Critical Design Review

Autonomous Water Taxi

Team Members (Team B):
Tushar Chugh
Shiyu Dong
Bikramjot Hanzra
Tae-Hyung Kim
William Seto

Mentors:
John Dolan
Dimitrios Apostolopoulos

Sponsor:
Jeremy Searock

THE ROBOTICS INSTITUTE

December 18, 2015

THE ROBOTICS INSTITUTE

Contents

List of Figures

List of Tables

1. Project Description

Figure 1.1: 27 ft SeaHawk Aluminum Welded Boat

The aim of this project is to create a proof of concept that demonstrates a viable water taxi system for the City of Pittsburgh. Many of Pittsburgh's attractions are located along the three rivers and would be easily accessible by boat. Currently, there are only a few luxury cruise lines that are mainly used for entertainment and tourism, rather than as serious form of transportation. Although the federal government has allocated money to the city for building docks to support transportation, neither the city nor any companies have found a good business model for such an operation. We believe developing an autonomous vessel would help bring the water taxi operation to fruition. A fleet of autonomous water taxis would reduce commute time for citizens, boost tourism for the city, and grow businesses located along the rivers.

To support this effort, National Robotics Engineering Center, the applied research lab of the Robotics Institute, has recently acquired a 27 ft SeaHawk aluminum welded boat (Figure 1.1). The boat has been converted to a test bed for maritime autonomy research, with drive by wire controls and a defined interface. The boat is outfitted with autonomy sensors, a positioning system, and also has a cabin and heating/air-conditioning to facilitate year-round use for developers onboard. In this project, we will develop algorithms and tools to enable autonomous navigation of the boat to defined destinations along the river.

2. Use Case

James would like to spend a nice day with his family, beginning with some shopping at Walmart, followed by an exciting game at PNC Park, and finally a nice dinner at the Cheesecake Factory in Southside Works. Unfortunately, as these locations are located in completely opposite directions, he has been unable to make this perfect day materialize.

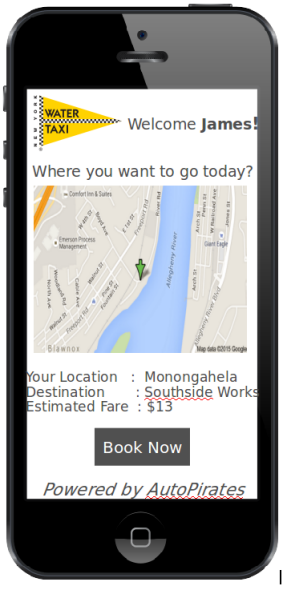

Figure 2.1: AutoPirates App

Luckily, the new Auto-Pirates service and smart phone app has just launched, which allows anyone access to their own private water taxi ride. After installing the app, James sees some predefined pickup points/destinations on the map. James heads over to the closest pickup location, the Monongahela Wharf, and proceeds to request a taxi on his app (Figure 2.1). His location is sent to a server, which manages all the boats in the fleet. The system dispatches the closest boat which is currently unused, or is carrying passengers to James' current location. As the boat is en route, James also receives a special code on his app, which will allow him exclusively access to the boat once it arrives.

Figure 2.2: Operator Control Unit

The boat arrives at the specified Auto-Pirates dock in the Wharf. After James gets on the boat, he proceeds to the Operator Control Unit, which is a touchscreen map similar to his smart phone app (Figure 2.2). James enters the code he received earlier and can now select a destination. He chooses Southside Works, and then the boat begins planning its trip. As the boat autonomously backs out of the dock, James can see a planned path on the map. As James enjoys the beautiful view along the river, he notices a large cargo ship approaching. He looks at the OCU and sees that the ship shows up as a blip on the map and the taxi has recalculated a new path that goes around the obstacle. He can also see buoys and bridges pop up as obstacles on the map.

Figure 2.3: The boat arrives destination

Eventually, the boat arrives at Southside Works and autonomously parks at its reserved dock, where the next set of passengers are waiting (Figure 2.3)

3. System-level Requirements

3.1 Mandatory Functional Requirements

The system shall -

MF.1 Interface with low level controller of the boat, IMU/GPS and Radar.

NREC has provided us an interface to the low level controller in the form of ROS [1] messages. The path planning subsystem shall send the appropriate commands so that the boat can follow our desired trajectory.

MF.2 Design static obstacles in the river with minimum size of $2m \times 2m \times 2m$.

The minimum size specified will be adequate for the obstacles that the boat will absolutely need to avoid since it covers anything larger than buoys. The perception subsystem shall extract the obstacles from the radar image.

MF.3 Segment shores and bridges.

Due to the characteristics of the radar, bridges are detected as one large contiguous obstacle in the river. Bridges must be identified and removed from the radar image, leaving all other obstacles in the costmap. In order to achieve this, the perception subsystem shall segment the shores and bridges, so that the bridges can be removed, and shore information can be kept for obstacle detection.

MF.4 Autonomously navigate from source to destination at the maximum distance of 2 miles in not more than 15 minutes.

This requirement provides a guideline for the final validation experiment. The path planning subsystem shall operate the boat autonomously for 2 miles and ensure the boat travels at a reasonable speed.

MF.5 Avoid detected static obstacles with an accuracy of more than 95%.

The system shall navigate autonomously as to avoid obstacles in the river. The perception subsystem must correctly detect the obstacles in the river and then the path planning subsystem must correctly generate a costmap with the detected obstacles and then plan a path to avoid the obstacle. Finally, the boat must be controlled to adhere to the planned path.

MF.6 Update the path within 5 sec after detecting new static obstacles.

The code efficiency will limit the speed of the boat. So we are using 5 sec as our mandatory functional requirements that we need to update the path within 5 seconds after detect new obstacles.

MF.7 Record sensor data up to duration of 15 minutes.

This requirement demonstrates that we are able to record data. So we save data for the sensors on the field test and test our algorithms with the saved data.

MF.8 Replay saved sensor data. This requirement demonstrates that we are able to play back the sensor data. This is also a required function after we record the sensor data.

MF.9 Have a 2D simulator to test path planning algorithm.

For our path planning subsystem, we first need to run the planner on a static map and then update the path for detected obstacles. Because of the limited test trials, we need a simulator to test the path planning algorithm before we can go out for field test.

MF.10 Generate Occupancy grid map of environment.

To generate an occupancy grid map of environment is a requirement of our system, as the SBPL [2] library for path planning needs an occupancy grid map of environment and we need to update the obstacle in the occupancy grid map.

3.2 Desirable functional requirements

The system shall -

DF.7 Track dynamic obstacles with minimum size of $1m \times 1m \times 1m$.

As the mandatory requirement of our system is to detect static obstacles with minimum size of $1m \times 1m \times 1m$, we would like to detect smaller obstacles and be able to track the dynamic obstacles in the desirable functional requirements.

DF.8 Update the path with 2sec after detecting new dynamic obstacles.

Our mandatory functional requirement for the path planning is to update the path with 5 seconds after detecting new obstacles, but we'll try update in 2 seconds in the desirable functional requirements. This will allow the boat to run faster without hitting the obstacles.

3.3 Mandatory non-functional requirements

The system shall -

MNF.10 Be tested in 25-30 field trials.

This is a requirement due to our limited budget and time. We need to test all the functionalities of the boat within 25-30 field trials. As a result, building a simulator to test our algorithms and having good test plan are required.

MNF.11 Arrive within a radius of 15 meters of the desired destination.

This requirement specifies the accuracy of the path planner. So the boat should arrive the destination within a radius of 15 meters.

MNF.12 Be demonstrated in a video.

This is required by our sponsor that we need a video to demonstrate the functioning of the boat.

MNF.13 Sail at a velocity not more than 15 mph.

As limited by the refresh rate of radar and the code efficiency, the boat shall travel no faster than 15 miles per hour for the safety of the boat, and other obstacles in the river.

3.4 Desirable non-functional requirements

The system shall -

DNF.1 Obey the "rules of the road".

If possible, the system shall follow the International Regulations for Preventing Collisions at Sea (Colregs).

DNF.2 Estimate the shapes of the static obstacles in the river with a minimum of 50% overlap.

In order to better enhance the performance of the path planner, the perception subsystem shall be able to estimate the contour and size of the obstacles in the river.

---

4. Functional Architecture

4.1 Functioning with perception and path planning

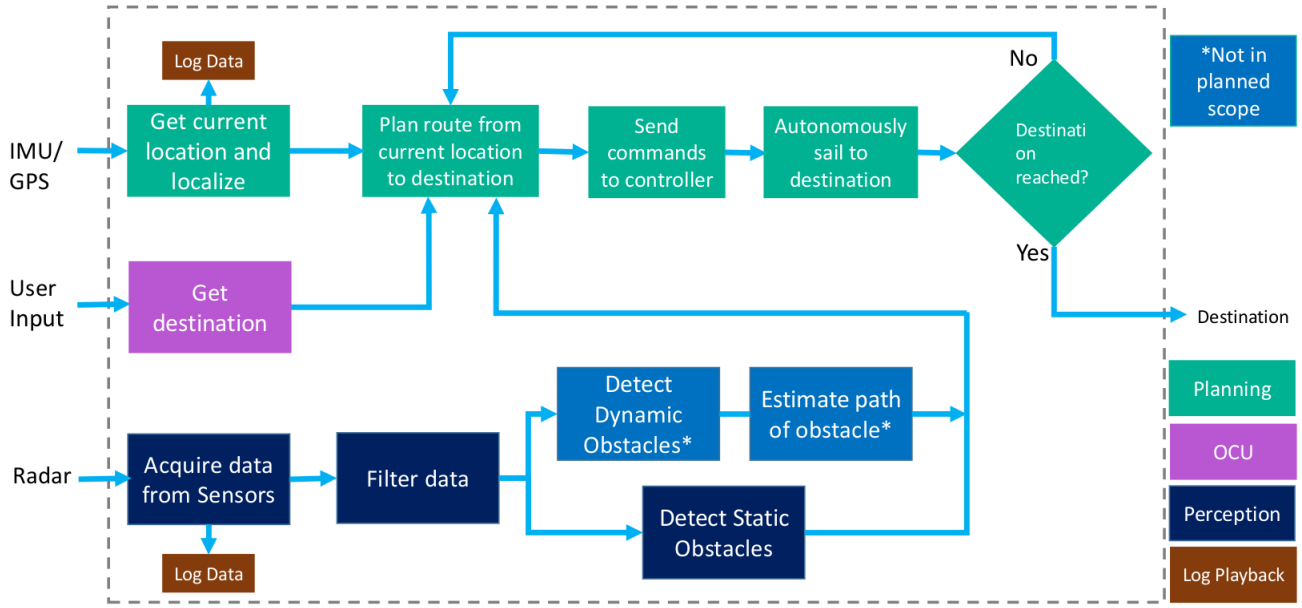

Figure 4.1: Functional Architecture

Figure 4.1 describes the functional architecture of the Autonomous Water Taxi. The functional architecture can be viewed as the building blocks of elements spanning across perception, planning, OCU and Log Playback. The desired output of the system is that the autonomous boat reaches the final destination by avoiding obstacles. The input of the system is user input, which specifies the way points and destination, and the data acquired by sensors, which includes IMU input that gives data of current position(GPS coordinates, heading), and Radar to detect the objects.

First part of the functional architecture is obstacle detection based on Radar. The system acquires data from the Radar sensor, which we can log and play back that data for further analysis. Since the radar data will potentially have a lot of noise, the next step will be filtering the data to get the useful information that indicates the size and location of the obstacles. For the obstacles, we need to divide the obstacles into two categories, static obstacles and dynamic obstacles and then design different strategies for them, because for dynamic obstacles we need to estimate path of it and then decide to follow its path or just avoid it. And we will get obstacle avoidance algorithm for this part finally.

The second part of the system is Log Playback. Log Playback allows us to log the data from the sensors (RADAR, IMU) and then analyze and implement algorithms based on the data. It also provides required information for the OCU module, and finally shows in the user interface.

The third part is Operator Control Unit (OCU), which allows user to specify the geometry location, usually the longitude and latitude, of waypoints and destination. The idea of the water taxi is that we implement an autonomous driving boat and users are able to use the mobile app to order a taxi service, and the water taxi will pick up passengers at each point and plan a path.

The final part is path planning which gathers data from IMU sensor to get the current position and location of the autonomous boat. The IMU sensor has an embedded high-accuracy GPS and digital compass to get an accurate location and gives to the path planning module as the input. The path planning function gets the input of current location, destination and the information of the dynamic and static obstacles. Then it plans a path for the autonomous boat to reach each way point without hitting any obstacles and sends commands accordingly to the low-level controller. Then the following function will estimate the current location and check if the autonomous boat reaches each way point and finally arrives the final destination successfully. If the autonomous boat has not reached the destination, then it will go back to the path planning function and design a new strategy for path planning with updated input from sensors.

4.2 Functioning of simulator with logged data

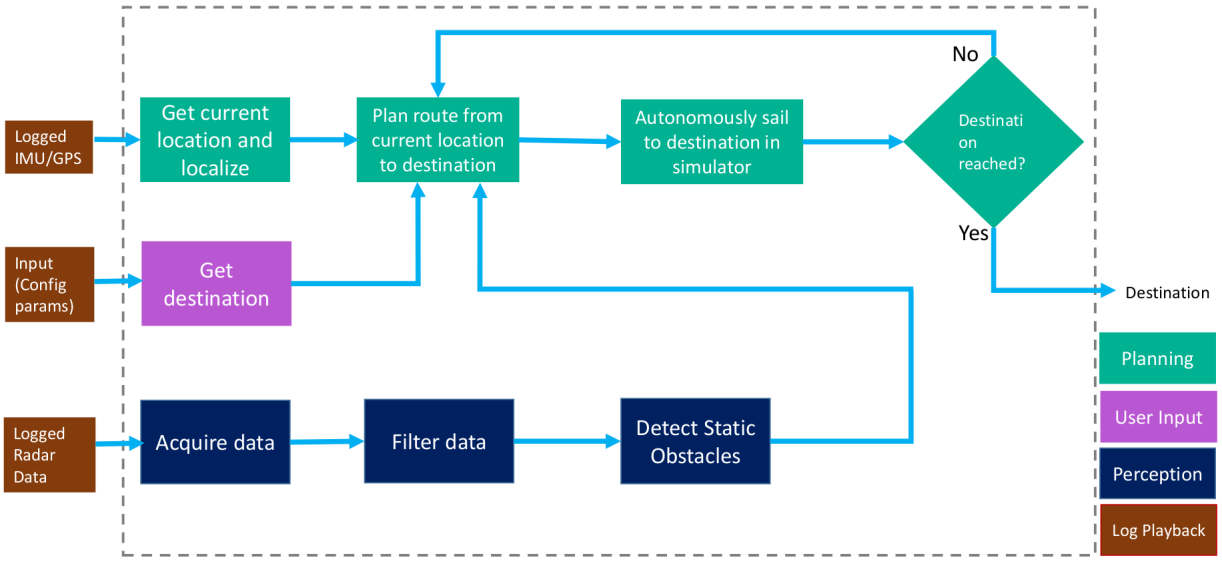

Figure 4.2: Functional architecture of logged data and simulator

Figure 4.2 shows the functional architecture of the simulator. This is very similar to our main functional architecture with a primary difference that it takes recorded data of radar and IMU/GPS as the input. In addition to it, the output is the visualization of the path planned rather than the commands sent to the low level controller to navigate through the waypoints.

5. Cyber-physical Architecture

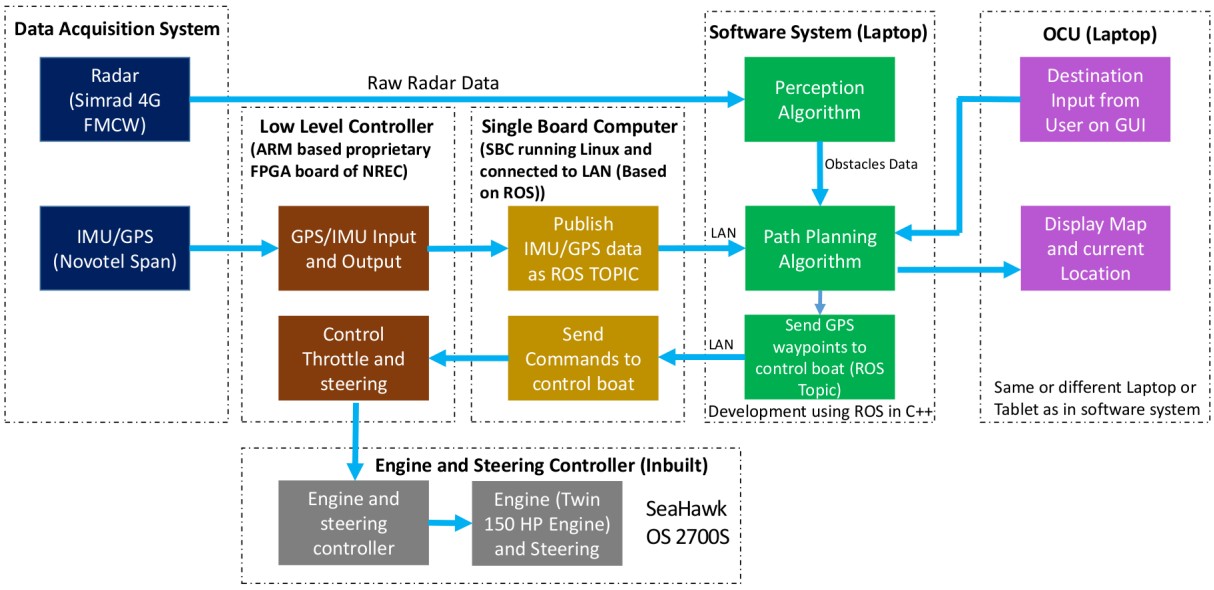

Figure 5.1: Cyber-physical Architecture

As shown in the Figure 5.1, the cyber-physical architecture is divided into 3 systems -

- Data Acquisition System
- Software System
- Mechatronics System

5.1 Data Acquisition System

As the name suggests, the primary function of the data acquisition is to acquire data from the various sensors that are outfitted on the boat. Currently, there are 3 sensors that are installed on the boat.

The Simrad 4G RADAR will be used to detect obstacles like ships, bridges, buoys and shores. The Novatel SPAN is an inertial navigation system. It provides us GPS coordinates, as well as velocity and orientation information from the IMU. There is also a mount on the boat for a LIDAR. As of now, as per the requirements of the sponsor we are not using the LIDAR sensor but we might use it if we do not get the desired results by using only the RADAR.

Once, we have the data from the sensors (RADAR and IMU), the next step is to use this data to find the GPS location of the boat using IMU data, detect obstacles from the RADAR data. Once, we have the location of the boat and the positions of the obstacles are in the environment, we need to plan how the boat navigates to the navigation. All these jobs are done by the software system. Since this is a software intensive project, this system is the heart of the project.

5.2 Software System

In the software system, we have 2 subsystems -

- Perception
- Path Planning

The perception algorithm uses the RADAR data to detect obstacles and then feeds this data to the path planning algorithm. This subsystem is responsible for interfacing with the radar, and then filtering and processing the data. There are 2 stages in the filtering process. The raw data comes as a polar image, in which data is given by range and azimuth. In this stage, we apply a simple filter based on the binomial distribution. In order to utilize OpenCV [3] and perform higher level image processing tasks like morphology operations and contour extraction, we need to convert our polar image to a cartesian image. In the 2nd stage, we segment our image so we can identify and remove bridges, and then perform the image processing operations mentioned above. After this, the locations of any obstacles of interest are sent as a message to the planning subsystem.

The path planning uses the obstacle data from the perception subsystem along with the IMU data to plan the path. The first step is to incorporate the obstacle information into the current costmap. Depending on the obstacle size and location of the boat, the calculated costs may vary. Next, the costmap will be used by the path planner to generate a safe path to the destination. We have decided to utilize SBPL (Search-Based Planning Library) for the core of our planning subsystem. The library provides several types of search algorithms which will allow us experiment with trade-offs between optimality and speed. Additionally, it allows us to specify motion primitives, in which we can constrain the path planner so that it mimics the dynamics of the robot. Finally, after generating the path, the planning subsystem will send waypoints to the boat in the form of ROS messages so that the low level controller can navigate to those waypoints.

5.3 Mechatronics System

The last system is the Mechatronics system. Most of the low level control has already been done by engineers at NREC. We only need to send high level commands in the form of ROS messages to the engine controller to change the speed and direction of the boat.

6. Current System Status

6.1 Targeted Requirements

Table 6.1 shows the targeted requirements for the fall semester. Our project is divided into two main parts: perception and path planning. For the perception part, our goal for the FVE was to interface with the radar and get the raw radar data, then detect static obstacles and be able to segment the radar images into shores, bridges and obstacles.

For path planning part, our goal for the FVE was to generate an occupancy grid map of environment. Based on the occupancy grid map, we should be able to run our path planner, generate a path and show it on the simulator.

For both perception and path planning, we need to be able to record and replay sensor data, so we can test our implementation using recorded data from our field tests.

Table 6.1: Targeted Requirements

ID	Requirements
MF1	Interfacing with low level controller of the boat, IMU/GPS and Radar.
MF2	Detect static obstacles in the river with minimum size of $2m \times 2m \times 2m$ with more than 85% accuracy.
MF3	Segment shores and bridges.
MF7	Record sensor data up to duration of 15 minutes.
MF8	Replay saved sensor data.
MF9	Have a 2D simulator to test path planning algorithms.
MF10	Generate Occupancy grid map of environment.

6.2 Subsystem Depictions

6.2.1 Perception

Gathering radar data and segmenting the types of obstacles

Our first challenge in the perception subsystem was to get data from the radar. No one at NREC had used this radar before so we were unsure how difficult the interfacing of radar would be. We found an open source library OpenCPN [4] through which we were able to collect data from radar. We integrated this with ROS as our entire software architecture of perception and path planning is based on ROS. Next, during the field tests we used an RGB camera in addition to the radar sensor to capture the ground truth. Figure 6.1 shows collected RGB camera and raw radar data. Our next step was to segment brides, shores and obstacles on the river. We did that by comparing it with original occupancy grid map (with bridges). The output is

shown in the rightmost visualization of figure 6.1. Here. Shores are represented by green, bridges by red and obstacles by blue.

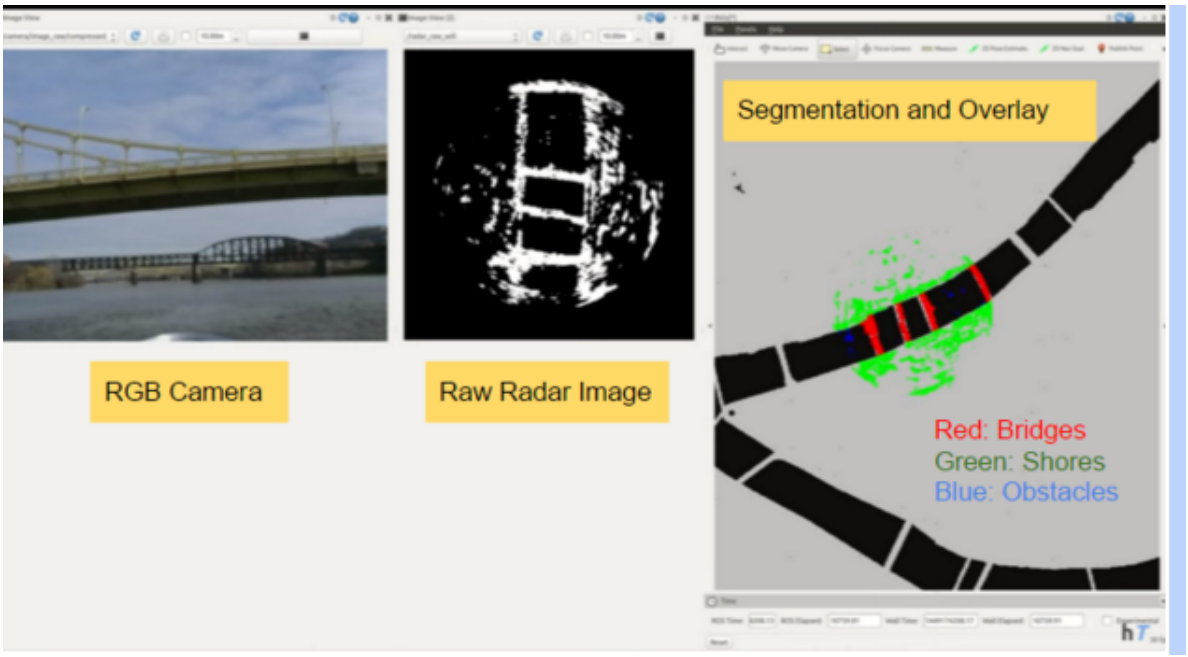

Figure 6.1: Obstacle Detection

Filtering raw radar data

Raw radar data has some false positives which are required to be filtered in order to get correct location of the obstacles. We used a binomial probabilistic filter [5] to get rid of the false positives. Figure 6.2 shows the result after we apply filters to the data. The center visualization in Figure 6.2 shows raw radar data and right image shows filtered radar data. We can see that a lot of the false positives are not present in the filtered radar data.

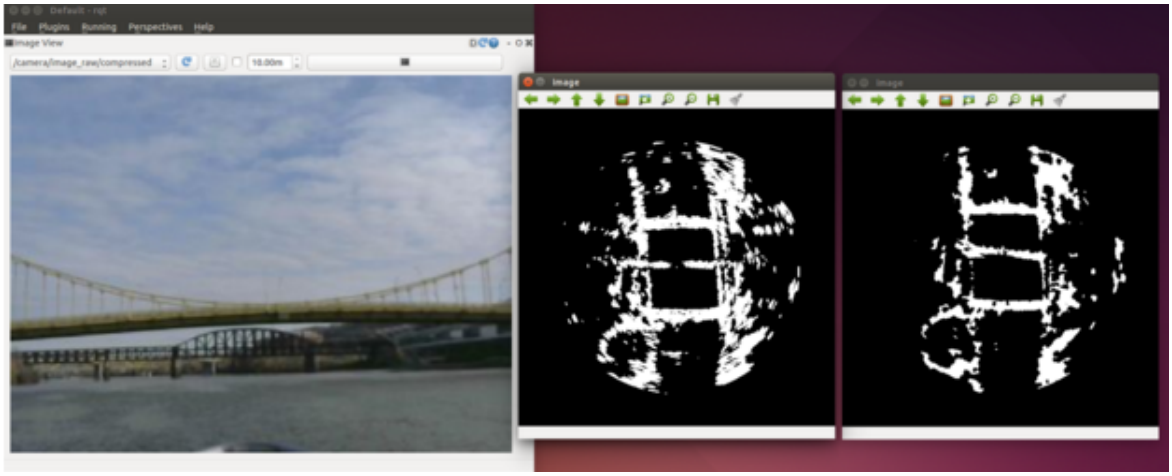

Figure 6.2: Radar Filtering

Putting bounding boxes around the obstacles

In order to measure the accuracy of detecting obstacles, we need to quantify how many obstacles we are able to detect from the given obstacles in the path. For that, we create boundary boxes around the obstacles. This can be seen from figure 6.3 which shows bounding boxes around the blue blobs, which are the obstacles. Two of the big obstacles are marked in figure 6.3. One of them is a big barge.

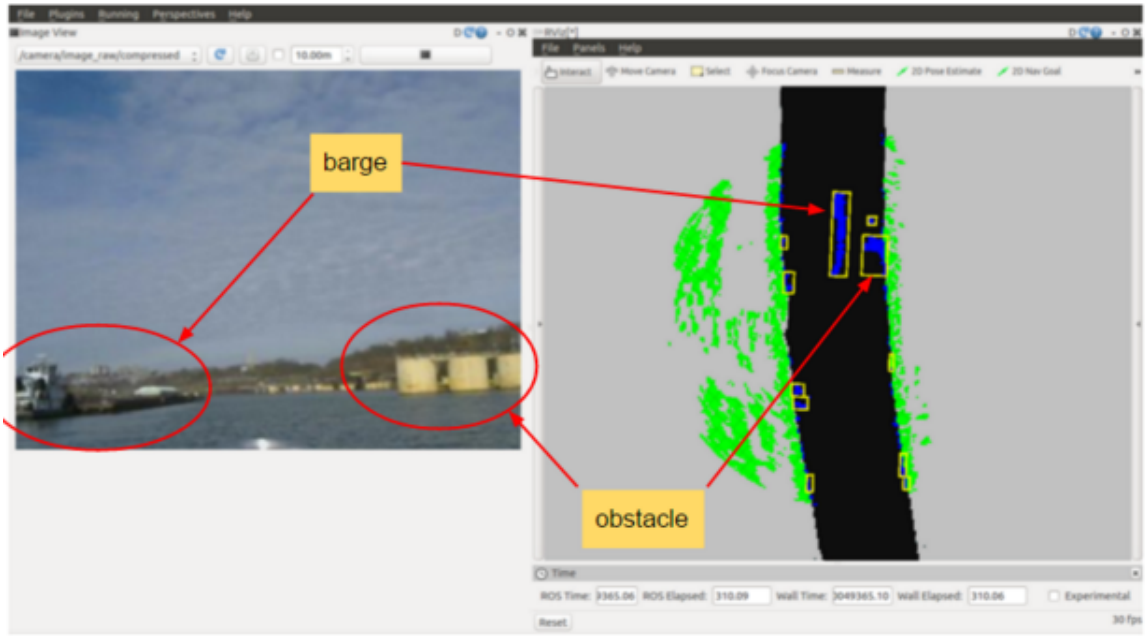

Figure 6.3: Radar Filtering

6.2.2 Path Planning

Occupancy Grid Map

We have created an occupancy grid map (OGM) which the path planning algorithm would be using to find the shortest path between our start position and the desired location. We used qGIS software and openstreet map to extract our area of interest i.e. the three rivers flowing through Pittsburgh. Figure below shows OGM where black (zero value of pixels) represents empty path and white (one value of pixels) represent obstacles. The resolution of the map is approximately 5 m.

Figure 6.4: Occupancy Grid Map

Inflating the cost near shores and obstacles in Occupancy Grid Map

The path planner algorithm tends to find shortest possible path between start and the goal locations. However, the path generated touches the shores or is very close to obstacles. To overcome this issue we need to inflate the cost (value of pixels) near the shores and the obstacles. We have created a tool which allows us to experiment with modifying these costs through a simple GUI (shown in figure 6.5). Also, this GUI enables us to add obstacles which we can use to test our algorithms. Figure 6.6 shows addition of obstacles to the image and the inflated costs (in grey) near shores and obstacles.

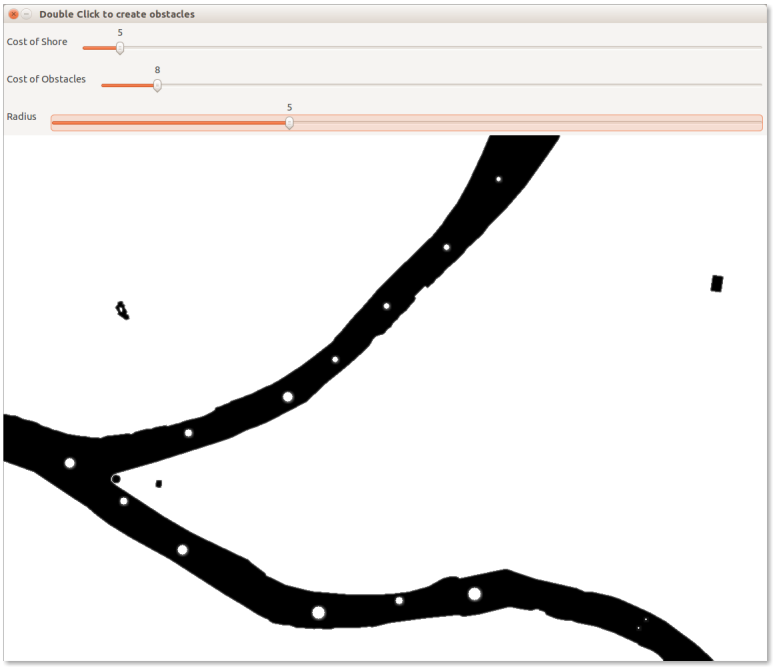

Figure 6.5: GUI to add obstacles

Figure 6.6: Added cost in Occupancy Grid Map

Path Planning Algorithms

We are using the SBPL library for planning the path. The library has built in planners like ARA* and AD* algorithms. We converted the OGM to SBPL compatible format and ran the ARA* planner by providing the start and desired location. The algorithms were tested on small, medium and large size obstacles and the visualization of the results are represented in Figure 6.7, 6.8 and 6.9 respectively.

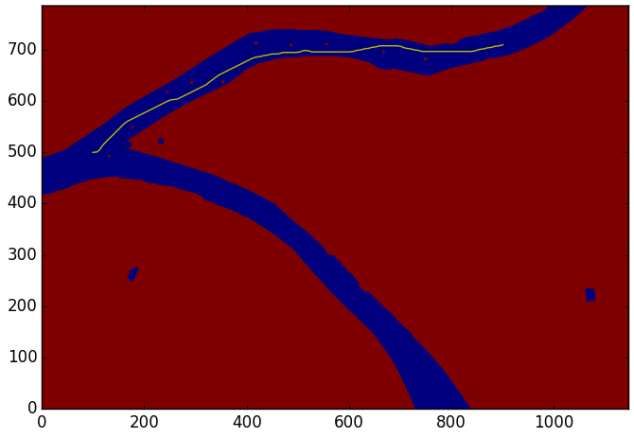

Figure 6.7: Path on Small Size obstacles

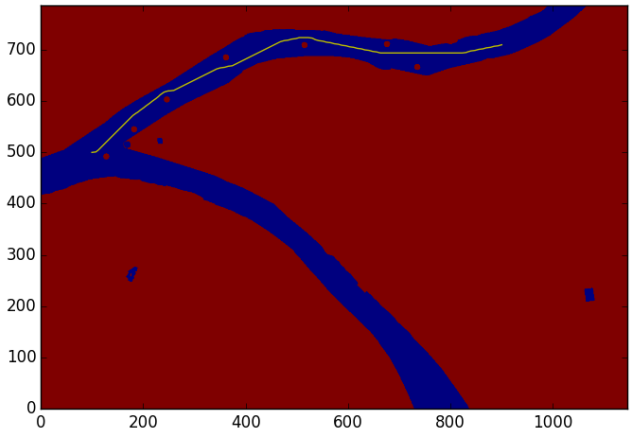

Figure 6.8: Path on Medium Size obstacles

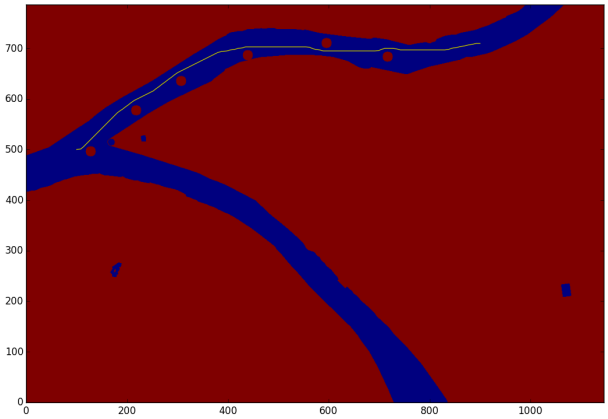

Figure 6.9: Path on Large Size obstacles

Also, we had to tune the motion primitives for the boat by penalizing motions the boat cannot take (for example, 90 degree turns right or left). Before tuning the parameters, the path of the boat is shown in figure 6.10.

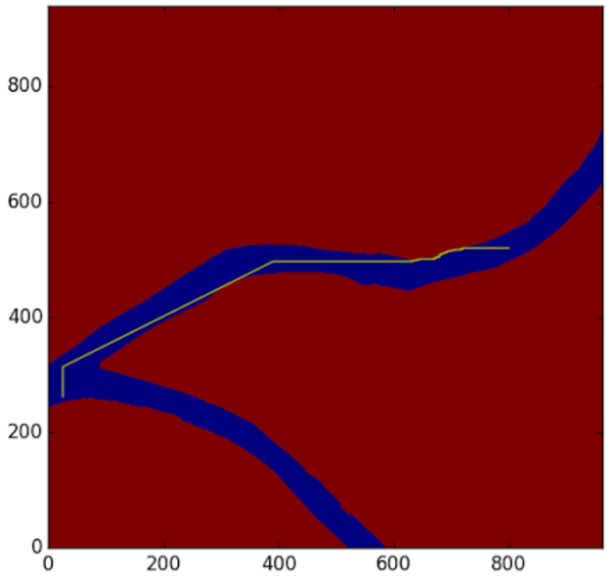

Figure 6.10: Path on Large Size obstacles

Simulator

As we have limited trials and it takes time for us to test the software on the boat, we needed a simulator to visualize the result of our algorithms. So, we created a simple simulator on RViz where we subscribe to the data given from the path planner, the current location (span pose), and the map server.

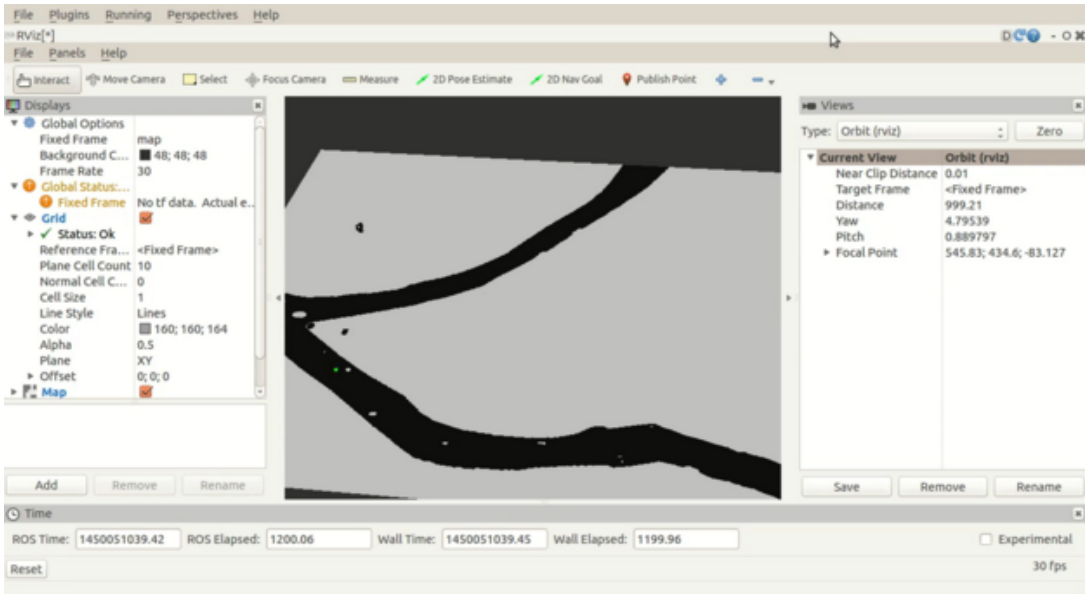

Figure 6.11: RViz Visualization

Added features to Graphical User Interface (GUI)

NREC provided us with the GUI which interfaces to the boat. But it didn't have a feature to add UTM coordinates to the list of waypoints to be navigated. We added this feature in the GUI which can be seen in

the bottom left part of the GUI.(shown as figure 6.12)

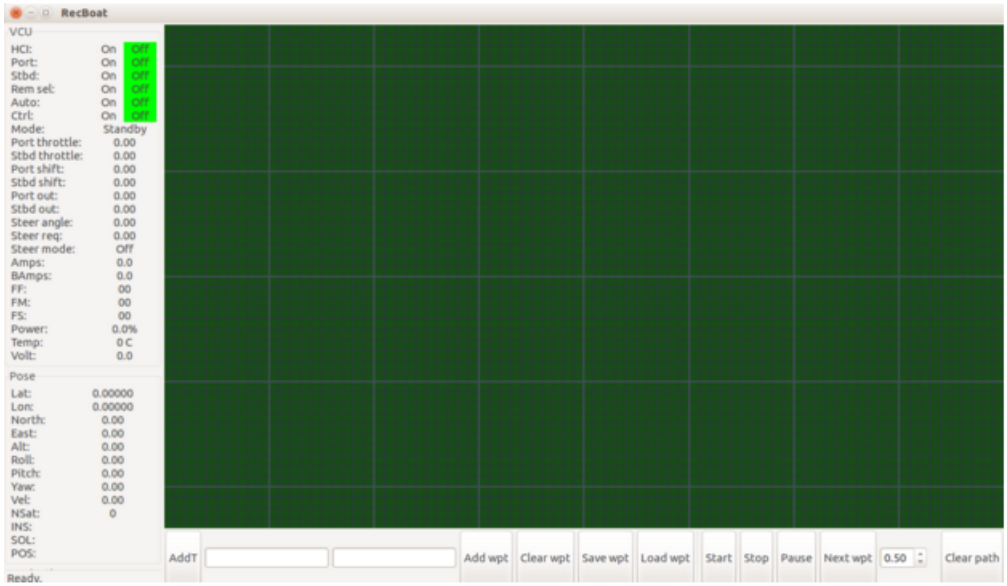

Figure 6.12: GUI

6.3 Modeling, Analysis, and Testing

6.3.1 Radar Analysis

The radar we are working with comes with a display interface installed on the boat. There are many settings we can tune such as gain, range, and scan speed. There are also features such as guard zone monitoring. Additionally, we can use OpenCPN to control a few of these settings on our laptops (Figure 6.13).

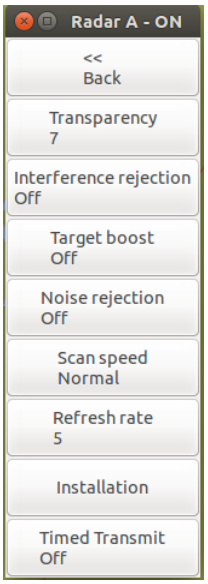

Figure 6.13: Radar Control GUI through OpenCPN plugin

Although we do not fully understand all of the settings and features, we have experimented a lot with the gain and range settings. For the range of the radar, we have currently set it at 125 meters for all of our logged data so far. We decided on this since we want a higher resolution radar image. However, we must consider the speed at which boats would be traveling the river. If we limit our speed to about 15 miles per hour (7 meters per seconds), and we assume a vessel coming for us at the maximum speed, 45 miles per hour, then an obstacle in the radar image moving at 30 meters per second will still be observed by our system. Currently, we receive radar updates at about 1-2 Hz, which ensures that there won't be an issue of not detecting obstacles. However, additional work will need to be done to develop a tactical path planner which should be able to react almost immediately if necessary. But given that most of our development and the spring validation experiment will occur outside of boating season, it is unlikely that this will cause much difficulty.

Considering the gain of the radar, the main trade-off we see is choosing a gain such that we don't introduce too much noise and false positives, while being able to consistently detect small obstacles in the river such as buoys. It is more important that we can detect as many obstacles as possible so we used this idea to guide our gain tuning. Specifically, we made many rounds back and forth at a dock where there were several thin poles in the river serving as the boundaries of the dock. We drove very close to the poles and then slightly further away, while adjusting the gains to make sure we saw sufficient detection and separation in the observed targets.

The 'target separation' feature of the radar is also important for the filtering of our radar data as well as the perception subsystem as a whole. This feature specifies the beam width of each radar scan. Currently, we've set it to the 'High' setting since we would like to be able to distinguish between all the poles we observed in the dock. Additionally, the beam width is crucial for our binomial filter, since the main parameter is specifying how many adjacent cells that we should also check to see if there was a positive return.

6.3.2 Path Planning

A lot of research was done early on to analyze and choose the right path planning library for our project. In general, we chose SBPL (Search-Based Planning Library) since it is relatively easy to use, provides several types of search algorithms, and support is readily available since it is developed here at CMU. Additionally, it has support in ROS so we can explore and integrate with ROS's navigation stack if necessary. As briefly mentioned earlier, the motion primitives SBPL provides are nice because it gives us an, although simplistic, but powerful way to define the kinematic constraints of the boat. Although we have not tested it extensively, we've seen in simulation that the paths generated are smooth, and in a couple of live, short distance path planning scenarios during our field test, the boat seemed to adhere well to the generated trajectory.

Another key part of the path planning subsystem is the mapping functionality. We need a fairly high resolution map, and properly geo-tagged as well so that we can properly position the boat and obstacles for the path planner. QGIS provided us the capability to generate raster maps, and currently we have decided on a resolution of about 5 meters per pixel for our map. This makes for us since it is a good balance between having a map that is a manageable size for the path planner, and having enough resolution to be on the same scale as the minimum obstacle size we expect, as well as the boat and our defined motion primitives.

6.3.3 Integration and Testing

Having ROS as our framework made integration and unit testing a lot easier. Generally, our workflow is to develop a small function or feature in isolation as a single ROS node, and then integrate or keep it modular as needed. It greatly speeds up our development since with some recorded data in a bag file, we can develop and test several different perception subsystem functions in parallel. We were able to quickly prototype the new filter as a separate node. We quickly realized that writing the filter in Python would be way too slow, and after switching to C++, the performance greatly improved. Similarly, having logged data has also allowed the path planning development to occur mostly in parallel as well.

Although our process has made unit-testing and minor integration relatively painless, this could foreshadow a potentially difficult total system integration later on when we put all the parts together and things end up running too slow than is acceptable of an autonomous system. This is described further in our risks, but the general mitigation strategy would be to continue integration as much as possible, as we have seen that it is one of our strengths so far.

6.4 FVE Performance Evaluation

Figure 6.14 shows the script for the Fall Validation Experiments for Perception and figure 6.15 shows the script for the Fall Validation Experiments for path planning.

Fall Validation Experiments: Perception

Step ID	Req.	Step Description	Success Criteria	FVE	FVE Encore	Conditions
PE.1	MF7, MF8	Start RGB video of field test	This is our ground truth	Successful	Successful	Location: NSH Demonstration on laptop (Simulation), Projector required
PE.2	MF7, MF8	Simultaneously replay and visualize saved raw radar data in ROS	We can see plots of the radar image	Successful	Successful	
PE.3	MF3	Show filtered data	The filtered image has less noise than the raw image	Few false positives detected	Improved but requires more testing	
PE.4	MF2	Demonstrate obstacle detection	Draw bounding box around the obstacles.	Successful	Successful	
PE. 5	MF2	Show detected objects from radar image while showing recorded video	The radar should detect obstacles that are in the video, at least the boats (75% success)	Few false positives detected	Need to categorize performance for quantitatively	

Figure 6.14: FVE Perception

Fall Validation Experiments: Path Planning and Simulator

Step ID	Req.	Step Description	Success Criteria	FVE	FVE Encore	Conditions
PL.1	MF1	Interfacing with IMU/GPS	Display logged data from IMU/GPS which we would be collecting during a field test	Successful	Successful	Location: NSH Demonstration on laptop (Simulation), Projector required
PL.2	MF10	Generate Environment Map	Display occupancy grid map for 2 miles with filled blocks depicting obstacles (For one of the rivers of Pittsburgh)	Successful	Successful	
PL.3	MF4	Demonstrate Path Planning	Boat successfully navigates from Point A to Point B in simulator which are 1 mile apart	Successful	Successful	
PL.4	MF5, MF9	Show obstacle avoidance	Boat successfully avoids obstacles in simulator in the above task. Obstacles are static and are simulated through occupancy grip map	Required to demonstrate with different size and number of obstacles	Successful	

Figure 6.15: FVE Path Planning

We were successfully able to demonstrate most of the experiments that we promised for FVE. The summary of the results of our experiments performed during FVE are shown above.

During the FVE, there were a few false positives observed in the radar data. We improved this by using the binomial probabilistic filters and showed the improved results during FVE encore. However, we need to improve the performance of filter and make the data more robust and cleaner.

In addition to this, we also need to measure the success rate of detecting the obstacles by comparing the number of obstacles present in the path and the number of obstacles inferred by our perception algorithms.

In the path planning subsystem, during the FVE, we didnt have the mechanism to simulate with different sizes and number of obstacles. We created a tool to add obstacles (of different sizes) and with an additional feature to inflate the costs near obstacles and shores. We demonstrated the improvement during FVE encore.

6.5 Strong/Weak Points

6.5.1 Strong Points

1. We are able to get continuous technical help from NREC. This is enabling us to cruise through the road blocks that we face.
2. Radar and IMU sensors on the boat are military grade and the performance of the sensors is more than we were expecting initially. High accuracy of the IMU/GPS is saving us from doing extra work in doing fusion of data from IMU/GPS and radar.

3. The hardware in the boat, including the low-level controller to control throttle and steering and Single Board Controller running ROS nodes are robust and highly reliable. This saves a lot of time for us.
4. We are using the SBPL library for path planning which has been developed by the research groups here at CMU. Any challenges related to it are easily solved using the help from the SBPL team.
5. We are using ROS as the platform to communicate between different modules. Also, we need to log and play back the data collected from sensors on the boat. Here, built-in tools of ROS like ROSBAG and the navigation stack are helping us a lot in the development process.

6.5.2 Weak Points

1. Radar only gives obstacles data in 2D format. So, it is not possible to segment bridges, pillars of the bridges and nearby obstacles. To resolve this issue, we will be using existing maps and include pillar data in the occupancy grid map. In addition to this, we will be assuming that there won't be any obstacles near to the bridges.
2. ROS doesn't have good support for radar. We had to interface radar data with ROS on our own. We are continuously improving the architecture and code reliability to make sure that this interface is robust.
3. Planners from the path planning library take around 3-4 seconds to find the shortest possible path. This is too slow and it is not practical considering the 10 MPH speed that we are targeting at.

7. Project Management

7.1 Work Breakdown Structure

The work breakdown structure is shown in the Figure 7.1.

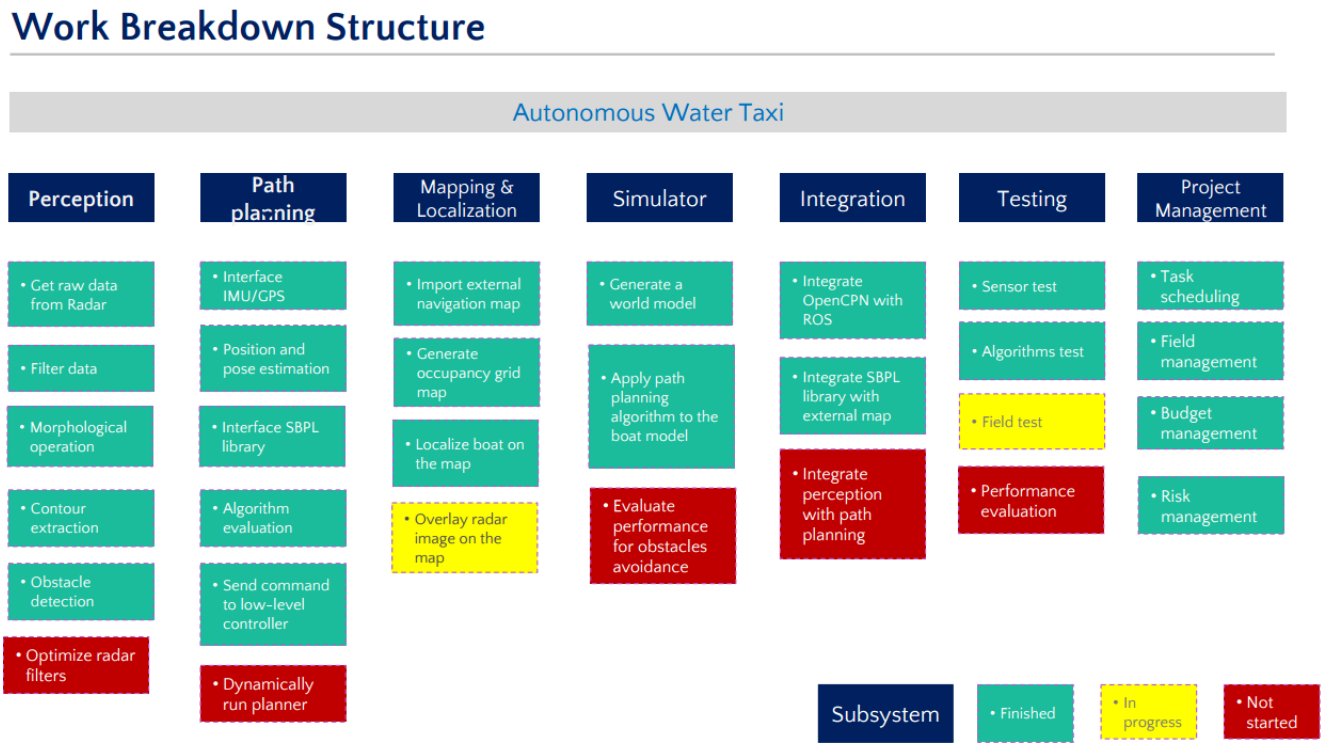

Figure 7.1: Work Breakdown Structure

In the work breakdown structure, the green blocks are what we finished in the fall semester, the yellow blocks are in progress and the red ones are what we haven't started yet. In the fall semester, for the perception part, we have successfully processed the raw radar data and used morphological operations to detect static obstacles and draw bounding boxes around the obstacles. We have implemented a binomial probabilistic filter but we still need to optimize the filter. For path planning, we have successfully interfaced with IMU/GPS and planned a static path using the SBPL library. We will work on running the path planner dynamically. Now we have successfully created an occupancy grid map for the three river area and overlaid the radar image on the occupancy grid map. However, after integrating with the radar filter, it takes more time for the code to run so we are still working on the code efficiency.

As the work breakdown structure shows, we will work on optimizing the radar filters in performance as well as efficiency and achieving the dynamic path planner. Finally, we will integrate the perception and path planning with the boat and tune the controller in the field test.

7.2 Schedule

Table 7.1: Schedule for Spring Semester

S No.	Task	Start Date	End Date	Category
1	Test path planner algorithm on boat	01/13/2016	01/20/2016	Path Planning
2	Improve radar filtering	01/13/2016	01/28/2016	Perception
3	Improve costmap functionality and path planning simulation	01/20/2016	01/27/2016	Simulation
4	Create GUI to enter destination	01/27/2016	02/03/2016	Path Planning
5	Quantify and measure radar efficiency	01/28/2016	02/03/2016	Perception
6	Dynamically generate OGM	02/03/2016	02/17/2016	Perception
7	Integrate radar obstacles with path planner	02/03/2016	02/23/2016	Integration
8	Dynamically plan the path (Global Planner)	02/17/2016	02/23/2016	Path Planning
9	Dynamically Plan the path (Local Planner)	02/23/2016	03/09/2016	Path planning
10	Improve perception efficiency	03/09/2016	03/20/2016	Perception
11	Improve path planning efficiency	03/09/2016	03/20/2016	Path Planning
12	Identify and practice exact scenario for SVE	04/01/2016	04/20/2016	Integration
13	Buffer to complete miscellaneous items	04/20/2016	End of Semester	Integration

For the FVE, we were successfully able to meet the goals which we had set for the team. So, we were on track with the schedule. During the first half of the spring semester we want to target the integration of perception and path planning algorithms and the testing of these algorithms on the boat during our field test. In the later half we would be tuning the algorithms to get better performance. Table 7.1 shows our major milestones for spring semester.

7.3 Test Plan

7.3.1 Progress Review Milestones for Spring Semester

Figure 7.2 shows the milestones for each progress review in spring semester. As we have already finished some basic functionality for both subsystems in the fall semester, the goal for spring semester will be integrating the subsystems and improving the performance and code efficiency. We have prioritized improving our costmap functionality since it will need to be fairly robust to ensure that the generated paths don't take us too close the shores. While perception is more crucial when considering the final system, we may not encounter many other boats in the river until the spring, so we have left the milestones for demonstrating better obstacle detection later in the semester.

Table 7.2: Progress Review Milestones

PR No.	Time	Milestones
PR7	Late January	Improve costmap functionality and path planning simulation
PR8	Mid February	Integrate radar obstacles with path planner
PR9	Late February	Integrate with boat
PR10	Mid March	Improve path planning efficiency
PR11	Early April	Improve radar filtering and efficiency
PR12	Mid April	Identify and practice exact scenario for SVE

7.3.2 Spring Validation Experiments

Table 7.3 shows the test script for Spring Validation Experiments (SVE). Although we still need to finalize the logistics of our spring validation experiment with our sponsor, here is a rough idea of the test conditions.

- Starting Location: South Side Bridge, Monongahela River.
- Equipment: Boat, Radar, IMU/GPS, Fuel, Laptop in boat, Camera (to record video)
- Maximum number of people on Boat: Driver + 5 team members + 2 Professors
- Maximum time allocated: 4 hours (30 minutes to starting location, 30 minutes to get the boat to water, 30 minutes for setup, 1 hour for SVE tests, 1 hour buffer time, 30 minutes to dock the boat)
- Operating Area: The test will span a distance of 2 miles. In this area, there should be at least one bridge, and a buoy. Additionally, there should be 2 - 3 boats in the area. We will need to figure out the logistics of how to make this happen.

Table 7.3: Spring Validation Experiments Script

Step ID.	Step Description	Success Criteria
SV.1	Get the boat on the river	Boat is in river and engines start
SV.2	Keep 5 static obstacles in the path	Size more than $2m \times 2m \times 2m$
SV.3	Start AutoPirates autonomous system	The system starts successfully
SV.4	Enter the destination through OCU	OCU takes the location correctly
SV.5	The boat navigates autonomously and avoid obstacles	Boat avoids at least 4 obstacles
SV.6	The boat arrives the destination	Boat arrives to destination (15 mts radius)
SV.7	Show logged file	Play back logged data
SV.8	Turn off the autonomous system and engine	System switches off without any error

- SV1: Get boat on the river.
  - The safety driver and everyone else gets onto the boat while it is on shore (Figure 7.2). Another NREC employee will unhook us from the trailer.
  - When the boat is fully in the river, start the engines and generator.
  - The safety driver will drive us to the desired start location of our experiment.

Figure 7.2: Boat at launch

- SV2: Keep 5 static obstacles in the path.
  - At the starting location of our experiment, we should be able to see some of the obstacles that the boat will have to navigate around.
- SV3: Start AutoPirates autonomous system
  - The software should start without crashing
- SV4: Enter the destination through OCU
  - Figure 7.3 shows tentative visualization of OCU. The operator needs to click the destination on the OCU. OCU should then show the curve for the planned path to the user. Essentially it would be built on RViz.
  - Figure 7.4 shows the GUI where the trails of the path and the relevant data of the boat would be dispayed.

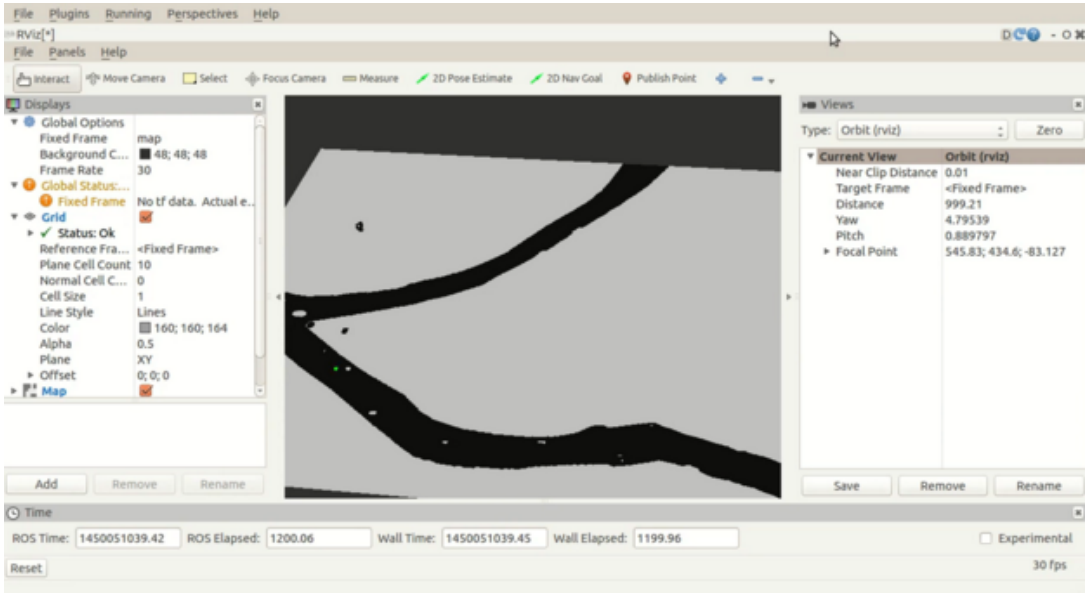

Figure 7.3: RViz Visualization OCU

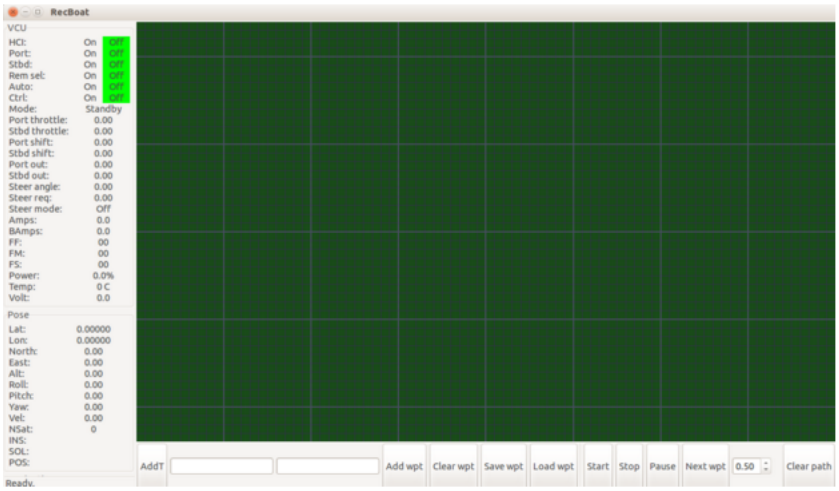

Figure 7.4: OCU

- SV5: The boat navigates autonomously and avoid obstacles
  - Boat should be able to follow the planned path shown on the OCU
  - Boat should avoid at least 4 obstacles out of 5 obstacles coming in the path shown on the OCU
  - Boat be able to go under the bridges. We are making an assumption here that no other obstacle should be within 30 meter of the bridges.
- SV6: The boat arrives the destination
  - Boat should arrive to the destination. We are taking a buffer of 15 meters between the final location of the boat and the desired destination.
- SV7: Show logged file
  - During the test run we would be recording data from radar, RGB camera and IMU/GPS.
  - We should be able to play back and visualize the data at the end of the test. Figure 7.5 shows the play back of radar data and Figure 7.6 shows the playback of IMU data.

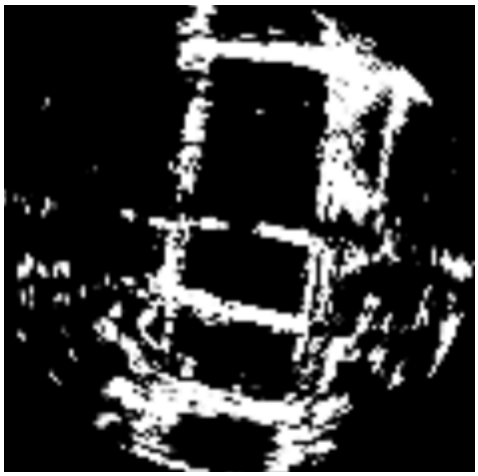

Figure 7.5: Play back Logged radar data

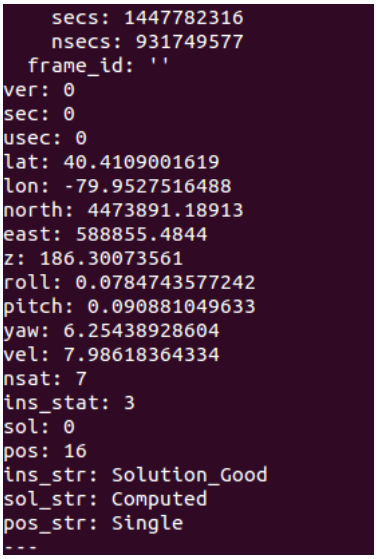

Figure 7.6: Playing back Logged IMU data

- SV8: Turn off the autonomous system and engine
  - Exit the AutoPirates software.
  - Switch off the engines.
  - Drive the boat back to our starting point.

7.4 Budget

Table 7.4 shows the hardware and any equipment that is provided by our sponsor (NREC). We are provided with the 2015 SeaHawk OS 2700S boat, the SimRad 4G FMCW Radar and Novatel SPAN with IMU positioning. We've also got 25 trials for field test for our project.

Table 7.4: Hardware/Equipment/Budget provided by the sponsor

S.No	Part	Part Name	Quantity
1	Boat	2015 SeaHawk OS 2700S	1
2	Radar	SimRad 4G FMCW Radar	1000
3	IMU/GPS	Novatel SPAN with IMU positioning	1
4	Laptop	Dell i7	5
5	Proprietary Boards	SBC and Microcontroller boards	2
6	Trails for testing	$800 per trial	25 (trials)

For the fall semester, we have gone for 2 field test. Since most of the integration part will be done in spring semester, we will use the rest of the 23 trials for testing our integration.

Table 7.5 shows our MRSD project budget.

Table 7.5: Hardware/Equipment/Budget provided by the sponsor

S.No	Category	Description	Qty.	Cost	Total	Spent	Left
1	Transportation	Travel to NREC (5 people)	64	25	1600	50	1550
2	Videography	Professional video for NREC	1	500	500	0	500
3	Backup for Trails	In case we run out of allotted 25 trails	2	800	1600	0	1600
4	Miscellaneous	To buy camera, contraption, books, etc	1	300	300	54	246

We have a $ 4000 budget as part of the MRSD project course. Since we've been provided all the hardware from our sponsor, we don't need to use our budget to buy any hardware devices. Three main major items in our budget are transportation, videography and backup for field test trials. The transportation budget is used to get to NREC faster, especially when we are working late in NREC on night when there are no buses and field test days when we need to be at the boat launch at 7:30 am. As one of the requirements from our sponsor is a demonstration video of our project, so we will use some of the budget for professional videography. Also we will save $ 1600 for backup trials in case we run out of all the trials.

In the fall semester we have spent $ 104 out of $ 4000 for our budget. Half was spent on a book about programming robots with ROS, and the other half was spent on Uber trips to the boat launch location.

7.5 Risk Management

We identified seven risks in Preliminary Design Review (PDR) and discovered two more risks as we went on the field test. One issue is that the radar is unable to detect bridge supports and the other one is the code efficiency.

In the course of risk management, we mitigated five risks, ID 1, 2, 3, 6, and 7 in the Preliminary Design Review (PDR) and have been keeping track of four risks, ID 4, 5, 8, and 9 until now.

Table 7.6: Risk Management

ID	Risk Title	Req	Type	Description	Likelihood	Consequences	Mitigating Actions
1	Test environment availability	All	Testing	Weather conditions delay the field test	4	3	Test radar perception on shore for static obstacles. Use a simulator for path planning.
2	Limited testing trials	MNFR2	Project management	Only 25-30 field tests	5	3	Appoint a field manager to plan logistics of each test.
3	Radar Performance	MF2 MF3 DF1 DF2	Technical	Radar signal is too noisy	3	4	Incorporate additional sensors like Lidar and camera
4	Localization accuracy	MF4 MF5	Technical	Path planning algorithm relies on precise localization given by GPS	3	5	Filter GPS data, Limit the maximum speed of the boat.
5	Insufficient boat protection	All	Testing	No automatic stop in case of a possible collision	2	5	Develop fail-safe software modules.
6	Technical support from NREC	All	Technical	Limited availability from NREC engineers	4	3	Ask for software documentation.
7	Team Communication	All	Project management	Language barrier in the team	4	2	More preparations before the meetings.
8	The Radar only has 2D information	All	Technical	Can't plan path without information of bridge supports	5	5	Research nautical maps that has information for the bridge supports
9	Code Efficiency	All	Technical	Code may run too slow after integration	5	4	Integration early and often to identify bottlenecks.

This is how we responded to the risks:

Risk ID 1: Thanks to the warm winter, we've been able to go on 2 field tests so far. As there is opinion that it would be trouble in field test in March and April because of lumps of ice on the river, we should do more field tests in January and February.

Risk ID 2: We have logged radar sensor data and INS sensor data of the boat while doing our field tests. As a result, we could implement and test perception and path planning algorithm using the logged data. It helps us minimize the field test trials.

Risk ID 3: Although the radar signal had lots of noise on the shore, the radar noise was significantly decreased on the river. For improving perception performance more, we have been applying several filters to our radar signal.

Risk ID 4: Until now, the INS accuracy for boat localization was not as bad as we expected. We've been tracking this issue while implementing the path planning algorithm.

Risk ID 5: We are investigating the way to create an automatic stop in case of a possible collision.

Risk ID 6: For path planning, we found that there is a SBPL software researcher on campus, so we've been taking support from him in the path planning. Except for that, we have support on how to use the existing functions of the boat for doing field test from NREC engineers. Consequently, we could operate the boat for field test now.

Risk ID 7: Our team has regular meetings for communication to overcome the language barrier.

Risk ID 8: We plan to add coordinates of the bridge supports to the occupancy grid map by referencing geographical information.

Risk ID 9: As there are a lot of subsystems, it has possibility reducing the code efficiency. For this reason, we should integrate subsystems into the entire system as soon as possible, identifying bottlenecks and resolving them. The risk likelihood-consequence table is shown as figure 7.7.

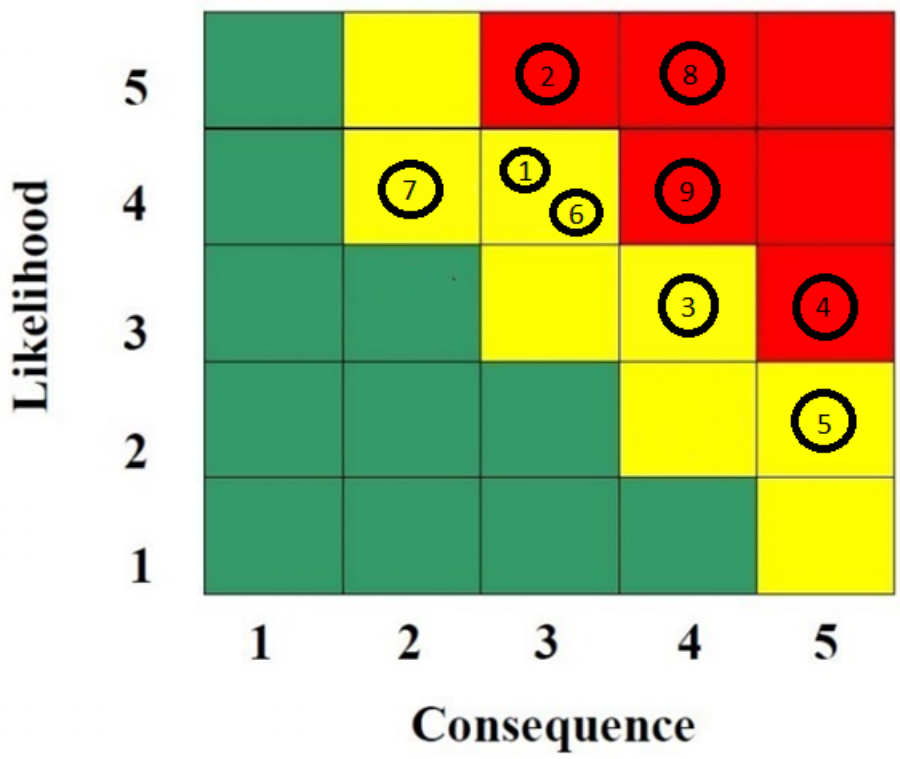

Figure 7.7: Risk Likelihood-Consequence Table

8. Conclusion

8.1 Lessons learned

Our team has learned several lessons after experiencing the design and embodiment of the autonomous boat system like the following.

Prototyping is made with adjustment of schedule. Before fabricating the prototype, we couldn't fully figure out the scope of our development and determine the schedule of the project. Scheduling should keeps pace with development of the project which is what Agile iterative development process focuses on.

Code review is important. Code written by original writer may have some bugs. The other team mates could find these bugs in code review, which would have resulted in initial appearance of bugs. As well, within limited time like field test, code review helps teammates spend less time with finding errors and problems in the code.

8.2 Key activities in the spring semester

We achieved good progress in path planning and perception for the fall validation experiment. Based on this accomplishment, we outlined the three main key activities that we should focus on in the spring semester.

We should implement Dynamic path planning to consider the dynamic environment like other boats navigating nearby our autonomous boat.

Perception is the key aspect for the autonomous boat. We implemented binomial filtering of radar data in the fall semester. In addition, we'll implement and optimize several more filters for radar data.

Ultimately, we need to integrate dynamic path planning and optimized perception into our boat to improve overall performance and safety of autonomous boat.

References

[1] Robot Operating System
http://ros.org/

[2] SBPL Reference
http://sbpl.net/

[3] OpenCV
http://opencv.org/

[4] OpenCPN
http://sourceforge.net/projects/opencpn/

[5] Schuster, Michael, Michael Blaich, and Johannes Reuter. "Collision Avoidance for Vessels using a Low-Cost Radar Sensor." World Congress. Vol. 19. No. 1. 2014.