Why Yes, That is Plywood: A Simulation Based Approach to RobotX 2016

Patrick Meyer, Samuel Seifert, Coline Ramee, Ethan Evans, William Roberts
Dr. Kenneth Cooksey, Dr. Kelly Griendling
Georgia Institute of Technology

***Abstract*—This paper describes the development of The Wambulance, a Wave Adaptive Marine Vehicle (WAM-V) adapted for participation in the AUVSI RobotX competition by the Georgia Tech ADEPT Lab. A highly capable simulation environment was developed in parallel to the hardware systems. This simulation environment was critical in allowing the rapid iteration of autonomous behaviors and other algorithms. This approach was validated in our successful participation in both the Roboboat and Robosub competitions earlier this year. Experience gained from both of these competitions has been incorporated in both the simulation and hardware systems. We have high hopes that this process will lead to further success at this year's RobotX competition.**

I. Introduction

RobotX is a biannual competition held over 9 days in Honolulu, Hawaii. The final day of the competition is reserved for the Final Round, with the preceding 6 used for practice and qualifying time. Even if a team is fortunate enough to average 3 hours per day of testing, teams will only have roughly 18 hours of on water time. This is simply not enough time to develop, debug, and tune an entire software stack for all of the tasks outlined in the competition, let alone solve hardware issues that may arise. To adequately prepare for the competition, teams are left with three main options to supplement this time: use of previous years' data, additional testing at a mock competition area, and software simulation. The most successful approach would be to combine the three options, though this is not always feasible. Unfortunately, data from previous years' competitions is limited. Additionally, the nearest suitable testing location is a 40 minute drive from the Georgia Tech campus. Collecting real world data at this site requires a significant time investment, and would be impossible to test on a day to day basis. Due to these challenges, the team has decided to augment our testing capabilities through the development of a comprehensive simulation environment, called the ADEPT Autonomous Vehicle Simulator (AAVS). Using this environment, different autonomy and control algorithms can be designed, implemented, and tested without needing to be at the lake. This allows lake time to be used more efficiently, verifying and tuning behaviors instead of creating and debugging them.

The simulation environment has proved incredibly useful in non-software aspects of the design process as well. Ambiguous hardware questions, like *what LIDAR should be used?* and *where should the hydrophones be mounted?*, can be answered through virtual experimentation. By running the entire competition in AAVS using different sensor configurations, the performance of each configuration can be compared using the scoring guidelines outlined by the RobotX rules as an objective function. This technique ensures we get the most out of the sensors we have, and can be used to determine which sensors will actually help improve performance of competition tasks before purchases are made. This same technique was employed in our successful participation Roboboat and Robosub earlier this year.

The remainder of this paper will outline the development of the hardware and software used to compete in this year's RobotX competition. This begins with a discussion of our overarching design philosophy in II. Design Strategy. Next, the implementation of this strategy in the development of AAVS and the vehicle hardware used is outlined in III. Vehicle Design. Finally, preliminary results available at this time are presented in IV. Experimental Results.

II. Design Strategy

RobotX is ultimately an autonomous vehicles competition. Though novel hardware systems can improve performance, the most important factors in having a successful system are the autonomous behaviors. With this in mind, the Georgia Tech team adopted a software first approach to the design process. This is similar to a common view in the UAV community, where the vehicle itself is merely a "truck" to get the payload where it needs to be. In this case, the behavioral algorithms themselves could be considered the payload. The "truck" necessary could be any maneuverable floating platform, provided it supports the correct sensors. This is particularly relevant for our team, as we are involved in other maritime robotic competitions and research, which require to use different classes of vehicles: a 16 foot WAM-V platform for RobotX, a smaller than 6 feet boat for Roboboat, an underwater vehicle for RoboSub. Each vehicle is capable of supporting a varied suite of sensors and actuators and runs the same software.

To further this software first view, it was decided early on that a fully featured simulation environment would be developed to aid in rapidly designing, testing, and iterating on behavioral algorithms. To further simplify the software stack necessary for competition, the simulation would also be used as the control software on the actual competition vehicle. The intent behind this simulation environment was to have a proving ground to test new ideas before going through the efforts of implementing and testing them on the full system.

This virtual testing capability greatly accelerated the behavior development process, but it is important to note that the simulation environment did not replace hardware testing. This follows from following two points:

1) What *does not* work in simulation *will not* work in the real world.
2) What *does* work in the real world *will* work in the simulation.

A key takeaway from this is that an idea being successful in simulation does not guarantee success in the real world. As such, it is important to maintain a semi-regular testing schedule that uses full system tests to validate and update the simulation with new data and eliminate algorithms that may only work in simulation.

It is also important to note the subtle differences between these two points that highlight different strengths of our chosen simulation based design approach. The first of these points reinforces the usefulness of the simulator as a proving ground. Though significant effort has been expended to accurately model the on-board sensors and dynamics of the vehicle, the simulator generally overestimates system performance. As such, ideas that have been implemented in the simulator without success are unlikely to perform any better in the real world, where additional noise and non-optimal conditions are constantly present. These ideas can then be eliminated from consideration. The implications of the second point are a little more subtle. One interpretation of this point can be taken is that the performance of behavioral algorithms that show success in the real world will show qualitatively similar performance in both the real world and the simulator. Therefore, ideas that have been implemented and validated in the real world can be compared in the simulator to differentiate their performance.

III. Vehicle Design

As the design philosophy described above is very much software first, this section will largely discuss the development and capabilities of AAVS. This includes a broad overview of AAVS itself and it's capabilities, the dynamic model and state estimator used, and a sampling of the implemented autonomous behavior algorithms. Following this will be a brief description of the hardware systems used.

A. Simulation Environment

AAVS was built from scratch in C# and utilizes the OpenTK library for graphics. Using data extracted from Google Maps, knowledge retained from previous competitions, and the preliminary rules for this year, the 2016 RobotX course has been laid out in the simulation environment as shown in Fig. 1. Using this environment to develop algorithms and the overarching software stack has proved invaluable.

Initial analysis showed four sensor subsystems were required to gather adequate information from the environment to complete the Roboboat tasks:

- GPS and IMU to determine vehicle state.
- LIDAR system to detect physical obstacles.

Figure 1: The 2016 Roboboat course in simulation.

- Camera system to recognize color and pattern.
- Sonar system to locate the pinger.

Models for sensors in each of these four categories have been developed for use in AAVS. These models have been validated against real world experiments to show they can generate representative data. In the current version of the simulator, simulated data for GPS, IMU, and LIDAR can be realistically generated while camera and sonar models remain at a lower level of fidelity. Figure 2 illustrates simulated LIDAR data that has been projected onto the world frame using simulated GPS and IMU data. Because of this, most of the autonomy algorithms developed rely primarily on GPS, IMU and LIDAR data, and only use the cameras and sonar systems when specifically needed. However, this has proven to be sufficient in real world tests. A suitable dynamic model has been implemented to estimate the vehicle's response to command inputs as well as an Extended Kalman Filter and will be described in further detail in the following section.

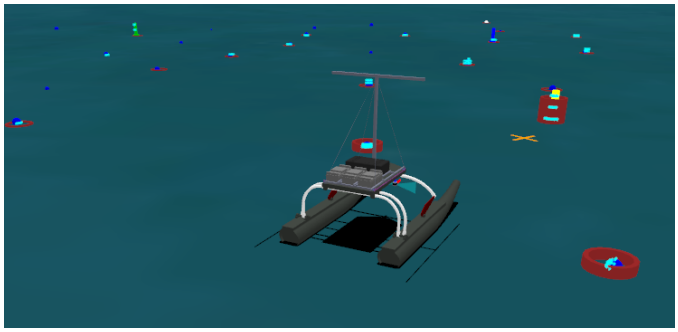

Figure 2: Simulated LIDAR, GPS, and IMU Data.

In Fig. 2 the ground truth location of the vehicle is represented by the transparent vehicle model. The EKF estimated boat position (based off simulated GPS and IMU data) is represented by the opaque vehicle model. The vehicle is equipped with a 3D LIDAR unit. The cyan lines are the simulated LIDAR data projected onto the world frame using the estimated boat position, with the red cylinders represented perceived buoys (with height). The LIDAR used in this project does not pick up the surface of the water, so the return data

is clean.

Figure 3: Sister image to Fig. 2, with the vehicle approaching first speed gate.

B. System Dynamics

For the simulation environment to generate representative training runs, a reasonable model for how the vehicle moves through the water is needed. While there has been work on three-dimensional dynamic models for small marine vehicles [1], a two-dimensional model, considering *yaw*, *surge*, and *sway* while ignoring *pitch*, *roll*, and *heave* has been implemented. Ignoring *pitch*, *roll*, and *heave* is common practice for small surface vehicles [2] in low sea states where affects like broaching can be ignored. For propulsion, the vehicle is equipped with two Torqeedo Cruise 2.0R thrusters in a skid steering configuration. One thruster is attached to the pontoons on either side of the vehicle, and can be used to accelerate/decelerate in the longitudinal direction and apply a torque to induce rotation. As such, the dynamic model consists of these states:

- θ - angular position (yaw) in world frame
- ω - angular velocity in world frame
- x - position in world frame
- y - position in world frame
- u - linear velocity (surge) in vehicle frame
- v - linear velocity (sway) in vehicle frame

And the following inputs:

- m_l - force applied by left motor
- m_r - force applied by right motor

Integration of the Torqeedo motors into the software stack allows for the recording of commanded input and actual motor RPM. This data, combined with state estimation data, was used to build a transfer function to estimate the applied forces for given throttle inputs. The state update equations for the dynamic model are dictated by the following equations:

$$\theta_{k+1} = \theta_k + \omega_t \cdot dt \tag{1}$$

$$\omega_{k+1} = \omega_k + dt \cdot \left(\frac{m_{r,k} - m_{l,k}}{c_1} - c_2 \cdot w_k - c_3 \cdot w_k \cdot |w_k| + c_9 \cdot \left(u_k^2 + v_k^2\right) \cdot \sin\left(2 \cdot \text{atan2}\left(v_k, u_k\right)\right) \right) \tag{2}$$

$$x_{k+1} = x_k + dt \cdot \left(u_k \cdot \cos\left(\theta\right) + v_k \cdot \sin\left(\theta\right)\right) \tag{3}$$

$$y_{k+1} = y_k + dt \cdot \left(v_k \cdot \cos\left(\theta\right) + u_k \cdot \sin\left(\theta\right)\right) \tag{4}$$

$$u_{k+1} = u_k \cdot \cos\left(\omega_k \cdot dt\right) - v_k \cdot \sin\left(\omega_k \cdot dt\right) + dt \cdot \left(\frac{m_{r,k} + m_{l,k} + 2 \cdot m_{c,k}}{c_4} - c_5 \cdot u_k - c_6 \cdot u_k \cdot |u| \right) \tag{5}$$

$$v_{k+1} = v_k \cdot \cos\left(\omega_k \cdot dt\right) + u_k \cdot \sin\left(\omega_k \cdot dt\right) + dt \cdot \left(-c_7 \cdot u_k - c_8 \cdot v_k \cdot |v_k|\right) \tag{6}$$

In all of the above equations, the k^{th} state is the current state, while the $(k+1)^{th}$ state is the updated state for the next time step. dt is the time step between updates of the vehicle's state.

For the heading update equation, Eq. 1, only the first order angular velocity term is considered. Higher order angular acceleration terms are insignificant with a sufficiently small time step.

The angular velocity update equation, Eq. 2, consists of four terms to define the angular acceleration. The first part term consists of the input torque $m_r - m_l$ divided by the vehicle rotational inertia c_1. The length of the moment arm for these torques is captured in the rotational inertia term. The next two terms approximate the first (c_2) and second (c_3) order angular damping of the vehicle. The final term represents a straightening phenomena, designed to capture the vehicles natural tendency to glide straight through the water.

The global position update equations, Eq. 3 and 4, only contain the first order velocity terms without higher order acceleration terms. This is done with the same assumption of a sufficiently small time step. The $\sin(\theta)$ and $\cos(\theta)$ terms transform the velocity terms u and v from vehicle frame to world frame.

The surge update equation, Eq. 5, involves transforming the velocity in vehicle frame as the vehicle frame rotates. The next term is the linear force divided by the system inertia (c_4). The last two terms are first (c_5) and second (c_6) order drag approximations. The sway update equation, Eq. 6, is identical to surge with the exception of the removed force/inertia term.

For this model to be useful, values for the nine constants that appear in the state equations need to be assigned, estimated, or measured. Some of these constants (like mass) can be measured directly, while other constants need to be estimated. Initially, online parameter estimated with a recursive least squares (RLS) estimator was attempted. However, the online version proved unstable and an offline parameter estimator was used to determine these coefficients. For the offline parameter estimator, an hour's worth of GPS and IMU data (of the vehicle driving on the lake in a predetermined pattern) was

recorded. GPS data consists of the vehicle global position at a 4 Hz update rate, and the IMU data consists of 3D Magnetometer, Gyroscope, and Accelerometer data at 100 Hz. 95% of the GPS data was withheld and used to train and determine dynamic model coefficients. A gradient descent optimizer was configured to minimize the mean-square-error (MSE) of the withheld GPS data with the predicted model location. In other words, we:

1) Split data into 5 second time intervals.
2) Withheld all the GPS data (except the very first data point in each time intervals) from each interval.
3) Seed dynamic model with an estimate of what the boat is doing at that very first point for each interval.
4) Evolve the dynamic model 5 seconds into the future, using only the recorded input (motor voltages) for that interval.
5) Compare the withheld GPS data with predicted path from dynamic model.
6) Perform gradient descent on model parameters to minimize the MSE between withheld and prediction data.

This approach was tried on several dynamic models before settling on the model described above. Due to simplifying assumptions (ignoring wind, waves, wakes), the training and prediction data will not match perfectly. Figure 4 compares the training and prediction data for a few samples of the training set. In this clip, the vehicle is moving from left to right. The grid lines correspond to meter increments. The red arrows correspond to estimated vehicle locations at the start of each interval, and the green path corresponds to the predicted boat path from the dynamic model. Each black X is a GPS data point that was withheld. Note that during the first 5 second interval the predicted (green) and withheld (black) paths are nearly on top of each other. During the second 5 second interval, the two paths diverge after the boat makes more abrupt maneuvers. However, even with this occasional erratic behavior, the predicted coefficients perform sufficiently well for our needs.

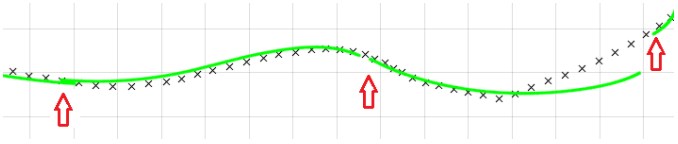

Figure 4: Withheld training GPS data vs dynamic model prediction.

C. State Estimator

The dynamic model state equations were intentionally laid out to be easily transitioned into an Extended Kalman Filter (EKF). A Kalman Filter (KF) is an optimal estimator for linear systems assuming both gaussian process and measurement noise, and the EKF is a modified version of the KF that can handle nonlinear systems. There is a significant amount of literature on EKFs, with much of the work beyond the scope of this paper. Only the most pertinent details of the implementation used for this work are described below.

The quality of the EKF output is directly related to how well the dynamic model, process noise (Q matrix) and measurement noise (R matrix) represent the actual system. Typically, the dynamic model, Q, and R are measured with available ground truth data[3][4]. A novel process has been developed to estimate the full Q and R matrices without ground truth data for this project and is the subject of another paper to be published at a future date.

The EKF uses Bayesian inference to combine information from the dynamic model (*priori*) and from sensor measurements to predict the vehicle state (*posteriori*). A good fitting dynamic model is needed to maximize EKF performance. Tuning our dynamic model, as described above, requires a good estimate of what the vehicle is doing at the start of each time interval. This is a chicken or egg conundrum, as tuning the EKF requires a good dynamic model, and tuning the vehicle model requires a good EKF. The problem can be overcome by alternatively tuning the EKF, then the dynamic model, then the EKF again in an iterative fashion.

D. Autonomy

The software developed conforms to the following ordered structure:

1) Raw data from sensors (like LIDAR point clouds) is transformed into usable data like global positions and orientations of docks and buoys.
2) Buoys are identified from the list of obstacles and labeled (i.e. which buoys are most likely to be speed gates, or obstacle entrance and exit gates, or the buoy with the active pinger).
3) Gate, dock, and pinger locations are transformed into a destination based on the planner that takes into account the current and completed tasks for the overall mission.
4) The arbiter takes the destination, obstacles list, and other sensor data and determines how to get there without hitting anything.
5) The controller transforms the arbiter command (direction and heading) into motor voltages, keeping the vehicle on course and stable.

Some specialized algorithms combine two more more of these steps, but for most configurations the above list represents how data flows through the software. There are many different moving parts, and many of these components have multiple behavior algorithms implemented. It would be impossible to cover all of these in the space of this paper, so a representative sample is presented here.

1) Acoustic Pinger Localization: The vehicle is equipped with the same three hydrophone layout as was used on the vehicle used for the Roboboat Competition in July. The hydrophone layout for the Roboboat vehicle is shown in Fig. 2, with the blue cylinders representing hydrophone locations. Each hydrophone returns a raw audio signal which, when filtered and amplified, can be turned into a series of timestamps that correspond to when the hydrophone detected a ping. There are several ways to use these timestamps to estimate pinger location. The best performing algorithm that has been implemented is a RANSAC locator[5]. As such, this algorithms

was used in simulated testing to determine the placement of the hydrophones on the WAM-V. What follows is a brief description of the pinger localization algorithm.

If two hydrophones recorded the same event at the same time, the pinger must be equidistant from both hydrophones. This is illustrated in Fig. 5, with the red circles representing hydrophones and the blue line representing the continuous set of possible pinger locations.

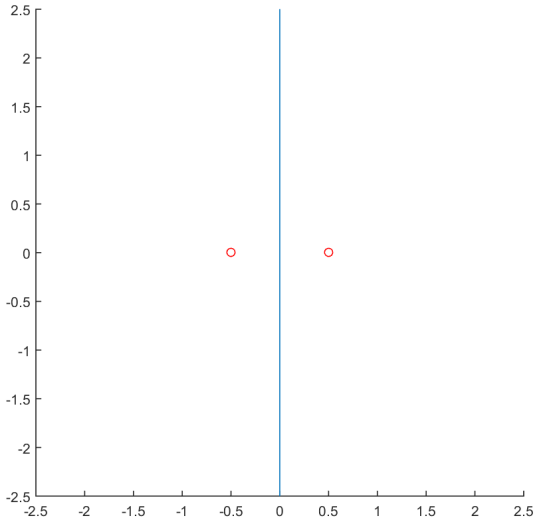

Figure 5: Continuous set of pinger locations (blue) given that some event was recorded by two hydrophones (red) at the same time.

This blue line is also called a contour line, signifying that for any point on the line, the time difference between when the sound reaches both hydrophones is constant. A nonzero time difference, would correspond to a different contour line, as shown in Fig. 6. The increment in time difference values between adjacent contour lines in this graph is also constant. The nonlinear behavior of the angular resolution highlights the fact that relative accuracy of an estimated pinger position is highly sensitive to the orientation of a given hydrophone pair.

With multiple hydrophone pairs, it is possible to combine the information from the contour lines to triangulate the position of the pinger. Alternatively, a single hydrophone pair could estimate the position of a stationary pinger by moving through space and using the hysteresis of contour lines as additional virtual hydrophone pairs. For either of these methods, an accurate estimate of the vehicles state, and the relative orientation of the hydrophones to the vehicles frame is necessary. In practice, it helps to both have more than two hydrophones, and move the vehicle while data is being recorded. This is illustrated using the simulation environment in Fig. 7 and Fig. 8. Finding an optimal configuration for hydrophone placement analytically, especially considering the coupling with the EKF and autonomous behavior, is impossible. Using the simulation environment, however, a local optimal solution can be found.

2) DAMN Arbiter: Distributed architecture for mobile navigation or (DAMN) is a reactive architecture that arbitrates

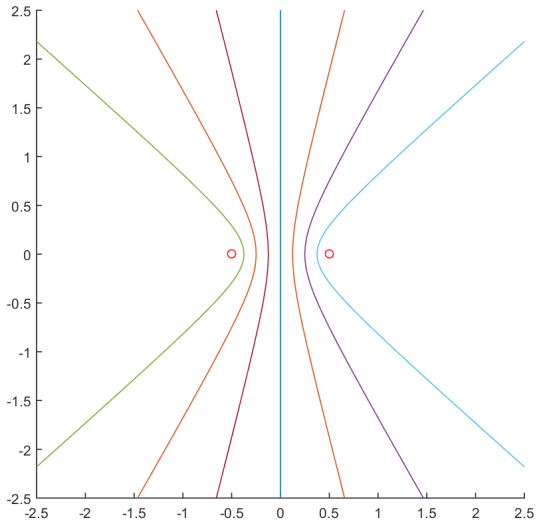

Figure 6: Contour lines for a two hydrophone sonar array.

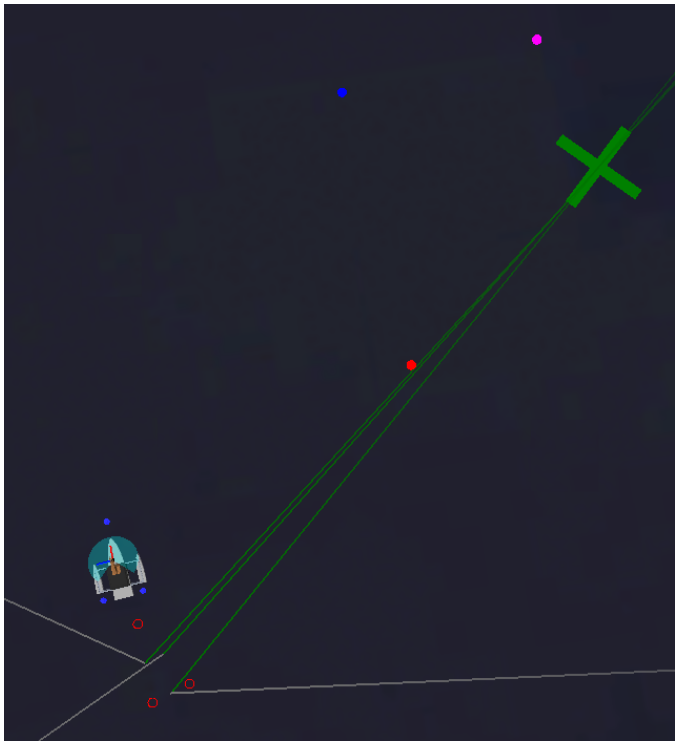

Figure 7: RANSAC for single ping on a three hydrophone vehicle. The system misses (green cross) the pinger (red buoy) because the contour lines are almost all parallel.

through voting [6]. The local region around the vehicle is broken up into smaller sub regions, and each behavior (in this case both *go to waypoint* and *avoid obstacles*) votes on how willing that behavior is to travel to that region. These regions are illustrated in Fig. 9; where color corresponds to vote total. Green indicates a high vote, or a willingness for the vehicle to head to that region. Red indicates a low vote or an unwillingness to head to that region. Different behaviors have a different voting weight. The avoid obstacles

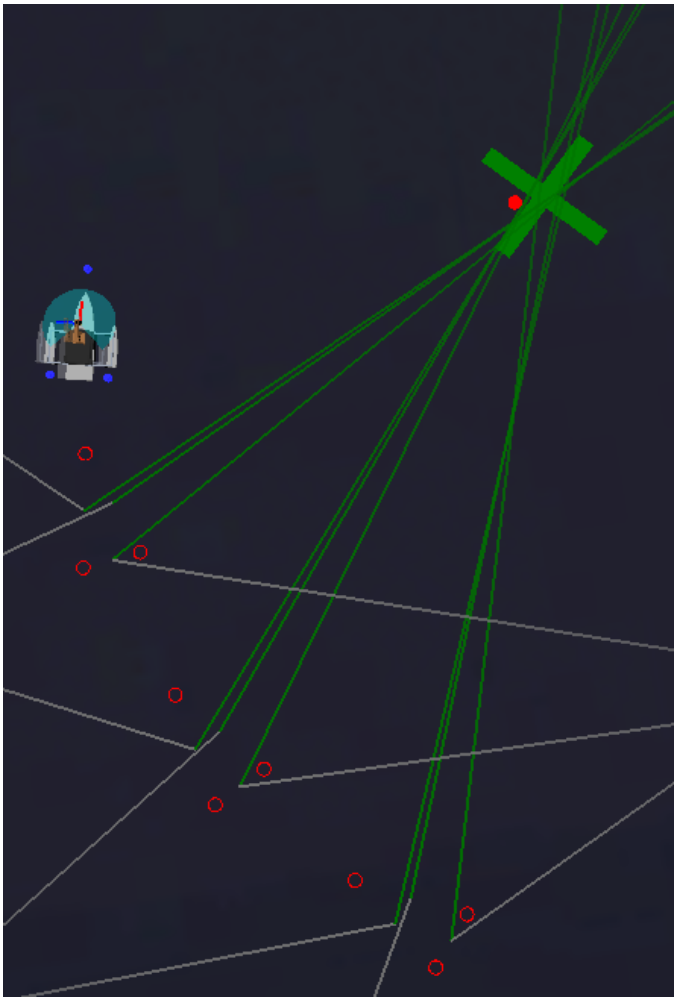

Figure 8: RANSAC for three pings on a three hydrophone vehicle. The system returns (green cross) a much better estimate for pinger (red buoy) location because the contour lines are not all parallel.

behavior has the strongest vote, which is why the regions near perceived obstacles (indicated by red circles) are dark red. To avoid situations where the goal point is directly behind an obstacle, the avoid obstacle behavior also negatively votes for any areas that are obstructed by known obstacles. An easy way to visualize this is to think of the vehicle as a light source, with obstacles casting shadows. Anywhere a shadow is cast is heavily downvoted. The orange cross indicates the current destination, and, as expected, is surrounded by the greenest regions. The arbiter commands the vehicle to head toward the region with the highest vote total, with a speed that's proportional to how far the region is from the vehicle.

3) Potential Fields Arbiter: Another arbiter that has been implemented and tested is arbitration through a potential field abstraction. Each behavior now acts as either a source, such as avoiding obstacles, or a sink, a waypoint destination. A weighted average of the resulting fields is taken, again with avoid obstacles having the dominant weight, and heading and velocity is returned. This return is produced via a simple gradient descent through the potential space. Figure 10 illustrates

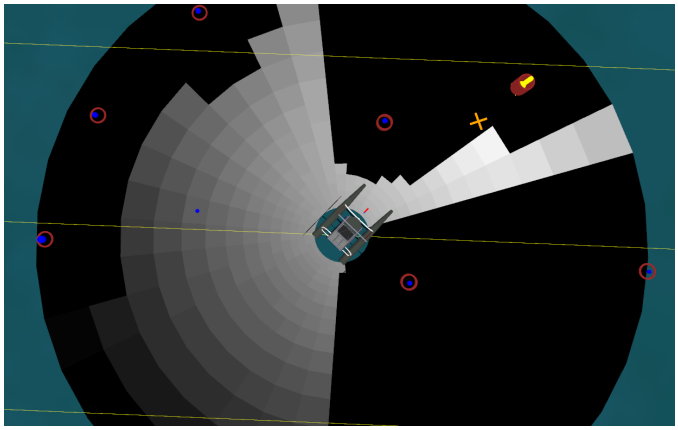

Figure 9: DAMN arbiter.

the average vector returned by the arbiter at various locations, indicated by the white arrows. The orange cross denotes the waypoint sink and red circles denote perceived obstacles acting as sources in the potential field.

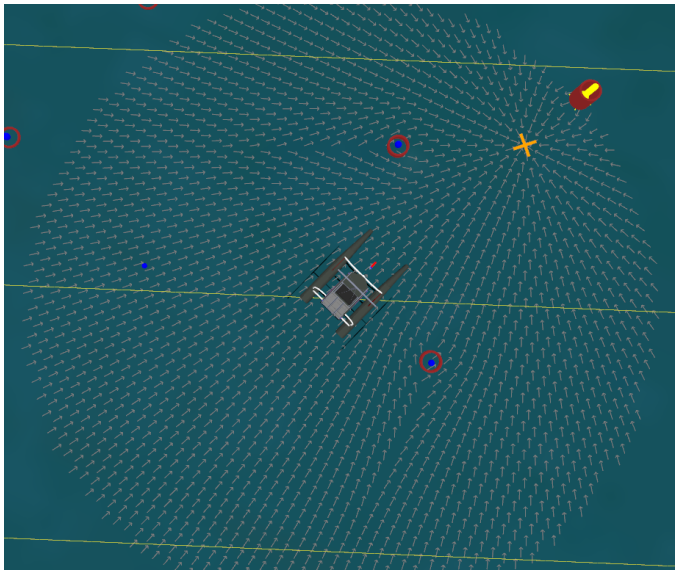

Figure 10: Potential fields arbiter.

These two arbiter implementations highlight the usefulness of the simulation environment for behavior comparisons. Not only can the performance of competing behaviors be compared, visualizations within the software allow for the algorithm performance to be understood in a more concrete way.

E. Hardware Description

The Wambulence utilizes the WAM-V platform. Propulsion is provided by two Torqeedo Cruise 2.0R electric motors arrayed in a skid steering configuration. Onboard computing is handled by an Intel NUC with a Core i7 processor running Windows 10. The computer is connected directly to the Torqeedo motors to provide control through a proprietary API. A Velodyne PUCK VLP-16 3D LIDAR is used for 3D obstacle

detection and classification. A Microstrain 3DM-GX3-45 INS is used for an onboard GPS and IMU system. All electronic components are housed in a waterproof Pelican Storm iM2400 case. The case has been outfitted with a modular component rack and custom power rail system. The motors and other electronics are decoupled on separate circuits, with a 2 Torqeedo LiPo battery used for motor power, and a 1 Torqeedo LiPo battery used for the computer and sensor systems.

IV. Experimental Results

Much of this paper has discussed the value of the AAVS environment. One of the key features of this environment is the integrated stack used for simulation, hardware-in-the-loop testing, and control of the competition vehicle. This has allowed for testing of algorithms, design choices, and hardware implementation on multiple levels. Software simulations are being run near constantly to further refine and compare behavioral algorithms. Hardware-in-the-loop simulations are largely done at the Georgia Tech ADEPT Lab and around the Georgia Tech. Various tests are done multiple times a week, including motor control, manipulator actuation, and object identification/classification. Full system tests are conducted at Sweetwater Creek State Park on a self constructed mock mission course, and have been done on a roughly monthly basis. Previous sections have covered the uses of pure software testing, while this section will focus mainly on the hardware-in-the-loop simulation and the playback of real world testing data.

A. Hardware in the Loop

As previously discussed, the simulation environment is also the software used to control the actual competition vehicle. This means that all sensing and actuator systems aboard the vehicle have been interfaced with the simulation software. This allows for these sensors, and the data returned by them, to be used to accurately model, predict, and validate real world performance. This has been an invaluable asset in debugging the implementation of the software-hardware interfaces.

An example of this is in the motor controller interface. By allowing the vehicle to actively actuate its real systems in a simulated mission run, their performance and potential faults can be identified before planning, or in preparation for, a lake trip. During early testing, an unacceptable amount of lag was present in the initial implementation of the motor controller interface. This was partially due to motor control through mimicry of components in a separate Torqeedo motor controller. By identifying this lag using hardware-in-the-loop testing, the effort required to proceed with a direct control interface with the motors themselves could be justified without wasting any precious lake testing time. This hardware in the loop testing also allowed for verification of the new control system it progressed. Many hours have been spent using similar simulations of other components to debug and improve their implementations before even getting to the lake.

B. Playback & Visualization

This integrated software stack also allows for another additional capability: the playback of real world testing data. By logging data taken during real world tests, whether at the lake or in a hardware-in-the-loop simulation, this recorded data can be used instead of simulated sensor data in play backs of the test. This capability allows for the visualization of what the vehicle was *thinking* as it went through a test. Figure 11 shows both a frame from a video taken during a lake training run of the Roboboat vehicle, and the cumulative sensor data up to that point for that run. During this run the vehicle found and navigated the speed gates successfully, which is not surprising given the two distinct LIDAR point clouds that have been picked up on the right. In this case, the visualization is interesting but not useful. However, when the system fails to navigate through the speed gates (or do any other task), the visualization is incredibly useful. By observing what goes on and what the behavior algorithms were planning, the problem can be diagnosed (*was it a sensor blind spot error? sensor filtering error? arbiter error? or controller error?*). This information can be used to solve the underlying problem and ultimately improve the vehicle instead of just alleviating the symptoms of the problem with ad-hoc debugging methods.

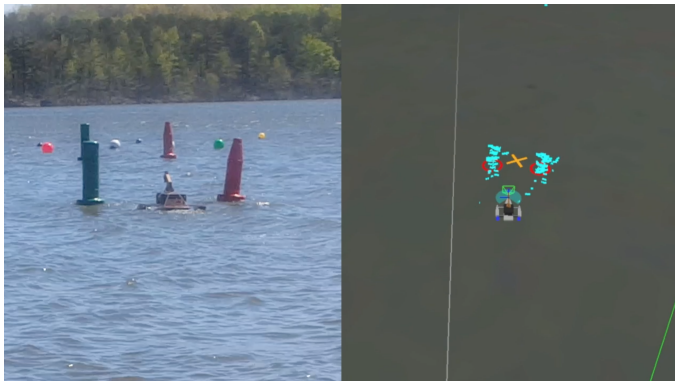

Figure 11: Playback capability of simulation environment. Left is part of a video from one training run. Right is the visualization of the cumulative sensor data up to that point for that training run.

V. Conclusion

At the beginning of this year, the team decided that we were going to win this competition with software. A highly capable simulation environment was developed to aid in the rapid development, implementation, and iteration of various autonomous behavior algorithms. This software stack was developed to also be used on-board the vehicles such that a single software stack could be used for simulation, testing, and control of the vehicle during competition. This software stack has seen incredible growth throughout this process. Through the use of the AAVS integrated software simulation, hardware-in-th-loop testing, and full tests at the lake, our ability to consistently perform the necessary tasks for the

RobotX challenge has grown. Now we're excited to see our system compete on the actual RobotX course.

REFERENCES

[1] A. W. Browning, "A mathematical model to simulate small boat behaviour," *Simulation*, vol. 56, no. 5, pp. 329–336, 1991.

[2] H. Ashrafiuon, K. R. Muske, L. C. McNinch, and R. A. Soltan, "Sliding-mode tracking control of surface vessels," *Industrial Electronics, IEEE Transactions on*, vol. 55, no. 11, pp. 4004–4012, 2008.

[3] C. Goodall and N. El-Sheimy, "Intelligent tuning of a kalman filter using low-cost mems inertial sensors," in *Proceedings of 5th International Symposium on Mobile Mapping Technology (MMT'07), Padua, Italy*, pp. 1–8, 2007.

[4] P. Abbeel, A. Coates, M. Montemerlo, A. Y. Ng, and S. Thrun, "Discriminative training of kalman filters.," in *Robotics: Science and systems*, vol. 2, p. 1, 2005.

[5] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[6] J. K. Rosenblatt, "Damn: A distributed architecture for mobile navigation," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 9, no. 2-3, pp. 339–360, 1997.