Georgia Institute of Technology

Edward Macdonold, Patrick Dillon, Chris Taylor, Sean Culpepper, David Moroniti

May 23, 2011

Abstract

For the second year of competing Georgia Tech Marine Robotics completely overhauled their design from the previous year. Using a trimaran design for stability and speed, new and improved sensors including LIDAR, IMU, GPS and CCD firewire camera the new design is capable of performing each mission. Algorithms include using Kalman Filtering all data to get good approximations of our states, localization using known buoy locations and employing multiple behaviors through DAMN (Distributed Architecture for Mobile Navigation).

1 Mechanical Design

1.1 Hull

The hull was completely redesigned form last year's vehicle. This year a trimaran hull form was selected. While more difficult to construct, this design was chosen for its increased stability, speed, and maneuverability. The shape of the hull was designed in Paramarine and analyzed for buoyancy and stability. The hull consists of polystyrene foam covered in several layers of fiberglass for increased strength with out adding very much weight. The two pontoons are only polystyrene foam, as they do not need the added strength required of the main hull. The hull and the pontoons were cut on a 4-axis CNC hotwire foam cutter. The hull was cut in 11 sections then glued together before applying the fiberglass. The pontoons were cut as a single piece then shaped by hand and covered in epoxy. The vehicle is 4 feet in length, 3 feet in width and three feet in height. Two aluminum rails run the length of the hull and provide easy mounting to the hull any where along the deck. This design allows for flexibility and modularity and on the fly configuration changes if desired.

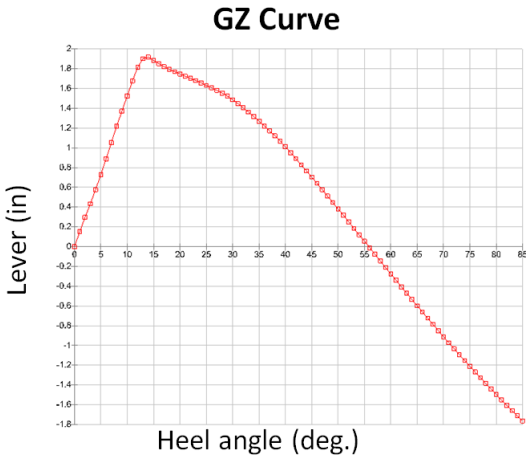


Figure 1: GZ Curve

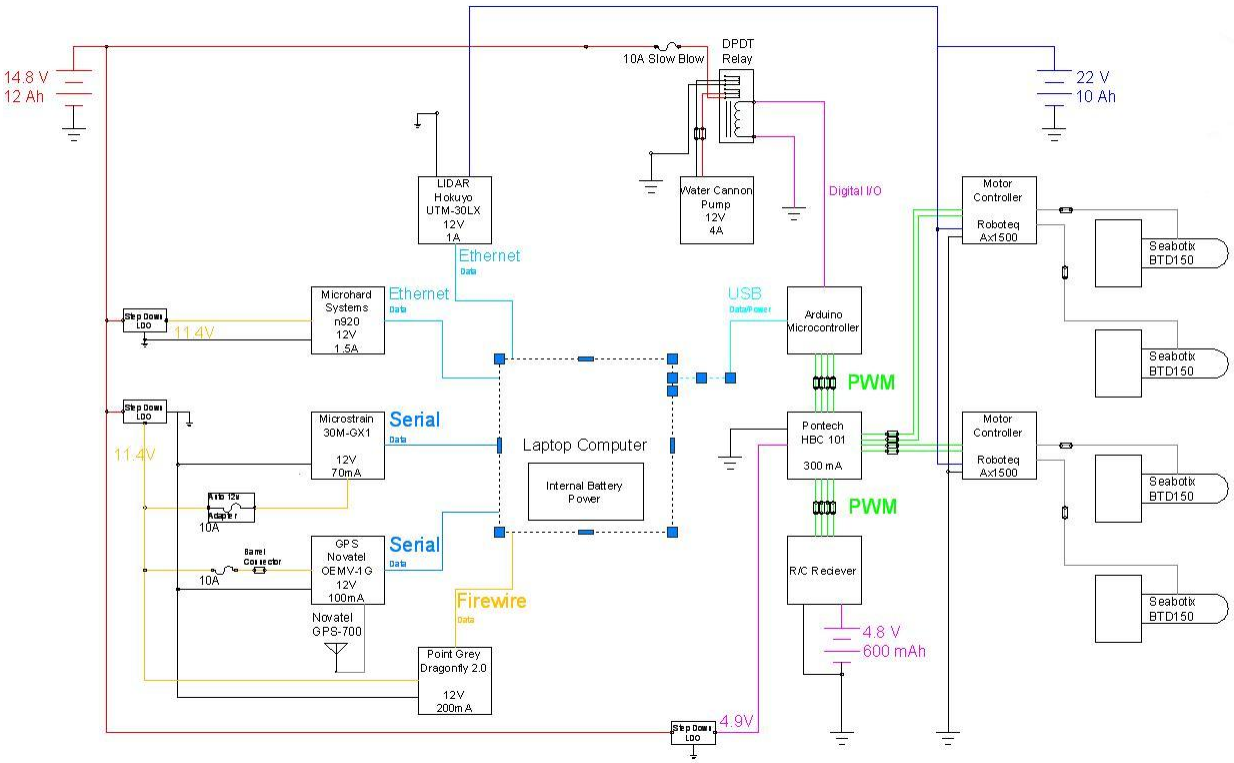


Figure 2: Wiring Diagram

1.2 Electronics Box

The electronics box consists of a modified storage container. While last year's ASV used a mater proof pelican case, this year a smaller and lighter box was used. The smaller and lighter box allows for easier assembly and transportation. The electronics box contains the computer, batteries, and all electronics except sensors. The motors connect via 4 Sub-Conn wet connects, and all other wires run through a custom-built pass-through port. This is made of several layers of weather stripping, which allows any wires to easily pass into the box with out needing separate connections for each wire. This allows the team to easier swap in and out parts freely while still maintaining the integrity of the box. By having nearly all the electronics in the same box we reduce the number of wires that need to run outside the box which makes waterproofing the ASV easier. Inside all electronics are mounted to Lexan sheets, which are attached to the box with Velcro. This allows the electronics to be moved around within the box without compromising the water resistance.

1.3 Electrical Design

This year's electrical system also under went a complete re-design. Major new features include LIDAR, IMU , and GPS. Almost all connections are made inside the box which helps to reduce possible water exposure. A custom built slot in the boxes allows for easy access to the box for any number of different wires that are needed. Due to many problems we had with last year's electronics system this year's was built completely from scratch. For a complete wiring diagram see figure 2.

1.4 Power

This year's power system is completely redesigned. This year there are two separate power lines. There is a 14.4 v line and a 22.2 V line. The 22.2V line consist of two 5000 mAh 22.2v Li-Po batteries in parallel, these are used to power the thrusters because the motors produce max thrust a 19V. All the other electronics are run off a single 14.4 V 600 mAh battery, except the Laptop which has its own battery. Voltage regulators

drop the 14.4 V to 5 V and 12 V for various components. Additionally the voltages of the batteries are monitored in real time in addition an audible tone is produce if the voltage drops enough informing the team that it is time to change the batteries. This helps to extend the life of the batteries.

2 Software

2.1 Internal Communication

Internally the various components of the boat's software each run on their own thread. Instead of message passing, shared memory is used to create data that is globally accessible from any component. This allows rapid development of components which have access to any data as soon as they need it without the need to wait for a message to be sent and a response to return. Making extensive use of the Boost Asynchronous Input/Output and Boost Thread libraries the prototype for the components and the shared memory mechanism were developed to be generic and easily extensible. In order to solve the problem of data ownership ambiguity the data is all owned by a central data manager which distributes shared access to the data structures upon request. Once shared access to a data structure is received a shared mutex is locked and either downgraded or upgraded to provide read or write access respectively. In order to keep the code as readable as possible the shared objects are a templated class that are requested using a simple static templated Get function, but because it is not necessary to keep this shared object around all of the time, it is usually created automatically by either a templated ReadAccess or WriteAccess class. This allows read or write access to any data on the system to be acquired in one line of code without the need to explicitly release it.

2.2 Drivers

Custom drivers were written for all of the devices to maximize efficiency. These drivers were written to rely on the cross-platform Boost Asynchronous Input/Output and Boost Thread libraries. They are each running at their maximum rate, determined by the hardware, which is immediately feed into the data filters and estimators. Because they also use the shared memory scheme mentioned above, all of the processing components of the boat are instantly updated with the latest information about the state of the boat. In order to decrease the room for error, when the drivers start, they are able to identify if they are connected correctly and automatically correct for any mistakes.

2.3 External Communication

Externally the boat and the ground station communicate via Joint Architecture for Unmanned Systems (JAUS) messages. This was done to provide a standard interface for communicating with the boat. The data passed between the boat and the ground station is compliant with the JAUS standard if at all possible, and as a result, all data stored internally exists in side classes that constrain data to be compliant with the JAUS message payload format. Once the class is told to write its data to a buffer the data is converted into the correct format, buffered, and eventually sent across the network.

2.4 Operator Control and Monitoring Interface

In order to control and monitor the boat throughout its development phase, the user interface (referred to as the ground station) was designed with the goal of expediting the debugging process. The ground station was written as a web server and a website using the emerging standard for bi-directional full-duplex communication over TCP for servers and clients called WebSockets. This was accomplished by adapting a Boost Asio web server to handle WebSockets using a handshake procedure established in late 2010. WebSockets enables the interface to keep an open connection to the ground station server where messages can then be relayed to the boat itself. In order to keep the communication protocol simple, all communication between the ground station server and the client are done using comma separated value (CSV) format. Once the connection is established, the server sends all known data about the boat to the client every 30 milliseconds. Once the client receives this data it is parsed and displayed to the user in either a dynamically updated div

or canvas element. Because raw data provides the worst data visualization possible, both a 2D and a 3D real-time display of the state of the boat was created using WebGL. By drawing a 3D model of the boat at the correct orientation, as well as vectors for velocity, and other data, the web interface accomplishes its goal of increasing data understandability, which in turn aids the software development process.

3 System Architecture

Navigation of the surface vehicle is one of the greatest challenges of the competition. This large and complex goal is broken down into many smaller tasks. Our overall navigation architecture can be split into 3 fundamental categories: Sensing, Thinking, and Acting. Each category feeds into the next.

The purpose of the Sensing block is to collect information about the outside world. This is accomplished by an array of sensors: LIDAR, Camera, Infrared Sensor, Inertial Measurement Unit (IMU), and GPS. The raw sensor data is conditioned, filtered an analyzed so important features of the environment can be extracted. Some of the most important features include boat velocity, boat position, and buoy position. This information is then fed into the "Thinking" stage to influence how the boat behaves. The sensing stage is discussed in more detail in the State Estimation, and Buoy Detection sections.

The sole task of the thinking block is to generate a relative target position at every time step. Choosing the target position is a difficult task as we may have many simultaneous goals and constraints. For example, we may want to drive to an assigned waypoint, but we also want to avoid hitting any obstacles along the way. The challenge is picking a target position trajectory that accomplishes both of these goals. Autonomous behaviors are used to evaluate each individual goal, then the behaviors are combined using the Distributed Architecture for Mobile Navigation (DAMN). More details on this behavior based architecture appear in the DAMN Architecture section.

The position target generated by the "Thinking" stage is fed into the "Acting" block. The acting block consists of two components: Position Controller, and Velocity Controller. The position controller determines desired linear and rotational velocities that will drive us to our position target. These desired velocities are achieved by PID velocity controllers. The position and velocity controllers are discussed in detail in the Controllers section. An overall block diagram of the navigation system is shown in figure 3.

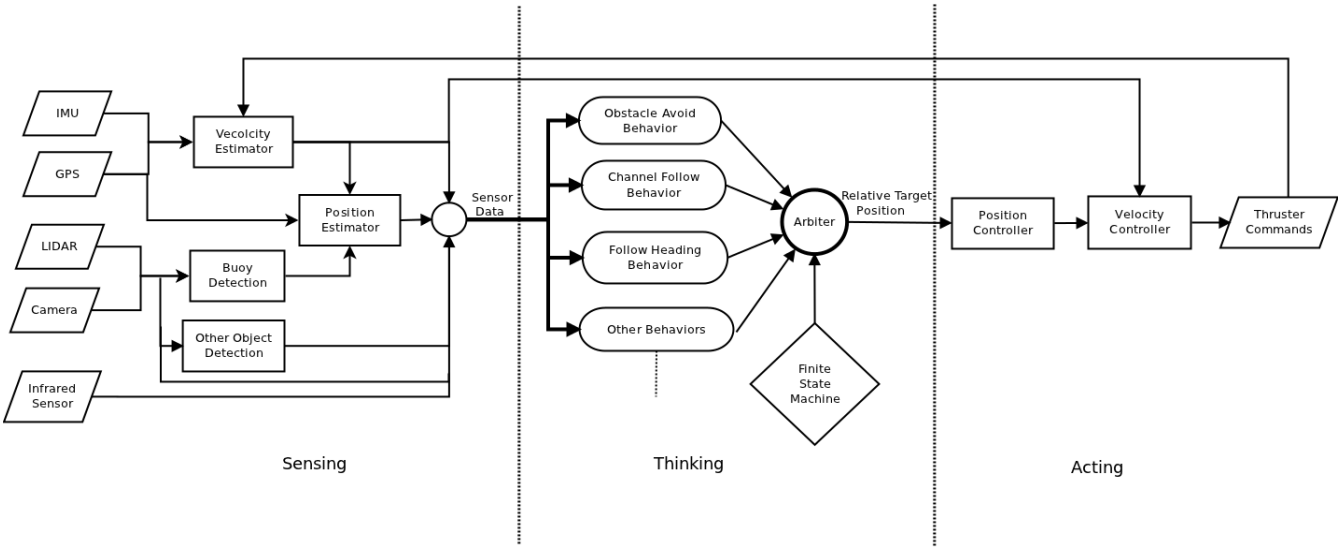


Figure 3: Navigation and Control System Architecture

4 System Identification and State Estimation

4.1 Identification of the Boat Dynamics

After the construction of the boat was complete, one of the first tasks we set out to accomplish was to obtain a model of the boat dynamics. Specifically we needed the input-output relationship between motor thrust and boat velocity. Of course the dynamics of boats is a well studied area, and a good approximation for the boat linear velocity dynamics is given by a quadratic equation where the drag produced by the water depends on the square of the velocity. Thus the boat dynamics are inherently non-linear. We chose to obtain a linearized model around V=0, assuming a linear approximation would be sufficient. There are many benefits of using a linearized model including ease of system identification, observer design, and control system design. For these reasons we chose to use a linear, state-space model to characterize our system in the following form:

$$\begin{bmatrix} \dot{V}_x \\ \dot{W}_z \end{bmatrix} = \begin{bmatrix} \frac{P}{M} & 0 \\ 0 & \frac{Q}{M} \end{bmatrix} \begin{bmatrix} V_x \\ W_z \end{bmatrix} + \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ u_{21} & u_{22} & u_{23} & u_{24} \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} \tag{1}$$

Here P, Q, M, and $u_{11}...u_{22}$ are unknown constants. V_x and W_z are the linear and rotational velocities, respectively. $T_1...T_4$ are the four motor thrusts. Notice we are also assuming that rotational and linear velocities are completely decoupled at low speeds.

4.1.1 Experimental System Identification

To determine the unknown parameters in our linearized model, we set up a series of experiments that would allow us to observe the input-output relationship of the system. The inputs to the system are simply the thrusts of our 4 motors. The outputs we would like to measure are the linear and rotational velocities. The outputs could be directly measured using GPS, and IMU data. The thrusters were tested using a fish scale, and calibrated, to remove any non-linearities in their thrust to power relationship.

A series to tests were designed to excite the system in various ways. We used a matrix of different thrusts to cause the boat to move forward in a straight line, rotate in place, or drive in a circle. With inputs and outputs logged and compiled from these various tests we were able to enter the information into matlab and obtain a linear system model. The model was derived as to minimize the error in a least squared sense between the model predicted system response, and the measured system response. Following this, we obtained the following complete system model:

$$\begin{bmatrix} \dot{V}_x \\ \dot{W}_z \end{bmatrix} = \begin{bmatrix} 0.4807 & 0 \\ 0 & -0.726 \end{bmatrix} \begin{bmatrix} V_x \\ W_z \end{bmatrix} + \begin{bmatrix} 0.05276 & 0.05276 & 0.05276 & 0.05276 \\ 0.03863 & 0 & 0 & -0.03963 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} \tag{2}$$

The plot below shows an example of a single experiment where the boat moved forward in a straight line. Both the measured response and model predicted response for linear velocity are shown in figure 4.

We found that by linearizing the thruster behavior, our linearized model was able to predict the actual behavior of the boat rather well at low speeds.

4.2 Velocity State Estimation

Notice in figure 4 that the measured velocity from the GPS is not only noisy, but also exhibits about a 1 second lag. If we were to close our velocity loop directly around this sensor data we would surely have problems. Additionally we have some additional, yet noisy, acceleration data from our IMU. The goal here was to combine the knowledge of our system model, with the accelerometer measurements and GPS velocity measurements to obtain a good estimate of our linear velocity. The first step was to condition raw sensor data so that it was in a usable state.

The accelerometer data tended to exhibit high frequency noise from the motor vibrations. This noise was attenuated first by taking the difference between the measured acceleration and the acceleration predicted

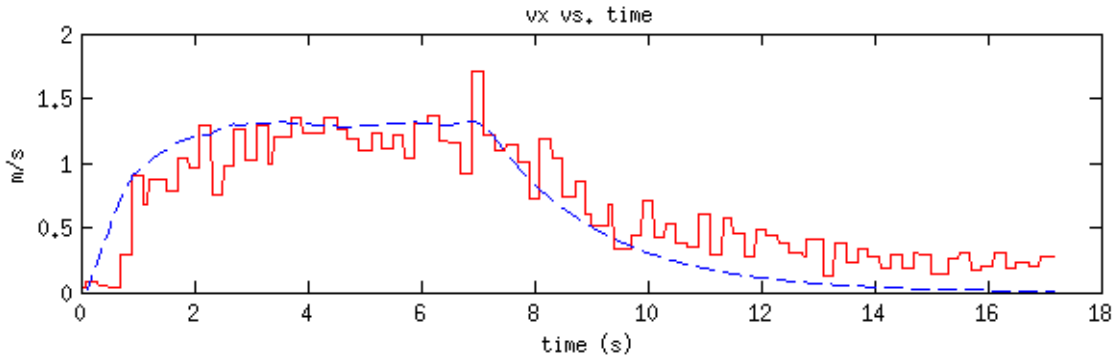


Figure 4: Simulated and measured system response. Actual data shown by red solid line, model predicted behaviors shown by blue dotted line.

by our system model. This difference was then passed through a low pass butterworth filter with a cutoff frequency of .5hz. Then, the filtered difference is added to the predicted acceleration. This allowed us to use a very low cutoff frequency on our filter, yet not attenuate accelerations that "should be there".

The GPS velocity data needed suffered from a different problem. The GPS antenna is not at the center of rotation of the boat. If the boat rotated in place, the GPS would measure a linear velocity as it traveled in a circle. To remove this phenomenon we multiplied the velocity by the cosine of the difference between the compass heading on the IMU and the heading given by the GPS velocity measurement. This would remove any velocity measured by the GPS that is not in the direction of the boat heading.

With the raw sensor data conditioned we then passed everything through a Kalman filter. The Kalman filter used the boat model obtained above as well as some knowledge of the sensor noise characteristics obtained through additional testing. The plot below shows one test where we moved in a straight line. The raw velocity measurement and estimated velocity are shown in figure 5.

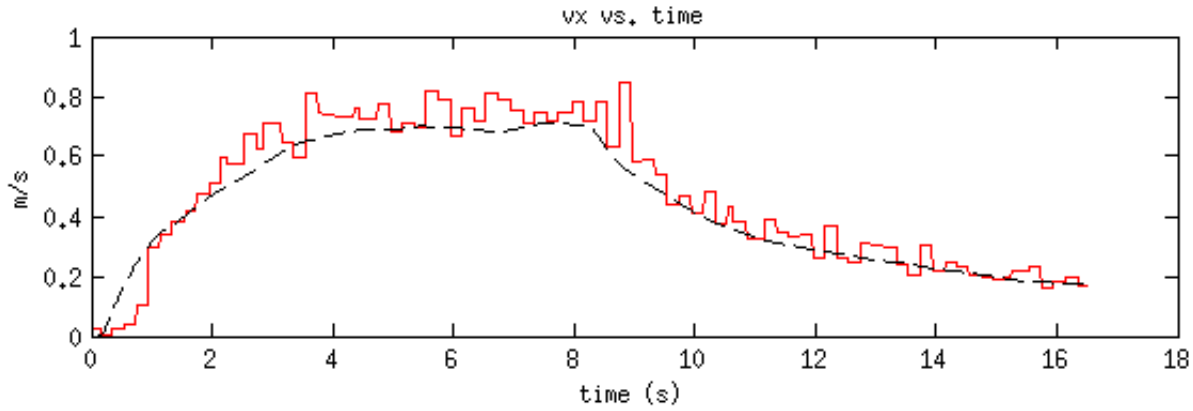


Figure 5: Measured and estimated system response. Actual data shown by red solid line, estimated velocity shown by black dotted line.

Using the Kalman filter we were able to greatly improve our velocity estimate, and eliminate the lag inherent in the GPS velocity measurement. We then repeated this approach for rotational velocity estimation using our system model and gyroscope data.

4.3 Position State Estimation

After extensive testing, we found the positional accuracy of our GPS sensor to be inadequate. We wanted to somehow get a much more reliable fix on our location. Here we sought to employ concepts from SLAM and apply them to our particular case. In addition to the rough GPS position we also have the ability to

perform dead reckoning by using our velocity and heading estimates obtained above. These two things alone could provide reasonable localization, however it could drift over time. To further improve upon this we wanted to use the additional information we obtained through our buoy observations. Since the buoys are fixed in space, and we need to locate them for the channel navigation task, they are well suited to be used as landmarks for SLAM.

Each buoy that is observed using LIDAR and computer vision is added as a state into a high-dimensional state-space system. The boat's position also acts as a state within this system. The buoy dynamics are trivial: they have none, since they are stationary. The boat dynamics are also simple because we know the boat velocity. The output of the state space system is our GPS measured position, and observed buoy locations. Thus the complete system model is given by equations 3 and 4:

$$\begin{bmatrix} \dot{Boat}_{x-pos} \\ \dot{Boat}_{y-pos} \\ \dot{Buoy1}_{x-pos} \\ \dot{Buoy1}_{y-pos} \\ . \\ . \\ . \\ \dot{BuoyN}_{x-pos} \\ \dot{BuoyN}_{y-pos} \end{bmatrix} = \begin{bmatrix} Vcos(heading) \\ Vsin(heading) \\ 0 \\ 0 \\ . \\ . \\ . \\ 0 \\ 0 \end{bmatrix} \tag{3}$$

$$\begin{bmatrix} GPS_{x-pos} \\ GPS_{y-pos} \\ Observed - Buoy1_{x-pos} \\ Observed - Buoy1_{y-pos} \\ . \\ . \\ . \\ Observed - BuoyN_{x-pos} \\ Observed - BuoyN_{y-pos} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & . & . & . & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & . & . & . & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & . & . & . & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & . & . & . & 0 & 0 & 0 \\ . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . \\ -1 & 0 & 0 & 0 & 0 & . & . & . & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 & 0 & . & . & . & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} Boat_{x-pos} \\ Boat_{y-pos} \\ Buoy1_{x-pos} \\ Buoy1_{y-pos} \\ . \\ . \\ . \\ BuoyN_{x-pos} \\ BuoyN_{y-pos} \end{bmatrix} \tag{4}$$

This is in the form of a standard state-space system. If we know the noise characteristics of our boat and buoy position measurements, and are able to correctly associate buoys with their states in the model, then we can use a Kalman filter to simultaneously update our boat and buoy position estimates. This should theoretically give us a much better estimate of both the boat position, and buoy locations.

At the time of this writing, the complete system has yet to be integrated and tested so we do not have any results to report.

5 Position and Velocity Control

5.1 Position Controller

Since this system can be treated as a "unicycle" type drive robot a non-holonomic position controller must be used. The robot is only able to have linear velocity along the direction it is facing, but it can also rotate about its center. Thus the position controller must determine the desired linear and rotational velocities based on the desired target position. This is done using the non-holonomic controller proposed by Olfati-Saber [?].

The method involves partial feedback linearization on the translation dynamics of the vehicle. This allows us to achieve global asymptotic stability with respect to the target position. Since the output from the DAMN arbiter is a target position relative to the robot, no direct feedback to the position controller is required (i.e. the input is already the position error).

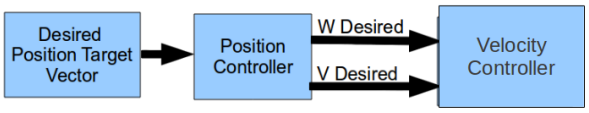


Figure 6: Position Control block diagram

5.2 Velocity Controller

The Velocity controller follows a basic PID type model. The linear and rotational velocity dynamics are assumed to be decoupled, and as such they each have their own PID loops. These two loops must be somehow combined to give the ultimate 4 thruster commands. This is done by making the sum of the 4 thrusters equal to the output of the linear velocity loop, and making the difference between the two outside thrusters equal to the output of the rotational velocity loop. The complete block diagram, including Kalman filter blocks, is shown in figure 7.

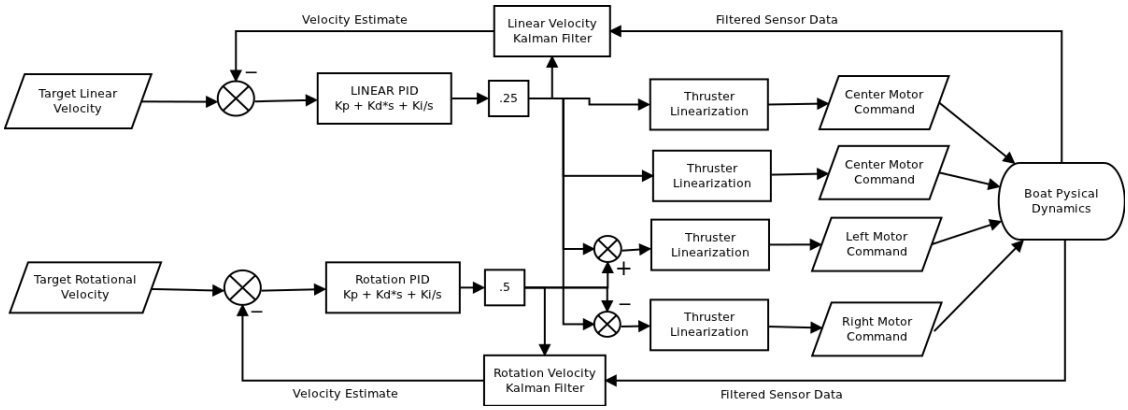


Figure 7: Velocity Control Block Diagram

There are linearization blocks between the output of the PID loops and the thruster commands to remove the non-linear force-power relationship of the thrusters.

6 Behavior Based Control

6.1 DAMN Architecture

To combine multiple behaviors we employed DAMN (Distributed Architecture for Mobile Navigation). DAMN operates by discretizing the space around the robot. Each behavior votes for each spacial block according to how beneficial it would be for that behavior to set that block as the target position. Votes range from -1 to 1. A vote of 1 corresponds to a block being a highly beneficial choice for that behavior, and a vote of -1 corresponds to a detrimental choice. These votes are then weighted and summed, and the block with the greatest vote is chosen at the position target. DAMN was a logical choice over motor schema or subsumption since it allows all behaviors to always be active, but also decreases the chances we get stuck in a local minimum. The challenge with DAMN was to discretize the space around the robot as to create cells on which each behavior could vote. This was solved by using a polar co-ordinate system centered about the robot. Cells were created by 180 evenly spaced radial lines extending from the origin, and several concentric circles of different sizes, centered about the origin. Since we care more about space closer to the robot, the diameters of these concentric circles grow exponentially with the distance from the robot. This way we can have a limited number of cells, yet the area of cells close to the robot are small allowing precise positioning (see figure 8).

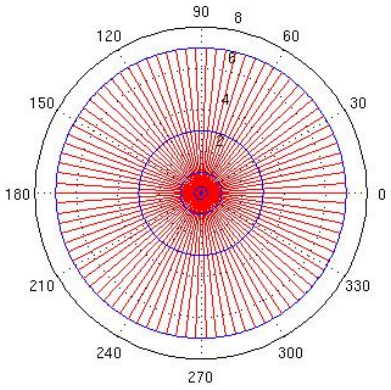


Figure 8: DAMN cell division. 180 radial lines, 6 concentric circles: 1080 total divisions. Angles are given in degrees, circle radius' are given in meters.

In in our case, we have many behaviors but 3 are fundamental: Avoid Obstacles, Go-to Waypoint, and Navigate Channel. The Avoid Obstacle behavior places a binary vote for each cell, either -1 or 0. If entering a cell could potentially cause us to his an obstacle then the vote is -1; it is 0 otherwise. This behavior is given sufficient weight so that any cell with a vote of -1 will never be the chosen destination. The behaviors are combined by the arbiter, with the weights of each behavior's votes dictated by the state machine. For example, when we want to navigate the channel, that behavior's vote is given significant weight. However, when we don't care about staying in the channel that behavior's weight is set to zero.

6.1.1 Go-to waypoint

The Go-to waypoint simply drives the robot to a given global reference point. This is converted to a vote according to the equation:

$$V_i = \frac{1}{||C_i - P|| + 1} \tag{5}$$

Here V_i is the vote for cell i, P is a vector pointing to the target position, and C_i a vector pointing to the center of cell i. In other words, the vote for a given cell is inversely proportional to the distance between the center of that cell and the target position. The end result of the combined behaviors results in the robot driving towards the closet point to the target position where it will not collide with an obstacle.

6.1.2 Channel Navigation

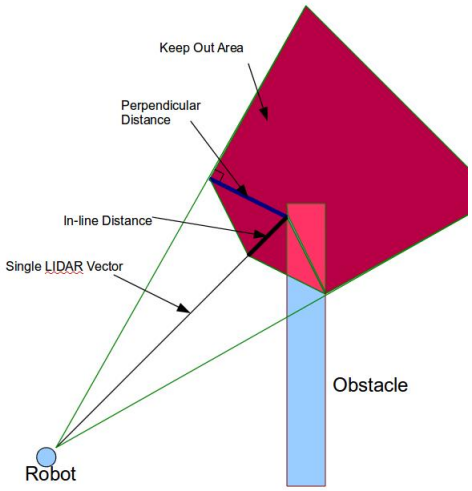
The channel navigation behavior attempts to keep the boat within the channel. Given the buoy locations, a polygon is formed using the buoy locations as vertices. Edges are formed by building a Gabriel graph which tends to create edges only between neighboring vertices. Excessive edges are trimmed until we are left with a well defined, closed polygon. The behavior then simply votes 0 for any points within the polygon and -1 for any points outside the polygon. We are also considering adding slight negative votes for points inside the polygon, yet close to the borders. This would tend to cause the vehicle to favor the center of the channel as opposed to the edges.

6.1.3 Obstacle Avoid

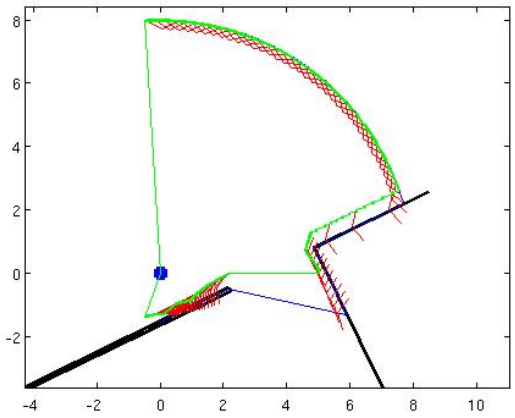
The reactive Obstacle Avoid behavior attempts to calculate the distance the robot can travel in every direction before colliding with an obstacle. Since this is a reactive behavior, it relies only upon the present laser scanner values. For the purposes of "growing" obstacles, the robot is treated as a rectangle, and it is assumed it can only drive in the forward direction (ie, it is non-holonomic).

For each angle between -135 and +135 degrees, the laser scanner returns the distance to the nearest detected obstacle. For each of these points a "keep out" area is generated according to the how close the robot is allowed to get to the obstacle. Each laser scanner ray is treated as a vector, and the "keep out" area is defined by two distance values: In-line and Perpendicular distance (see figure 9a).

When this is repeated for all laser rays a complete "keep out" area is generated where the robot could potentially hit an obstacle (see figure 9b)). Notice the robot cannot drive too close to an obstacle, yet it can go past an object with the minimum clearance required. If the robot drives to any point within the green boundary line, it will not collide with an obstacle. If a DAMN cell overlaps with any of this area, the behavior's vote is -1 for that cell.



(a) Single Laser Ray



(b) Full Laser Scan. The robot is the blue dot, obstacles are black lines, the laser data is the blue line. Each LIDAR vector's "Keep Out" boundary is shown in red. The inner boundary of the resultant "Keep Out" area is shown by the green line.

Figure 9: Visualization of the Obstacle Avoid Behavior

7 Conclusion

Georgia Tech Marine Robotics boat went through a total overhaul not only in mechanical design but also algorithm and software design. A complete hull redesign from PVC pontoons to the current design provided a much more hydrodynamic shape. A much needed upgrade to the computer system provided the capabilities to process all incoming data on time with the possibility of GPU utilization in the future. A complete restructure in the code provided JAUS compliant communications and easy component creation while also providing easy readability to future programmers. Lastly the algorithms developed and used provided means of state estimation and buoy localization each necessary to complete every task in the 2011 competition.