Georgia Tech Marine Robotics Maritime RobotX Challenge 2018

David Dulle, James Kenny, Patrick Meyer, Coline Ramee
Rahul Rameshbabu, William Roberts, Vinayak Ruia, Anthony Velte
Michael Steffens

***Abstract*—This paper describes the design of Georgia Tech Marine Robotics entry for the 2018 Maritime RobotX Challenge. By focusing on consistency and reliability in both our hardware and software, we look to build upon our experiences from the 2016 competition. This strategy led to the decision to make incremental improvements to the hardware platform and significantly overhaul to the software used at the 2016 competition. The sensor package remains largely the same as was used in 2016, with the removal of the mast used for the detect and deliver task as the most major update. On the software side we have transitioned to ROS in place of the custom software environment used in previous competitions. This decision was made for interoperability and integration reasons, as the open source nature of ROS is expected to encourage collaboration with other teams and ease the learning curve for future team members. Additional software changes include a focus on navigation capabilities in state estimation and a neural network approach to modeling vehicle dynamics.**

I. Introduction

The 2018 Maritime RobotX Challenge presents a difficult challenge for engineering teams pushing themselves to further the state of the art in autonomous marine systems. The Georgia Tech Marine Robotics team is excited to once again take part in this competition and test our system against the difficult course. This challenge provides and opportunity for our students to develop skills in systems thinking and design. The students who partake in this competition take these experiences and continue pursuing research in broad areas, from vehicle design to artificial intelligence. The challenge also provides an avenue to develop and test an autonomous surface platform, ensuring the platform has autonomous capabilities necessary to navigate complex environments and make decisions based on sensor information. These aspects make participation in RobotX an invaluable educational experience.

As the complexity of the challenge grows, it is necessary to develop a strategy to ensure the effort of team members results in maximum payoff during the competition. Our design strategy focused around the consistency of the system and is discussed further in Section II: Competition Strategy. The implementation of this strategy towards our hardware and software solutions is discussed in Section III: System Design. Testing of our system to ensure its consistency is discussed in Section IV: Experimental Results.

II. Competition Strategy

The primary strategy we have taken in working towards this year's competition is consistency, consistency, consistency. This strategy is born from observations made by members of the team at past AUVSI marine robotics competitions. Namely, the most successful teams are not those that do everything halfway, but those who do a few things very well. This focus on consistency can be seen throughout our design process, from the development of our software to the testing of our system. This strategy is also expected to yield large benefits to at competition development. Because our focus has been in increasing the reliability and robustness of the platform, development efforts at competition can focus on increasing capabilities of the system. This is a lesson taken directly from the 2016 challenge, where much of our team's time was taken in debugging our hardware and basic operations instead of enhancing the artificial intelligence required to complete the tasks.

Because of this guiding focus, when faced with a trade between reliability and complexity, we nearly always chose reliability. As with many strategies, this was a guiding idea, not a hard rule. In some cases, increasing complexity at the cost of system reliability was required. Some specific cases of this include navigation, discussed further in Section III: System Design. These cases are definitely exceptions though. In general, when faced with two possible solutions to a problem, the simpler solution was accepted.

Our second guiding strategy for the competition was to develop a robust navigation platform. While the tasks provided by the technical staff require significant perceptual capabilities, a robust navigation capability allows for the completion of the majority of the tasks. Similarly, the addition of obstacles throughout the competition area leads to the observation that autonomous navigation is focus of the competition. Finally, navigation capabilities are unlikely to have large changes required at the competition. While the sea states at the competition site are likely to be more severe than what is possible to replicate at our primary testing location, we expect these to minimally impact our navigation capabilities due to our modeling approach and control design. These are both dicussed in further detail in Section III-B: Software Design.

A third guiding strategy for the team was the elimination of so called "black boxes" from the system as much as possible. The idea is simply that the team should be able to access and alter any hardware and software component used on the vehicle as much as possible. A result of this goal was a major change to the software architecture used on the system. The team made the decision to shift from a software stack developed at Georgia Tech, ARCS [1], to the open source architecture ROS [2]. This change is discussed further in Section III-B: Software Design. Primarily, this change built upon the positive experiences new team members had in developing software in ROS for RoboSub 2018 compared to developing in ARCS for RoboBoat 2018. Additionally, the use of ROS brought our team in line with the larger robotics community, allowing for reuse of libraries and the potential for our advances to be shared as well.

A major exception to this guiding principle against black boxes in the system is the selection of motors for our vehicle, the Torqeedo Cruise 2.0. While these motors communicate over RS-485 and provide significant information about their current status, this communication is done over a proprietary communications packaging protocol. As such, our ability to communicate with the motors is limited to simple motor commands that are handled by an on board controller. More on this can be found in Section III-A: Hardware Design, but this illustrates our final guiding principle: when there is no feasible solution that avoids a black box, use what is familiar. We have used these motors with the WAM-V platform since our first work with the vehicle. Though not perfect, our familiarity with these motors and their performance relative to others makes adjustments not worth the effort required to develop a new solution.

III. System Design

As required by the competition rules, our vehicle utilizes the Wave Adaptive Modular Vessel (WAM-V) platform developed by Marine Advanced Research Inc. This vehicle was outfitted with two additional buoyancy pods at the rear of the pontoons to provided additional safety margin for the motors, batteries, and other equipment necessary to compete in the RobotX challenge. The vehicle subsystems are discussed further in Section III-A: Hardware Design. Compared with the incremental hardware changes that have occurred since the 2016 competition, the software used by the team has been replaced. The team decided to pursue the use of ROS as the backbone of the system software, in place of in-house software, ARCS. This decision, and further details on the software used in the vehicle can be found in Section III-B: Software Design.

A. Hardware Design

The vehicle used for the competition is the 16' WAM-V platform developed by Marine Advanced Research Inc., a catamaran style surface vessel. A payload tray is supported between the two pontoons of the vehicle and is used to hold a Pelican case containing the computing hardware. Two buoyancy pods are attached to the aft of the two pontoons. Two Torqeedo Cruise 2.0 motors are used for propulsion, each powered by a Torqeedo Power 26-104 lithium ion batteries. An additional battery is used to power the main computer and other sensors. With this configuration, we have been able to test in excess of four hours at a time without recharging.

The main onboard CPU is an Intel NUC, with Arduino Megas providing access to various sensors that cannot be directly accessed over USB or Ethernet. A pictorial layout

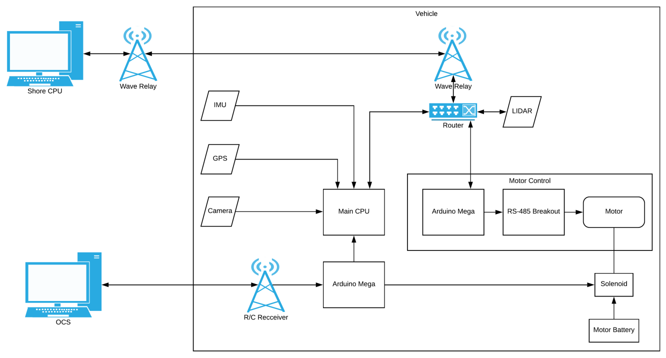

Figure 1: Basic configuration of onboard sensors and computing equipment.

of the computing equipment used on the competition vehicle can be found in Figure 1.

As was mentioned in the strategy section, our selection of the Torqeedo motors breaks our guiding principle of "no black boxes." However, this choice was justified as their performance was significantly greater than other feasible options. Many teams at the 2016 competition attempted to use off the shelf trolling motors, but struggled to provide enough propulsive power. At the same time, these trolling motors would present themselves in essentially the same way as the Torqeedos currently do, providing a basic interface for simplistic control. In this way, they would be no better than the Torqeedo motors. Other teams either used custom built propulsion solutions, a costly and time consuming effort that was not feasible for our team. Due to our familiarity with the Torqeedo motors, we decided to continue using them. To ensure safe operation of the vehicle, a simple kill switch circuit was developed using a solenoid to switch high current power to the motors using a control signal from the Arduino connected to our R/C receiver. If power to the main CPU is lost, the signal is forced to ground and current cannot flow through the solenoid. The same occurs if connection to our remote kill switch cannot be established, is lost, or is set to the killed position. The circuit is completed by the hardware kill switches placed around the vehicle, ensuring they operate as expected.

Communication with the motors is handled over RS-485. The packets transmitted by the Torqeedo motors provide a lot of relevant data including motor temperature, battery voltage, motor speed, and others, however this protocol is proprietary to Torqeedo and we have not been able to access it. As such, the only interface we currently have with the Torqeedo motors is a request for a throttle value, with no feedback from the motors. While this is frustrating, we are at a loss for how to improve the system. As it stands, we are investigating reverse engineering the protocol, but have had little success. It is hoped that the dynamic modeling approach we have chosen, detailed in Section III-B: Software Design, is sufficient to account for this lack of feedback. Additional problems that are associated with these motors include tight timing constraints on the communications and a general lack of obvious debugging capability. This problem was previously encountered during the 2016 competition, and has continued to occur. Testing of this, detailed in Section IV-B: Lab Bench Testing, revealed the timing issue was related to the RTS line of the RS-485 breakout board between the Arduino and Torqeedo. It is believed that this issue is due to the half-duplex nature of RS-485 implemented both on our end at the breakout board and on Torqeedo's end in their controller, but this has not been confirmed. This led to the implementation of a minor delay before sending a response to the internal motor controller providing a throttle command. While this seemed to resolve the issue in the detailed stress testing of the motors, it is kept in mind in case the issue appears again at the competition.

B. Software Design

Our approach to software took a more aggressive attempt to improve the consistency of the system. The most significant strategic shift was the change to using ROS as the backbone of the software stack instead of the previous custom software stack, ARCS. [1] This shift was taken for a few reasons, among them integration with a more mature software community. This is advantageous as it eases the learning curve when bringing newer team members onto the project. This was especially important, as the team is entering a phase of significant turnover as much of the initial team behind the custom software has moved on. This choice also brings our software stack more closely in line with much of the rest of the robotics community at large.

This process began in late January 2018, when the Georgia Tech RoboSub began transitioning to ROS for exploratory purposes while the RoboBoat 2018 team continued using ARCS. As both of these teams largely consisted of students that were not among the group that first developed ARCS, it would provide for a interesting comparison of the learning curve for each software stack. While ARCS has significant performance advantages over ROS in its tight integration and focused development, it was found that it was difficult for new students to pick up and provide significant contributions at the competitions. This can largely be attributed to a lack of documentation and somewhat ad-hoc nature of its development. While unfortunate, this provides a cautionary lesson to other teams developing custom software stacks. It doesn't matter how good your software is if others can't learn and contribute to it. The importance of documentation is difficult to overstate for teams that will experience any sort of turnover.

ROS on the other hand was largely adopted easily, and new students found the available documentation and community useful in developing new contributions. With this positive experience in mind, the decision was made to pursue ROS as the primary software stack for the RobotX Challenge.

C. Navigation

Along with increased support, ROS has many ready-to-use packages that can be implemented for robot control. Among these libraries is the navigation stack, a foundational component of ROS that has been very useful in developing our navigation capabilities. The ROS navigation stack allows for the relatively simple implementation of autonomous navigation and localization using prebuilt ROS packages.

The navigation stack uses sensors information and goal poses to generate a path to the intended goal using the Navfn navigation function. Paths are generated using Dijkstra's algorithm, information from local and global costmaps, and current sensor and localization information. Velocity commands are developed using the Dynamic Window Approach (DWA) local planner in ROS. The DWA planner generates several trajectories, applies forward simulation to predict motion over a short time period, and scores each trajectory on proximity to path, goals, and obstacles. Initial implementation and testing of the navigation stack was completed using the Virtual Maritime RobotX Challenge Simulation environment. [3] These tests are further detailed in Section IV-A: Simulation Based Testing.

In the current implementation, the velocity commands generated by the navigation stack are transformed into thrust commands sent to each motor and controlled with a PID controller. The primary purpose of this controller is to provide a tunable interface to translate the desired trajectory into motion of the robot. Initial testing has shown poor results in both real world testing and simulation-based testing. These experiments and past experience suggest that the system is simply too complex to be appropriately controlled by a basic PID controller on heading and velocity. These complexities are due in part to the transfer function between command and action of the motors involving significant non-linearities and a non-trivial ramping implemented by the onboard controller within the Torqeedo. A model-based controller that was previously used is being ported from ARCS and improved to allow for more accurate control. This controller uses a sampling based approach to generate vehicle trajectories. Control trajectories are sampled using a gridded approach and a dynamic model of the vehicle is used to predict the resulting vehicle trajectories. Based on these predictions, a near optimal control trajectory can be chosen simply by choosing the trajectory with lowest cost. This cost is defined by the required control effort and the confidence the resulting vehicle trajectory will remain collision free. While this implementation is ongoing, it is expected to be completed by the beginning of the competition.

D. Dynamic Modeling

In the past, our approach to dynamic modeling has been to assume a 2-dimensional model (ignoring pitch, roll, and heave). Unknown model parameters for mass properties and drag terms were inferred from raw data during an optimization process that concurrently tuned the dynamic model and an extended Kalman filter. Although this approach served us well, it falls short in two important respects. First, our model assumes nothing about wind or ocean currents - a factor that led to significant difficulty during the 2016 competition. Second, our thrusters have a noticeable ramping function and, because of the proprietary communication protocol, the true motor speed and thrust at any given time is unknown.

For the 2018 competition, we are taking a new approach to address these issues: training a neural network to infer a dynamic model from data collected during remote operations. In general, neural networks are useful when 1) data is cheap and abundant, 2) the form of the underlying model is not well

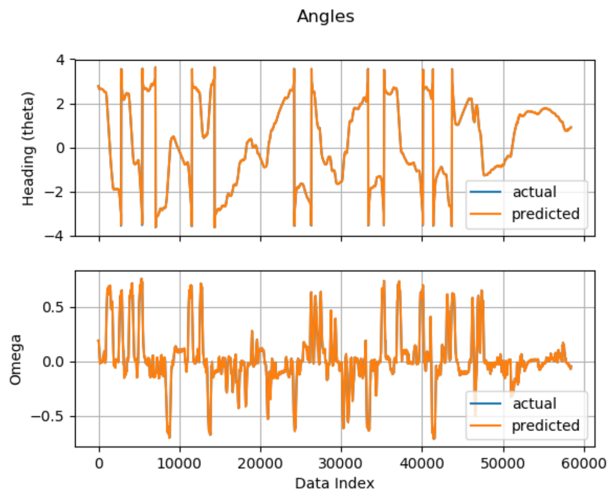

Figure 2: Actual and predicted heading and angular velocity from the neural network during a training run.

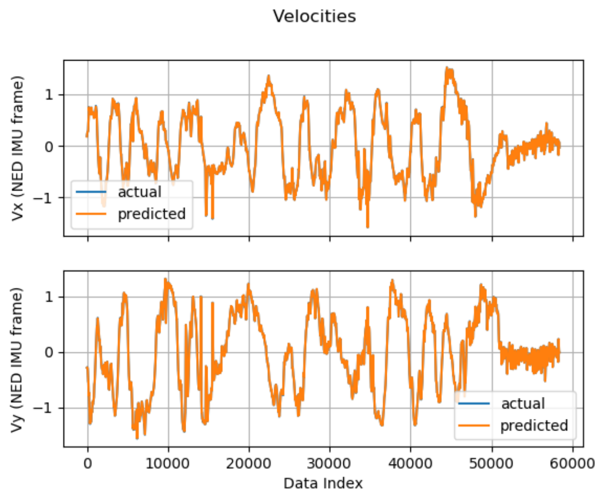

Figure 3: Actual and predicted velocities from the neural network during a training run.

understood, and 3) the utility and accuracy of the trained model is more important than its understandability. These criteria are met in our case (although the form of the equations of motion are described in [4], the ramping function for our motors is unknown).

The question our dynamic model should answer is, “Given an initial state and motor command at time t, what is the predicted state at time t+δt?”, or

$$\vec{x}(k+1) = f(\vec{x}_k, m_L, m_R, \delta t) \tag{1}$$

In practice, there is considerable flexibility in assembling input-output pairs for training from the raw data, and we found that supplementing the initial state with motor commands from the previous 0.5s and 1.0s was sufficient to capture the motor ramping function. We implemented our dynamic model in Keras (an extension of the popular TensorFlow library for Python). Finding a suitable architecture for any neural network can be a challenge, but after some experimentation, we settled on a two-layer network that offered high accuracy, as shown in Figures 2 through 5. The trained model can be saved for execution in both Python and C++ for integration with our state estimation and path planning modules.

E. State Estimation

The purpose of state estimation is to generate a hypothesis for the current pose of the vehicle. In our case, the states include position, velocity, heading, and angular velocity. For estimating the state of the robot at any given time, the vehicle will utilize an Unscented Kalman Filter to integrate predictions from the neural network dynamic model and measurements from onboard IMU and GPS. An Unscented Kalman Filter (UKF) is used in this case due to the complexity and non-linearity of neural network equations. A more popular approach, the Extended Kalman Filter (EKF), involves linearization of non-linear dynamics using Jacobians. There are two problems with this approach for our case. The first problem is that the equation of the neural network is much too complex,

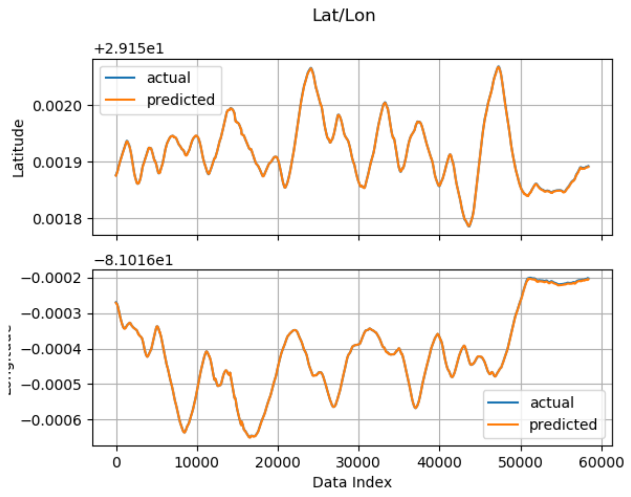

Figure 4: Actual and predicted latitude and longitude of the vehicle during a training run.

which means that analytically computing the Jacobean is not feasible. Second, the neural network is highly non-linear, which means that using a first order linear approximation could possibly result in an inaccurate state estimate. On the other hand, a UKF works by using a systematically sampled set of points to approximate the state rather than linearization.

The UKF and dynamic model are tuned concurrently in an optimization loop. First, the neural network is trained using raw data. Next, the neural network model is used to tune the UKF. Finally, the filtered data from the UKF is used to re-train the neural network and the process continues until both the UKF and the neural network have converged.

F. Perception

Vision and perception are implemented on the boat using a Velodyne Puck^{TM} (VLP-16) 3D LiDAR and USB web camera. The LiDAR sensor provides ranging information to populate the navigation stack costmaps with obstacle information. A

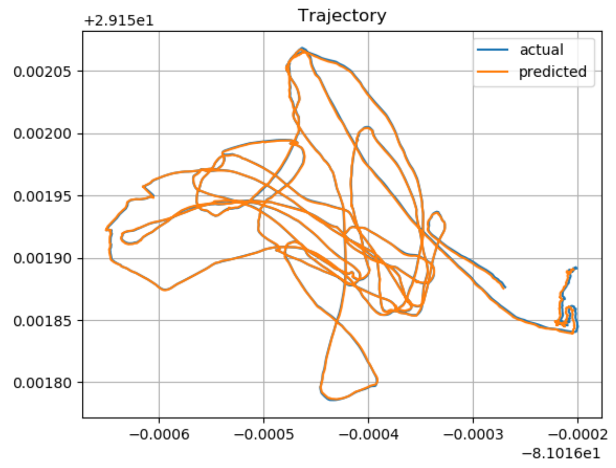

Figure 5: Actual and predicted trajectories, a composite of man queries to the neural network over a dataset spanning half an hour of remote operation.

method for buoy identification and classification is currently being developed to enable further autonomous missions involving decisions based on the type and color of buoy. Testing and training of the classification methods will be conducted on the simulation environment before deployment on the physical system.

G. *Mission Planning*

The complex mission architecture required to carry-out intricate autonomous tasks necessitates the use of a structured mission planning system. The mission planning system utilizes hierarchical state machines to track robot states and transitions between states. The foundation of these state machines is the SMACH ROS package that provides various debugging and introspection tools for state machines. Using this architecture complex missions can be broken into smaller components with many possible states and sub tasks. A task specific interface was defined in python based on the SMACH library to allow for easy integration of new task models as they are developed at competition.

IV. Experimental Results

To ensure our designs are adequate to perform well at the competition, a rigorous testing plan was laid out. This testing plan included simulation based testing, lab bench testing, and full system testing at Sweetwater Creek State Park near Atlanta, Georgia. The results of these tests are discussed below.

A. *Simulation Based Testing*

While ARCS included a simulation environment within the control environment, this is not the case with ROS. As such, the Virtual Marine RobotX Challenge (VMRC) environment set up within Gazebo was used [3]. This was developed to have an accurate representation of the vehicles hydrodynamic

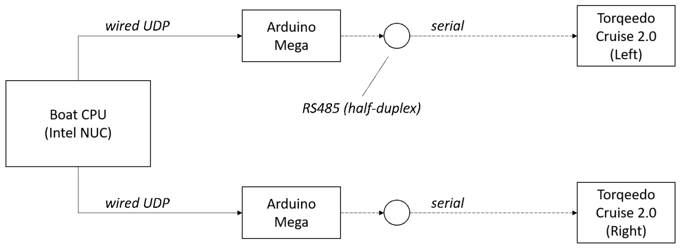

Figure 6: Motor commands are sent via UDP from the main CPU to a pair of Arduino Megas, which communication with the motors over RS-485 through a half duplex breakout board.

performance, propulsion performance, and relatively high fidelity sensor implementations. The environment has been used to provide initial tuning of the PID controllers used for control of the vehicle, as discussed in Section III-C: Navigation. This has also been used to develop proof of concept algorithms for perception as described in III-F: Perception. As our vehicle has already shipped at the time of writing, we plan to continue using the VMRC environment to further develop these algorithms.

B. *Lab Bench Testing*

Our success in the 2016 competition was hindered by unpredictable loss of one (and sometimes both!) of our thrusters. This was a difficult issue for us to fix even after returning from the competition as many factors were confounded to produce the failure. Motor stress testing was conducting by setting up an experimental test stand inside our lab environment. This stand allowed the motors to be run simultaneously, fully immersed in water, for extended periods of time. Stress testing was conducted by varying the commands to the motors along simulated control trajectories. Sinusoidal, step functions, and randomized control trajectories were used to elicit and identify failure modes.

After many hours of controlled motor testing in the lab, we traced the problem to the communications between the CPU and the motors. The failure was identified to be a timing related issue in the response providing a command to the internal Torqeedo controller. This controller follows a master-slave architecture for connection to many possible components, and so the failure of communications can be very sensitive to the timing of a response. It appeared this was causing the motor failures, and a simple delay before sending response appeared to solve the problem. This solution was verified with further bench testing. The communication architecture we have now implemented is shown in Figure 6. Motor commands generated from the controller module are transmitted from the CPU to a pair of Arduino Megas over UDP (one Arduino for each motor, assigned a static IP). The Arduino will wait to receive a packet from the Torqeedo, then transmit the motor command. Packets are converted through an RS-485 transceiver in between the Arduino and motor.

C. *Lake Testing*

Due to the WAM-V's size and complexity, it is difficult to conduct live testing. Additionally, Atlanta is a land-locked city.

The nearest body of water that the WAM-V can be deployed in is a roughly 40-50 minute drive out of the city. This means full system testing on the water is a rare resource. However, it was a focus for this years competition to increase in-water time to discover as many hardware issues as possible before the vehicle was shipped to Hawaii. Prior the 2016 competition, we had time to complete one three-hour fully integrated system test before shipping the vehicle. At competition, we were plagued with hardware failures. This year, the team logged six trips to the lake, each lasting the better part of a workday. This yielded a total of roughly 20-30 hours of in-water testing. In addition to logging hours in an attempt to catch rare errors, data was gathered to fit the neural network dynamic model, evaluate control algorithms, and gather realistic perception data for developing autonomous behavior algorithms. While nowhere near the level required to root out all problems, confidence in our system is much higher than it was in 2016.

V. Conclusion

The Georgia Tech Marine Robotics team is excited to participate in the 2018 Maritime RobotX Challenge. This challenge presents a difficult use case to further the state of the art in autonomy and marine navigation. We have participated in previous competitions including the 2016 Maritime RobotX Challenge, and the 2016, 2017, and 2018 RoboBoat and Robosub competitions. Each of these provides an opportunity to build upon previous experiences and further engineering skills. Each competition has brought about new challenges, such as navigation in environments without the aid of GPS in RoboSub, vehicle interoperability challenges in RoboBoat, and the challenging dynamics of the large vehicles used in RobotX. These challenges have pushed us as engineers to grow and provide innovative solutions.

The experience we have gained has pushed us to value reliability in our systems, favor simple solutions when possible, and avoid black boxes. We have taken these guiding principles and applied them to our vehicle development for the 2018 RobotX competition, leading to a relatively simple hardware system and the use of an open source software architecture. Consistency also drove the testing of our system with in-water time greatly increased relative to 2016, continued lab testing, and substantial simulation-based development. We look forward to seeing the results of all our hard work!

Acknowledgments

We would like to thank the GT Aerospace Systems Design Lab (ASDL) for supporting our work on this project. Within the ASDL, we would also like to give special thanks to the ADEPT Lab for use of space and manufacturing tools. Finally, we would like to thank all the previous team members here at Georgia Tech, your experience and guidance has been invaluable throughout this process.

References

[1] S. Seifert, P. Meyer, C. Ramee, E. Evans, W. Roberts, K. Griendling, and D. Mavris, "Arcs: A unified enironment for autonomous robot control and simulation," in *OCEANS 2017 MTS/IEEE Anchorage*, IEEE, 2017.

[2] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, Kobe, Japan, 2009.

[3] C. Aguero and B. Bingham, "Vmrc." https://bitbucket.org/osrf/vmrc, 2018.

[4] T. I. Fossen, *Handbook of marine craft hydrodynamics and motion control*. John Wiley & Sons, 2011.