Why Yes, That is Hot Glue: A Simulation Based Approach to Roboboat 2016

Samuel Seifert, Patrick Meyer, James Wittig,
Vinayak Ruia, Daniel Findeis
Georgia Institute of Technology

***Abstract*—This paper describes the development of Burnadette, a fully autonomous surface vehicle for use in the AUVSI Roboboat competition developed by the Georgia Tech ADEPT Lab. The vehicle uses a previously manufactured platform to allow for added emphasis on software capabilities. A tightly coupled simulation environment was developed in parallel to the hardware solutions. This allowed for the development, testing, and integration of various autonomous behavior algorithms for use in the competition. The simulation environment developed also includes hardware in the loop capabilities for use in debugging and testing. The added simulation capability has allowed for rapid iterations of these behaviors while lessening the need for time consuming full system tests. We have high hopes that this process will lead to success at this year's Roboboat competition.**

I. Introduction

Roboboat is held over seven days. For the first five days, participating teams vie for several 30 minute time slots for testing on the competition course. Over the course of these days, teams can expect to get roughly 10 hours of time to test. This is simply not enough time to develop, debug, and tune entire software stack for all of the tasks outlined in the competition. To adequately prepare for the competition, teams must to supplement this time either by: using data and knowledge from past competitions, testing before the competition at a mock competition site, and testing in simulation. Due to lack of foresight, recorded sensor data from previous Roboboat competitions was not available. Additionally, the nearest suitable testing site to Georgia Tech is a 40 minute drive away. Collecting real world data at this site requires a significant time investment, and it would be impossible to test at this site 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 too. 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 Roboboat competition in AAVS using different sensor configurations, the performance of each configuration can be compared using the scoring guidelines outlined by the Roboboat 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.

The remainder of this paper will outline the development of the hardware and software used to compete in this year's Roboboat 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

Roboboat is ultimately an autonomous vehicles competition. Though the vehicle design 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. To minimize the design work necessary to get an adequate platform, a previously designed vehicle was used. This platform was the vehicle used by the 2014 Georgia Tech team.

To further reduce workload, the software stack used for simulation was designed to control the actual competition vehicle (as opposed to being two separately developed entities). The simulation and the actual vehicle software are the same stack, which has greatly accelerated the development process. The simulation environment is a great tool because *real world success implies simulation success*: i.e. the ideas that fail in simulation can be abandoned because they won't work in real life. However only testing in simulation is inadequate because *simulation success does no imply real world success*. To compensate for this phenomena, the team has worked hard to constantly validate the simulation with real world tests.

It is important to understand these two points that highlight different strengths of a simulation based design approach. 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. Similarly, ideas that work in the real world will almost certainly work in simulation where conditions are ideal.

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#. Using data extracted from Google Maps, knowledge retained from previous years competitions, and the preliminary rules for this year, the 2016 Roboboat course has been laid out in the simulation environment as shown in Fig. 1 inside AAVS. Using this environment to develop algorithms and the overarching software stack has proved invaluable.

Figure 1: The 2016 Roboboat course in simulation.

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

- GPS & IMU to determine vehicle state.
- LIDAR system to detect physical obstacles.
- Camera system to recognize color & 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, GPS, IMU, and LIDAR data can be realistically generated (as shown in Figure 2), while camera and sonar data remain unrealistic. 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.

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 four SeaBotix BTD150 thrusters in a skid steering configuration. Two thrusters are along the centerline of the vehicle, and they can be used to accelerate/decelerate in the longitudinal direction. The remaining two thrusters are 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_c - force applied by center motors
- m_r - force applied by right motor

SeaBotix provides a thrust to voltage curve for these motors at low speeds as shown in Fig. 4. This thrust-voltage relationship 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:

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

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

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

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

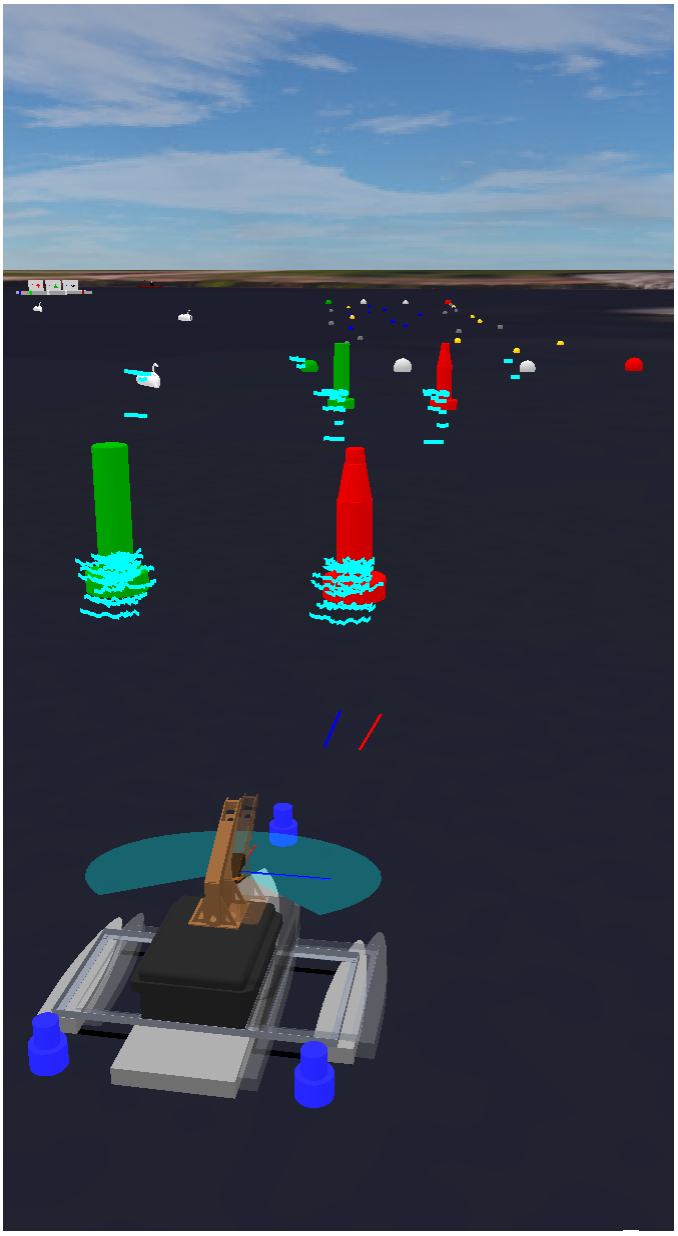

Figure 2: Simulated LIDAR, GPS, and IMU Data. The ground truth location of the vehicle is represented by the transparent vehicle model. The EKF estimated boat position (based off simulated GPS and IMU data) is represented by the opaque vehicle model. The vehicle is equipped with a 2D LIDAR unit, that is configured rotate up and down. The cyan lines are the simulated LIDAR data projected onto the world frame using the estimated boat position. The LIDAR's used in this project do not pick up the surface of the water, so the return data is clean.

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

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

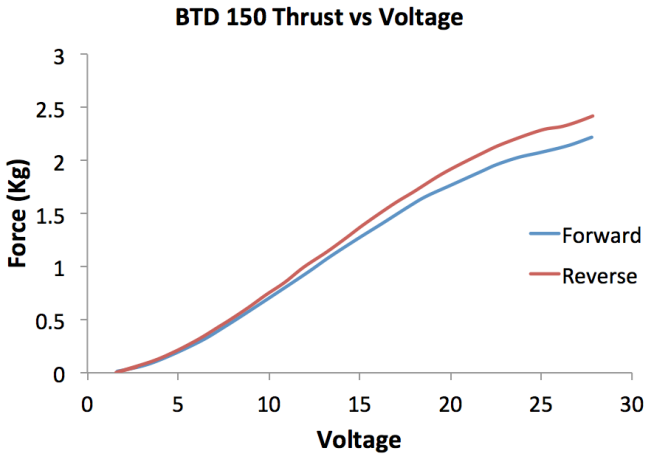

Figure 4: Thrust vs Voltage curve for SeaBotix BTD150 at low speed.

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

In 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 discrete time step.

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 hours 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 & 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 won't match perfectly. Figure 5 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 an erratical left turn. However, even with this occasional erratic behavior, the predicted coefficients perform sufficiently well for our needs.

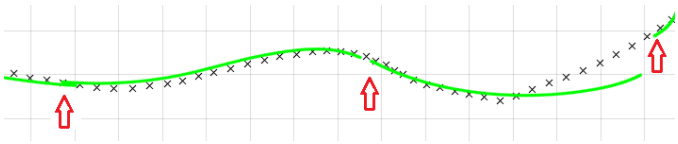

Figure 5: Withheld training GPS data vs dynamic model prediction.

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 & 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 & 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 steps have been implemented in more than one way. It would be impossible to cover the entire software in the space of this paper, so a representative sample is presented here.

1) Acoustic Pinger Localization: The vehicle is equipped with three hydrophones. The hydrophone layout 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 locater[5].

If two hydrophones recorded the same event at the same time, the pinger must be equidistant from both hydrophones. This is illustrated in Fig. 6, with the red circles representing hydrophones and the blue line representing the continuous set of possible pinger locations.

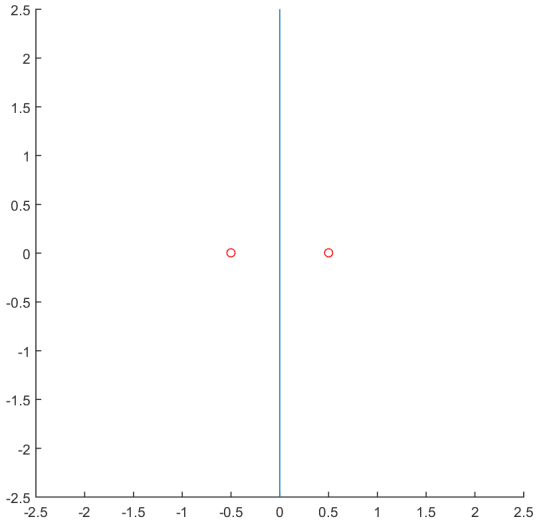

Figure 6: Continuous set of pinger locations (blue) given that some event was recorded by two hydrophones (red) at the same time.

This blue line is also called an 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. 7. The increment in time difference values between adjacent contour lines in this graph is constant. The nonuniform angular resolution illustrated in Fig. 7 highlights the fact that estimated pinger position accuracy is sensitive to the orientation of the hydrophone pair relative to the pinger.

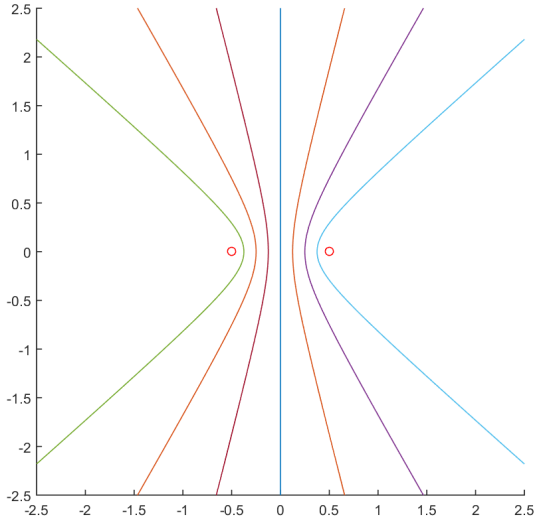

Figure 7: Contour lines for a two hydrophone sonar array.

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 collecting several data from several different locations. 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. 8 and Fig. 9. 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 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. 10; 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 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. The orange

Figure 8: 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.

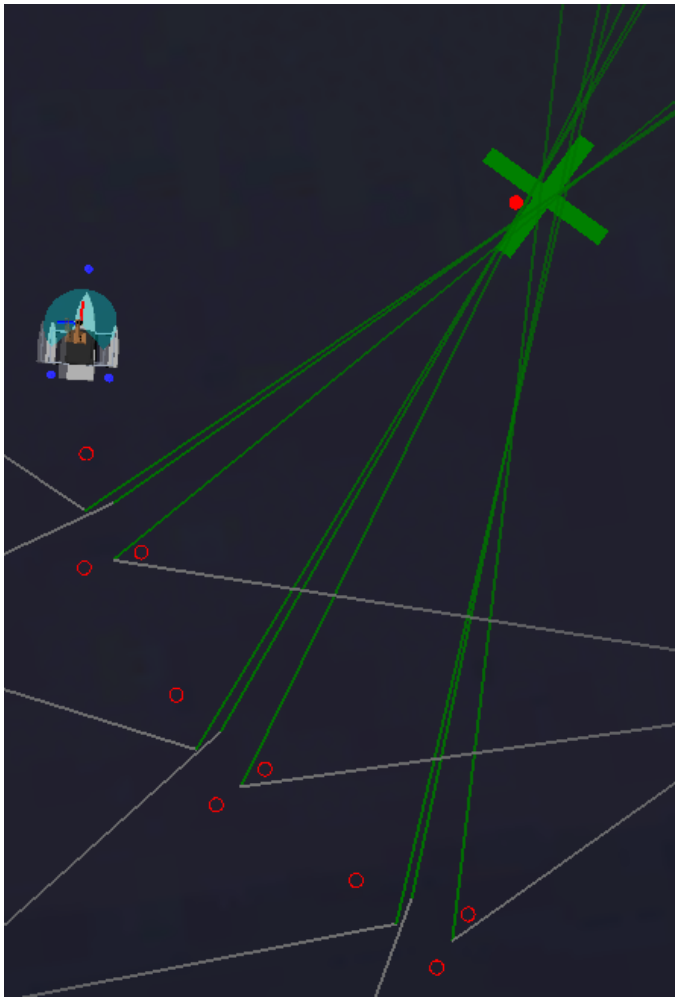

Figure 9: RANSAC for three pings on a three hydrophone vehicle. The system returns (green cross) a much better estimate for pinger (red buoy) location because the contour lines are not all parallel.

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, such as 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 11 illustrates 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.

The simulation environment has been used to compare performance of these two (and other) arbitration techniques. Visualizations within the software provide insight to not only which algorithms perform best, but also why they perform best.

E. Hardware Description

An existing platform that had been used in past Roboboat competitions was repurposed for this year's competition. The vehicle frame itself is a trimaran design. Propulsion is provided by four Seabotix 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 interfaced with the DC motor controllers using an Arduino Mega microprocessor and packetized serial communication. A Hokuyo UTM-30LX planar LIDAR actuated by a Dynamixel AX-12A servo 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 cases. 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 5 cell LiPo battery used for motor power, and a 4 cell 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

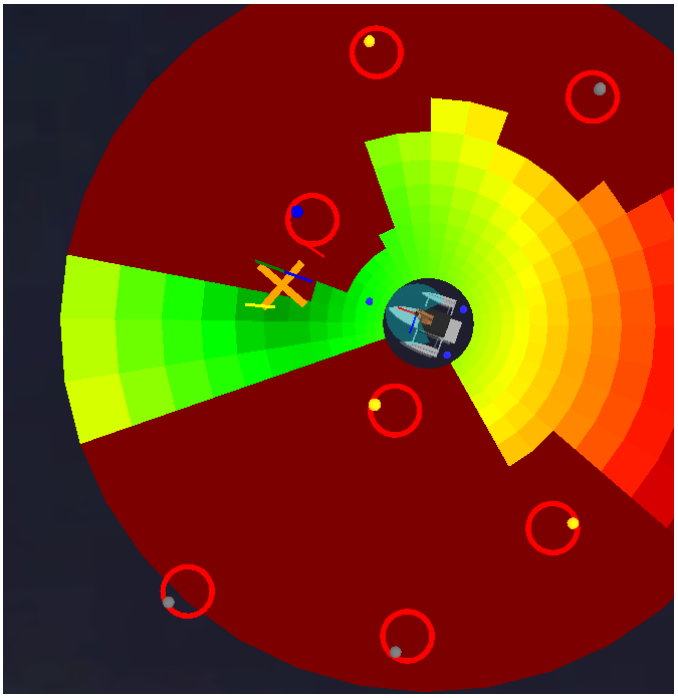

Figure 10: DAMN arbiter.

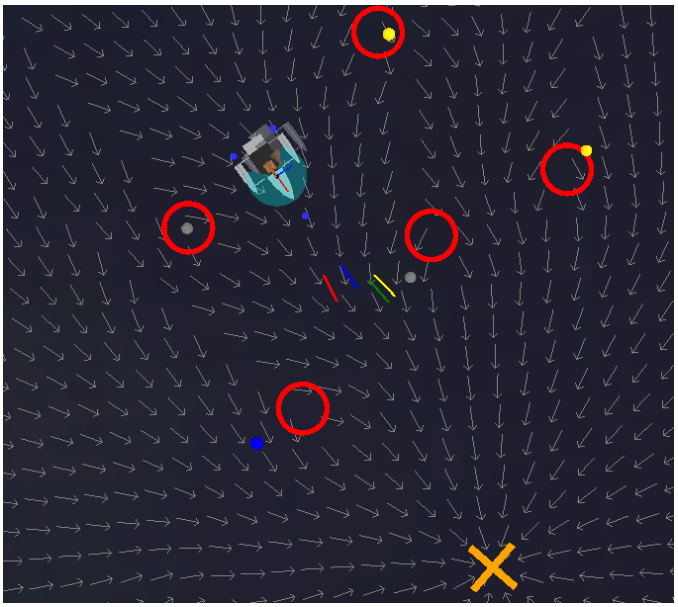

Figure 11: Potential fields arbiter.

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 campus with the vehicle being drug around in a wagon. Various tests are done multiple times a week. 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 sensors and actuators aboard the vehicle have been interfaced with the simulation software. This allows for these sensors, and the data returned by them, can 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 it's 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. By identifying this lag using hardware-in-the-loop testing, the problem could be solved in the lab and a sufficient controller interface was implemented without wasting any precious lake testing time. 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, the simulation can then use this recorded data instead of simulated sensor data. This capability allows for the visualization of what the vehicle was *thinking* as it went through a test. Figure 12 shows both a frame from a video taken during a lake training run, 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.

V. Conclusion

At the beginning of this year, the team decided that we were going to win this competition with software. We opted to spend as little time and effort on hardware as possible, improving and iterating on software capabilities instead. We've re-purposed

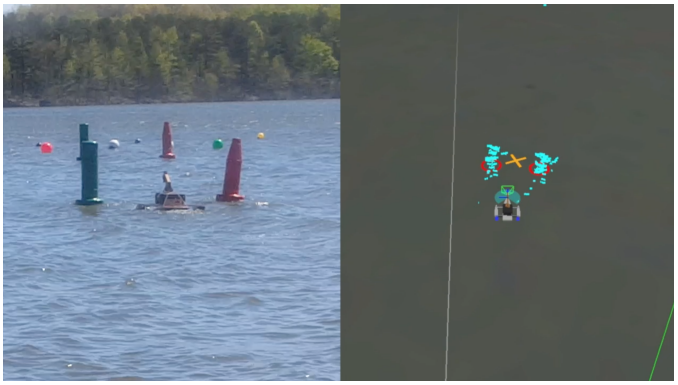

Figure 12: Playback capability of simulation environment. Left is par of a video from one training run. Right is the visualization of the cumulative sensor data up to that point for that training run.

a vehicle from past Roboboat competitions, and performed minimal amounts of hardware changes. The software stack that leverages this vehicle has seen incredible growth. Through the use of the AAVS, hardware-in-the-loop testing, and full tests at the lake, our ability to consistently perform the necessary tasks for the Roboboat challenge has grown. We're excited to see how it will all play out come competition time!.

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.