TEAM MIT-OLIN

RobotX Journal Paper

Student Team Members:
Arthur Anderson, Thom Howe, Nicholas Rypkema, Erin Fischell, Arturo Parrales, Tom Miller, Jonathan Garcia, Allan Sadun, Michael Defilippo, Victoria Coleman, Victoria Preston, Devynn Diggins, Mindy Tieu, Shivali Chandra, Nikolay Lapin, Paul Titchener, Alex Kessler, Madeline Perry, Jay Woo, Zoher Ghadyali, William Warner

Supervisor Team Members:
Michael Benjamin, David Barrett, Alon Yaari, Liam Paull

Abstract

The MIT/Olin RobotX team is comprised of students from the MIT Department of Mechanical Engineering and Olin College. Initial mechanical development took place at Olin with later mechanical and software development at MIT. For the propulsion system, two fixed-position Min Kota trolling motors are used. The sensor system consists of a single forward-look camera and a Velodyne-32 lidar and a Hemisphere Vector 102 GPS. The software system uses the MOOS-IvP middleware-autonomy architecture along with OpenCV libraries for vision processing.

1 Introduction

Figure 1: **Athena-Nike**: The MIT/Olin vehicle.

In 2013, the Office of Naval Research announced a competition, for 15 university teams from 5 different countries to compete in an Autonomous marine robot competition called RobotX in Singapore in October 2014. The goal of the competition is to complete 5 separate tasks without any human control or intervention. To compete, MIT partnered with Olin to create a competitive vehicle with advanced sensors and autonomy software architecture developed at Oxford and MIT called MOOS-IvP. The vehicle's name was named Athena-Nike, named after the Greek goddesses of wisdom and victory.

The vehicle is a 12 foot, double pontooned hull with a dynamic suspension system that supports a platform for the vehicle's sensors and electronic components above. The bareboat, not including the electronics, propulsion and power, was provided by the competition so that all RobotX competition teams are starting from the same baseline. This paper describes the design process and approach used for developing the propulsion and power, electronics, and sensing systems for Team MIT-Olin.

2 Technical Approach

Design decisions were driven by a minimalist approach to hardware and through leveraging existing software. Hardware decisions were governed by reducing the number of waterproof connectors, the number of individual components, and power requirements. Sensor interface, data message transport, and user interface operate within MOOS, a freely-available robotic middleware. Autonomy is achieved through IvP, a public open source behavior-based autonomy package [2].

2.1 Computing, Mechanical Systems

Hardware and mounting decisions are based on institutional experience of marine vehicle assembly. Multiple methods of achieving propulsion were modeled using software tools and a 1/10th scale model of the vehicle. Azimuth thrusters, front-mount motors, and a quad-motor design were investigated. Trade studies narrowed our selection of thrusters, batteries, and other on-board hardware components.

The physical relationship and important data connections between major system components is presented in Figure 2. With the exception of the acoustic acquisition systems, all computing and electronics systems are contained within a single waterproof enclosure. External sensors are cabled to the main enclosure for processing. In addition, a wifi radio connects to a shoreside computer where a user interface provides deployment control and monitoring of autonomous navigation progress.

2.1.1 Computing and Support Systems

Three main computers (supplied by sponsor American Technology Portwell, Inc.) operate the vehicle control and autonomy system. Autonomous decision-making, localization, state machine, and vision processing run on the main system computer. The second computer is dedicated to processing the Velodyne point cloud and extraction of features from the cloud. The third auxiliary computer is used exclusively for acoustic processing. The on-board enclosure also houses power conditioning circuitry, an Ethernet switch, and the Velodyne sensor's controller board.

2.1.2 Thrusters

The propulsion system consists of two Riptide Transom 80 Saltwater Transom Mount Trolling Motors. These motors were selected for their affordability and ease of use. Most importantly, their low weight precluded the need for installing hull extensions. The motors are attached to a mount system affixed to each of the pontoons. The mount system conveniently allows for retracting the motors during deployment and retrieval and allows the vehicle to rest on a floor when on the dock. During water operation, the motor shafts are lowered such that the motors are positioned completely below the pontoons. The motor shafts are fixed and do not rotate in place, so control is managed via differential drive.

2.1.3 Batteries

The batteries selected are two Torqeedo Power 26-104 batteries. The Power 26-104 is a top of the line lithium battery that is specifically designed for marine applications. These batteries are IP67 waterproof and have a built in automatic shutoff in case of flooding for added safety and protection. Besides built in safety of the Torqeedo, the most important feature of these batteries is that they have superior energy density to weight over that of a comparable lead acid marine battery. Each Power 26-104 has

2685 Wh of energy and only weighs 55 lbs. They provide sufficient power for the Riptide motors for a full day of testing on the water.

Computing power was designed with flexible power inputs. Lithium or nickel-metal-hydride batteries may be installed inside the computing case or a lead-acid battery can be connected externally. The emergency-stop system is also powered by a separate, dedicated battery.

2.2 Networking Design

The system design is fairly simple, and can be seen in the diagram presented in Figure 2. The heart of the system is the main computer box, which houses the majority of the electronics. The main computer box serves as a hub through which all the important information travels This computer box receives inputs from the the three main sensing systems - the LIDAR, camera, and GPS systems, and also connects to the acoustics box, which houses its own computer that talks to the hydrophones used for acoustic sensing. The main computer box also communicates through a wifi antenna to a shoreside computer, which is used to control the vehicle, as well as monitor it's autonomous state when operating autonomously.

The main computer ultimately outputs two engine orders to each of the engines to direct the vehicle, but before those orders given the the engines, they need to pass through a safety mechanism: the Emergency stop, or E-stop, box. The E-stop box severs as an intermediary between the motors on computers. The E-stop box overrides the computers when it receives any other orders from the the Operator Control Unit (OCU), or any of the manual emergency stop systems installed. More details of the emergency stop system are described in Section /refsec:e-stop.

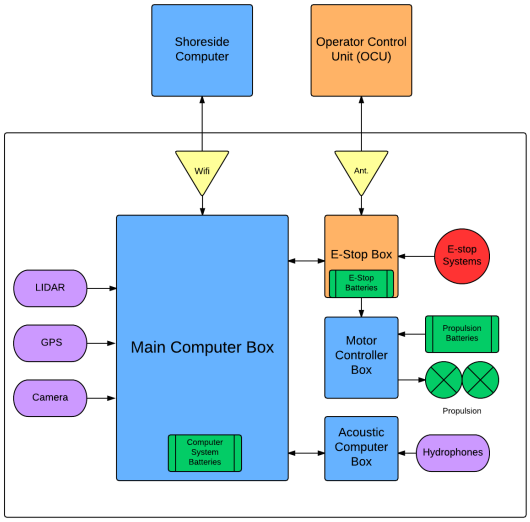

Figure 2: **Overall System Design:** The major hardware components on and off the vehicle, and how they connect with one another.

2.3 Emergency Stop System

The main objective of the on-board emergency-stop system is to stop the motors with fault conditions or when commanded to do so. Power for motor thrust is regulated by a Roboteq motor controller. Thrusters and drive batteries are connected directly to the controller outputs contacts and engaged in a differential-thrust mode. Roboteq controllers include sophisticated and field-proven logic for cutting power to the motors. On-board e-stop buttons and pull-cords are wired directly to dedicated inputs on the Roboteq.

An additional level of safety is achieved with an Arduino micro-controller. Thrust commands to the Robtoteq are commanded over serial line only from the Arduino. The Arduino receives commands from the on-board autonomy computers or preferentially from manual control on the dockside OCU. Watchdogs in the Roboteq and Arduino will stop the thrusters when communications is lost in any link be-

tween the Roboteq, Arduino, or dockside OCU. In addition, the OCU contains an e-stop button that stops the thrusters when engaged.

Figure 3: **E-Stop Button:** One of two E-stop buttons on the boat that cut power to the engines.

2.4 Sensing Systems

The key sensor components used to complete the taste are shown in Fig. 4. The GPS is the main navigation sensor and provides us with reliable global position and orientation of the vehicle. Objects in the environment such as obstacles and buoys are detected with the Velodyne HDL-32E 3D laser scanner that produces a dense point cloud at 2Hz. Consequently detecting objects on the surface fairly straightforward and robust. The final sensor used is a standard RGB camera mounted below the laser. The camera is used for identifying buoy color, the placard shapes for the docking task, and the LED panel sequence on the light buoy.

2.4.1 Object Detection

The Velodyne HDL-32E lidar is the vehicle's primary sensor for buoy and obstacle detection. Object detection from the 3D point clouds takes place in the ddConsoleApp app (Sec. 2.5.5).

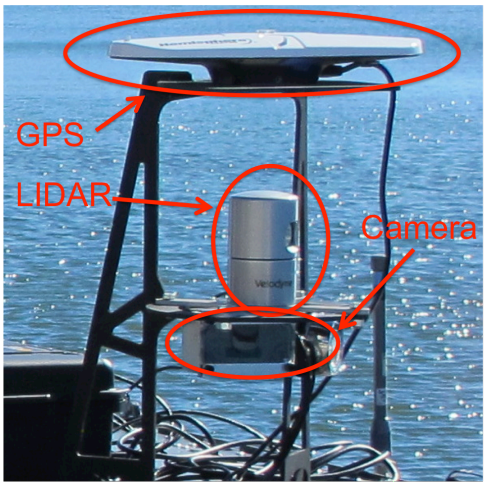

Figure 4: **Sensor Tower:** The sensor tower holds the camera, Velodyne, and GPS sensors.

Point-cloud culling is the first stage of the pipeline. In this stage, points are removed based on three criteria: firstly, points that are too close or too far from the lidar are removed (typically points within 4-30m are kept) , in order to mask out returns from the vehicle; secondly points that are outside an arcsweep are removed (typically only the points within an arc in front of the lidar are kept); and finally points that are outside a specified intensity range are removed, in order to remove low intensity returns from the water surface, which typically occur due to the formation of bubbles.

The second stage of the pipeline is clustering. In this stage, the points that remain after culling are down-sampled using a voxel grid with a user-specified leaf size, so as to reduce the clustering computation time. These down-sampled points are then grouped into clusters using Euclidean clustering, whereby points within a specified distance of each other are grouped together. In this stage, clusters with too few or too many points are removed. The centroid of the remaining clusters are then computed and stored for persistence checking.

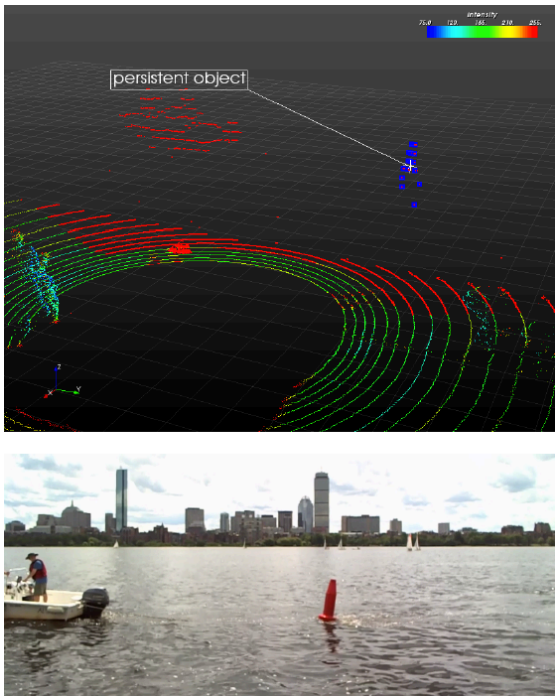

Figure 5: **Point Cloud Object Detection:** Top: Point cloud from 3D laser with buoy identified. Bottom: Corresponding image from camera used for buoy color detection GPS sensors.

Once these clusters are obtained, the final stage of the pipeline is persistence checking. During this stage, the cluster centroids from the ten most recent lidar sweeps are compared to the latest sweep in order to determine whether or not a cluster within the current sweep is a stationary object. If the centroid of a cluster in the current point-cloud sweep has a corresponding cluster centroid (their centroids are within a specified radius of one another) in each of the previous ten sweeps, then it is flagged as a true obstacle. This final stage is important for removing noisy lidar returns which may be grouped and incorrectly identified as an obstacle. An example of a successful buoy detection is shown in Fig. 5. The point cloud is shown on top where the the points that remain after culling are shown in red and the final persistent feature is shown indicated. The corresponding image from the camera is shown below is subsequently sub-windowed around the detected object for color detection as described in Sec. 2.4.2.

2.4.2 Vision

The system comprises three main vision processing tasks: 1) identifying buoy colors, 2) finding the correct placard feature for docking and 3) correctly determine the sequence of colors from the lighted buoy.

Buoy Color Detection The buoy color detection system is triggered upon the detection of a feature from the object detection system (Sec. 2.4.1). Upon reception of a new image detection, a frame is requested from the camera. The location of the detected image is converted from global coordinates to image pixel coordinates and then a sub-windowing operation takes place to grab only small region of the image around the detected feature. This was essential in practice to remove false color detection particularly for the white colored buoy.

Figure 6: **Buoy Color Detection:** Left: Raw image from camera after sub-windowing based on detected location laser. Right: Output from "red": color segmentation filter showing a correct red color detection.

The sub-windowed image is then converted to the Hue-Saturation-Value (HSV) color space and undergoes a set of thresholding operations to identify colors. The HSV colorspace is beneficial for color detection in images because it is less sensitive to lighting conditions as the majority of the color information should be contained within the hue channel.

Placard Feature Identification The key design of our placard detector is 1) robust to degradation caused by motion, scale and perspective transformation from different viewing positions, warp and occlusion caused by wind, and variants of color from light condition, and 2) fast and accurate enough to support real-time decision-making. We tackle this problem by detection and decoding stages. Our pipeline aims at high recall rate in detection stage, and then rejects false positives by decoding.

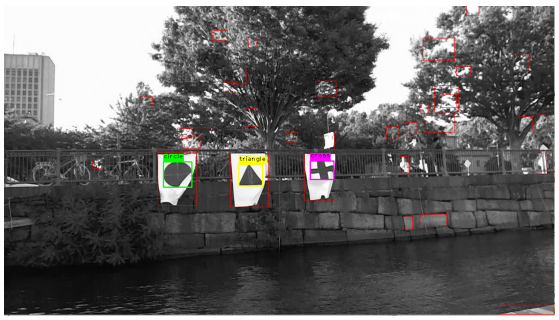

Figure 7: **Placard Detection:** The three placards were made and hung from a railing. Here we show correct detection of all three placards even though one of the (the circle) is partially obscured and another (the cruciform) is significantly distorted.

The detection is based on Maximally Stable Extremal Regions (MSER) [4], which is fast and robust to blur, low contrast, and illumination. In the decoding stage, we detect SIFT [3] and FAST [5] keypoints in each candidate region, which is then encoded using a junction-style descriptor. A rotation-invariant histogram of junctions is used to compare each candidate region with target placards. Fig. 7 demonstrates the candidate regions (red rectangles), which are rejected by decoding stage; the placards are shown in green, yellow, and pink rectangles, respectively. The computation runs at 2 frames per second for an imagery of 1280 X 720 pixels.

LED Light Sequence The LED sequence detection from the lighted buoy requires a similar capability as the buoy color detection except that there is an added temporal component required to detect the sequence. Color detection is done in the same way (with different thresholds for the color segmentation) with an example shown in Fig. 8.

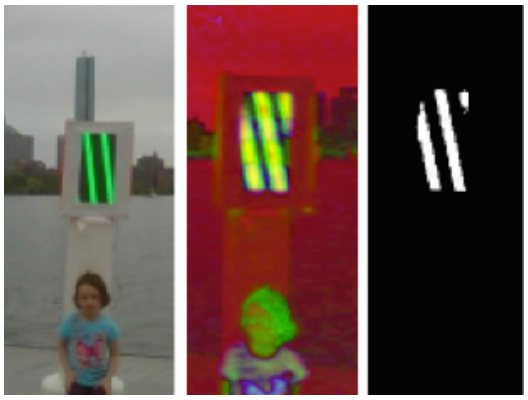

Figure 8: **Light Buoy Pattern Detection:** Left: Original RGB image. Middle: Image transformed to HSV colorspace. Right: Detection of green color resulting from addition of several subsequent images.

In order to detect the sequence, the detector follows a flow of operations:

1. Wait until first detection is made
2. Wait until no detection is found for 2 seconds
3. Record color of detection

4. If color detected is different from previous frame add it to the sequence
5. If no detection return to Step 2
6. If sequence is length 3, report and end

This system requires the lighted buoy to be within the field of view of the camera for at least 4 seconds. If no detections are being made the segmentation thresholds are adapted automatically to be more admissive. Similarly, if the pause in the sequence is never being found (caused by false detections) then the thresholds are adaptively made more restrictive.

2.4.3 Acoustics

The acoustic system, used for completing Task 2, consists of 4 HTI-96-MIN hydrophones, an amplification/filtering board, and a PC104 stack. The PC104 stack contains a Eurotech CPU-1484 computer, a 24DSI12-PLL analog to digital converter, and a HE-104 DX power supply. This system takes in raw data from the hydrophones, performs filtering/amplification on each channel, then does the signal processing and tracking required to localize a pinger.

The hydrophone elements are arranged in a "T" shape, to give both bearing and elevation information on active pingers. Each hydrophone goes into one of the 5 channels in the custom amplification and filtering board. Each board channel contains a Sallen-key high-pass filter, 2x amplifier, then low-pass filter.

The amplified and filtered signals from the hydrophone elements are passed into the 24DSI12-PLL analog to digital converter. The 24DSI12-PLL is triggered in data acquisition at the start of each second, and exactly a second of data is collected each second by a daemon running on the computer, 24dsi12_boardread. The acoustic data is recorded synchronously from the four channels.

MOOS-IvP is used for the acoustic signal processing chain. Array processing is used to determine the bearing and elevation to a pinger at a specified frequency, and a bearing-only tracker is used to determine the pinger location from a sequence of bearings.

2.5 Autonomy and Software

The software architecture used on the MIT boat is based on the MOOS-IvP autonomy architecture MOOS-IvP is an open source project consisting of the MOOS middleware from the University of Oxford and the IvP Helm autonomy architecture from MIT, `www.moos-ivp.org`. The MOOS-IvP distribution is designed and distributed with a nested or layered software concept - MOOS is a subset of MOOS-IvP consisting of the MOOS middleware architecture and core MOOS application. MOOS-IvP consists of many additional MOOS applications and the Helm behavior-based architecture with several core helm behaviors. The MIT RobotX codebase is a further layer of software with MOOS-IvP as a subset. Additional RobotX MOOS applications and Helm behaviors were written to provide the additional capabilities needed by the WAM-V vehicle and RobotX mission goals.

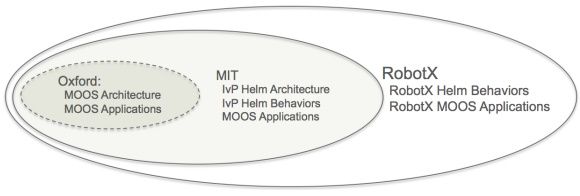

Figure 9: **Nested Software:** The RobotX codebase can be viewed as an additional layer on top of the public MOOS-IvP codebase with additional MOOS applications and Helm behaviors for the WAM-V and RobotX mission.

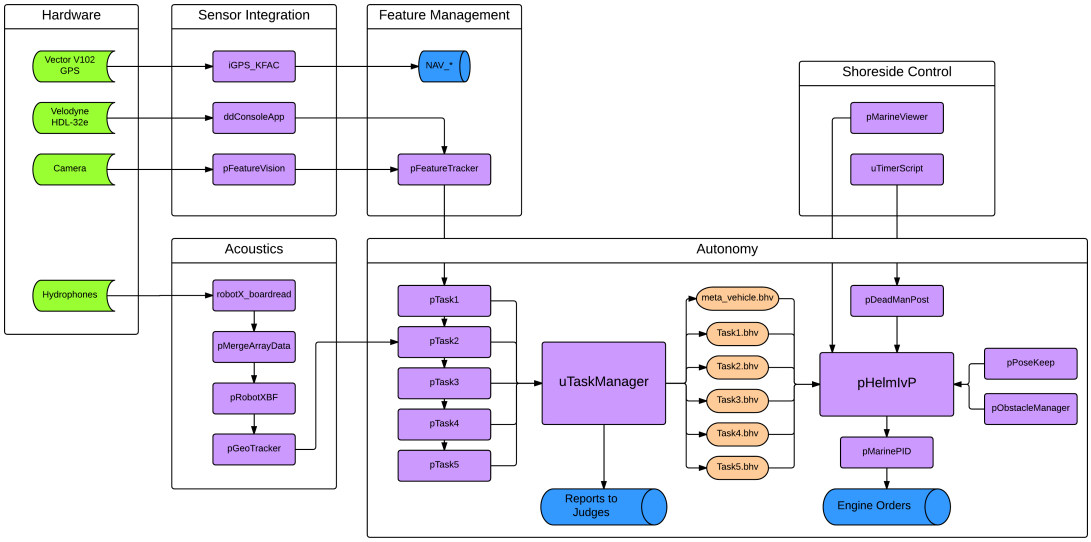

Figure 10: **Software Architecture:** The diagram above shows the general structure of the dataflow and relationship between the different processes.

The general idea behind the RobotX software was for each task to be controlled as a subset of a larger task. A MOOS application called uTaskManager serves as the overall structure. This app sends and receives signals to enter and exit each of the individual tasks. This app also reports the status of the vehicle to the judges, such as the heartbeat and task reports.

In addition, An app was built for use in each task, as well, labeled pTask1, pTask2, etc. These apps take in inputs from a plug file, with information such as course boundaries and general course set-up, and perform any calculations necessary for their respective tasks. The processes keep track of information such as individual task-specific object locations, and update them accordingly as new sensor information becomes available and is reported through pFeatureTracker (Sec. 2.4.1). These apps will also feed this information into a number of standard and custom MOOS-IvP behaviors which determine the next set of directions the boat will take. These behaviors are mostly waypoint behaviors, but it also includes obstacle avoidance behaviors, pose-keeping behaviors (see Sec. 2.5.2), and behaviors for backing up.

The software modules used on the MIT/Olin RobotX vehicle are comprised of (a) public MOOS applications from Oxford (b) public MOOS applications from MIT with the IvP Helm and behaviors, and (c) newly created or non-public applications specific to RobotX.

The public MOOS applications from Oxford are: MOOSDB, pLogger, pShare.

The public MOOS applications from MIT are: pHelmIvP, pNodeReporter, pMarinePID, pEchoVar, pHostInfo, uProcessWatch, uFldNodeBroker.

The public IvP Helm behaviors from MIT are: Waypoint, AvoidObstacles, StationKeep, OpRegion.

The non-public or RobotX new apps are:

- pDeadManPost
- pPoseKeep

- pObstacleMgr
- uTaskManager
- pGeoTracker
- pFeatureTracker
- pFeatureVision
- pTask1
- pTask2
- pTask3
- pTask4
- pTask5
- pMergeArrayData
- pManageAcousticVars
- pRobotXBF
- iVeloPy
- ddConsoleApp

The public MOOS applications and behaviors are well documented at `http://oceanai.mit.edu/ivpman` and are not covered in detail here. The non-public or RobotX new applications are the focus of the next sections.

2.5.1 The pDeadManPost Application

The pDeadManPost app may be used to queue one or more postings to the MOOSDB that will NOT be made so long as a named MOOS variable continues to be written to. It is a generalization of a physical "dead man switch" which must be continuously pressed or held by a human operator, otherwise cutting the power to an engine or other physical device.

On the RobotX vehicle, this application ran in the MOOS community with hearbeat messages sent continually from the shore. On the shore a simple script (the uTimerScript MOOS app) posted heartbeat messages shared over WiFi with the pShare app to the vehicle MOOS community running pDeadManPost. This ensured that if the shoreside command and control computer went down, or if WiFi went down, the pDeadManPost app on the vehicle would immediately post commands to either all-stop or station-keep in place.

2.5.2 The pPoseKeep Application

The pPoseKeep application is used on the WAM-V vehicle to directly talk to the left and right thrusters to maintain a desired direction. The basic idea is shown below.

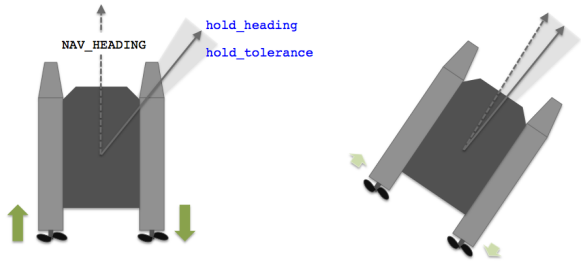

Figure 11: **PoseKeeping:** A vehicle with differential thrust applies opposing thrust of equivalent magnitude to turn a vehicle in place until it achieves a desired `hold_heading`, with a given `hold_tolerance`.

The application was designed to be invoked at any time during the mission, effectively overriding temporarily the Helm when it is active. It's sole purpose is to rotate the vehicle to a given goal direction for a period of time. This is used in the mission to (a) position the vehicle to look at the light buoy, (b) position the vehicle to look at sections of the course where docking placards may be prior to docking, and (c) to perhaps rotate the vehicle prior to backing out of the docking station.

The application is configured accept goal directions either through a direct MOOS message such as `HOLD_HEADING=45`, or indirectly by specifying a point in local coordinates, such as `HOLD_POINT=10,-90`. In the latter case, the vehicle's own local coordinates are used to derived the goal direction from the specified point. The application may be configured to run (a) indefi-

nitely until deactivated by another MOOS message, (b) for a fixed duration after which it may be automatically deactivated, or (c) until the present vehicle direction is within a threshold tolerance of the goal direction, after which it may be automatically deactivated.

2.5.3 The pObstacleMgr Application

The pObstacleMgr app accepts `TRACKED_FEATURE` reports from the pFeatureTracker app and in turn generates `OBSTACLE_ALERT` messages to the helm that spawn an avoid obstacle behavior for each obstacle.

The tracked features are grouped by pFeatureTracker and are posted as a point in the x-y space with a cluster ID. The obstacle manager simply keeps track of the clusters and maintains a convex hull for each cluster.

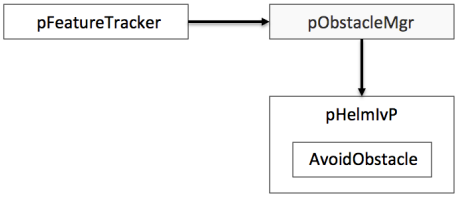

Figure 12: **Obstacle Manager Flow of Information:** The obstacle manager receives detections from the feature tracker, and alerts the helm to spawn a obstacle avoidance behavior for each new obstacle.

The obstacle manager uses the convex hull to generate an alert, containing the polygon, received by the helm. The helm is configured with an obstacle avoidance behavior template that will spawn a new behavior with each new alert with a unique ID.

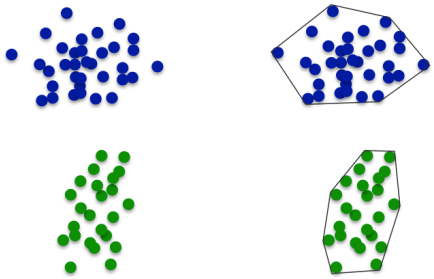

Figure 13: **Conversion of Cluster to Convex Hull:** As new points (tracked features) arrive, they cluster and convex hull are updated.

2.5.4 The Helm Obstacle Avoidance Behavior

An obstacle avoidance behavior is spawned by an alert from the obstacle manager described in Section 2.5.3. The alert contains a convex polygon and unique ID. Subsequent updates from the obstacle manager may change the shape of the polygon representing the obstacle. For each obstacle, a dedicated behavior is launched. The behavior is configured in the helm as a template to allow unlimited spawning. The behavior template names a buffer distance to be applied around the original obstacle polygon as shown in Figure 14.

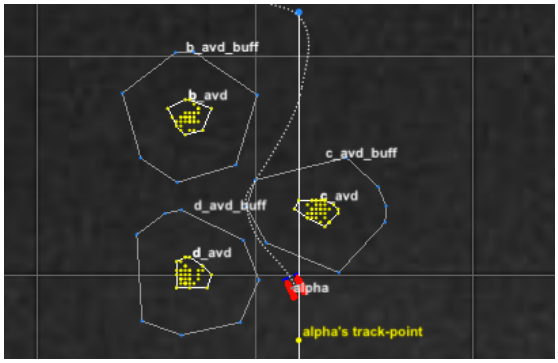

Figure 14: **Obstacle Avoidance:** The vehicle proceeds around known obstacle polygons with a given safety zone.

The actual vehicle trajectory is a result of the helm's multi-objective optimization engine

running [1], continuously balancing the objective functions from obstacle avoidance against the objective function for transiting through the obstacle field, finding a buoy, or other primary tasks.

2.5.5 The ddConsoleApp Python Application

The program 'ddConsoleApp' provides the functionality to interface with the HDL-32E lidar, as well as to process and visualize the HDL-32E's point cloud data and is not a MOOS application. 'ddConsoleApp' leverages software from the MIT DARPA Robotics Challenge Team, and is essentially a C++ based program with bindings to Python; it provides direct access to the point-cloud generated by the HDL-32E, which can then be manipulated within a Python script passed as an argument to the program, and which can also be visualized using a VTK-based viewer. Processing of the point-cloud is achieved using VTK filters, which include a wrapped subset of PCL algorithms; in addition, since manipulation of the point-cloud is performed within a Python script, Numpy/Scipy functions are available for use. In the context of the MIT/Olin RobotX vehicle, 'ddConsoleApp' operates on a single Python script that defines the point-cloud processing pipeline that performs buoy detection. This processing pipeline is fairly simple, and consists of three main stages: the point-cloud is first culled to isolate points within a region of interest; the isolated points are then grouped into clusters; and finally clusters are filtered so that only those which do not move over a given number of lidar sweeps are kept. its position is stored in an LCM message and relayed to the iVeloPy MOOSApp for use by the vehicle in buoy detection and obstacle avoidance. More details of the object detection pipeline were given in Sec. 2.4.1.

In addition to buoy detection, the Python script also performs a secondary function for placard detection. When a placard is detected by the camera, the azimuth and elevation from the lidar to the placard are relayed to the Python script via LCM. The azimuth and elevation are used to obtain points within a spherical pyramid defined by those angles, which are then averaged so as to estimate the location of the placard.

2.5.6 The iVeloPy Application

The iVeloPy moos application is essentially a MOOS wrapper application designed to marshal data between ddConsoleApp and the MOOSDB. When an object detection is received from ddConsoleApp a new feature is instantiated and published to the MOOSDB containing attributes such as the location of the object in global coordinates and its size (e.g. `FEATURE="x=20.3, y=14.2, size=10"`).

During the placard detection task, the iVeloPy app also receives `PLACARD_RAY` detections from pFeatureVision and sends a requests ddConsoleApp for the range associated with this ray. The result is published (e.g. `PLACARD_FEATURE="x=10.2, y=7.5, placard=triangle`) which triggers the system to begin parking if it corresponds to the correct placard.

2.5.7 The pFeatureVision Application

pFeatureVision is the MOOS application responsible for interfacing with the camera using an OpenCV VideoCapture. It is also responsible for distributing the grabbed frames to the various image processing tasks which are triggered in different ways:

- **Buoy Color Detection**: Each time feature detection is received (from iVeloPy)

the current frame and feature information are passed to the buoy color detection algorithm to try and identify a color to the feature. Following this the feature is republished with color included (e.g. `COLOR_FEATURE="x=20.3, y=14.2, size=10, color=red"`).

- **Placard detection**: Once the placard detection begins, each frame is passed to the placard feature detection. If a detection is made, the pixel location of the center of the placard is used to generate a `PLACARD_RAY` that indicates the angle from the camera to the placard. This is required since we only have 1 camera and we do not know the size of the placard beforehand, therefore we cannot resolve the distance to the placard, only the bearing. The range is subsequently determined by querying the laser sensor for the corresponding ray.

- **Light buoy detection**: Once the light buoy task is reached, all frames are passed to the light buoy detector (see Sec 2.4.2 for details) until a sequence is determined, at which time it is published as a `LIGHT_BUOY_SEQUENCE` variable.

2.5.8 The pFeatureTracker Application

The main task of the pFeatureTracker application is to associate the colored features received from the output of the vision system. To accomplish this, a list of active features is maintained. Each time a new feature detection is received, the location is checked against all active features. If it is sufficiently close (in terms of Euclidean distance) to any of the features in the list, then a new feature is not instantiated. The list is periodically published as set of tracked features (e.g. `TRACKED_FEATURE="x=20.3, y=14.2, label=a, size=10, color=red"`) which is ultimately what is used for buoy gate traversal and obstacle detections.

2.6 Failure and Recovery

In late August during on-water testing, team MIT-Olin suffered a catastrophic failure in one of the WAM-V's joints while the vehicle was floating in the water dockside. The recovery effort took approximately 45 minutes, all the while the electronics on board were submerged underwater. Some components did work and could be salvaged, but others, including all the computers, did not make it.

Figure 15: **Boat Failure:** In late August, on one of the initial water tests, the boat suffered a catastrophic failure, and many of electronics were ruined.

Fortunately, the team was able to make a full recovery. The failed boat part (see Fig. 16) was quickly replaced by the WAM-V manufacturer, the damaged hardware components including were replaced with extra funding from MIT's mechanical engineering department. While this set back the testing schedule several weeks, the team was still able to design

and create a fully capable, competition-ready robot.

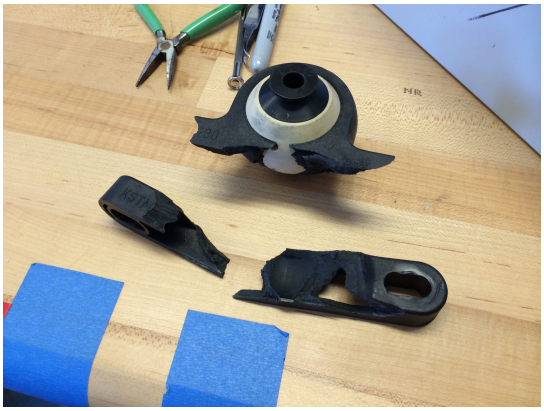

Figure 16: **Failed WAM-V part:** The piece that failed that caused the boat to flip upside-down.

3 Collaboration

- Olin - Clearly delineated task where Olin concentrated on hardware design for vehicle components and ancillary parts (light buoy, etc.). MIT concentrated on system integration and software.
- Portwell - provided embedded computing solutions. Learned practical experience of selecting and working with highly specific technical pieces.
- Sea Grant - Provided equipment, access to Sea Grant engineers, and support for MIT research staff mentoring.

4 Conclusion

MIT and Olin have worked hard to create a capable, well-designed marine robot. There has been much innovative design throughout the process, including 3D point cloud analysis to find objects on the water, advanced vision processing techniques applied to placard and light buoy detection, innovative thinking in autonomy, and new methods developed for maneuvering, such as turning in place and backing up to a goal point. With these innovations and a sound strategy, Team MIT-Olin will make a strong showing in Singapore at the RobotX 2014 competition.

References

[1] Michael R. Benjamin, Henrik Schmidt, Paul M. Newman, and JohnJ̃. Leonard. *Unmanned Marine Vehicle Autonomy with MOOS-IvP*, chapter 2, pages 1--100. Springer, 2012.

[2] Mike Benjamin, Henrik Schmidt, and John J. Leonard. `http://www.moos-ivp.org`.

[3] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91--110, 2004.

[4] L. Neumann and J. Matas. Real-time scene text localization and recognition. 2012.

[5] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *Computer Vision--ECCV 2006*, pages 430--443. Springer, 2006.