Alfie 2.0: Seeking Greater Autonomy

Daniel Yang, Alex Fay, Kyle Rong, Yoo-Jin Hwang,
Sidney Taylor, Francine Wright, MuddSub team members
Harvey Mudd College
Claremont, CA, USA

***Abstract*—This article discusses the ongoing modifications of *Alfie*, MuddSub's AUV first introduced at the 2019 RoboSub on-site competition. Given the additional constraints of remote testing and deployment in the past two years, we took this opportunity to fundamentally re-evaluate the structure of *Alfie's* mechanical, electrical, and software stacks with the criteria of novel and sustained contributions to vehicular autonomy. The result is a framework of unique subsystems, each operating under strict, predefined specifications of information and command flow. Under this structure, there are two distinct levels of design decisions. High level design decisions affecting overall competition strategy, including inputs and outputs, are defined by the design graph. Low level decisions affecting individual subtask strategies are considered by the relevant interdisciplinary team. The ensuing sections document both high-level and low-level design decisions as well as provide empirical results for current subsystems in the hope of future comparison.**

Fig. 1. A render of *Alfie 2.0*.

I. Competition Strategy

In 2019, MuddSub entered RoboSub for the first time with *Alfie*, an autonomous underwater vehicle. *Alfie* was designed with first-year constraints in mind, including limited funding, small team size, and limited pool test time. It was built to attempt a small number of the obstacles, but did so reliably. This year, we present *Alfie 2.0*, the second iteration of our robot. Compared to our previous robot, *Alfie 2.0* is designed under different constraints. For example, COVID-19 made it impossible for our organization to test *Alfie 2.0* the water. Under these unique constraints, MuddSub's competition strategy has evolved to set up our organization for long term competition success by catalyzing development of robust and flexible autonomous vehicles.

Rather than directly optimizing for points, *Alfie 2.0* focuses on what we call minimum-prior autonomy - the ability to operate under as little prior knowledge of the testing environment as possible while maintaining full functionality. The most compelling reason to adopt this architecture is the adaptability of MuddSub's work and generalization of tasks to RoboSub's changing ruleset, allowing MuddSub to transfer prior work to the next competition. Furthermore, the objective clarifies the extent of complexity required to make a reliable system. Over-complex systems tend to focus too much on task details, which goes against our objective. At the same time, under-complex systems do not have the power to generalize which is also counterproductive. Minimum-prior autonomy specifies a middle ground where the system is both sufficiently complex and reliable.

This objective may initially seem unrelated to RoboSub's point system, but it is in fact not the case. Between every competition year, there is a limited amount of time teams have to work on the robot. As such, the time teams have in a single year is not sufficient to develop a top-performing robot. However, if a robot is built under *minimum-prior autonomy*, the time spent each year can be allocated to fine-tuning the robot to the specific competition as general systems are already in place.

In spite of the challenges brought by a pandemic, MuddSub used this time to expand Alfie's capability and optimize for minimum-prior autonomy. This was driven by two factors. First is the expansion of our team. In the 2019 RoboSub on-site competition, our organization consisted of a mere 6 people. This small member-count served effectively to get a basic robot in the water through fast iteration and deployment. The sacrifice was in the breadth of the tasks - our organization only attempted the gate and buoy tasks. Our organization now consists of around 50 members, each with a variety of expertise and background. With more members on the team, MuddSub organized subteams of members each independently researching and prototyping for a single task. This allowed MuddSub to exhaust the RoboSub spectrum, meaning every obstacle in the competition was addressed. Second, due to COVID-19 restrictions, robots were not judged based on their performance in a testing environment, but rather evaluated from an developmental standpoint.

The rest of this article is organized as follows. In the design creativity section, we document the design decisions of our individual subteams. In the experimental results section, we document particularly compelling results with respect to relevant evaluation metrics.

II. Design Creativity

This section describes the motivations and decisions of MuddSub's subteams in developing specific systems to solve various RoboSub obstacles.

A. Computer Vision

In the RoboSub competition, it is crucial that the robot is able to sense the world around it. One of the most reliable ways of doing so is with cameras. In the past decade, neural models have become the state of the art for computer vision systems. For such models, model performance is commensurate with the quality and number of the training data.

MuddSub created a system for remote image labeling, which allows the team to quickly and accurately get bounding boxes for our computer vision neural networks to train on. Test images were divided into batches and assigned to two people each. A web server and a connected desktop app distributed those images to members of the MuddSub team to label remotely. To verify the accuracy of the labels, the two sets of labels are compared. If they are not close enough, the image was sent to a third person to gather evidence on which label was more desirable. Ultimately, this system ensured that MuddSub had accurate data to train neural networks on, and involved the whole team so labeling could be done more efficiently.

There are a variety of deep architectures to localize and classify objects. In order to determine the best model, the team considered four different neural architectures – YOLOv3, MobileNet, U-Net, and SqueezeNet. These architectures are chosen because they are single-shot detectors, which means that localization and classification are done within a single neural network. Single-shot detectors favor speed over accuracy. In the case of Robosub, this is favorable because external filters such as SLAM are used to remove noisy predictions over time.

After comparing these architectures, the team determined that the best model to use is a modified version of YOLOv3. Specifically, the model blends elements of YOLOv3, which had 100+ residual layers total with three detection layers, and YOLOv3-tiny, which consisted of about 20 residual layers and one detec-

tion layer. The resultant model is YOLOv3-medium, which uses about 40 residual layers and two detection layers. This network design allows the team to achieve both optimal performance and speed.

B. Simultaneous Localization and Mapping

Given the objective of minimum-prior autonomy, the team decided that SLAM was an optimal solution for robot localization, as it reduced the need for human heuristics through generalized environment mapping.

Given that Alfie's primary perception system uses cameras, a popular SLAM method to adopt would be visual SLAM. However, the team decided against visual SLAM and opted for FastSLAM to increase reliability, extensibility, performance, and learning opportunities.

Based on prior competition experience, the over-exposure of the afternoon light and the water quality degradation often rendered many images unusable, causing purely visual SLAM systems to become less reliable. To overcome this challenge, MuddSub opted for FastSLAM which depends only on the range and bearing information of the obstacles and not the particular sensor. This enables the team to diversify perception dependencies and increase system reliability. In addition, the team wants to be able to easily incorporate more cameras and a wide variety of sensors, such as sonar and hydrophones, in the future. As such, FastSLAM's sensor-agonistic feature also supports another design principle, extensibility.

The biggest reason for choosing FastSLAM 2.0 was performance. Through research, the team determined that among the more well known algorithms such as Extended Kalman filter (EFK) SLAM, FastSLAM is one of the best performing algorithms. It can match EKF SLAM with far less computational resources and with more noise data. Additionally, compared to FastSLAM 1.0, FastSLAM 2.0 can better maintain multiple hypotheses by maintaining particle diversity, which will be helpful in Robosub's challenging competition course.

Choosing FastSLAM 2.0 also provides a great learning opportunity. To understand the algorithm, the team was required to gain an understanding of many concepts, including EKF, particle filters, localization, and mapping. Moreover, FastSLAM 2.0 is an exciting implementation challenge: in the authors' own words, "FastSLAM 2.0's main disadvantage is that it is more difficult to implement" (Montemerlo and Thrun 63). Since there are not any existing implementations, the team was driven to learn through experimentation.

C. Navigation

Navigation is implemented via breaking high level commands into smaller setpoint goals. Through mapping obstacle commands to point locations in SLAM's map, *Alfie 2.0* is able to determine the paths of multiple goals and execute a predefined motion style: splines or sinusoidal. Harmonic motion was added to increase SLAM's known probability of an obstacle's location through changes in the robot's orientation and camera view angle.

The first iteration of the navigation system used Rapidly-Exploring Random Trees (RRT). The model used randomly generated nodes from the plant state to the obstacle's setpoint. Each segment of RRT's trajectory is then determined through a customized method that generates either an additional randomized node or the shortest distance between the current point and end goal.

The second iteration used A* , implemented in three dimensions, as it performed better on the map given by the SLAM. A* uses a heuristic function on a grid of nodes to find the shortest path to the end pose. This algorithm is optimized to deal with multiple start states and changes in the obstacle map as information is updated. However, as SLAM also gives navigation covariances for each of the given obstacles, this algorithm can be improved by adding a level of randomness to incorporate these probabilities and add portions of RRT.

D. Controls

The purpose of Controls is to take in a trajectory from navigation, and compute the optimal thruster forces to follow that trajectory. This goal is further broken into the subgoals of computing optimal wrench vectors and then breaking down those wrench vectors into the optimal combination of thruster forces. *Alfie* is overactuated; this means that any given wrench vector can be produced with only six of the eight thrusters. With this characteristic in mind, the thruster allocation is designed to minimize strain on any individual thruster and to cope with potential thruster failure, thereby enhancing reliability.

The first control algorithm used to calculate wrench vectors was a proportional–integral–derivative controller (PID). The team used one PID controller for each of surge, sway, heave, roll, pitch, and yaw. These controllers calculated the error between the plant state and target state, and the derivatives and integrals of those errors. They then returned a weighted sum of those three values as the corresponding entry in the wrench vector. However, this algorithm has two deficiencies. First, this algorithm cannot plan for the future trajectory beyond the target state. Second, the ideal coefficients used in this algorithm have to be experimentally determined, and are dependent on the physical structure of the robot.

In order to overcome these shortcomings, the team explored the use of a model predictive controller (MPC). MPC looks at multiple time steps through the trajectory, and uses a model of the system to predict how well different sets of control inputs would follow that trajectory. The controller then optimizes a set of control inputs over some small window of time. Once optimization is complete, the robot takes only the first step of the optimal set of controls. At the next time step, this entire process is repeated. The primary concern with MPC is the computational intensity of the controller; however, the team can tune various parameters, such as the prediction horizon, to reduce that intensity. Since the system is so complex, another difficulty is the creation of an accurate model. However, the team finds this trade-off acceptable, since this model can be mathematically approximated, and does not have to be experimentally determined every time the structure of the robot changes in some way.

E. Mission

Mission is *Alfie's* planning system, which takes in perception information and outputs action commands. In order to design the system, the team adopted the philosophy that use cases inspire design principles and design principles guide system architecture and implementation. One main use case is relating to safety and time constraints: when the diver activates the kill switch, *Alfie* must respond immediately, and to recover without having to be reconnected to a computer to save competition time. Another use case is that sometimes, the camera cannot capture any obstacle, so *Alfie* should develop a routine to find the obstacle accordingly. These two use cases translate into the principle of reliability: stop motion from kill switch, recover from kill switch, and recover from uncertainty due to lack of perception information. The last use case the team emphasizes is coding efficiency. During the last in person competition, the team found itself writing many of the same routines for different tasks, and modifying multiple similar areas in our code when a part of planning heuristics changed. This use case calls for code modularity and reusability.

With the principle of reliability and modularity in minds, the team adopts a hierarchical state machine that is based on the ROS SMACH library. First, this architecture consists of a kernel layer, which responds to start and kill switches and their resets. This addresses reliability by providing an infrastructure for restarting after the kill switch is activated. Next, the mission layer outlines a series of tasks, where each task is represented by a task layer. Task layer represents the actual task *Alfie* needs to perform, such as going through the gate.

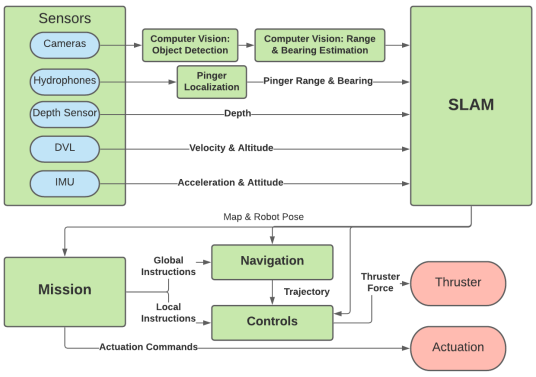


Fig. 2. An improved schematic of how information flows between subsystems after scoping.

Each task is decomposed into three components: locate obstacles, go to obstacles, and perform task specific behaviors. In particular, the first two components are shared across all tasks, obtaining modularity and reusability. Finally, each component above is an action layer, which is a concurrent container with two states: a state that monitors for the kill switch signal, and a custom state defined to perform the said actions. This layer realizes concurrency that is necessary to stop kill switch activation, and thereby addresses the principle of reliability.

F. Integration

With an expansion of robot features, keeping track of each subsystem and integrating them become more difficult. For example, SLAM requires range and bearing measurements to obstacles as input and relies on the vision system to provide them. However, before the coordination between subsystems was established, the vision system was only able to output bounding boxes, which SLAM cannot use. In addition to some systems expecting incompatible inputs, the necessary inputs for each subsystem were subject to change as implementation approaches changed during research and experimentation. To avoid miscommunications and compatibility issues, the team conducted subsystem reviews to establish inputs and outputs, designed software communication channels with specificity, and implemented graphical user interface (GUI) tools to increase usability.

Starting from a broad overview of how information should flow between subsystems, each subteam was asked to give a list of the types of expected input and output data. Just a survey of subsystems was not enough as many subsystems changed and the gap between experimental code and integration-ready code was too wide. Thus, a second series of team-wide scoping meetings was scheduled to record updated inputs and outputs (Figure 2) as well as provide assistance on implementing subsystem communication.

As *Alfie* is based on ROS, the bulk of the communication between subsystems is done through ROS topics, which are channels that can be published to and subscribed to, and ROS messages, the data sent through those channels.

Following the established inputs and outputs, the team planned the flow of information to be specific and efficient using custom messages and topics. The team splits subsystem outputs so that the clients of a message do not need to be familiar with the system that provides it. For example, instead of having a single “depth_sensor” topic that contains all of its output, the team uses “depth_sensor/depth” and “depth_sensor/pressure”, making it clear that the depth sensor provides pressure information in addition to the depth measurement. Another way to improve code readability and reusability is by strategically publishing processed data. Publishing to ROS topics is very computationally light and data transfer only occurs when there are subscribers to the topic, making it a better option to publish some of the outputs in a preprocessed format. Therefore, as long as the processed data is useful to at least one subsystem, publishing helpful data is at least as efficient as having each subscribing subsystem process the data internally.

G. Hydrophones

The main goal for hydrophones is to locate the pinger and use this information to estimate the relative position of the robot. For context within the competition, the pingers in the pool send out pulses of sound between 25kHz to 40kHz and if implemented and denoised correctly, the hydrophones can help derive the position of the robot with respect to the pinger source.

The analog signal goes through a two stage front-end including a charge amplifier and a multiple feedback band pass design. The team implemented a charge mode amplifier as our first stage in order to produce an output voltage that is proportional to the charge generated by our hydrophones. Charge amplifiers are useful to amplify signals with piezoelectric transducers, such as hydrophones, as charge amplifiers minimize the effects of interconnect capacitance.

Additionally, for our second stage, the team decided to use a multiple feedback band pass filter for 25kHz to 40 kHz instead of other common architectures like Sallen-Key due to its ability to achieve a high quality factor and high noise gain. Since the ADC expects a differential input, the second stage additionally converts the single-ended signal to a differential signal. In addition to the analog filtering, some other important features in our hydrophone board include:

Choosing an ADC, specifically the ADAR7251, that has two programmable gain stages to allow the team to adjust the amplification depending on the distance from the pinger. The team also included a subcircuit for a battery voltage monitor by using a voltage divider and a simple buffer with an RC anti-aliasing filter. And lastly, the four channels are sampled synchronously.

H. Motherboard

There are many communication ports between the computer and peripheral devices. In order to organize communication lines, regulate power, and provide additional I/O ports, the team was tasked with designing a motherboard. The motherboard requires a steady DC power supply; the power will be provided by the power distribution system and PCB, as discussed in the Power Distribution section. This motherboard's functionality includes PWM/UART/I²C communication for sensors, a battery voltage monitor for the two robot batteries—Turnigy High Capacity 10000mAh 4S 12C Lipo Packs, thruster kill signals, and communication to the Teensy 4.1 microcontroller and the computer, a Jetson AGX Xavier. The motherboard was designed in Altium Designer to address space constraints and simplify cable connections.

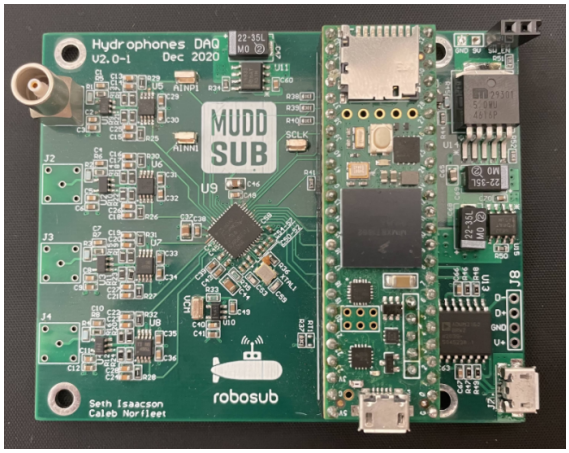


Fig. 3. Hydrophone board layout.

The current sensors include Blue Robotic's Bar02 Ultra High Resolution Depth/Pressure sensor, VN-100 IMU, Waterlink's DVL 450, and a custom hydrophone array (see Subsection 3.8: Hydrophones). These sensors require different types of connections from I²C, Logic-Level 3.3V, analog, UART for the Doppler Velocity Log (DVL), and PWM signals for LEDs, servos, and Blue Robotics ESC and T200 thrusters. One of the main reasons the Teensy was chosen as the motherboard microcontroller was due to its expansive and diverse set of communication I/O ports compared to other microcontroller platforms, such as Arduino. In addition, the Teensy has a smaller footprint, is easy to connect to the Jetson through USB, and

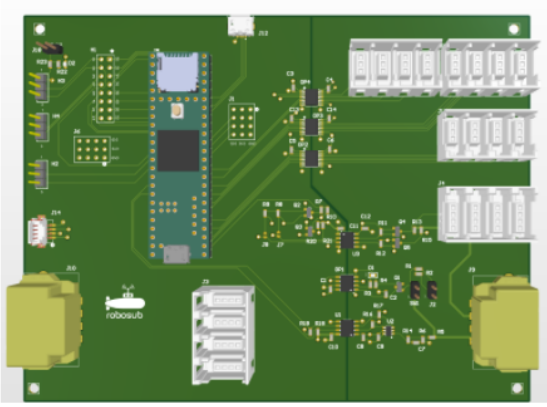


Fig. 4. Render of motherboard PCB.

has great CPU performance due to the ARM Cortex-M7. There are five main subcircuits within the motherboard: the PWM isolation, I²C level shifters, battery monitor, and circuits for the kill and start switch. One of the central features of the motherboard is a digital isolation line that separates the connections between the computer and the thrusters. Specifically, optoisolators were used in order to eliminate potential noise from ground loops between the analog and digital signals. In order to prevent the electromagnetic interference generated by high frequency PWM signals from interfering with other digital signals, a digital optoisolator is used before connecting to the Teensy. The thruster battery monitor first uses a voltage divider to step down the battery voltage to around 2.5V before going through a low-pass filter in order to be in the correct voltage range for the ADC while filtering out potential high-frequency noise. Then a digital isolator is used, outputting a I²C signal for the Teensy. The third main subcircuit is the I²C level shifter, which provides 3.3V and 5V ports for both power sides of the isolation line for the many sensors that operate using I²C serial communication. The motherboard includes additional I²C ports to account for additional sensors, and to create a more modular and robust system. The kill switch is connected to the HDC60D120 Solid State Relay as further examined in the Subsection 3.10: Power Distribution section, and the current is passed through a N-channel MOSFET. A digital isolator is used as the relay can potentially cause voltage spikes which can disrupt the kill switch signal before passing into the Teensy. The start switch simply consists of current limiting resistors with an indicator LED.

I. Power Distribution

The team's power distribution system consists of a high-current physical subunit and a low-current PCB subunit. The physical subunit is responsible solely for supplying power to the thrusters, whereas the board subunit is responsible for powering the computer, sensors, and other electronic components.

The mechanical component of the system consists of a large solid-state relay that activates and deactivates the robot thrusters in response to the kill switch. Power is then routed via thick copper bus bars through a bank of eight separate fuses. This circuit is also constructed to protect the ESCs from damage in the case that a motor is caught or malfunctions, drawing excess current. Although it would be simpler to use a single fuse for current protection, as the team has done in the past, having separate fuses for each thruster allows the robot to continue functioning on its remaining thrusters even after blowing a fuse. Using glass fuses allows for easy detection of blown fuses, and having an easy-to-access fuse and connector bank allows the team to quickly swap fuses and/or ESCs when necessary.

The printed circuit board component of the subsystem is tasked with accepting battery power and level shifting the supply voltage down to power the computers and sensors, specifically to 3.3V, 5V, 9V, and 12V. This is especially important because the variety of sensors and electronic components within the robot run relatively standard, but different voltages. This requires a variety of level shifters and additional wires throughout the robot unless voltages are regulated at the source. Therefore, the team determined that a circuit board

should be constructed to level shift the battery voltage down to appropriate levels and provide a bank of power ports such that voltage regulation is minimized elsewhere in the electrical system. The system does so by using switching voltage regulators, which were chosen for their significantly better efficiency at medium-low current draws. The array of switching voltage regulator circuits is routed to a bank of connectors so that components requiring different supply voltages can be powered directly from the power distribution board, and components can be added and replaced with ease. The subcircuits for level shifting to each voltage are rated for 8 amps to allow ample margins for potential inrush current spikes.

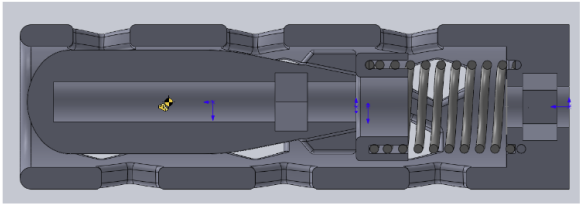
J. Torpedoes

Torpedoes worked to launch a maximum sized 2" by 2" by 6" torpedo through a 5 inch hole from three feet way. Initially, the torpedo subteam worked on a self motorized torpedo in order to avoid slight kickback on the robot. However, after testing, complications arose due to difficulties waterproofing the seal between the driveshaft and the torpedo body. Due to this shortcoming, the team decided to develop two additional prototypes: flywheel and spring.

The flywheel design used two wheels that would spin using a waterproof motor and propel the torpedo using friction. This allowed the torpedo body to be smaller and more hydrodynamic. The spring design included a compressed spring and tube. This design had similar benefits to the flywheel, the size of the body could be decreased due to a lack of onboard electronics. After completing rough prototypes of all three designs, they were evaluated using the same criteria.

The main criteria was consistency, accuracy, and ease of use. Applying these criteria, it was clear that the motorized torpedo was not a viable option. Between the flywheel and spring design, the spring prototype was the better option. It was the most consistent, accurate, easy to fire and simple. Thus the team decided to focus on the spring design.

Fig. 5. Sliced view of torpedo subsystem



Our main goal was to perfect the whole spring mechanism. Hexagonal holes were added to prevent negative pressure. It was found that if there were no holes in the tube, a negative pressure would exist behind the torpedo and it would fail to launch. Besides the spring, all components were rapidly prototyped through a 3D printer. Different torpedo bodies were tested to find out which would be the most hydrodynamic. Torpedos with a dull nose and fins flush to the body provided the best results.

K. Gripper

The gripper system was a new addition to *Alfie 2.0* this past year, expanding on both functionality and ability to interact with the RoboSub course. The goal of the gripper was to pick up and drop game pieces and to interact with lever handles on the course.

The first design iteration was an arm with two actuation joints. The arm could be actuated relative to the belly pan and a finger on a hand could be actuated to grasp game pieces. This first prototype was a proof of concept for aspects of the design such as surgical tubing for maintaining a closed position when unpowered.

The second design iteration had many modifications. The arm and hand were made more compact, reducing the gripper's size and impact on the robot's movement during actuation. In addition, this design included a sensor to detect game pieces within the gripper. The second gripper iteration also included a second finger, allowing it to open wider when grabbing a game piece. Due to the variety of possible objects that the gripper might need to interact with, this iteration was designed with modular fingers.

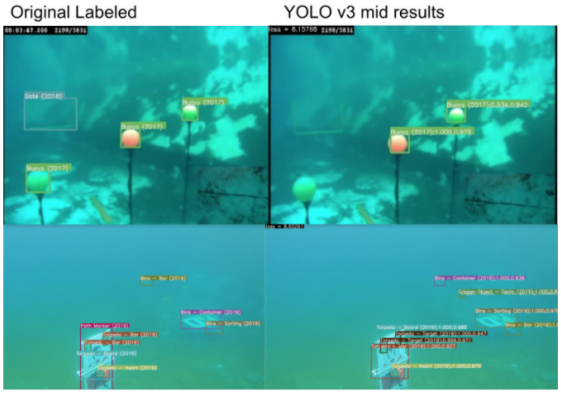


Fig. 6. Example Modified-YOLOv3 Predictions.

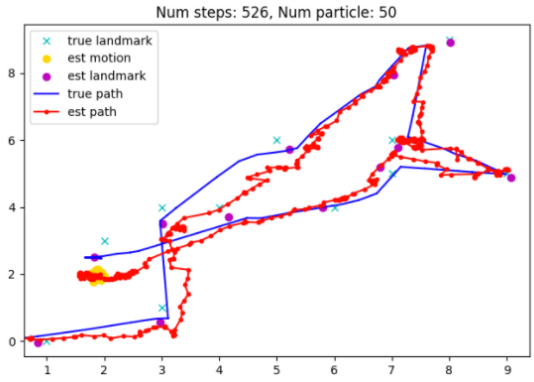


Fig. 7. FastSLAM path estimation on test datasets.

This improves the gripper's likelihood of a successful pick up since there is more area over which it can pick up a game piece. Lastly, the second design also included an electromagnet locking mechanism to hold the gripper against the belly pan when not in use.

III. Experimental Results

This section documents the experimental design and performance of several subsystems.

A. Computer Vision

To train the networks, the team used the previous competition images and then used the labeling software to mark bounding boxes and categorize objects. The initial results from training on this dataset were less than satisfactory. As a result, the team focused on

image pre-processing and data augmentation. The team implemented data augmentation techniques including cutout, cut-mix, flips, rotations, and mosaics, mixing and matching to train each network. Generating this augmented data during training, while computationally expensive, allowed the team to constantly be feeding the network novel images to train on, resulting in little to no overfitting of the models even when running for thousands of epochs.

The team analyzed the performance of neural networks using a wide variety of metrics, including classification accuracy — both on average and broken down by class — detection accuracy, detection precision. The team mostly prioritized detection precision, since erroneous detections can mean the robot will make unfavorable decisions.

The final model has around 60% precision and 85% class accuracy, but during the best training session, the team achieved 70% precision and 90% class accuracy. This is below the 90% precision and 95% class accuracy that the team hopes for, so the team will continue to experiment with the YOLOv3 architecture.

B. SLAM

Since the pandemic limited the team's ability to work with the robot, the primary methods of testing were through existing datasets and simulations. The team used the multi-robot cooperative localization and mapping (MR.CLAM) dataset, published by the University of Toronto Institute for Aerospace Studies. This consists of sets of bearing and range measurements, as well as velocity commands for the robot, which ware ideal for FastSLAM. The team also developed a simulator with similar data, which has the additional advantage of demonstrating how a robot will respond using the map SLAM produces. Figure 6 shows an example simulation, where red is the estimated path that fast slam calculates, and blue is the actual path that robot takes using the map fast slam generates. In this simulation, the algorithm initializes landmarks noisily, and the robot is tasked to

move toward the landmark until it hits a close-enough zone. As shown in the diagrams, the robot had reached the landmarked area using the map generated by SLAM. In addition, the trajectory slam generates closely matches the real trajectory. Moreover, during the simulation, the landmark poses are corrected as the robot moves toward that region. This gives the team the confidence that this algorithm will perform well in real competition.

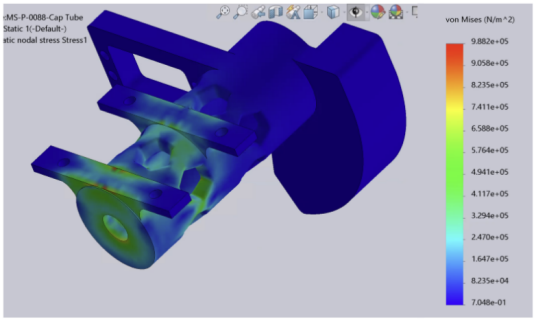


Fig. 8. Torpedo Launch Mechanism FEA Analysis

C. Navigation

We considered four different heuristics for the A* pathfinding algorithm: Euclidean, Chebyshev, Octile, and Manhattan. These methods are evaluated based on the accuracy and efficiency. Accuracy of each function was calculated by comparing the length of the path's output compared to the distance calculated by an exhaustive search algorithm. Efficiency is determined by comparing the number of visited nodes between the algorithms.

Both Euclidean and Chebyshev obtained the shortest path, determined by exhaustive search, 100% of time. Octile Distance and Manhattan Distance produced paths that were, on average, 0.58% and 2.50% longer respectively. In terms of efficiency, compared to Euclidean Distance, Chebyshev Distance and Octile Distance 230.75% and 13.75% worse while Manhattan Distance used 64.75% less amount of cells. As such, the team decided to use Manhattan Distance as it has an extremely high efficiency rate and a relatively small accuracy error.

D. Torpedoes

There were two rounds of tests for the torpedo design. The first round was in COMSOL, a fluid dynamics simulator. After parameters such as drag coefficient were tuned, the second round of tests were underwater.

This suite of tests revealed two things. First, the speed of the motorized torpedoes were slower than anticipated. Second, both the flywheel and spring designs launched the torpedoes at higher velocities, and travelled farther and straighter than the motorized torpedo. Finally, some of the torpedo bodies were reveled to not be waterproof, which can be destructive to the interior electronics. The team ended up choosing a spring launching mechanism.

After deciding on the final torpedo body and launching mechanism, the team conducted more fine-tuned tests. In addition, with the new spring release tube, the team ran SolidWorks Simulation on it to confirm that it would not compromise due to the force from the spring. This was confirmed through the simulation because there was minimal displacement after the spring force was applied to the back of the tube. This aligned with the tangible results, as there were reports that the 3D printed tubes were not noticeably deforming during the launches. Testing will continue with regard to the three release mechanism designs, but the testing methods should remain the same.

IV. Acknowledgements

MuddSub would like to thank all the individuals who contributed to the continued development of *Alfie*. We would like to thank Harvey Mudd College's Shanahan Fund, which served as our primary source of money. We would also like to thank Oliver Kwan who funded MuddSub for the entire summer. Last but not least, we would like to thank our advisor Zachary Dodds.

V. References

Howard, Andrew G. , Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun

Wang, TobiasWeyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural net-works for mobile vision applications. 2017.

Iandola F.N., Han S., Moskewicz M.W., Ashraf K., Dally W.J., Keutzer K. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size.

Leung K Y K, Halpern Y, Barfoot T D, and Liu H H T. “The UTIAS Multi-Robot Cooperative Localization and Mapping Dataset”. International Journal of Robotics Research, 30(8):969–974, July 2011.

Montemerlo, Michael, and Sebastian Thrun. Fastslam: a Scalable Method for the Simultaneous Localization and Mapping Problem in Robotics. Springer, 2010.

Redmon, J. and Farhadi, A. (2018). YOLOv3: An Incremental Improvement.

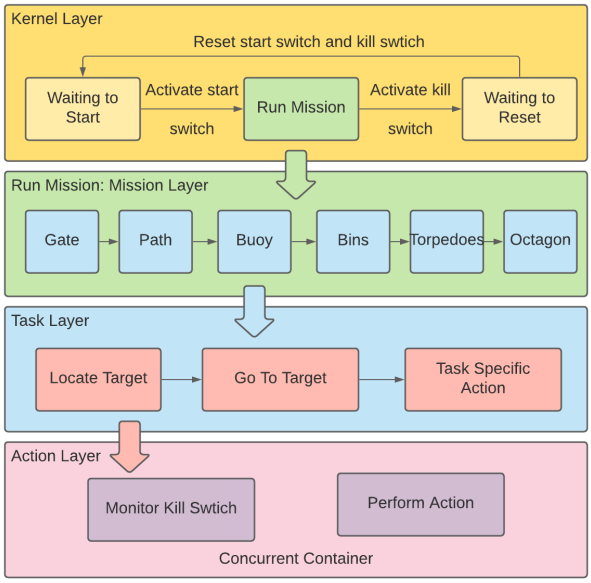
Ronneberger, Olaf; Fischer, Philipp; Brox, Thomas (2015). "U-Net: Convolutional Networks for Biomedical Image Segmentation".

APPENDIX A: COMPONENT SPECIFICATIONS

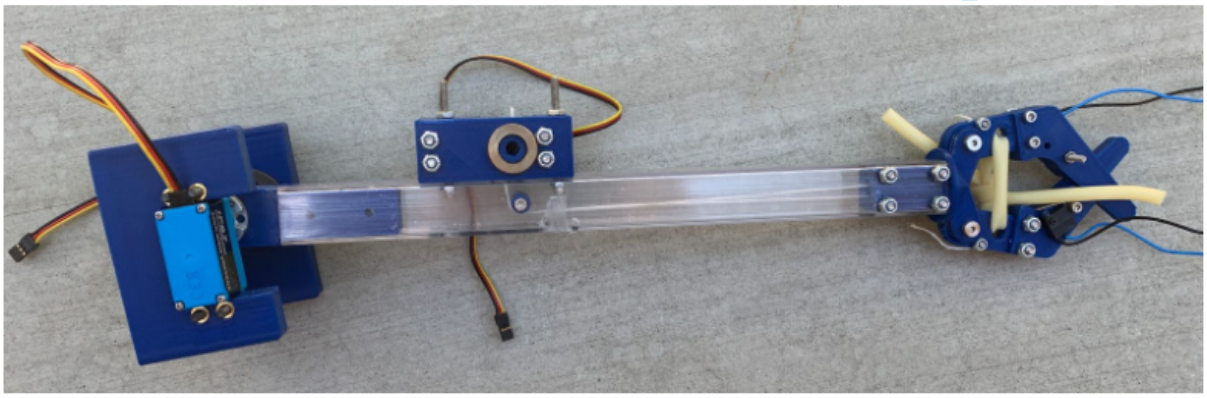
Component	Vendor	Model/Type	Specs	Cost(if new)
Buoyancy Control		Dive Weights		
Frame	In-house machined			
Waterproof Housing	In-house machined			
Waterproof Connectors				
Thrusters	Blue Robotics	T100		8x$119
Motor Control	Blue Robotics	BasicESC		4x$27
High Level Control				
Propeller				
Battery	HobbyKing	Turnigy	4S, 10 AH	2x$99
Converter				
Regulator				
CPU	NVIDIA	Jetson AGX Xavier	30W, 32TOPS	$650
Internal Comm Network				
External Comm Interface				
Programming Language 1	Standard C++ Foundation	C++17		Free
Programming Language 2	Python Software Foundation	Python 3		Free
Compass				
Inertial Measurement Unit	VectorNav	VN-100T		Donated
Doppler Velocity Log	Waterlinked	DVL A50	0.1mm/s Resolution	$5200
Camera(s)	FLIR	Chameleon 3	1.3MP	2x$310
Hydrophones	Teledyne	TC-4013		4x$1443
Manipulator				
Algorithm: Vision	Conv Net			
Algorithm: Acoustics	Unitary ESPRIT			
Algorithm: Localization and Mapping	FastSlam 2.0			
Algorithm: Autonomy	state machine			
Open source software	Canonical Ltd.	Ubuntu 20.04		Free
Team size	48 team members			
HW/SW expertise ratio	1:1			
Testing time: simulation	0			
Testing time: in-water	15 hrs			

APPENDIX B: ADDITIONAL FIGURES

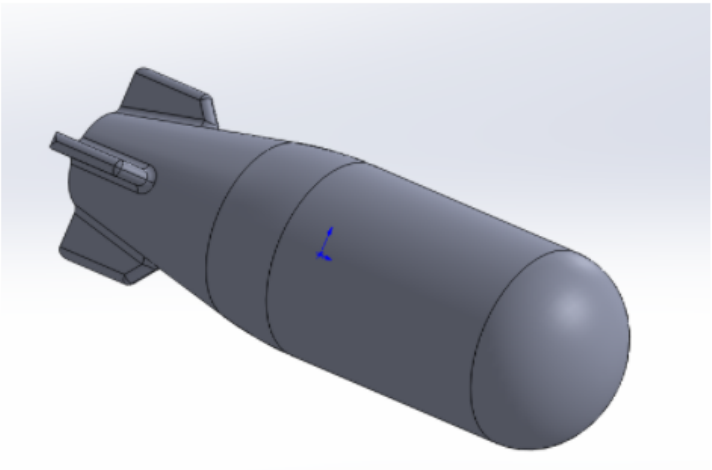
Alfie's Mission block diagram.



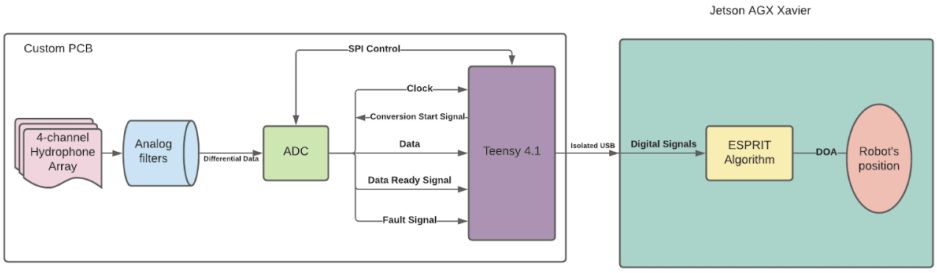
Alfie's gripper mechanism.



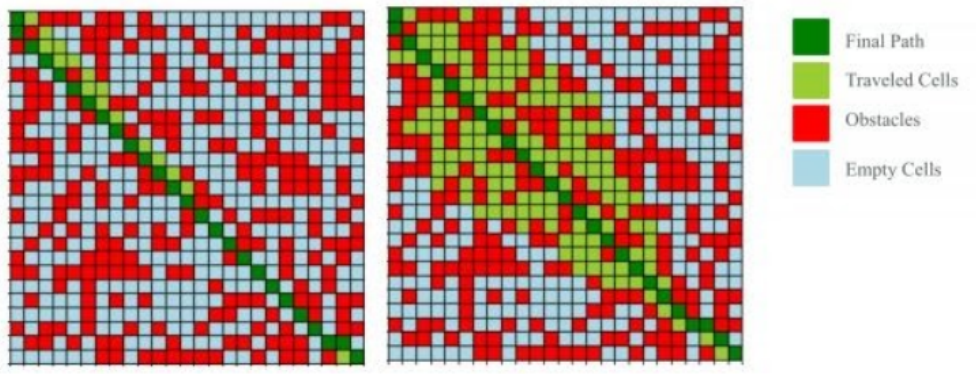
Torpedo body design.



Torpedo Launch Stress Analysis

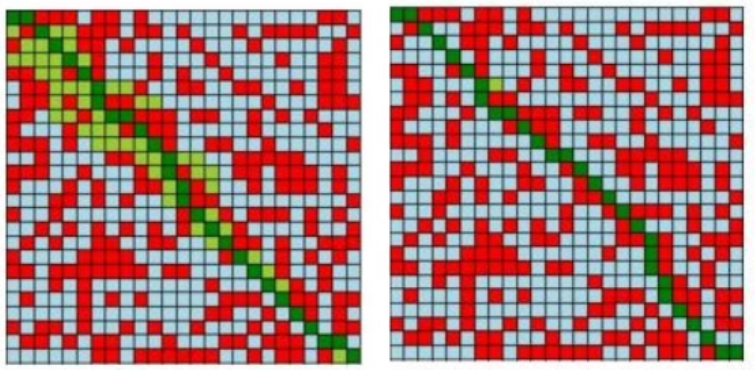


Results of various navigation models.



A. Euclidean Distance

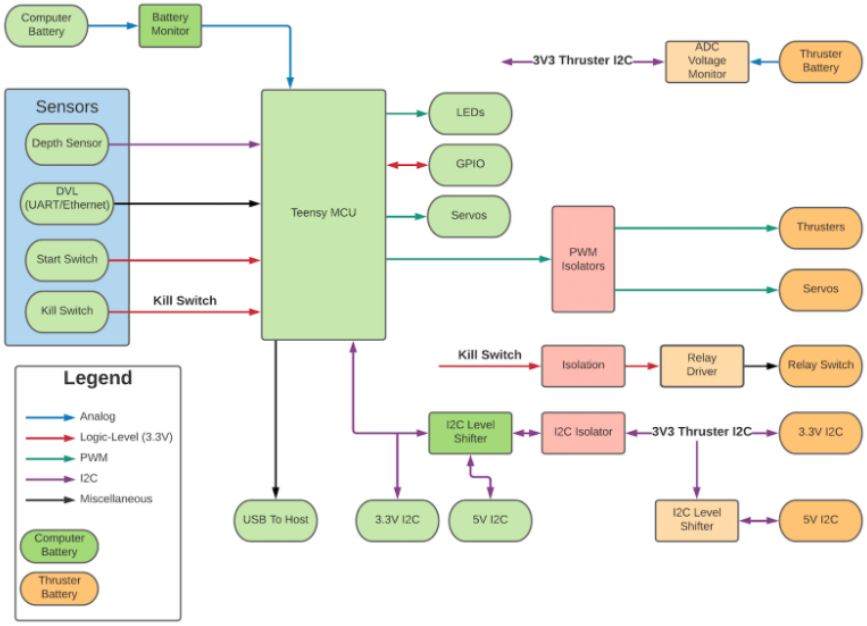
B. Chebyshev Distance



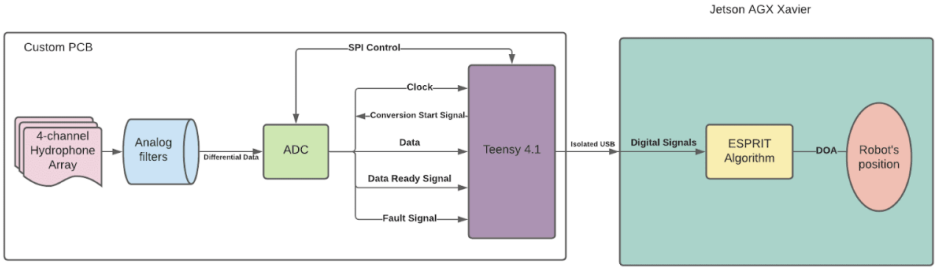
C. Octile Distance

D. Manhattan Distance

Schematic of motherboard.



Hydrophones information pipeline



Underside of *Alfie* with DVL, Gripper, Torpedoes, Hydrophones.