed back to the backseat computer and Helm navigates the Oystermaran back to the dock using the predefined waypoints and behavior based path planning.

Chapter 5

Conclusion

The work described in this thesis is part of an ongoing effort by researchers at MIT Sea Grant to continue the previous works on the Oystermaran system in an effort to create a robust and autonomous oyster basket flipping autonomous surface vehicle. Further work is necessary to integrate the system proposed in this thesis into the Oystermaran to fully test the integrated system.

5.1 Discussion

The use of a rigorous approach to the autonomy system provided a framework for autonomy development to move forward on the Oystermaran system. The value of this project is its enabling of future iterative development of the perception and motion planning software for the system. By providing a starting point for autonomy, the system can be advanced upon in multiple ways to create a final, robust system.

The major development hurdle for this project is the ability to freely test the software in an physical environment. As work on the vessel was on going during the project, testing of the systems in water was difficult. The limitations this created led to most work being conducted in simulation. The state this thesis concludes at is the working parts of an autonomy system. To actualize the system, its integration into the vessel and testing within the vessel must conducted before claiming the product as an autonomy solution for the Oystermaran.

The limitations of perception mainly results from the lack of data. For typical object detection application, the dataset of images consists of a minimum of thousands of images. The greater the dataset, the more likely there are image variants that are new to the system, and the more robust the recognition can become. The current working model may be sufficient in a controlled testing environment; however, it like will not stand up to varying conditions on an actual farm.

The path planning worked well in nominal sense in a controlled environment. Further testing will need to be to done to see if is adequate for an actual basket fields that is subject to drift and other factors such as the vessel's shape and dynamics.

Overall, the project provides a platform for further development and advancement for an autonomous system. While a full autonomous system was not deployed within the timeline of this project, future works will be able to make considerable progress once the system is integrated into the Oystermaran.

5.2 Future Work

The main work left for the system falls within three main categories:

1. Integrating the software systems with controls
2. Testing the system in the Oystermaran
3. Collecting additional image data and iterating on the robustness of the system

5.2.1 Integration

The integration of the system is likely a straightforward step. The autonomy system maintains a control on a high level: the vessels position, heading, speed, next waypoint, operational state and state transitions. These aspects need to be translated into the hardware controls of the system: thruster controls, flipping motors and servos, rudder controls. This connection was not completed in this project mainly due to the lack of expertise I have in this area, and taking a tangent on this work would have slowed the development of the autonomy system.

5.2.2 Physical Testing

Once the system is fully integrated, testing tank and river testing would quickly determine possible issues missed in simulation. The work within this area will be the most meaningful and fruitful as direct feedback from testing will show the strengths and shortcomings of the vessel as a whole. Depending on the failure modes, work in this area may be some resolving a combination of design, structural, dynamics, and software issues. These issues may or may not be resolved through only one subsystem; it is very likely that multiple subsystems need to be shifted in conjunction in order to resolve an issue.

5.2.3 Developing Robustness

Once the system works, development towards better autonomy can resume again. At this point, testing on-site and in different conditions will allow for both a stress test of the system and a chance to gather data and insight on how to improve the system further. This step will likely never end, as there can always be improvements on the system. Whether that is advancements in algorithms, greater compute power, or more efficient sensing systems, the autonomous system can always be improved in some way.

Appendix A

Figures

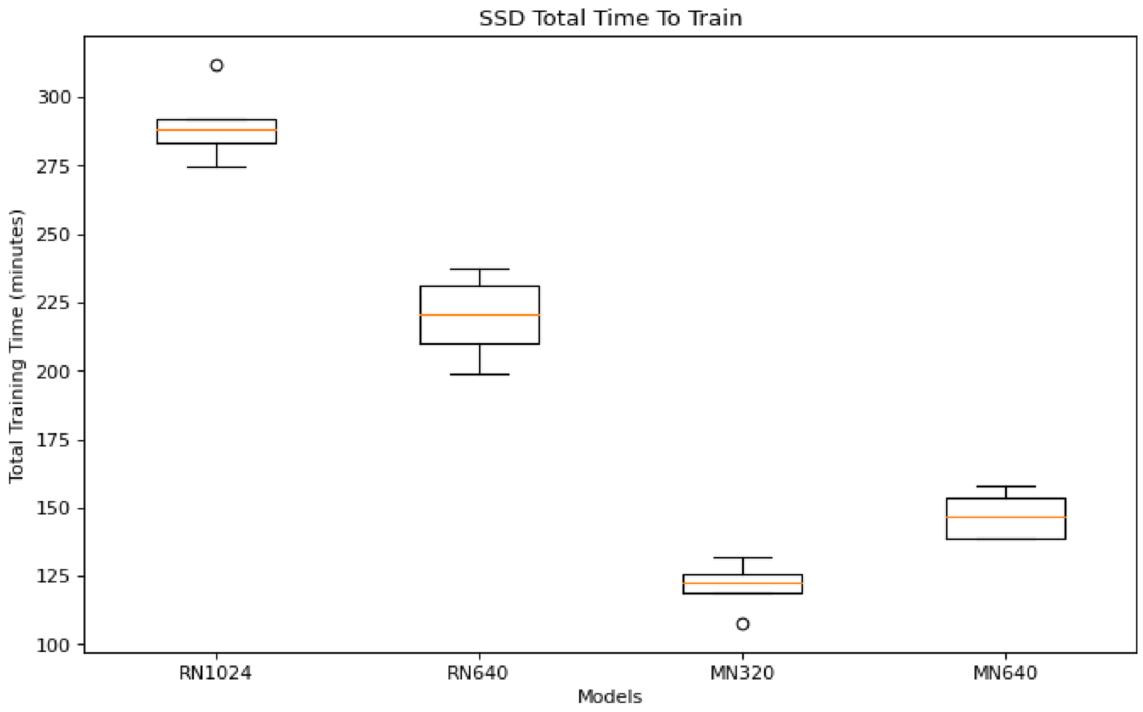


Figure A-1: Graph showing the total training time per SSD model

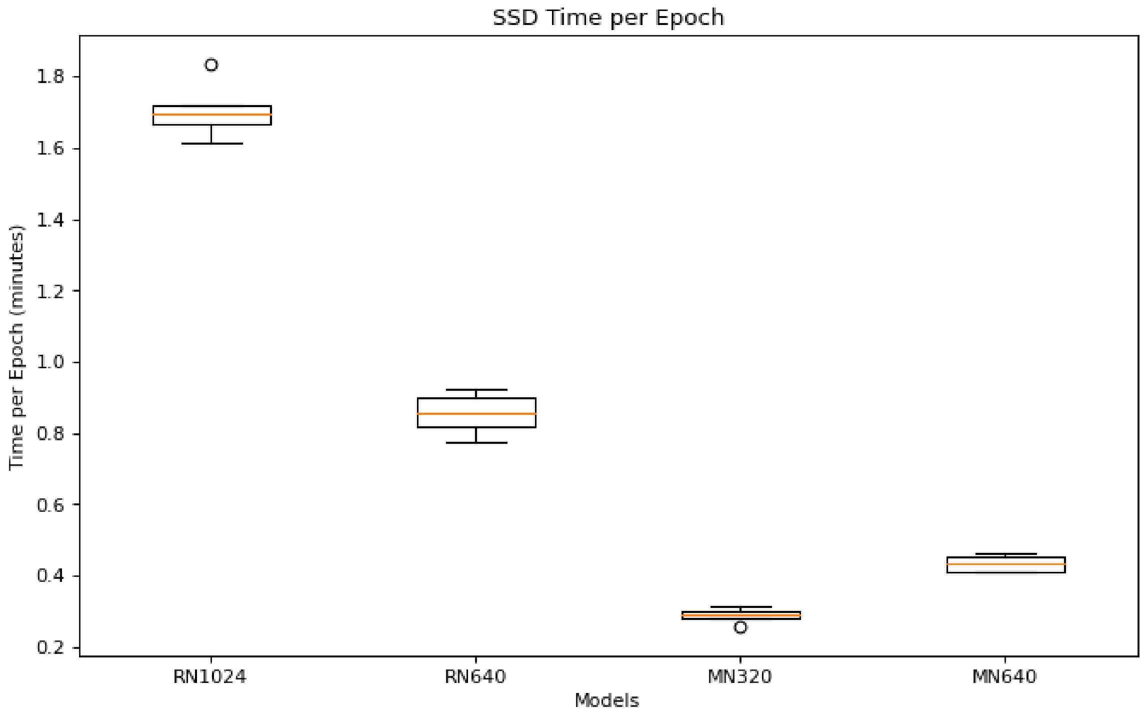


Figure A-2: Graph showing the training time per epoch per SSD model

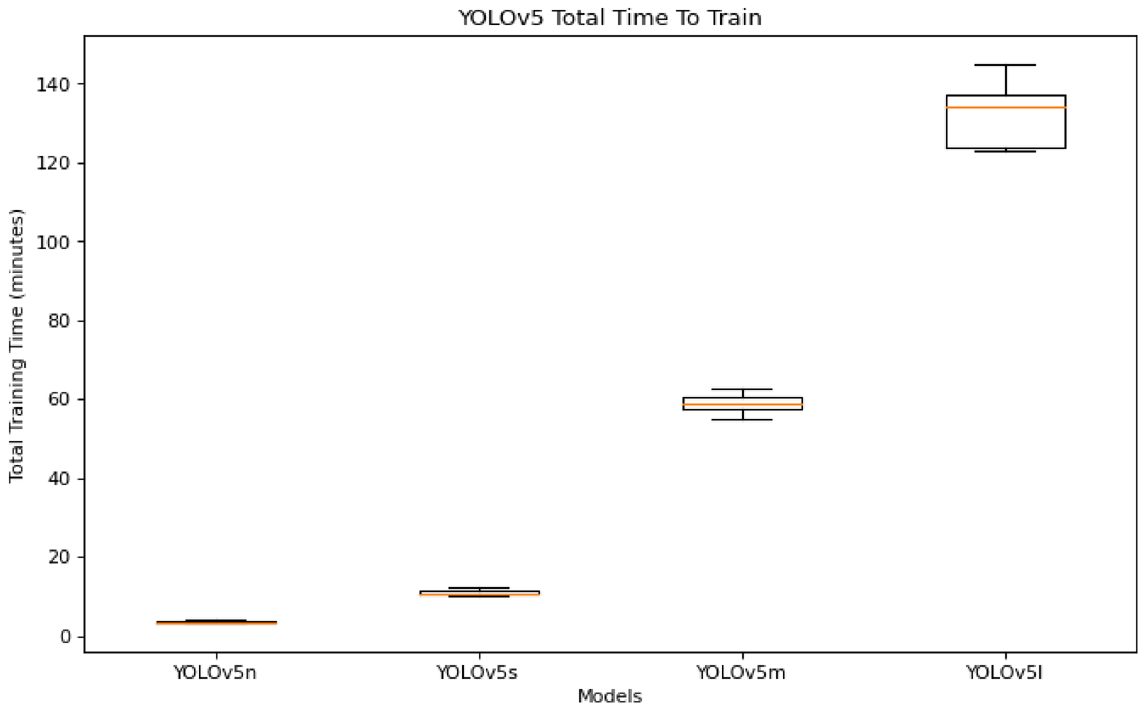


Figure A-3: Graph showing the total training time per YOLOv5 model

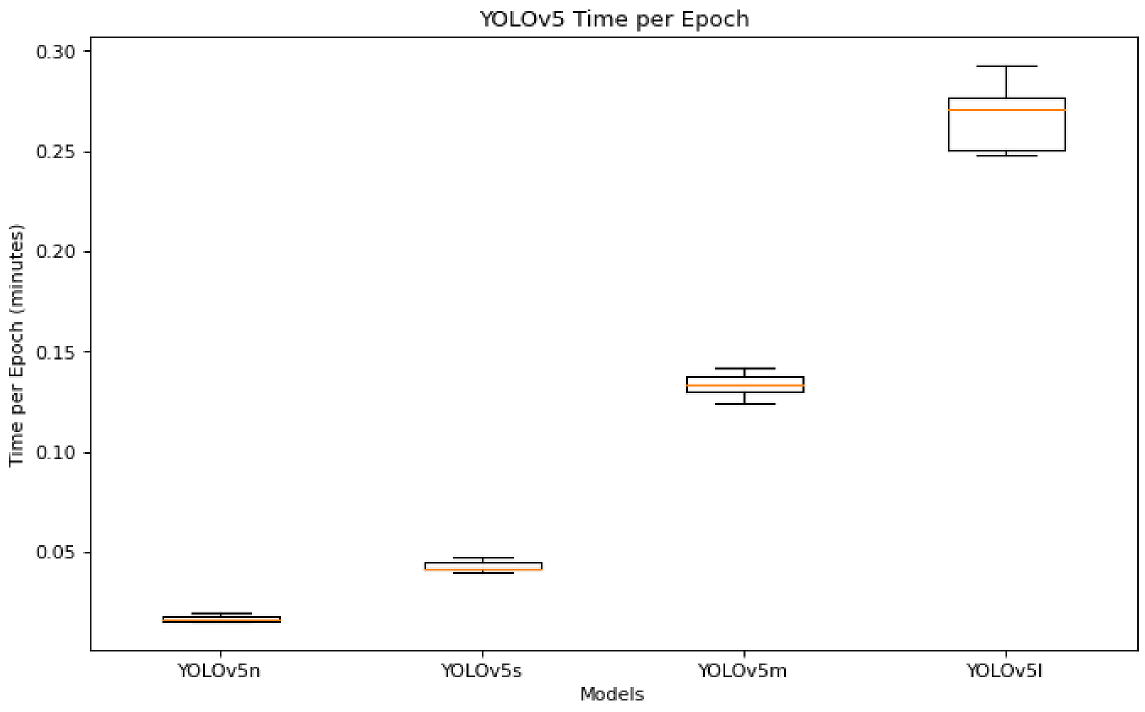


Figure A-4: Graph showing the training time per epoch per YOLOv5 model

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] John Adeyeye, Michelle Kornberg, Jacob McGuire, Joshua Padilla, Alexander Patton, and Margaret Sullivan. The oystermaran. Technical report, Department of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge Massachusetts, String 2020.

[3] Michael R Benjamin, John J Leonard, Henrik Schmidt, and Paul M Newman. An overview of moos-ivp and a brief users guide to the ivp helm autonomy software. *MOOS-IvP*, 2009.

[4] Michael DeFilippo, Michael Sacarny, and Paul Robinette. Robowhaler: A robotic vessel for marine autonomy and dataset collection. *IEEE*, 2021.

[5] FAO. Overview of the state of world fisheries. *Economics of Fisheries Development*, 2019.

[6] David Gonzalez Bautista, Joshué Pérez, Vicente Milanes, and Fawzi Nashashibi. A review of motion planning techniques for automated vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 17:1–11, 11 2015.

[7] Mark Hamming. Marine life series: The history of oyster farming.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. arxiv 2015. *arXiv preprint arXiv:1512.03385*, 2015.

[9] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

[10] Glenn Jocher, Ayush Chaurasia, Alex Stoken, Jirka Borovec, NanoCode012, Yonghye Kwon, TaoXie, Jiacong Fang, Imyhxy, Kalen Michael, and et al. Ultralytics/yolov5: V6.1 - tensorrt, tensorflow edge tpu and openvino export and inference, Feb 2022.

[11] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[12] Niall O'Mahony, Sean Campbell, Anderson Carvalho, Suman Harapanahalli, Gustavo Velasco Hernandez, Lenka Krpalkova, Daniel Riordan, and Joseph Walsh. Deep learning vs. traditional computer vision. In *Science and Information Conference*, pages 128–144. Springer, 2019.

[13] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[14] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[15] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[16] Ugo Rosolia, Stijn De Bruyne, and Andrew G. Alleyne. Autonomous vehicle control: A nonconvex approach for obstacle avoidance. *IEEE Transactions on Control Systems Technology*, 25(2):469–484, March 2017.

[17] Michael Triantafyllou. Autonomous surface vehicles for maintenance and intervention in aquaculture farming to improve occupational health and safety. *IEEE*, 2020.

[18] v7labs. Yolo: Real-time object detection explained.

[19] Daniel Ward. About ward aquafarms.

[20] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big data*, 3(1):1–40, 2016.

[21] Chunhui Zhou, Shangding Gu, Yuanqiao Wen, Zhe Du, Changshi Xiao, Liang Huang, and Man Zhu. The review unmanned surface vehicle path planning: Based on multi-modality constraint. *CoRR*, abs/2007.01691, 2020.