Captain Hindsight: An Autonomous Surface Vessel

Daniel Pickem [*] **David Morioniti** [*] **Chris Taylor** [*]
Santiago Balestrini-Robinson [*] **Dimitri Mavris** [*]

[*] *Georgia Institute of Technology, Atlanta, GA 30332 USA,*
{*daniel.pickem, ctaylor9, dmorionit3, sanbales*}*@gatech.edu*

Abstract: For the third consecutive year, the Georgia Tech Marine Robotics Group redesigned its autonomous surface vessel (ASV) entry to the AUVSI RoboBoat Competition.Using a trimaran design for stability and speed, new and improved sensors including a tilting LIDAR, an IMU, a GPS, and stereo cameras, the new design is capable of performing each mission. The ASV employs state-of-the-art algorithms to produce a highly capable and robust system. Algorithms include Kalman filtering for position and velocity estimation, a hybrid reactive-deliberative behavior-based control strategy, advanced point cloud generation, segmentation, and recognition algorithms, as well as stereoscopic vision.

1. MECHANICAL DESIGN

A new vehicle was created by the 2012 Georgia Tech AUVSI RoboBoat team building on previous experience to leverage existing advantages and address lingering issues. A new hull was built, maintaining the trimaran configuration while increasing displacement. This change reduces the overall drag of the vehicle and improves handling. The electronics box was completely redesigned to emphasize safety, reliability, and simplicity. The configuration of the hardware and power systems was updated at the same time to reduce complexity and clutter inside the box. The sensors and electronics are mounted to the main hull to maximize functionality. The vehicle has been thoroughly tested over the past semester and behaves as predicted by the initial analysis.

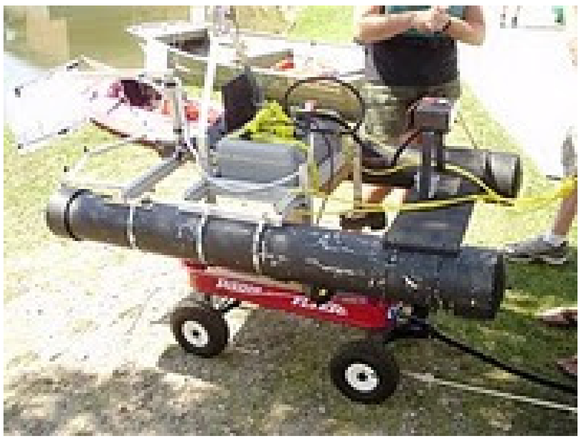
1.1 Hull Design

A trimaran hull design (see Figure 1(a)) was selected by last years team for the 2011 entry, to replace a heavy twin-pontoon design from 2010 (see Figure 1(b)). The selection was made because of the advantages that a trimaran can have over competing mono- and catamaran designs. Each option was qualitatively evaluated based on three criteria: weight, simplicity, maneuverability, speed, and stability. A monohull or trimaran has less structure than comparable twin-pontoon designs, and thus less weight. Many of the twin-pontoon teams in the competition use solid fiberglassed foam or heavy plastic for the pontoons, connected with plywood or aluminum bars. A trimaran can be hollow and requires very little structure to connect the pontoons. Pontooned designs, both twin- and tri-, provide superior stability in roll versus monohulls, and the overall design improves straight line tracking and performance. The combination of advantages that is found in the trimaran design made it the clear choice. Last years design decisions resulted in one of the lightest and best performing vehicles at the competition, and was key to our success. The same hull configuration was kept for this year, but redesigned to reflect some of the lessons learned. For instance, to minimize drag the draft of the vehicle should be such that the pontoons touch the water only to provide stability, not buoyancy. The hull was also designed with prior knowledge of the various sensors and electronics that the vehicle would be required to carry, so the design and configuration could be developed together.
Paramarine [1] , a ship and submarine design software, was used to design the hull based on the expected weight of the vehicle and the desired performance. This program allows us to accurately analyze the stability and the buoyancy of the vehicle without the need to build a prototype. Figure 2(b) shows the GZ Curve for the design as generated by Paramarine (see Figure 2(a)). The results indicated that the design is extremely stable and can recover from a 60 degree heel. The Paramarine analysis allowed us to optimize the freeboard of the vehicle to reduce the above-water profile in order to reduce the effect of wind on the craft. The resulting vehicle is three feet wide and just over four feet long, emphasizing the teams focus on weight: the vehicle is only as large as it needs to be.
Foam sections of the hull were cut with a waterjet and assembled around laser-cut plywood bulkheads. The entire assembly was fiberglassed and then the foam was removed leaving the bulkheads and shell in place. A gunwale was attached to provide stiffness along the top edge, and the decking was laser-cut out of thin luan plywood ensuring precise fit to the hull. The pontoons were shaped from foam similar to the hull sections, and mounted to an aluminum frame. The aluminum frame attaches to the main hull via a small rail, allowing for the center of buoyancy to be easily adjusted. Four Seabotix thrusters are mounted to the bottom of the hull for propulsion. The finished main hull is shown in Figure 12.

1.2 Electronics Box

The electronics box was redesigned with an emphasis on simplicity and reliability. The layout of the box last year

[1] http://www2.qinetiq.com/home_grc/products/paramarine.html


(a) 2010 twin-pontoon design

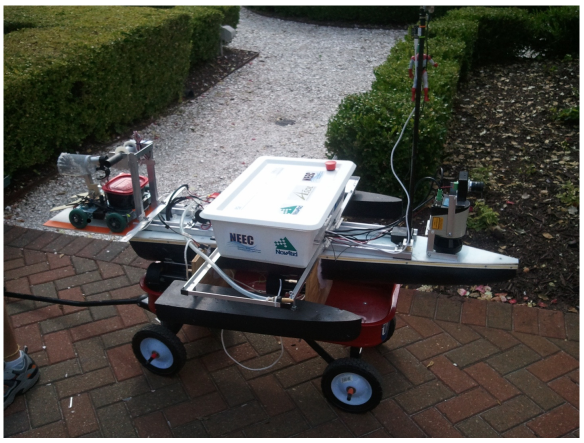
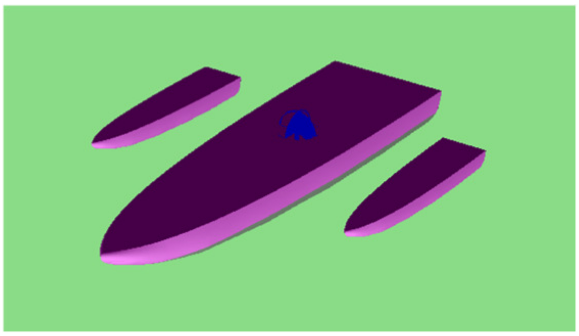
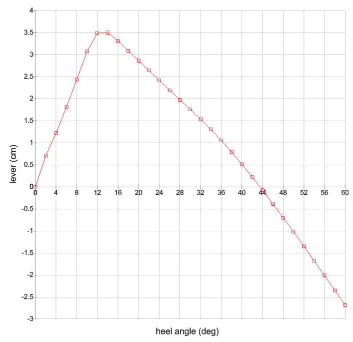

(b) 2011 trimaran design

Fig. 1. Georgia Tech MRG entry for 2010 and 2011


(a) 3D model of the trimaran design



(b) GZ curve

Fig. 2. Paramarine model of the 2012 hull and GZ curve

disorganized and produced through an ad-hoc process, shown in Figure 5(a), and could have led to significant mistakes or lost hardware. The team took two approaches towards addressing this issue. The first approach is to build a larger box, providing more room for components, better access to wires, and a cleaner layout. Component layout and wire paths were planned out in advance, and then the box was constructed around the configuration. This method relaxes the normal constraints of attempting to fit fixed-size components into a highly constrained space. The second approach is to reduce the number and lengths of wires that are required (see Figure 5(b)). This was achieved through a modular structured configuration based around the Deutsches Institut für Normung (DIN)

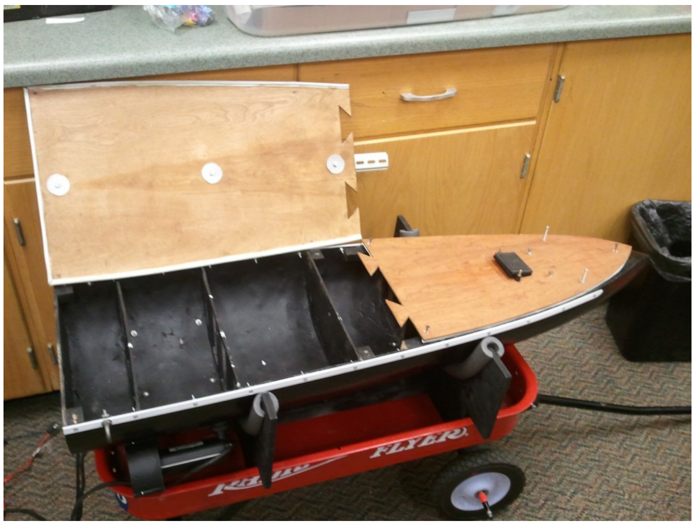


Fig. 3. Finished main hull

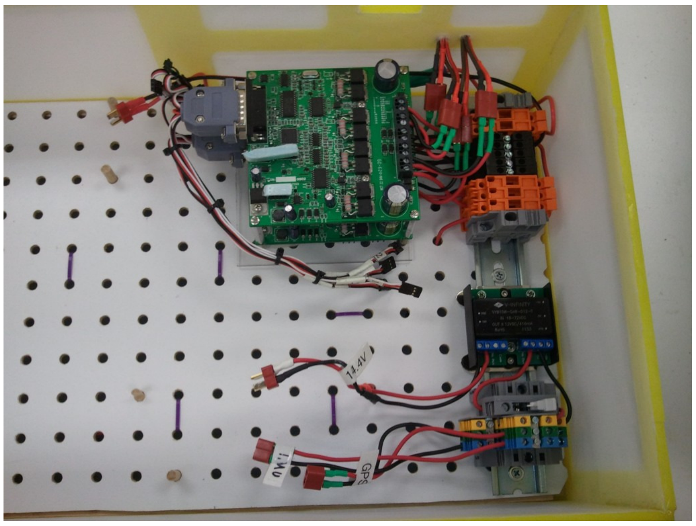


Fig. 4. DIN rail setup of this years electronics box

standard (see Figure 4). This method uses plastic terminal blocks that link forming circuits. The terminal blocks are all insulated and protected against shorts. These blocks are mounted on a common rail and secured in place. This structured layout supports the design process and allows for the flexibility to expand and adjust the configuration over time.

1.3 Electrical Design

Utilizing the DIN rail system for connecting components drives the design for configuring power and data connections to all the components. The main focus of the team during this process was on reliability, safety, and simplicity. Components should be protected against shorts by anything from water to a dropped tool. Wires and connections should stay secure while working on, transporting, and running the vehicle. The boat electronics are divided into two separate systems, propulsion and sensors. Each system is powered from a dedicated battery to prevent power surges, and to allow R/C control over the vehicle even if the sensors or computer are offline. The propulsion system is powered from a six-cell, 5000 milliamp-hour lithium polymer (LiPo) battery, while the sensors are powered from a four-cell 5000 milliamp-hour LiPo battery. The use of LiPo batteries reduces the weight and size of the vehicle while providing the necessary power requirements. A simplified power diagram is shown in Figure 6.

1.4 System Assembly

The finished electronics box sits inside the frame that connects the pontoons to the main hull. This attachment provides additional stiffness to the frame, and allows the box to shift with the frame to adjust center of mass and pitch. Two cameras for stereo vision mount to the corners of the frame, maintaining a set distance and maximizing depth perception at farther distances. The water pump sits behind the box, attached to the same rail, and the nozzle is mounted to an adjustable plate so that the angle can be changed for accuracy. The 3D LIDAR mount sits at the nose of the vehicle to provide an unobstructed view forward. The IMU mounts just aft of the LIDAR. The temperature probe is attached to the tilting LIDAR to allow the probe to be pointed at the Hot Suite. Finally, the lid of the box is actually a solar panel that provides all of the power to run the cooling fans, preventing the laptop from overheating. The solar panel generates enough power to also run both the IMU and GPS. However for overall system reliability during the competition those components will be connected to the batteries. The fully assembled boat is shown in Figure 7.

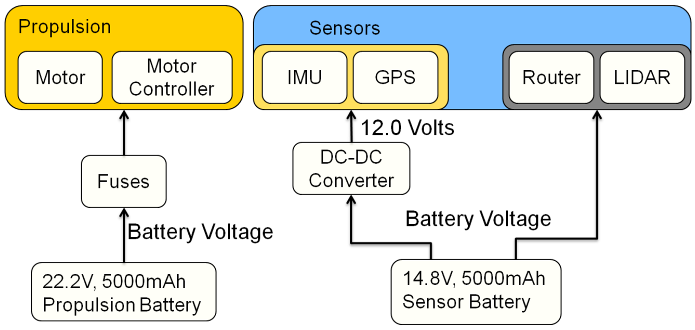


Fig. 6. Power diagram for propulsion and main sensor systems. Laptop operates from its own battery.

2. SOFTWARE ARCHITECTURE

This year, our boat's entire software architecture is based on the Robotic Operating System (or ROS, Quigley et al. [2009]) and Gazebo (Koenig and Howard [2004]), a 3D simulator. Both of these frameworks are open-source and are supported by a large and growing community offering a variety of packages. ROS seamlessly integrates Gazebo as well as external libraries like the OpenCV computer vision library [2], the PCL point cloud library [3], and the ODE dynamics engine for simulation. It defines a decentralized architecture based on message passing between individual modules (also called nodes). The biggest advantage of ROS though is the straightforward switch between simulation and real hardware. All that has to be changed are the drivers supplying the sensory inputs, i.e. whether the sensor data is published by Gazebo or by real hardware sensors. ROS itself and all modules that process sensor data are agnostic to the underlying low-level driver architecture meaning that once sensor data is published in the form of ROS messages, all higher software layers can use said data and are not concerned with whether the data was created by a simulation or by physical sensors. The hardware abstraction layers in the form of drivers is therefore completely transparent to all higher level layers. The only purpose of hardware drivers is to read data from sensors and publish that data as ROS messages.

2.1 Communication - Internal and External

All modules or nodes as they are called in ROS run as separate processes that communicate with each other via ROS's message passing architecture. Only one central instance is required that coordinates the message passing. This process is called the ROS master node and is on the boat's laptop. Every node registers with the ROS master and can then participate in message passing - either passively as a subscriber or actively as a publisher. Each

[2] http://opencv.willowgarage.com/
[3] http://pointclouds.org/

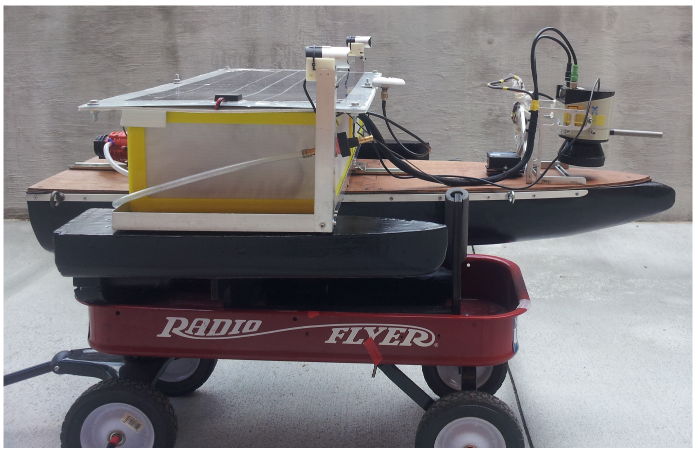
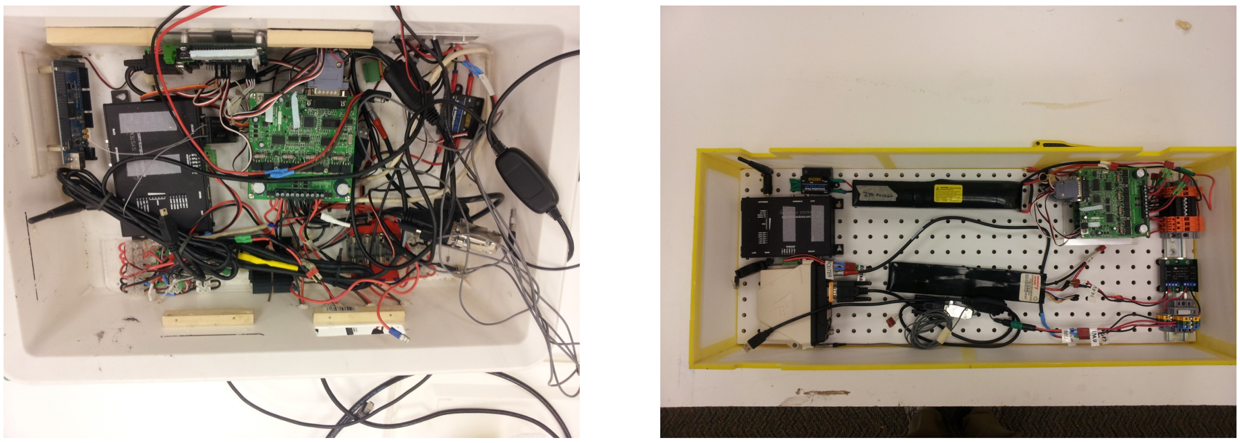


Fig. 7. Completely assembled boat.



(a) Layout of electronics box from 2011

(b) Layout of this years electronics box

Fig. 5. Comparison of last year's and this year's electronics boxes

message is strongly typed meaning that only messages containing a specific data structure can be published on each topic whereas a topic is the identifying string describing a message stream. Internal (on the boat's laptop and all its sensors and subsystems) and external communication (with the ground station) are fully identical, where each module is implemented as a ROS node that subscribes to the desired topics. In a networked ROS setup, it does not matter whether a node is executed locally on the boat or on any other machine connected to the same network. Last year's architecture used shared memory, mutexes, and locks. These were replaced with message passing and callback functions. A node can subscribe to a topic and define a callback function that is called every time a new message is published on that topic. Therefore, the system operates fully asynchronously. In addition to the networked ROS configuration, we have set up a remote desktop that gives the ground station full access to the boat's laptop via a Wifi connection, i.e. a wireless bridge between the boat's and the groundstation's wireless router. Therefore, we can remotely restart individual nodes, reprogram and recompile them directly on the boat.

2.2 Drivers

Last year's drivers were written utilizing the cross-platform Boost Asynchronous Input/Output and Boost Thread libraries that is fully supported by ROS. For this year's competition we ported all of these drivers to support the ROS architecture. Since ROS integrates the Boost libraries, no major changes to most of the drivers were required. Only the GPS driver underwent a major rewriting to comply with ROS. Additionally, as a result of switching our motor controllers we had to implement a driver that interfaces with the servo controller that in turn controls these controllers. The servo controller driver was implemented in C++ based on the original C# implementation and integrated into ROS. Another innovation on the hardware side - the tilting LIDAR of this year's boat that is controlled by a digital Dynamixel AX12 servo - required us to implement a driver for this type of servo as well.

2.3 Simulation Environment

Most of our software development had to happen in parallel to the building of the boat. Before an actual test on the real hardware could take place, we simulated the boat, its sensors, and dynamics in the 3D simulator Gazebo. Gazebo is integrated into ROS and seamlessly hooks into ROS's message passing architecture allowing for simple interfacing with all our code.
One of the biggest advantages of this ROS-Gazebo setup is the fact that it allows for simple switching from simulation to real hardware. ROS is designed with the goal of minimizing the gap between simulation and real hardware. Therefore, the only modules that have to be changed are the drivers that feed sensor data into our ROS-based software architecture. ROS itself only defines an interface to feed sensor data into the system, the underlying sensors and their hardware abstraction are fully hidden from ROS. Gazebo not only simulates physical properties of interconnected rigid bodies, such as friction, gravity, mass, etc. but also generates sensor data including GPS and IMU data and camera image streams. This setup allowed us to test every module of our system architecture for the boat in simulation before deploying it on the boat. We were able to test code for position and velocity estimation and control but also all behaviors outlined in Section 3.2 including obstacle avoidance, global waypoints, blob tracking and

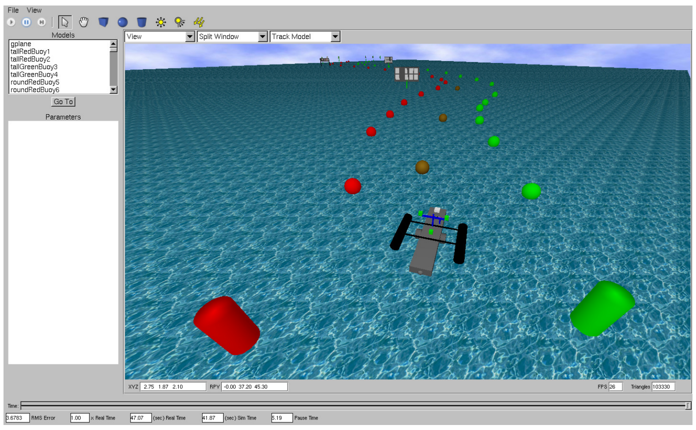


Fig. 8. Gazebo's simulation environment.

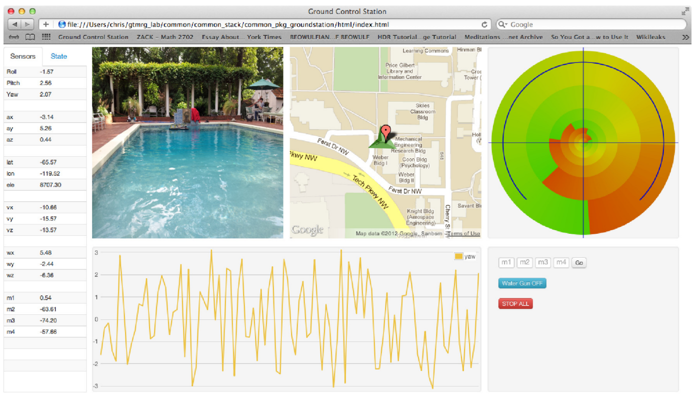


Fig. 9. Groundstation

even template matching and point cloud generation and segmentation. An example of a simulated buoy channel in Gazebo is shown Figure 8.

2.4 Ground Station

In order to control, monitor, and debug the boat throughout the development process and also in the deployment phase, we have developed a variety of visualization and control tools.

Monitoring Our webbrowser-based ground station features a fully integrated webserver, a ROS-interface, WebGL-based 3D visualization of LIDAR data and detected objects, as well as a Google Map overlay that allows us to drop GPS waypoints via mouse click. It interfaces with the ROS master node that runs on the boat by subscribing to the published topics and visualizes the data in any webbrowser supporting JavaScript and WebGL. This is also one of its main advantages and design motivations as it can be viewed from any device with a browser, even smartphones (see Fig. 9).

Controlling The control center is meant to complement the ground station in that it allows precise control over the boat, down to the level of issuing single motor commands. It allows the remote controlling of the robot via ROS and a networked setup between the boats laptop and any laptop on the shore that is connected to the same network. As Fig. 10 shows, one can de-/activate individual behaviors, set the boat into a specific mission mode, set local and global waypoints, and control the watergun. Additionally it shows the status of all connected sensors. The control center is implemented in C++ and makes use of the Qt library. Like any other node it subscribes directly to the topics and services published by the ROS master node running on the boat and utilizes ROS's message passing architecture.

Debugging For debugging purposes we have also implemented what we call the vote visualizer that shows the combined vote matrix and the current local target position that the DAMN arbiter outputs (see Section 3.2). In combination with the control center, this handy tool allows for the debugging of individual behaviors or any combination of behaviors (see Fig. 14). The vote visualizer is a C++ based ROS node that uses OpenCV's drawing functions.

3. SYSTEM ARCHITECTURE

The boat's system architecture resembles the Sense-Think-Act paradigm with an additional behavioral control aspect (see Brooks [1986]). In the literature this is commonly referred to as Hybrid Reactive-Deliberative Control (see Bekey [2005]). In the Sense-Think-Act paradigm, each of these three fundamental stages feeds data into the next. Therefore, sensor data acquisition (see 3.1), processing (see 3.2), and actuation (see 3.3) are done sequentially. Yet the processing stage (or thinking stage) runs multiple behaviors in parallel, where each behavior (e.g. obstacle avoidance, hold heading, channel navigation, etc.) outputs a local target position preference in the form of a vote matrix. All behaviors' votes are then fused in the Distributed Architecture for Mobile Navigation (DAMN) arbiter (see Section 3.2 and Rosenblatt [1997]) and fed into the acting stage that consists of the position and velocity controller as well as the actual thruster motors.

As can be seen in Fig. 11, our main focus this year lies in sensor data acquisition and processing, all part of the sensing stage. Unlike last year, we use a tilting laser range finder and stereo cameras to compute point clouds containing 3D data. In addition to point cloud segmentation and clustering, we also run a stereo blob tracker that computes positions of buoys in 3D space and a feature extraction module that compares the current scene with a set of templates to identify mission stations. The sensing stage is described in detail in Section 3.1.
Our thinking as well as acting stage remains largely unchanged from last years architecture, as we were able to use most of our behavior-based controls architecture and position and velocity control and estimation. Only the mission-related behaviors and the finite-state machine - our mission planning module - underwent a major redesign (see Section 3.2 and Section 3.3).

3.1 Sensing

System Model and State Estimation Both the system model and the state estimation modules could be reused

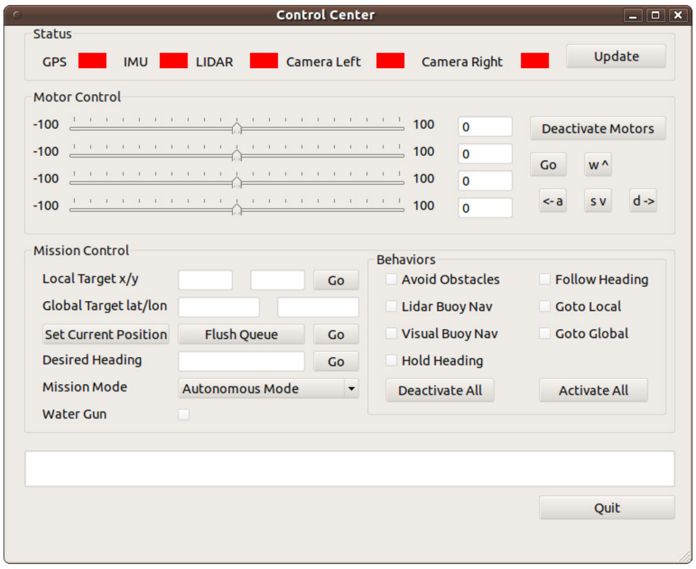


Fig. 10. Control Center

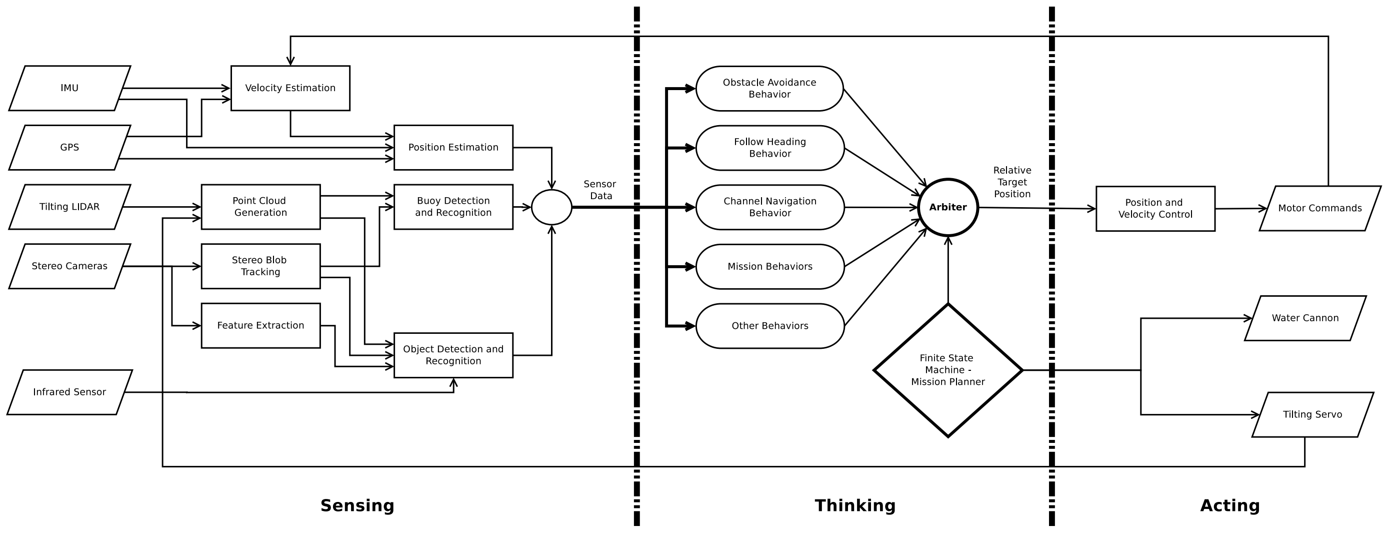


Fig. 11. Hybrid-deliberative system architecture.

from last year. The dynamics of our boat are described by a linearized system model around zero velocity. Even though boat dynamics are inherently non-linear, we chose a linearized approximation for ease of system identification, observer, and control design. Also, we assume that at low speeds, rotational and linear velocities are decoupled and can be estimated and controlled independently.
For linear and rotational velocity estimation, we feed conditioned and denoised sensor data into Kalman filters that use the boat model and additional sensor noise characteristics. These Kalman filters greatly improved our velocity estimates through fusing data from the GPS and IMU sensors. Position estimation is done in a similar fashion using a Kalman filter that fuses GPS data and a dead reckoning estimate based on our boat model and accelerometer data.

3D Sensing Our current boat features a tilting laser range finder that allows us to generate three-dimensional point clouds of the environment in a 270 degree horizontal field of view and a tilt angle that can be changed at runtime. The tilting LIDAR assembly contains a SICK LMS 111 laser range finder and a Dynamixel AX12 servo to move the LIDAR. Since LIDAR data is technically just an array of ranges corresponding to the horizontal field of view, we first have to project each ‘slice‘ into the boat's coordinate frame and then into a fixed global frame. These transformations depend on the current servo angle associated with each LIDAR slice and the current state of the boat, i.e. its orientation and position in 3D space. A point cloud is generated for each ‘sweep‘, i.e. all slices that are measured while the servo moves through its current angular range once. The resulting point cloud is roll, pitch, yaw, and velocity compensated.
The transformation from the moving LIDAR frame into the boat frame is done with a 3D rotation matrix using the current tilt angle of the servo, which is published at 50 Hz. Since the boat changes its current state during the measurement of one complete sweep, we have to compensate for the motion of the boat. This compensation is done using the roll, pitch, and yaw information from the IMU as well as the change in position that is obtained from the position estimator. A rigid body transform based on the changes in the boat state is then applied to each slice to transform all points into a fixed coordinate frame.

Point Cloud Segmentation Point clouds contain a large amount of data in the form of thousands of points from which we need to extract useful information, i.e. clusters, shapes, and features. This segmentation is done with the help of the Point Cloud Library (PCL). In a first step, we extract clusters of points together with their centroids, bounding boxes, and covariances. We then run a shape matching algorithm on each of these clusters, matching primitive shapes provided by PCL into those clusters. Currently, our algorithms can match spheres, cylinders, and planes into clusters of points. All of these recognized primitive shapes and also all clusters that could not be matched with any shape are then GPS-tagged, i.e. their global position is computed and stored. These GPS-tagged objects can then be used in channel navigation or camera-aided object recognition as explained in the following section. We chose this approach of GPS-tagging over SLAM because of its compatibility with our reactive behavior-based control approach.

Camera-aided Object Recognition The robot features a stereo camera system using two Microsoft Lifecam Studio webcams mounted on the electrical box of our boat. This setup results in a baseline between the cameras of 0.8 meters giving us a large field of view in which 3D information is available. The calibration of the stereo vision system, i.e. determining the intrinsic camera parameters and rectification and projection matrices was done using the camera calibration ROS package. In order to compute 3D information from a stereo vision system one needs to associate pixels in the left image to pixels in the right image. This is commonly done with a block matching algorithm that is fast enough to run in real time. In our experiments with the block matcher provided by OpenCV though, the resulting 3D point cloud was too sparsely populated. Therefore, we implemented the following two algorithms to extract 3D information of objects from both camera streams.

3.1.4.1. Stereo Blob-tracking with automatic color calibration We employ a stereo blob tracking algorithm based on the OpenCV computer vision library that is currently able to track red, green, blue, and yellow objects to find the corresponding buoys in the left and right camera frames and compute their location in 3D space. Independent blob tracking in both the left and the right image is necessary because only then we can compute a disparity value between the same object in both frames and project the blob from image coordinates into 3D coordinates. Disparity is simply the distance in pixels of the origin of an object in the left image and its origin in the right image. Therefore, a matching is required. After our stereo blob tracker has found blobs in both frames, we match buoys based on distance (i.e. the L2-norm) in image coordinates. A blob in the left image and the closest blob in image coordinates in the right image are assumed to be the same blob and allow us to compute a disparity value. The image coordinates together with the disparity enable us to project a blob into 3D space and compute the 3D location of each blob.
One inherent problem with blob tracking in an outdoor environment and outdoor computer vision in general though is how easily it is disturbed by changing lighting conditions, reflections, and glare. Specifically color values in the RGB color space that we use change significantly as lighting conditions or reflections change. To make our vision system more robust to these disturbances we implemented an automatic color calibration algorithm. It is based on the assumption that we know the color of a specific region in the image. We therefore place colored blocks showing all the colors that we want our robot to be able to detect (currently red, blue, green, and yellow) directly in front of each camera. The blob tracking algorithm continuously reads the color values of the corresponding pixels and adjusts the thresholds it uses for detecting colors.

3.1.4.2. Template matching In addition to tracking colored blobs, the successful completion of missions requires a method of recognizing mission stations. We chose an approach that combines point cloud segmentation and template matching. Specifically, we feed distances and sizes of objects retrieved from the point cloud segmentation into the template matching module. These objects are then projected into the current camera image to constrain the search space for the template matching algorithm. The template matcher then extracts local feature descriptors from both the template and the current object in the image and computes a confidence level of the match. Specifically, we use SURF (Speeded Up Robust Features, see Bay et al. [2006]) feature descriptors that are supported in the OpenCV computer vision library. The main advantage of SURF features is that they are scale- and rotation-invariant allowing the template matcher to recognize a match from any angle and distance. Additionally, this approach simplifies the creation of templates significantly, since a template is simply a cropped camera image of the mission stations.

3.2 Thinking

This stage combines all the sensory inputs through behaviors and generates motor commands to feed into the acting stage. We employ a behavior-based architecture using individual behaviors, a DAMN arbiter that generates target positions based on the behaviors, and high-level control realized as a finite state machine.

Finite State Machine Our high level control - or mission planner - is realized as a finite state machine that controls the weight of each behavior and therefore its influence on the target position in the DAMN arbiter. By setting weights to zero, it can also dynamically deactivate behaviors. All behaviors run continuously all the time - each as its own process - but just compute vote matrices if their weight is not set to zero. The transitions from one state to another are based on the detection of objects such as the blue buoy marking the end of the channel or any mission station. Additionally, the mission planner ensures that the boat always remains in a defined state and never gets stuck in an obstacle or attempting a task.

DAMN Arbiter The DAMN arbiter sole purpose is to combine the vote matrices of all behaviors and compute a local target position that it feeds into the position controller. The local coordinate frame is centered about the robot and uses a discretized polar coordinate system. 128 evenly spaced radial lines extending from the origin and 8 concentric circles centered about the origin ranging in distance from 0 meters to 8 meters result in a total of 1024 local target positions each behavior can vote for. Votes range from -1 to 1 where a 1 corresponds to a highly beneficial choice for a behavior and -1 for a detrimental choice. The DAMN arbiter computes a weighted sum of all vote matrices and choses the vote matrix element with the highest vote as its target position. The radii of the circles grow exponentially since precise control of the boat requires a higher resolution closer to the boat rather than further away.

Behaviors As mentioned before, all behaviors run simultaneously on the boat. Yet, the finite state machine

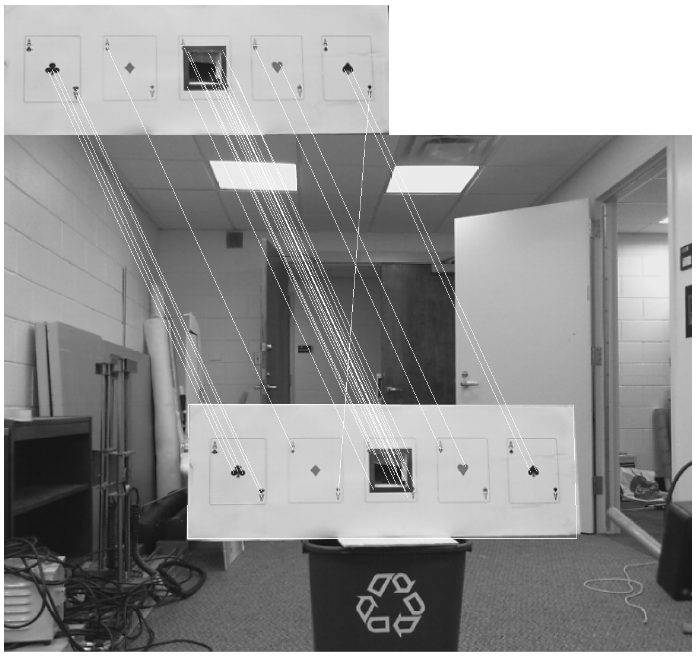
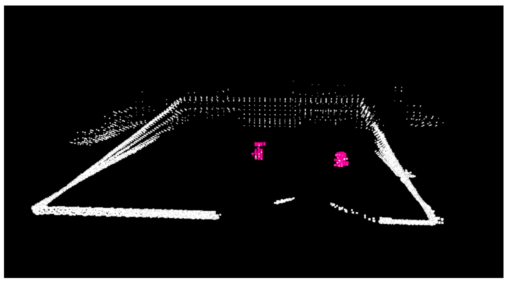
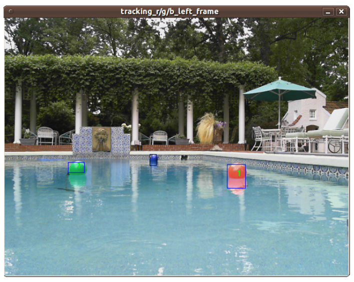
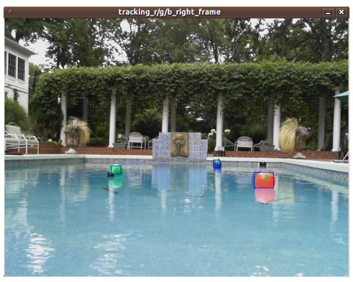


Fig. 12. Example of a matched template of the cheater's hand


(a) Point cloud generated with the tilting LIDAR



(b) Left camera image with tracked blobs indicated by bounding boxes



(c) Right camera image with tracked blobs indicated by bounding boxes

Fig. 13. 3D sensor data from both the tilting LIDAR and the stereo camera system with active blob tracking

determines, whether a behavior outputs a vote matrix or not, thus dynamically switching those behaviors on that are required for the current mission mode.

3.2.3.1. Obstacle Avoidance This reactive behavior determines how far the boat can move in each direction without colliding with any obstacles. It relies on sensor data from the laser scanner and is augmented by detected objects from point cloud segmentation. In order to take the boat's finite dimensions into account, this behavior also has to grow obstacles. The growth of obstacles depends on the length and width of the boat as well as on the distance to an obstacle. This dynamic growing and shrinking of obstacles makes the robot turn and react faster to closer obstacles. This behavior treats the robot as a rectangle with non-holonomic properties, i.e. that it can only drive in the forward direction. Also, this behavior is always given the highest weight, since it is most important that the boat does not run into obstacles and therefore does not get stuck.

3.2.3.2. Channel Navigation The channel navigation behavior combines input from the tilting LIDAR, the segmented point clusters and detected objects, as well as from the stereo cameras. The point cloud generated by the LIDAR is segmented into disconnected clusters with algorithms from the Point Cloud Library. We then apply basic shape recognition to those clusters, determining whether a cluster represents a plane, sphere or cylinder. Since LIDAR can't detect the color of objects, we have to augment these objects with color information from our stereo vision system. Therefore, we project the origin of each cluster into the camera image space and determine the color of each object in the image. This of course, requires calibrated stereo cameras as outlined in section 3.1. Once we have detected buoys (i.e. spherical or cylindrical red and green objects), we apply multiple behaviors to generate the next waypoint. In order to increase robustness of our channel navigation behavior, we combine three behaviors. The first picks the closest buoy pair and generates a waypoint between that pair. The second behavior computes the center of gravity of all found buoys. And the third behavior tries to identify two buoys of the same color and computes a channel direction. A waypoint is then generated such that the boats direction aligns with the channel direction.

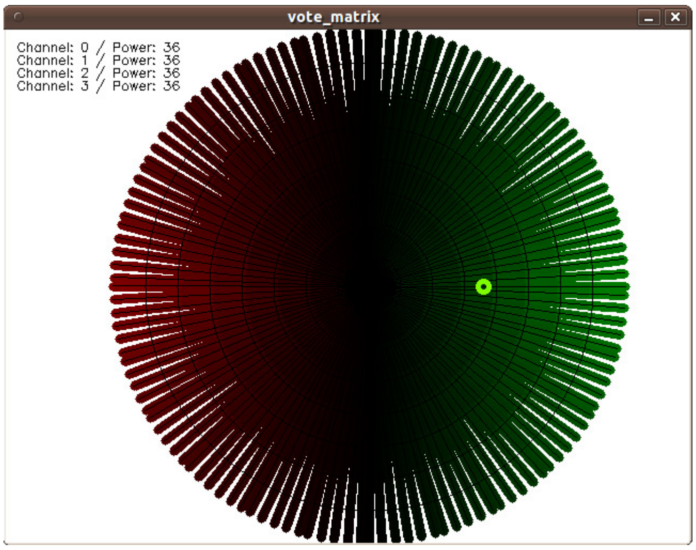


Fig. 14. Vote matrix visualization.

3.2.3.3. Go to Waypoint The go to waypoint behavior allows us to manoeuvre the boat to a waypoint in the local or global frame of reference. Therefore we can, for example, command the boat to move four meters forward or drive to a point with given latitude or longitude. The go to local waypoint behavior builds on the go to global waypoint behavior as it computes the global coordinates of the local waypoint given the boats position estimate and sends it to the go to global behavior. Go to global waypoint will also serve as a hold position behavior, as simply setting the waypoint to the current position will stop the boat from moving and only allow heading corrections.

3.2.3.4. Hold and Follow Heading The only difference between hold and follow heading is that hold headings highest vote is set to the innermost circle of the vote matrix and therefore only results in a rotational velocity command whereas follow heading also adds a linear velocity component. Both of these behaviors will be used in accomplishing mission tasks and the channel navigation. Follow heading will allow us to direct the boat to mission stations given the heading to each station from the end of the channel. Hold heading on the other hand will allow us to lock the boat's orientation such that it will face the mission station during an attempt to complete a mission.

3.2.3.5. Mission Behaviors Mission behaviors build upon the previously outlined fundamental behaviors and our 3D sensing capabilities.

The cheater's hand, the challenge of shooting the water gun at a target will be accomplished through a combination of the hold position and hold heading behavior, where the desired heading will change in such a way that the water stream sweeps across the cheater's hand sign until the raised flag indicating the completion of this station is detected.

For the jackpot challenge that requires the boat to push a button we will deactivate the obstacle avoidance and combine a follow heading behavior with visual blob tracking and a behavior that detects the impact of pushing the button. This combination of behaviors is similar to what's called visual servoing in the literature since information from the vision system of the boat will control its motion (see for example Hutchinson et al. [1996]).

For the hot suite challenge and reporting the hot target we employ both our template matcher and the infrared sensor mounted on our tilting LIDAR mount. Once the boat is holding its position in front of all four hot suite signs, it will rotate in place until it has template matched and measured the temperature of all targets with our thermal infrared temperature sensor. The hottest target will then be reported.

The poker chip challenge requires the boat to dock and deploy a subsystem to retrieve a poker chip from the dock. Again, it will use template matching to identify the dock and its orientation and hold its position in front of the dock. Since our subsystem is located at the rear end of our vessel, it has to rotate in place by 180 degrees and then deploy our subsystem - a six-legged walker. This hexapod runs the low-level control (i.e. inverse kinematics to control its legs and a gait control) onboard, while its directed to its target by the boat and its vision system.

3.3 Acting

Just as the position and velocity estimation, we were able to reuse our position and velocity controllers from last year. As mentioned in Macdonald et al. [2011], we use two PID control loops, one for linear and one for rotational velocity and a non-holonomic position controller. We chose this type of position controller because our boat can be treated as a "unicycle" type robot, which means that it can only have a linear velocity in the direction it is facing but also rotate around its center. The position controller therefore has to determine desired linear and rotational velocities based on the desired target positions according to the non-holonomic controller proposed by Olfati-Saber [2002]. These desired velocities are then fed into the two velocity PID control loops. The complete block diagram of our controls architecture including the Kalman filter-based velocity estimators are shown in Fig. 15.

4. CONCLUSIONS

The Georgia Tech Marine Robotics Group (GTMRG) boat went through a total overhaul not only in mechanical design but mostly in algorithm and software design. The hull as well as the electronics box has been rebuilt out of lighter materials based on last year's design and the wiring has been improved with a focus on simplicity and reliability. Additionally we have added a solar-powered cooling system for the electronics box to keep the main computer system cool even under the heavy load generated by processing all incoming sensor data in real time, including 3D point cloud generation and stereo image processing. Our main achievement on the software side was the porting of our entire codebase of GTMRG's participation in the 2011 competition to ROS and the use of the 3D simulator Gazebo. This approach allowed us to test most of our developed software in simulation before uploading it to the boat and testing it on the hardware.
Lastly the algorithms developed and used provide means of state estimation and object detection and recognition based on point cloud clustering and stereo blob tracking. Additionally, we have implemented template matching algorithms that will allow the boat to identify each mission station and a variety of objects necessary to complete every task in the 2012 AUVSI Roboboat competition.

REFERENCES

Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded-up robust features. In *9th European Conference on Computer Vision*, Graz, Austria, 2006.

George A. Bekey. *Autonomous Robots: From Biological Inspiration to Implementation and Control (Intelligent Robotics and Autonomous Agents)*. The MIT Press, June 2005. ISBN 0262025787.

R. Brooks. A robust layered control system for a mobile robot. *Robotics and Automation, IEEE Journal of*, 2(1):14 – 23, mar 1986. ISSN 0882-4967. doi: 10.1109/JRA.1986.1087032.

S. Hutchinson, G. Hager, and P. Corke. A tutorial on visual servo control. *IEEE Trans. on Robotics and Automation*, 12(5):651–670, October 1996.

N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2149 – 2154 vol.3, sept.-2 oct. 2004. doi: 10.1109/IROS.2004.1389727.

E. Macdonald, P. Dillon, C. Taylor, S. Culpepper, and D. Moroniti. Georgia institute of technology, May 2011. URL `http://mrg.gatech.edu/wordpress/wp-content/uploads/2012/03/Georgia-Tech-ASDL-2011-Journal-Paper.pdf`.

R. Olfati-Saber. Near-identity diffeomorphisms and exponential epsilon-tracking and epsilon-stabilization of first-order nonholonomic se(2) vehicles. In *Proceeding of the 2002 American Control Conference*, May 2002.

Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully B. Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

Julio Rosenblatt. *DAMN: A Distributed Architecture for Mobile Navigation.* PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, January 1997.

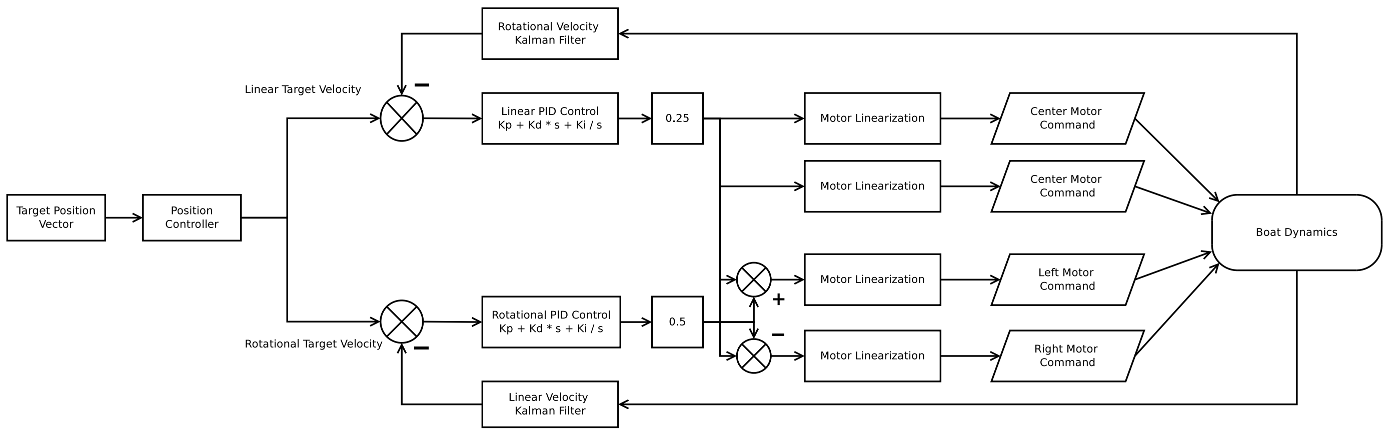


Fig. 15. Control architecture.