# Critical Design Review Report: Auto-Park for Social Robots

## Collaborative Parking between Autonomous Vehicles Utilizing Point-to-Point Communications

**Team Daedalus: Mohak Bhardwaj, Shivam Gautam, Dorothy Kirlew, Pranav Maheshwari, and Richa Varma**
**Masters of Science Robotics Systems Development**
**Sponsored by United Technologies Research Center**
**12/13/2015**

Carnegie Mellon
THE ROBOTICS INSTITUTE

# 1 Abstract

This report summarizes the progress made by Team Daedalus on the Carnegie Mellon University, Masters of Science in Robotic Systems Development project "Auto-Park for Social Robots" during the fall 2015 semester. The team has identified key functional and non-functional requirements of the project and has also defined relevant performance metrics. The system-level architectures representing the various subsystems and their interactions are depicted in the report. Detailed subsystem descriptions and their testing results are documented. The formulated project management plans with respect to work-breakdown, schedules, test plans, and budget are also discussed. The team has also identified key risks associated with the project and their mitigation strategies. Having gained valuable experience from testing, the team has identified the key lessons learned from this semester. This information has helped in planning for the completion of this project in the spring semester.

# 2 Table of Contents

# 3 Project Description

## 3.1 Keywords

- Optimal Spot – The optimal spot is the parking spot with the shortest route between that spot and the exit. The optimal spot must also be unoccupied.
- Optimal Route – Also called Optimal Path. This is the route is from the vehicle's current position to the optimal spot or to the exit, depending on the vehicle's status. It is optimal because it takes the least amount of time to traverse.
- Vehicle – Also called Mobile Platform or Robot. This serves as the test platform to implement and showcase the collaborative and autonomous aspects of the project, such as path planning, navigation, communication, and obstacle detection.
- Vehicle Status – The vehicle can be in the following states:
    - Free – The user has not yet told the vehicle to park. The vehicle is waiting for a command.
    - Parking – The vehicle has been told to park by the user. The vehicle is heading towards the optimal spot but has not reached it yet.
    - Parked – The vehicle is stationary within the parking spot.
    - Returning – The vehicle has been told to return by the user. The vehicle is heading towards the parking lot exit but has not reached it yet.
    - Returned – The vehicle is at a complete stop at the parking lot exit and is waiting for the user.
- Parking Lot – The parking lot is a single-level testing area with a known entrance, exit, and known parking spots. The lot will be proportional to the size of the vehicle.

## 3.2 Description

The imminent arrival of driverless cars has led to an increased focus on the development of an ecosystem that supports them. This project, "Auto-Park for Social Robots", aims at developing an autonomous system for collaboratively parking driverless cars. As envisioned by the team and the sponsor, United Technologies Research Center (UTRC), the project would allow a user with a driverless car to park their vehicle by simply pressing "Park" on their Android app.

The motivation for the project stems from key factors affecting the current parking system, such as poor parking safety standards, parking industry growth potential, and a competitive advantage of developing such a system. According to reports by Fayard and Stark, around 20% of all automobile accidents occur inside parking lots. About 90% of the people involved in these accidents are injured. The parking industry is worth $25-$30 billion and has potential to invest in upgrades [1] [2]. It regularly bleeds money to accident insurance claims, as well as public transportation due to people that take the bus rather than find a spot in a congested lot. The precious time wasted while searching for a parking spot, translated into lost revenue, is also a motivation behind the project.

This project consists of a scaled-down version of a car (mobile platform) that would be able to receive a "Park" command from the user's Android app, localize itself in a parking lot, collaborate with other cars to identify the best possible parking spot, navigate to the spot, and avoid any obstacles in the process. The vehicle would relay its status to the user's app at specific

times. Upon receiving the "Return" command, the vehicle would then exit the parking spot and navigate to the parking lot exit. The project's main focus is on establishing a robust system for effective collaboration between vehicles.

# 4 Use Case

Benjamin is a retired Armed Forces veteran. He has been driving in Pittsburgh, his home city, for many years now. The only thing that has made him think twice before taking his car out is the nightmare of parking. Recently, Benjamin was diagnosed with prostate cancer. Ben is receiving treatment at UPMC, 10 miles away from his home. The parking lot at the hospital is always extremely congested (Figure 1: Crowded Parking Lot) and it sometimes takes him longer to find a parking space than to drive from his home. He has to leave an hour and a half before his appointment and is frequently late.

**Figure 1: Crowded Parking Lot**

After his appointment last week, Ben was backing out of a parking spot when a reckless driver sped into his bumper at 30 mph. He spent the next few weeks handling insurance claims and getting repairs done. After the accident, Benjamin started taking the Port Authority bus to the hospital, even though he had to transfer buses several times to reach the hospital. Benjamin missed driving, but the scare from the accident had made him give it up altogether.

Five years have passed. It is 2020 and Benjamin has purchased an autonomous car equipped with CMU-UTRC Auto-Park System. His new car now takes him to the hospital for his weekly appointments – efficiently, safely, and without any hassle.

When Benjamin's car stops at the entrance of the hospital, he exits his car and presses "Park" on his smartphone app (Figure 2). The app sends the command to his driverless car and the status of Benjamin's car updates on his phone (Figure 3). Benjamin enters the hospital and his car autonomously enters the queue at the entrance of the parking lot.


**Figure 2: Virtual Valet App, Opening Screen**


**Figure 3: Virtual Valet App with "Park" Button Pressed**

As the vehicle enters the parking lot, it localizes itself and identifies nearby cars (Figure 4). His vehicle then connects to the network of cars already inside the parking lot (Figure 5).

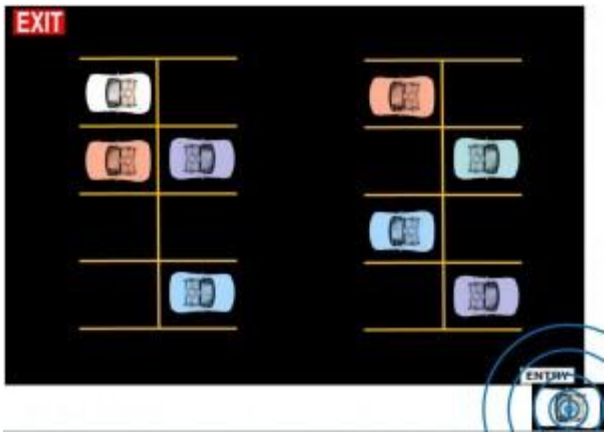


**Figure 4: Vehicle Localizes Itself Within Parking Lot**

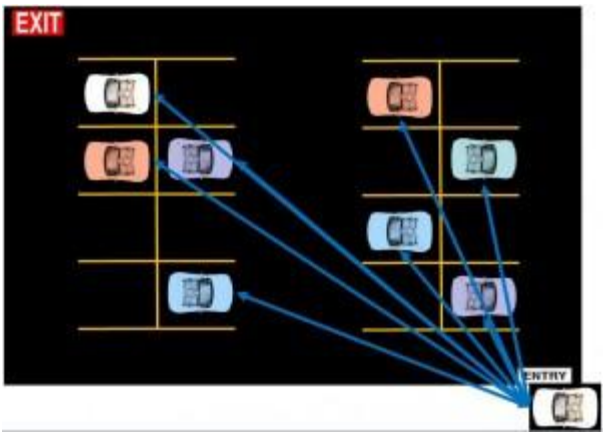


**Figure 5: Vehicle Initiates Collaborative Communication with Parked Vehicles**

Benjamin's car receives an occupancy map from the network of vehicles that allows it to identify the location of a free parking spot that is closest to the exit (Figure 6). His car autonomously plans a route to the spot and begins to follow the path (Figure 7).

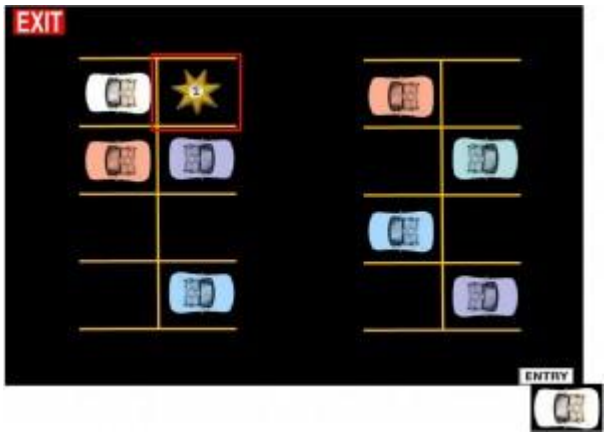


**Figure 6: Optimal Parking Spot is Identified**

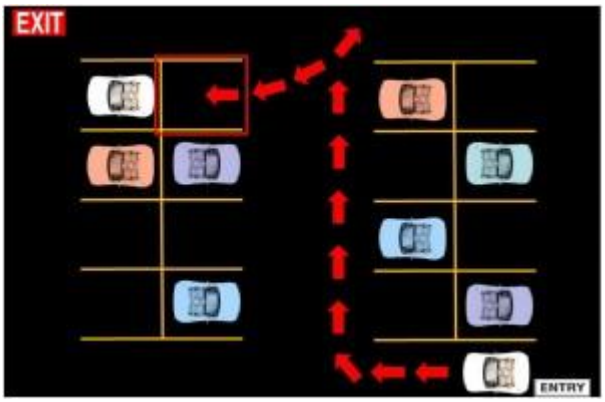


**Figure 7: Shortest Route to Optimal Spot is Planned**

While en route to the spot, another car receives a "Return" command and notifies nearby cars that it will be exiting its spot (Figure 8). Benjamin's car stops to give right of way to the exiting vehicle (Figure 9).

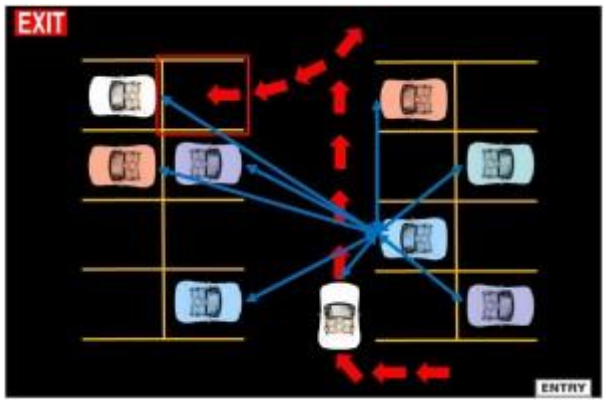


**Figure 8: Benjamin's Vehicle Stops as Another Vehicle Receives "Return" Command**

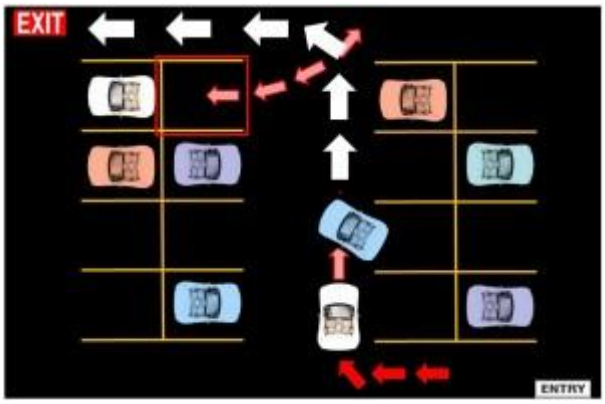


**Figure 9: Benjamin's Vehicle Gives Right of Way to Exiting Vehicle**

Benjamin's car waits for the exiting vehicle to be at a safe distance, and then continues on the original route. It then parks in the designated spot (Figure 10), notifies him that it has parked, and waits to receive the next command (Figure 11).

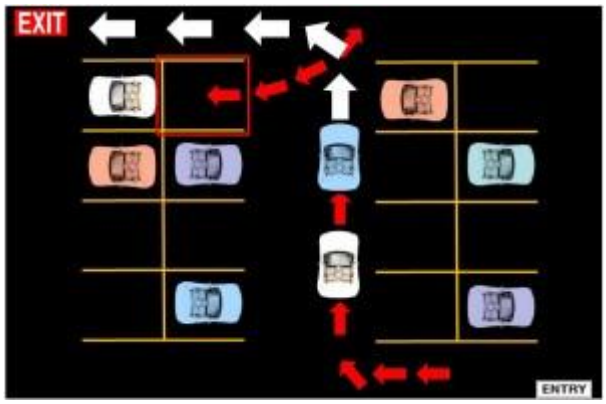


**Figure 10: Vehicles Continue on Path**

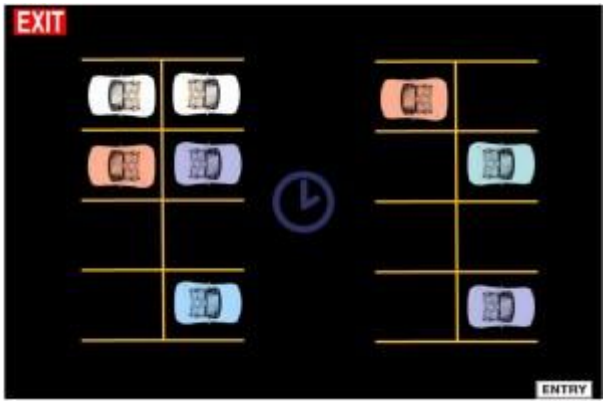


**Figure 11: Benjamin's Vehicle Waits for Return Command**

Benjamin is tired at the end of his appointment and wants to get home as quickly as possible. He hits the "Return" button on his app and walks towards the hospital exit (Figure 12).

His car receives the command and alerts nearby cars that it is about to exit the spot. Benjamin's app shows that his car is returning to him and displays the estimated time it will take to reach the exit (Figure 13).


**Figure 12: App Shows Parked Status**


**Figure 13: App Shows Returning Status and ETA**

Ben's car plans and follows the shortest route to the exit, where it waits for Ben (Figure 14).

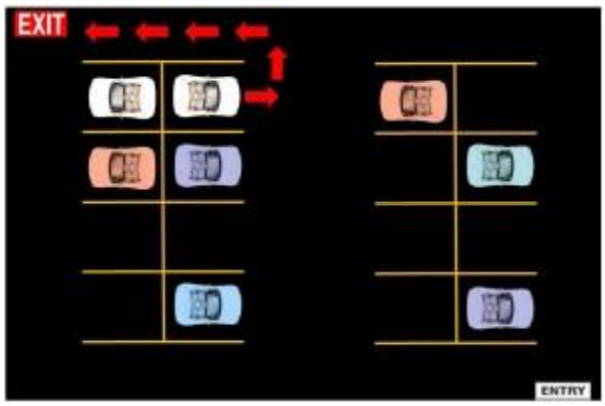


**Figure 14: Vehicle Plans Shortest Path to Exit**

By the time Benjamin reaches the hospital exit, where his car is waiting for him. He enters his car, grateful that he did not have to walk through the large parking lot after such a long day.

# 5 System Level Requirements

The system-level requirements are categorized as Mandatory (M) or Desirable (D), as well as Functional (F) or Nonfunctional (N). The requirements are meant to be read as "The system shall…", followed by the requirement. The performance metric(s) detail how the requirements can be validated. Furthermore, the subsystem column shows which of the following subsystems the requirement applies to Communication (P2P or Bluetooth), Perception, Mobile Platform, Software, or Control.

The requirements are much the same as those listed in the Preliminary Design Review, with the exception of the communication performance requirements. Those have been adjusted to measure P2P communication in terms of the percentage of messages received instead of the time it takes to establish communication. This is because XBees establish communication immediately, and it is more likely to lose a message, or for it to be misinterpreted, than for the XBees to not establish communication with one another. Another requirement change has moved the MF.11 from a desirable requirement to a mandatory one.

**Table 1: Mandatory Functional (MF) System Level Requirements**

| ID | Requirement | Performance Metric(s) | Subsystem |
|---|---|---|---|
| MF.1 | Receive "Park" and "Return" commands from user via smartphone app | 95% of messages will be received. | Communication (Bluetooth) |
| MF.2 | Share location, parking spot, and obstacle-related data with other vehicles. | Establish communication with other vehicles within the 10mx10m test area.<br>90% of messages will be received. | Communication (P2P), Perception |
| MF.3 | Navigate autonomously through parking lot. | 100% of navigation will be autonomous. | Mobile Platform |
| MF.4 | Plan optimal route to exit. | The vehicle will exit the parking lot within 90 seconds of receiving command. | Software |

| ID | Requirement | Performance Metric(s) | Subsystem |
|---|---|---|---|
| MF.5 | Follow optimal route to exit. | The vehicle will maintain a velocity between 0 and 10 cm/sec. | Mobile Platform, Software |
| MF.6 | Park inside parking spot. | Park 100% within a parking spot within 2 attempts. Be within 35º of parallel with the neighboring vehicles or the lines of the spot, as applicable. | Mobile Platform, Perception |
| MF.7 | Exit parking spot | Exit the spot within 2 attempts without collision. | Mobile Platform |
| MF.8 | Sense obstacles in the environment. | Avoid obstacles between 10-50 cm high and 10-120 cm wide. | Mobile Platform, Perception |
| MF.9 | Avoid infrastructure | The vehicle will maintain a distance of 30.48 cm (1 ft.) between itself and the parking lot infrastructure. | Mobile Platform, Perception |
| MF.10 | Stop in the event of an emergency | Stop within 3 seconds of an emergency (obstacle or internal vehicle error). | Mobile Platform, Perception |
| MF.11 | Maneuver efficiently through the lot | Vehicle has turning radius between 0 and 0.8 meters | Mobile Platform, Software |

**Table 2: Desirable Functional (DF) System Level Requirements**

| ID | Requirement | Performance Metric(s) | Subsystem |
|---|---|---|---|
| DF.1 | Identify optimal parking spot | Identify optimal spot 98% of the time | Communication (P2P), Software |
| DF.2 | Plan optimal route to spot | Optimal path is chosen 90% of the time | Software |
| DF.3 | Follow optimal route to spot | Vehicle maintains a velocity between 0 and 10 cm/sec | Mobile Platform, Software |
| DF.4 | Avoid other vehicles | Vehicle maintains at least 60.96 cm (2 ft.) between itself and the back of another moving vehicle | Mobile Platform, Perception |

**Table 3: Mandatory Non-Functional (MN) System Level Requirements**

| ID | Requirement | Performance Metric(s) | Subsystem |
|---|---|---|---|
| MN.1 | Use smartphone app to display vehicle status | 95% of messages are received | Communication (Bluetooth) |
| MN.2 | Communicate reliably between local vehicles | The network will be able to handle collaboration between 3 vehicles | Communication (P2P) |
| MN.3 | Efficiently exits the parking spot | Will take no more than 45 seconds to exit the parking spot | Mobile Platform, Perception, Software |
| MN.4 | Return to user as quickly as possible | The vehicle will arrive at the exit within 90 seconds of receiving the “Return” command | Communication (P2P and Bluetooth), Mobile Platform, Software |
| MN.5 | Make minimal changes to infrastructure | There will be ZERO changes to the infrastructure | N/A |
| MN.6 | Be within stipulated budget | Budget is $4000 | N/A |

**Table 4: Desirable Non-Functional (DN) System Level Requirements**

| ID | Requirement | Performance Metric(s) | Subsystem |
|---|---|---|---|
| DN.1 | Maintain scalable network of vehicles | Network is able to accommodate at least 3 vehicles | Communication (P2P) |
| DN.2 | Efficiently enter the parking spot | Vehicle backs into parking spot within 2 attempts<br>Vehicle takes no more than 45 seconds to back into spot | Mobile Platform |

# 6 Functional Architecture

The system is represented in the functional architecture, as seen in Figure 15. The inputs to the system are a preloaded map, the "Park" command from the user, and the "Return" command from the user. The outputs from the system are the "Car Parked" and "Car Returned" notifications to the user.

The entire flow can be divided into two phases: Park and Return. The structure of the flow diagram is based on the "Sense-Plan-Act" design.

In the Park phase, the vehicle receives a "Park" command from the user via the Android app and navigates to the entry queue, continuously localizing itself in the environment. It queries other vehicles in the parking lot for information needed to plan its route to the optimal spot. On selecting the best spot based on this data, it plans its route to the spot and starts navigation. Localization data is needed to continuously update the path planner and if obstacles are encountered along the way, the path is modified accordingly. Upon reaching the spot, the vehicles parks in the designated spot, sends a "Car Parked" notification to the user, and waits for the return command from the user.

In the Return phase, upon receiving the "Return" command from the user, the vehicle plans the optimal route to the exit based on its current location and the data provided by other cars regarding the conditions in the parking lot. It starts navigating towards the exit, sensing obstacles along the way and sending a notification to the user once it reaches the exit queue.

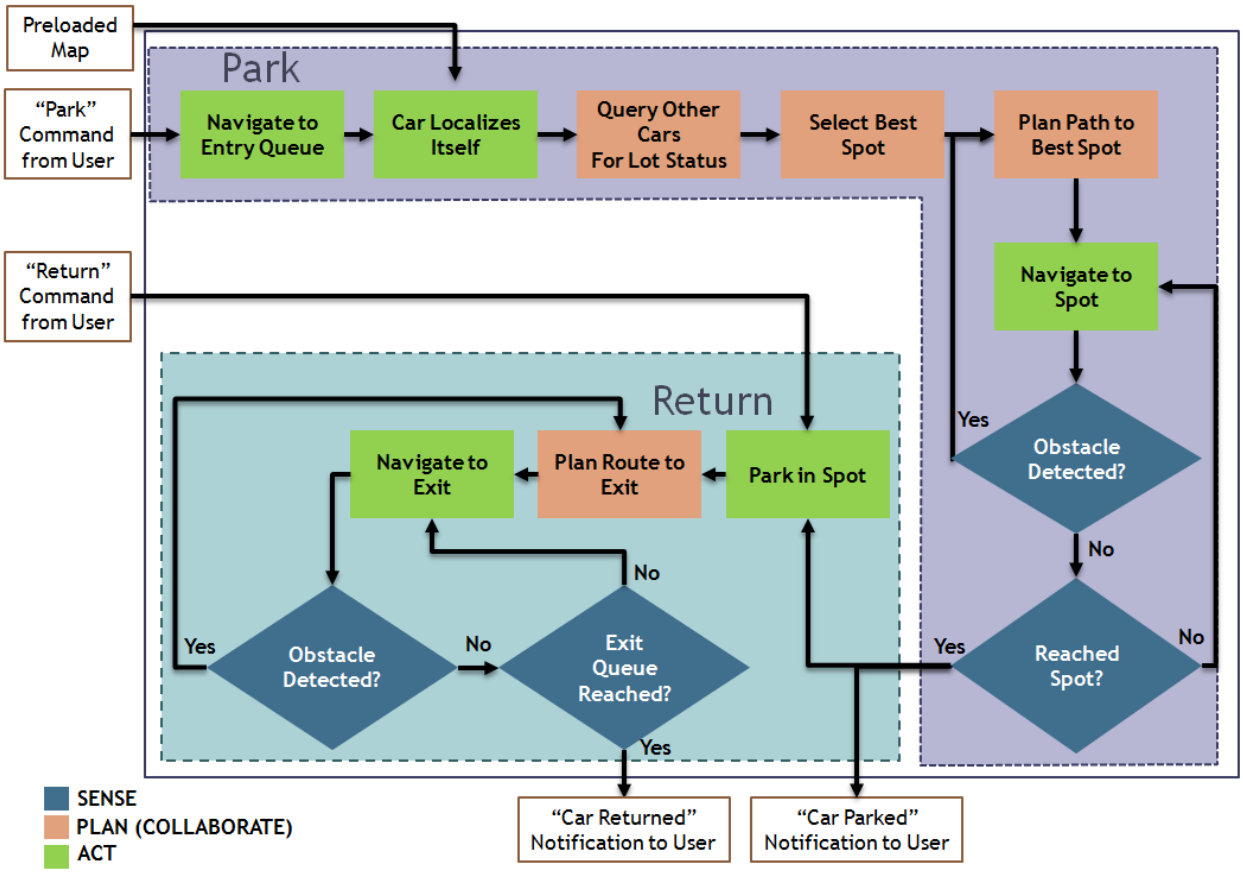


**Figure 15: Functional Architecture**

# 7 Cyber-Physical Architecture

The system can be divided into 5 main subsystems, as seen in Figure 16. The hardware and software interaction between these subsystems is detailed below.

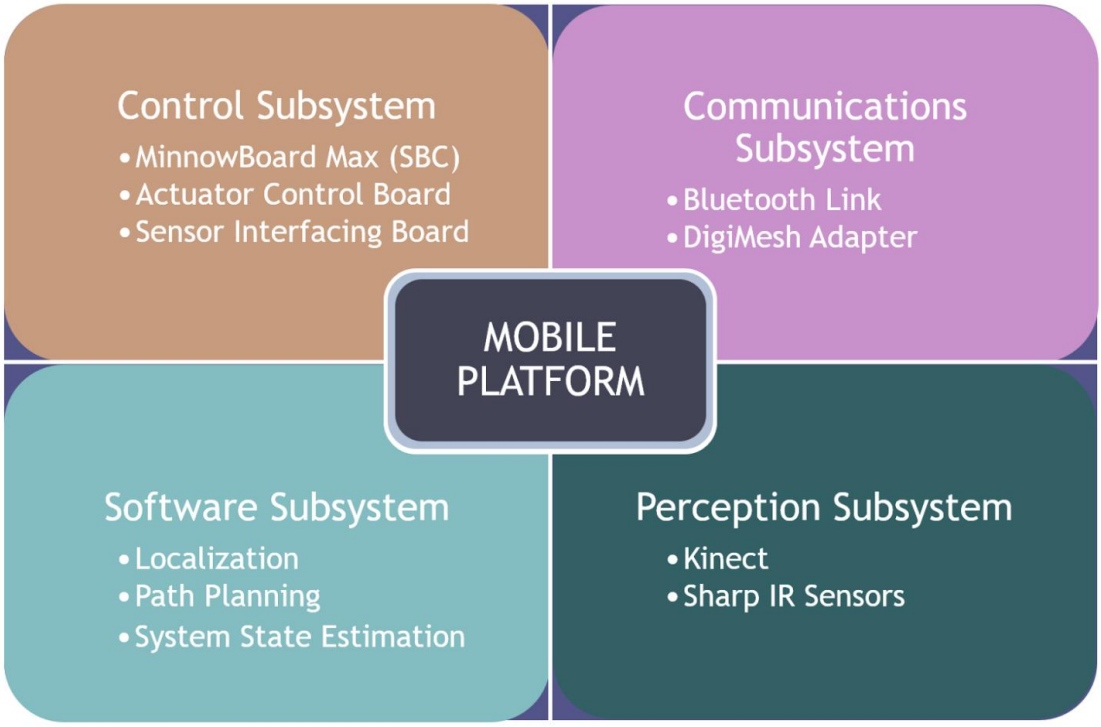


**Figure 16: System Overview**

## 7.1 Hardware Architecture

The mobile platform houses all the major subsystems, apart from the mobile app, which exists on the user's Android phone. Currently, the mobile platform uses a MinnowBoard Max SBC which is running Xubuntu 14.04 and ROS Indigo. The platform also has a Kinect v1 and three IR Proximity Sensors for obstacle detection. IR Proximity Sensors interfacing and actuator control are done by two Arduino Nanos. For the proximity sensor, the Arduino Nano is made to run as a ROS Node; for actuator control, the Arduino Nano communicates via Telnet Server. The DigiMesh XBee and Bluetooth 4.0 adapters are connected via USB to the SBC and act as Serial Ports for communication. The Oculus Prime platform comes with a power distribution board that accepts 12V from the LiPo battery and then powers the DC Motors and the SBC through it (Figure 17).

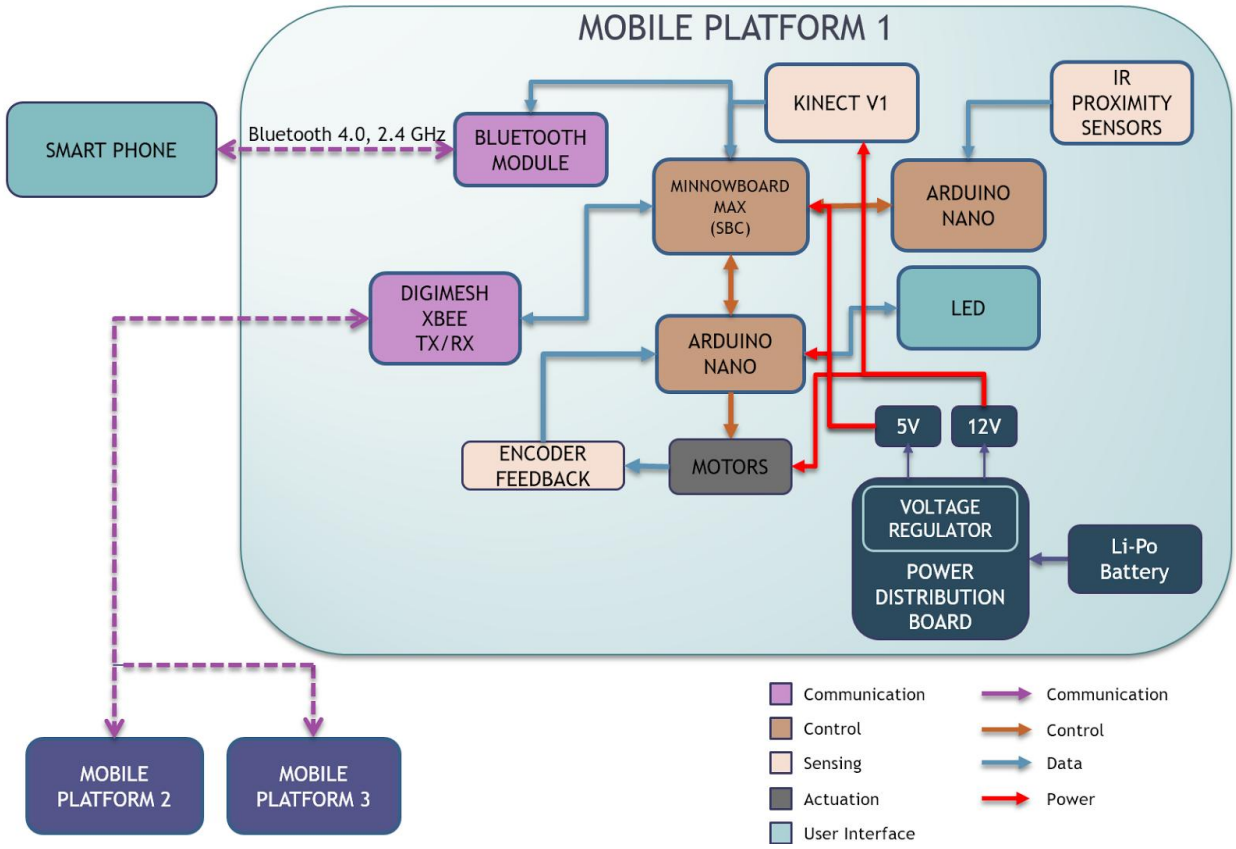


**Figure 17: Hardware Architecture**

## 7.2 Software Architecture

The software architecture (Figure 18) is based on the simple principle of sense, think, and act, denoted by Perception, Planning, and Control. Perception helps in interfacing with the environment and getting raw data, which then gets processed by Planning. Planning carries out path planning, localization, and uses the point cloud data to detect various objects in the vicinity of the robot. All of this information is then further transmitted to Control, where the robot carries out locomotion and also collaborates with other robots by sharing relevant data. The emergency node helps in bringing the robot to a halt in case of internal failures or the presence of an obstacle.

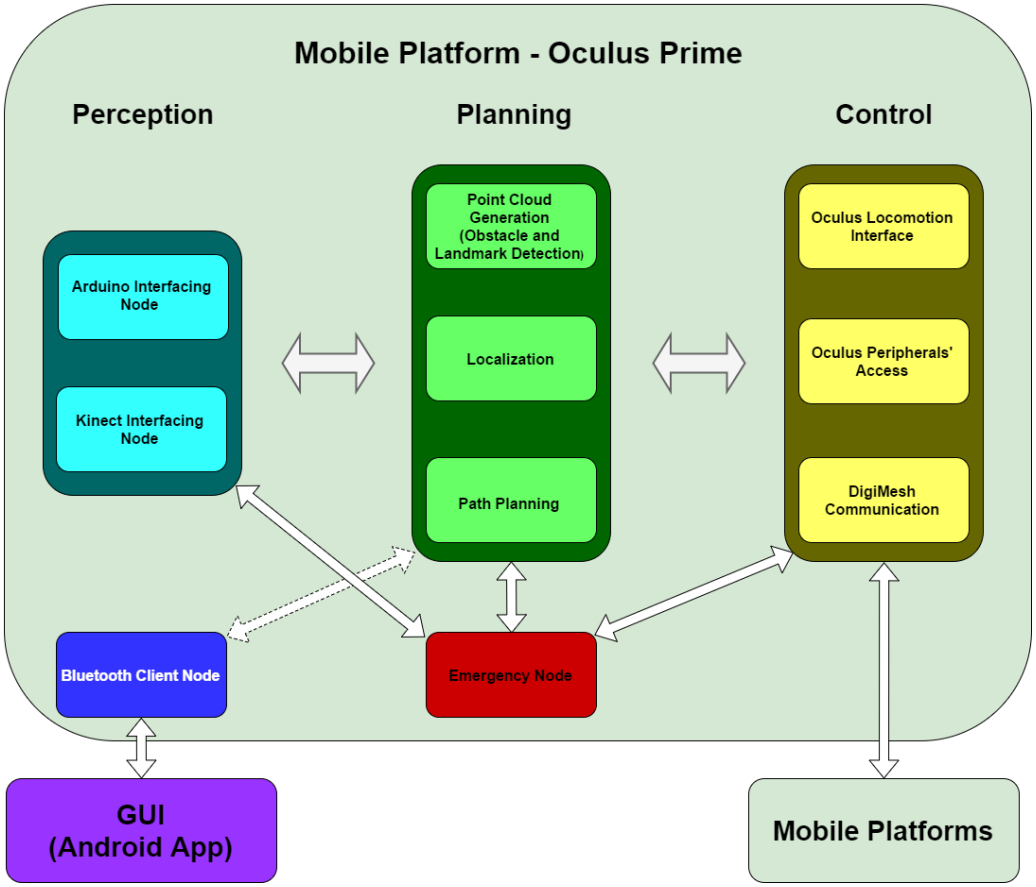


**Figure 18: Software Architecture**

The current algorithm to detect obstacles via Kinect can be seen in Figure 19.


**Figure 19: Kinect Obstacle Detection Algorithm**

For collaboration, a simple protocol is defined whereby a platform sends out a request for collaboration and other platforms reply back with relevant data. The flow of this interaction can be seen in Figure 20.

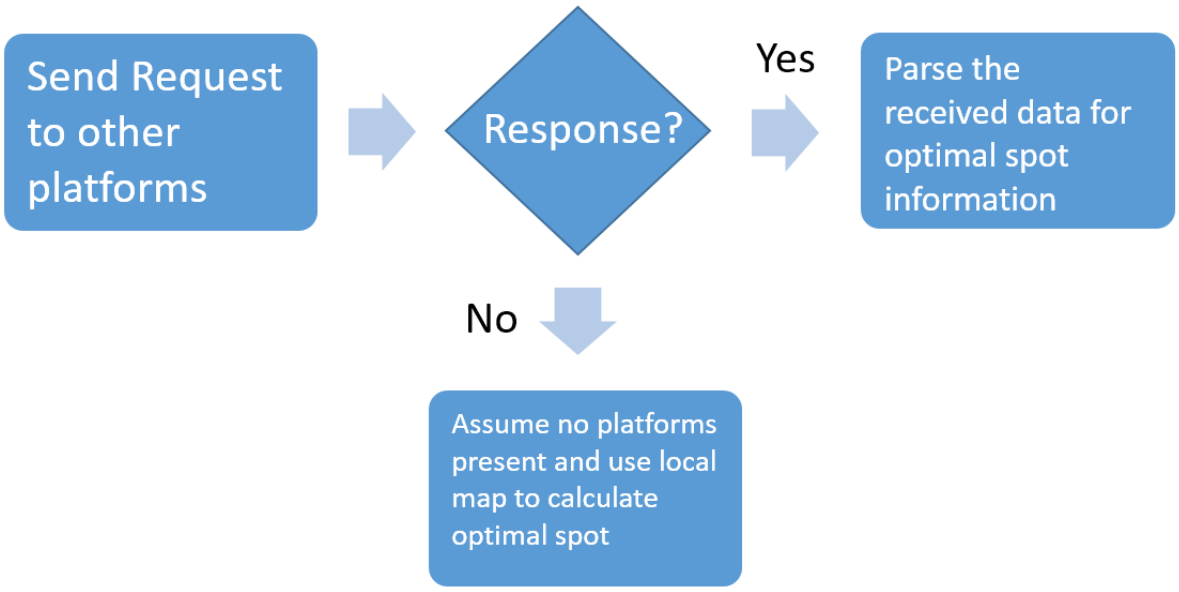


**Figure 20: Mesh Network Collaboration**

# 8 Current System Status

## 8.1 Fall Semester Targeted System Requirements

The following functional requirements were targeted during the fall semester can be seen in Table 5: Requirements Targeted in Fall.

**Table 5: Requirements Targeted in Fall**

| Requirement | Performance Metric |
|---|---|
| Receive commands from user via smartphone app (MF.1) | 95% of messages will be received |
| Share data with other cars (MF.2) | 90% of messages are received<br>Be able to handle collaboration between 2 vehicles |
| Sense the environment (static obstacles) (MF.8) | Detect obstacles within 20 cm of vehicle<br>Detect obstacles 10-50 cm high and 10-120 cm wide |
| Navigate through parking lot (MF.10) | Stop within 20 cm of a static obstacle and 3 seconds of an internal vehicle error |
| Maneuver efficiently through the lot (MF.11) | Vehicle has a turning radius between 0 and 0.8 meters |

## 8.2 Current System Description

An overview of the current subsystems of the project is depicted in Figure 21: Auto-Park for Social Robots Subsystems. The team focused on covering a larger breadth in subsystems so as to develop a good foundation for the spring semester. In addition to laying the groundwork in all of the subsystems, the team also worked on integrating all of the systems capabilities.

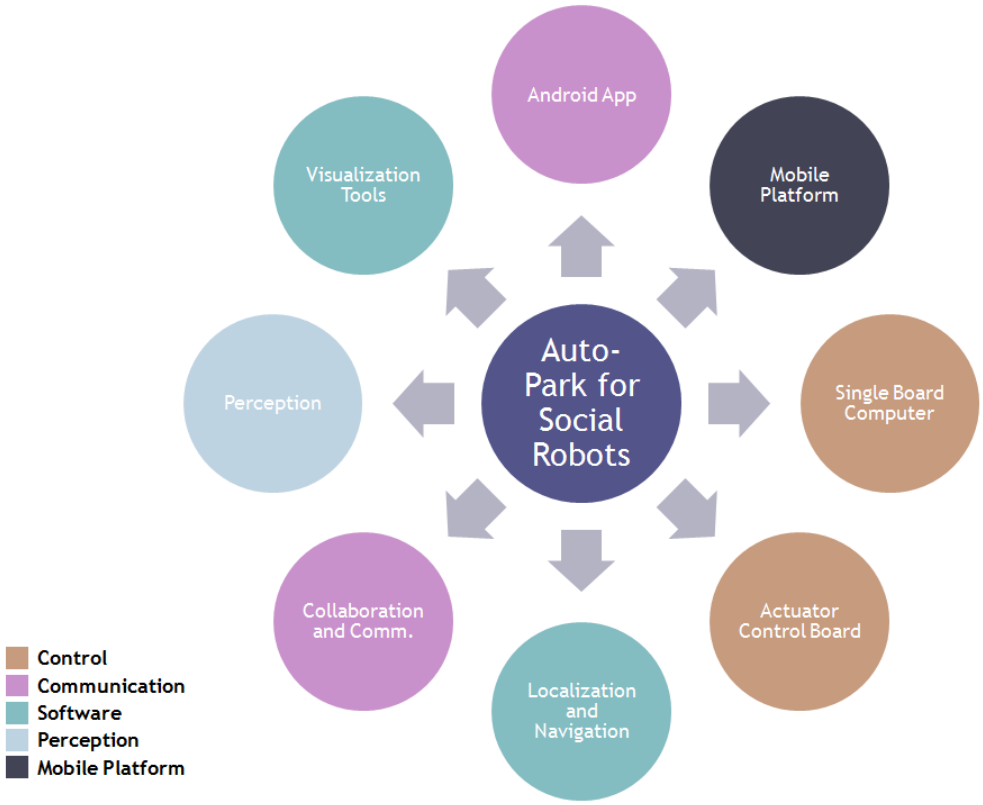


**Figure 21: Auto-Park for Social Robots Subsystems**

### 8.2.1 Android – ROS Bluetooth Communication

The Android application developed for this task utilizes the Bluetooth adapter present in the smartphone to establish communication with the Single Board Computer, which is running ROS. This bidirectional communication requires a server, client, and a service with a specific protocol through which all this interaction takes place. This specific subsystem can be broken down into two main parts:

- Android Application
    - The Android app developed utilizes the BluetoothChatService provided by Google for developers. This chat service helps to establish and manage connections with remote devices by running the appropriate threads. Data is sent from the app to the SBC whenever the user presses the Park or Return buttons on the app. The data received by the app is used to update the status and initialize the timer. The app can be made to connect with any Bluetooth device by entering the appropriate device name during startup. The app can be seen in Figure 22

with a status of Parking and a timer counting down. In Figure 23, the app shows that the vehicle has returned.

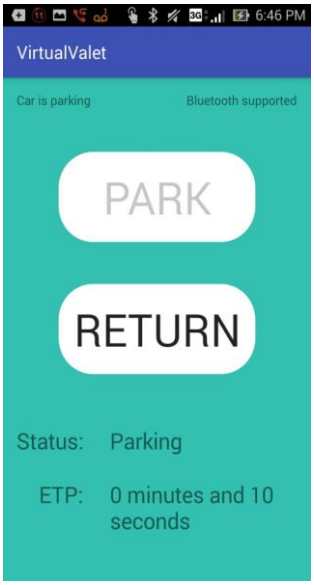


**Figure 22: Vehicle is Parking**

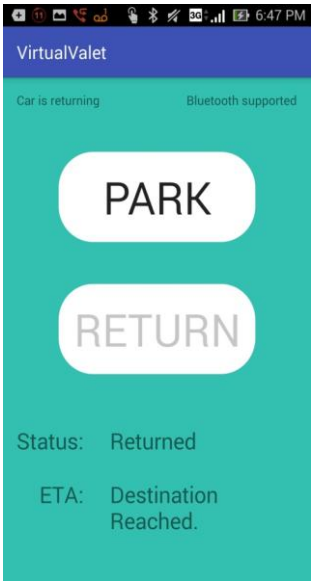


**Figure 23: Vehicle has Returned**

- ROS Node
  - The ROS Node running on the SBC communicates with the Android app by advertising a Bluetooth service to which the app can subscribe. All this is done through an RFCOMM socket port. Once this connection is established, the node starts two threads for sending and receiving data. These two threads have two ROS topics associated with them which the other nodes can use to push and pull data via Bluetooth. When the system is being shut down, an interrupt handler is called by the ROS node so that the ports can be closed properly before exit. Not doing so affects the ability of the system to reestablish a Bluetooth connection.

### 8.2.2 Mobile Platform

The Oculus Prime platform, depicted in Figure 24: Oculus Prime Platform, is the most suitable platform for the project needs. The platform is developed by a Canada-based company, Xaxxon, and is primarily used by the ROS community for surveillance related applications. The platform is made of ABS plastic with mounts for the Xtion, Microsoft LifeCam, four motors and external peripherals (Spotlights, Speakers etc.).

**Figure 24: Oculus Prime Platform**

The Oculus is powered by a 5000 mAh battery and a dedicated power management unit which supports onboard charging and voltage sensing. A charging dock is used to charge the battery without removing it from the chassis. The motor control board- MALG (Motors Audio Lights Gyro), is an ATMEGA 328 based microcontroller. The MALG is powered through the power management unit and serves as the actuator control board.

The assembly instructions in the Documentation for the Oculus Prime were followed to assemble the platform and the docking station. Later, the camera and the Kinect were mounted. The electrical integration of the platform needed customization according to the hardware. This involved the following major tasks-

- Increasing wire gauge and re-soldering existing joints on the platform
- Adding external voltage regulators for the 5V voltage level to power high current consuming devices like the SBC.
- Connecting power and data lines for peripherals like the motors, encoders, SBC, HD LifeCam, actuator control board, power distribution board, Kinect, USB hub and other components.
- Mounting the electronics using 3D printed mounts (Figure 25: 3D Printed SBC Mount and Figure 26: 3D Printed IR Mount), Velcro, etc.
- Integrating the proximity detection PCB.
- Adding external power supply to USB hubs to prevent overloading on a USB port.

**Figure 25: 3D Printed SBC Mount**

**Figure 26: 3D Printed IR Mount**

### 8.2.3 Single Board Computer/ Decision Unit

The SBC runs the nodes for all the other subsystems as well as a central, decision-making node (Figure 27). The decision unit is responsible for managing all the data flowing between various subsystems. It is important to keep a track of current system status, user requests, obstacles in the environment, as well as other tasks. The central node is responsible for fetching, processing, and making decisions based on all of this data. Commands for locomotion are also issued by this node.

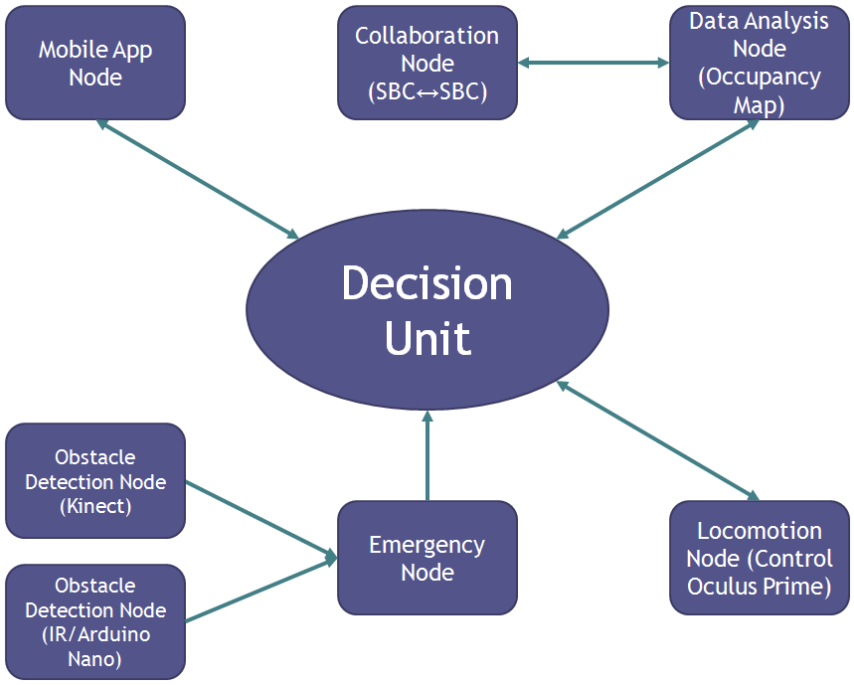


**Figure 27: Decision Unit**

As per the current routine, the decision unit gets triggered by a user command sent via the Android app. This command gets relayed to the decision unit through the Mobile App node. If a Park command is received, the platform sends a request to nearby platforms, asking for a destination. This takes place through the Collaboration Node. Once this data is received, it gets processed and final destination for the platform is calculated. This is then transmitted to the Locomotion Node as a waypoint. The Locomotion Node is connected to the Oculus Prime server and is able to control the platform. The Locomotion Node also keeps a track of whether or not the platform has reached its destination. When it has reached its destination, it publishing

this data back to the decision unit so that it can be sent back to the app through the Mobile App node.

The decision unit has a thread running in parallel to all of this which accepts data related to the presence of obstacles so that the platform can be stopped if a close range obstacle is detected.

### 8.2.4 Actuator Control Board

The MALG takes in serial data from the Oculus Prime server node based on a fixed protocol. Depending upon the data received, the MALG can carry out the following functions:

1. Move the platform by a given distance.
2. Turn on/off lights, speakers etc.
3. Fuse data from encoders and gyro to relay accurate odometry data.

There were a couple of problems with the MALG that required a custom board to be made. The first problem arose from a functional requirement (MF.11) that of the platform to be non-holonomic in nature. The default firmware is such that the platform can execute only one-dimensional motions and thus reach different positions only by turning on its axis. The firmware was modified to include "arc-turns," which better simulate the turning of a car. However, the minimum turning radius that can be achieved by these stock motors is about 0.6m.

The second problem arose when one of the USB hubs malfunctioned and took out the MALG connected to its USB port. The MALG, being an ATMega 328 based board, was replaced by a custom-made board (Figure 28 and Figure 29) and a motor driver (Figure 30), which mimic the functionality of the MALG, barring the external gyroscope.

**Figure 28: Front of Custom-Made Board**

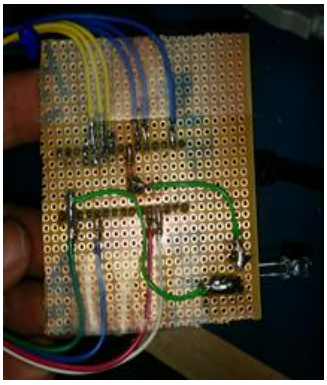

**Figure 29: Back of Custom-Made Board**

**Figure 30: Motor Driver**

### 8.2.5 Localization and Navigation:

The locomotion node communicates with the Oculus Prime server running on the SBC via TELNET. It functions include feeding waypoints to the platform, getting odometry data, and bringing the platform to a halt. The coordinates for locomotion and the command to stop or resume are received from the decision unit.

The node implements its functionality through a state machine. Whenever a command to move to a new point is received, the state machine enters a "Go-To" state. In this state, the platform receives a command to move to a specified destination. The locomotion node actively tracks the state of the system by getting odometry readings. Once the platform reaches its destination, the node publishes a message to decision unit, confirming successful locomotion and goes in "Waiting" state.

While in motion, if a command to stop is received by the locomotion node, it goes to "Emergency" state, which makes the platform stop. Later, when the emergency is cleared, the locomotion uses odometry readings to calculate the new waypoint in relation to the present position. This ensures that the platform continues to its original destination.

### 8.2.6 Collaboration and Communication:

The collaboration node running on the SBC uses DigiMesh XBee adapters to exchange serial data over 900MHz to collaborate with other platforms. The architecture for the system is depicted in Figure 31: Collaboration Architecture.

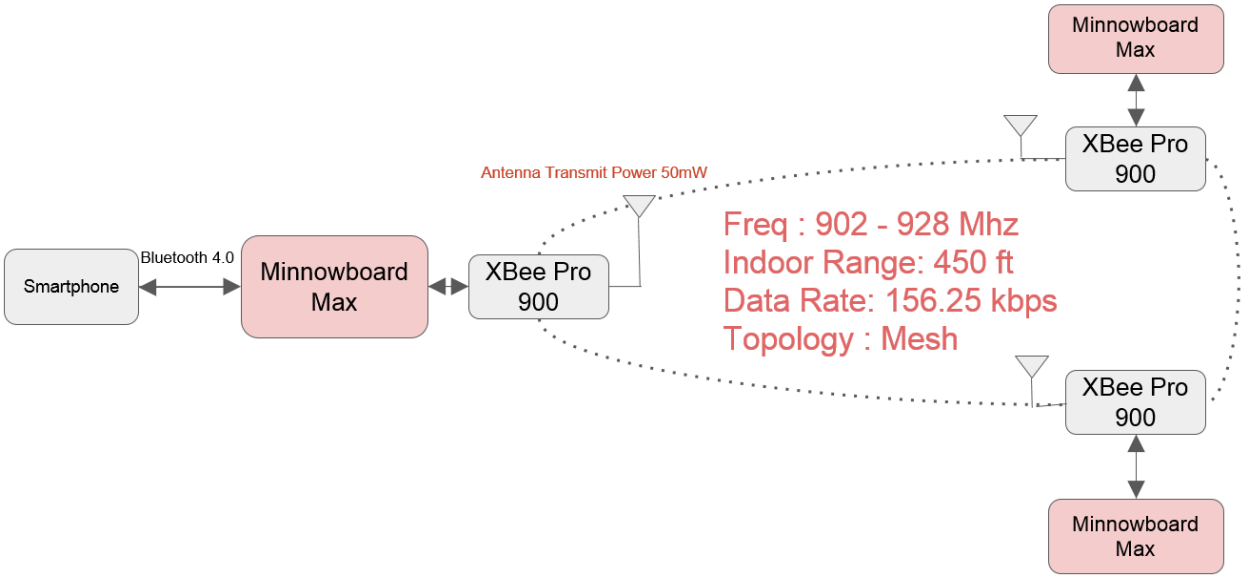


**Figure 31: Collaboration Architecture**

The data packets are encoded using a custom format that includes vehicle ID, data length, and checksum (Figure 32 and Figure 33), which is used to determine the origin and ensure the integrity of the data. The current data being sent is the occupancy map of the environment. This is a 5x5 grid, which the receiving platform can parse and find an empty spot to park. Two bits are used to calculate the state of each cell in the grid.

The collaboration node must be running on multiple SBCs for it to be used. This node runs two threads in parallel in order to send and receive data through the serial port. The received data is parsed to find the appropriate waypoint for the locomotion node. Once this waypoint is calculated, this data is published to the appropriate topic.

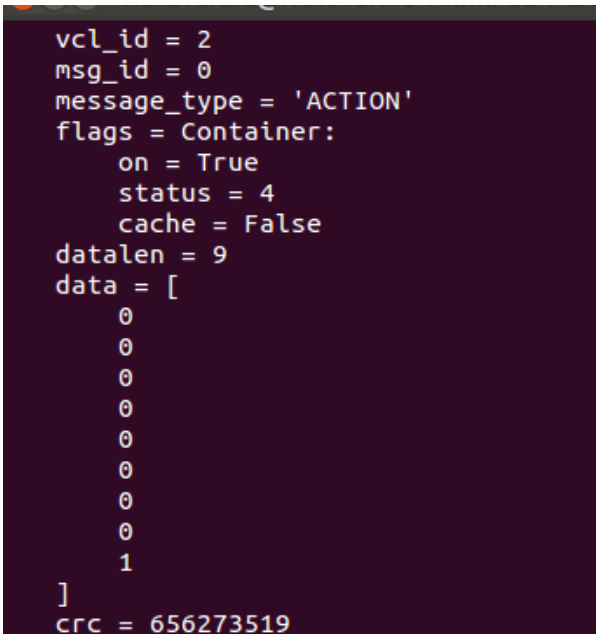


**Figure 32: Action Message**

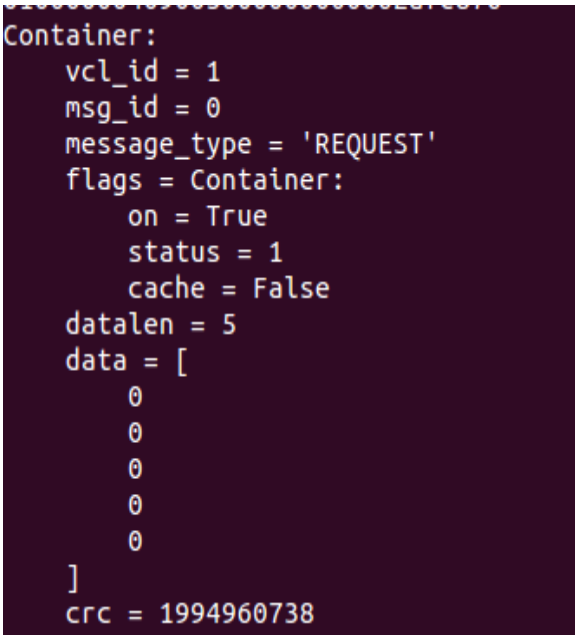


**Figure 33: Request Message**

### 8.2.7 Perception

Perception is used to detect obstacles so that the mobile platform will avoid any collisions. It is split into two types: the MS Kinect detects cylindrical obstacles that are in the range of 0.5 to 1.5 meters and Sharp IR sensors detect obstacles closer than 0.5 meters. This ensures that the vehicle has enough time to come to a complete stop regardless of the proximity of the obstacle.

**Obstacle Detection Using Kinect**

The Point Cloud Library (PCL) is used to detect cylinders of 10-50 cm height and 10-120 cm diameter. The point cloud data is segmented in the form of planes and cylinders. The algorithm implemented uses plane fitting of a cylinder model and RANSAC for outlier rejection. The parameters of these functions are varied in order to obtain an accurate segmentation of cylindrical objects. The objective of this node is to publish the detection of an obstacle on an emergency topic. The algorithm is shown in Figure 34.

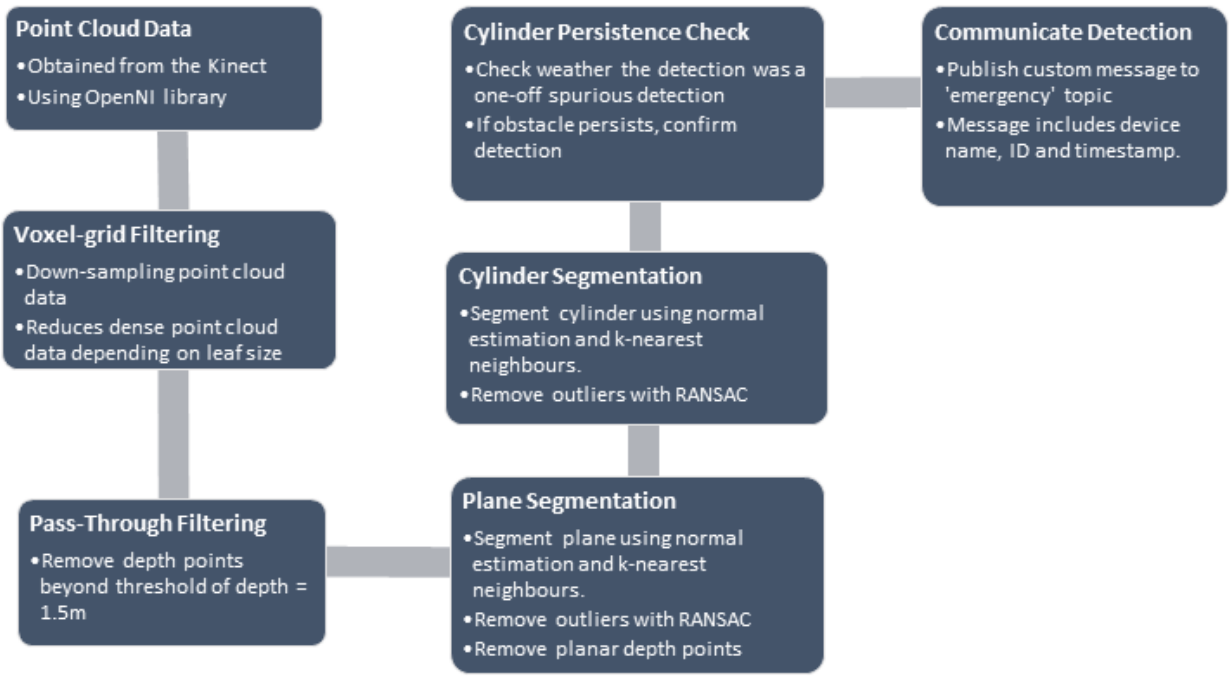


**Figure 34: Obstacle Detection Algorithm**

When the algorithm is applied to raw data from an initial point cloud data (Figure 35:), it produces a segmented image of a cylinder (Figure 36).

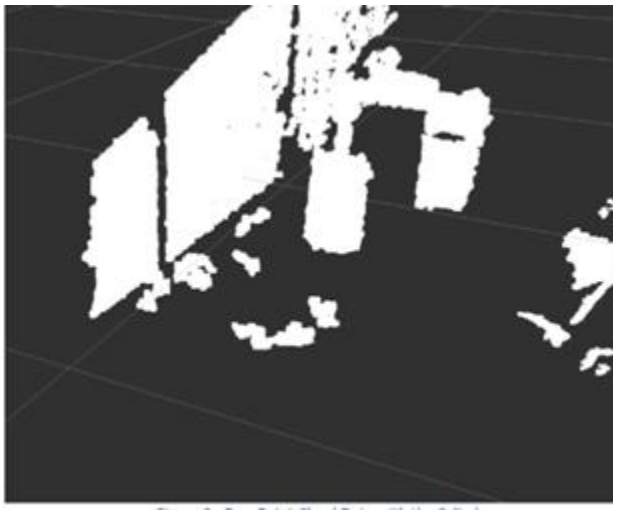

**Figure 35: Raw Data**

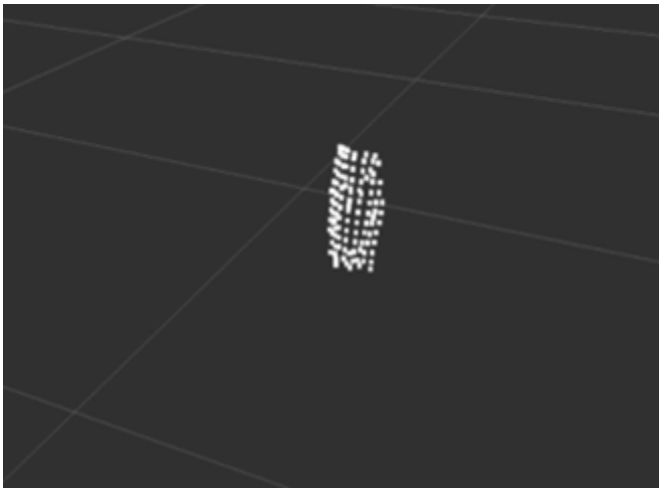

**Figure 36: Segmented Cylinder**

## Obstacle Detection Using IRs

A dedicated proximity detection subsystem is used to prevent collisions with close-range obstacles. The system is implemented in the form of a plug-and-play PCB with sensors and a microcontroller to interpret the readings of the sensors. The system is designed to detect obstacles that are within 50 cm of the platform and cannot be detected using point-cloud data

from the Kinect. Any obstacle less than 20 cm causes an emergency to be declared in the system, making the locomotion come to a complete halt. Three IR sensors are mounted on the front of the platform (Figure 37). A dedicated Printed Circuit Board (PCB) was also designed (Figure 38) to integrate the IR sensors within the system. The PCB houses an Arduino Nano, voltage regulation unit, and connectors for a power supply and three IR sensors (Figure 39). Open headers are present on the PCB for debugging and the addition of new peripherals.

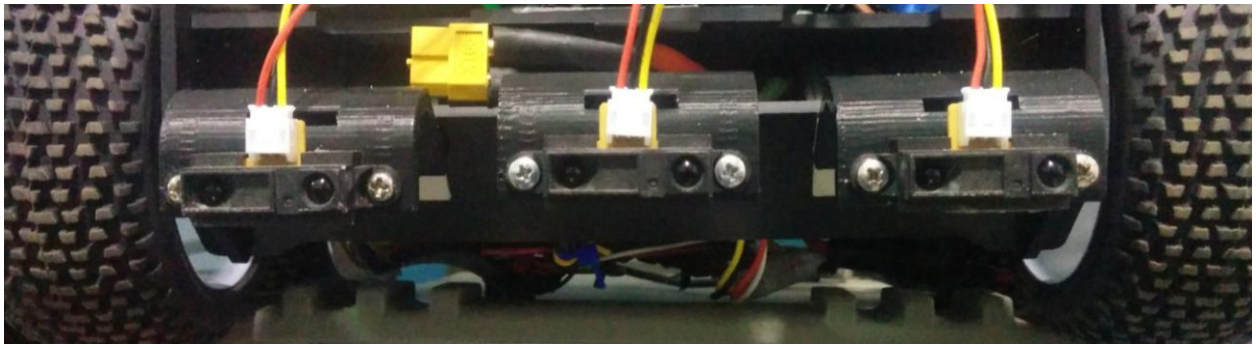

**Figure 37: Mounted IR Sensors**

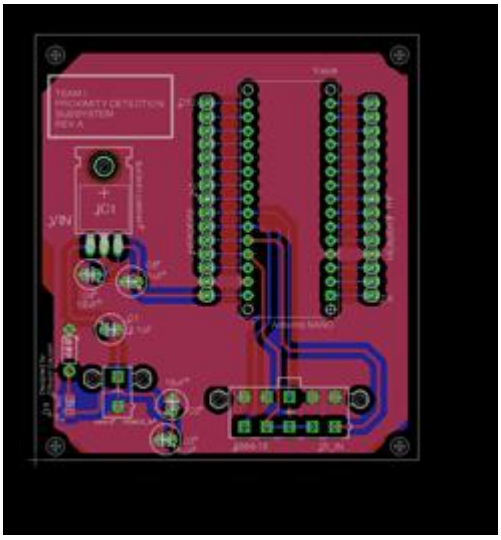

**Figure 38: PCB Design**


**Figure 39: PCB on Mobile Platform**

The Arduino operates as a ROS node and interfaces with the IR sensors to publish time-stamped range data on a ROS topic. This involves polling the three IR sensors and publishing the range data if the range is less than 50 cm. A separate "emergency" ROS node running on the SBC interprets range data and publishes the current state of the emergency. Figure 40 depicts the time-stamped range data being published on the left terminal window and the corresponding emergency state on the right. An emergency state of "0" indicates that no emergency has been detected and the system can continue normal operation. When a "1" is published on the "emergencyState" topic, the system is in a state of emergency and needs to stop immediately.

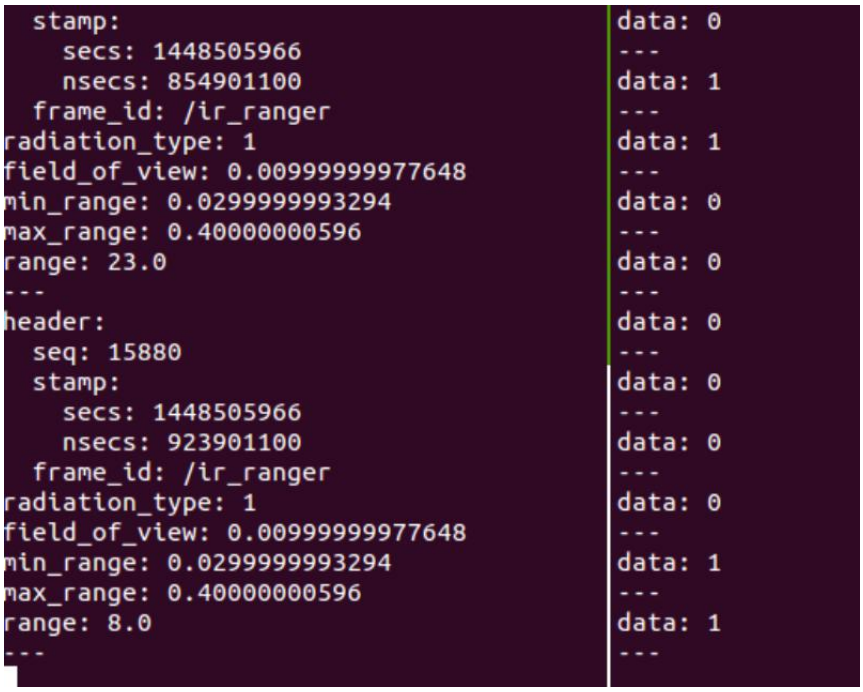


**Figure 40: Test Results from Proximity Detection Subsystem**

### 8.2.8 Visualization

A GUI was created to easily visualize and actively track the state of the occupancy map. Various elements in the environment such as the mobile platform, origin, destination, obstacles, etc., when visualized through a graphical user interface, can aid in achieving a better understanding of the working of the overall system. This is done by rendering a 5x5 grid and overlaying different shapes on top to signify the origin, destination, and current location of the platform. Once data is received from an occupancy map, the program goes through it and creates an image that represents the current state of the system. OpenCV is used to do all of this in Python.

In Figure 41, the platform original is shown as a blue square and the destination is shown as a green square. The estimated location of the platform itself is depicted as a gray circle.

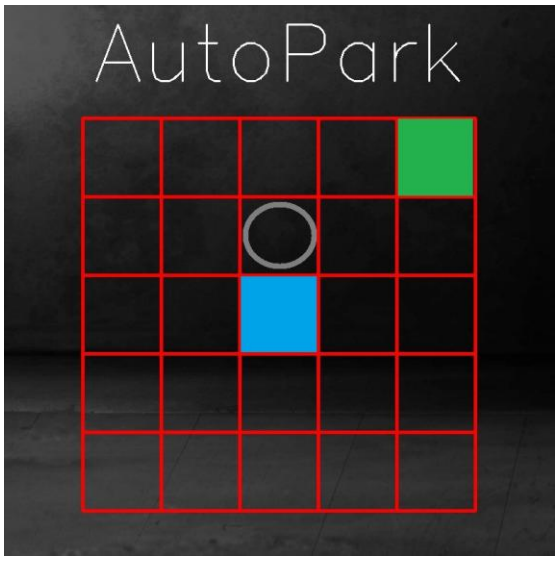


**Figure 41: Occupancy Map GUI**

## 8.3 Testing

The following is a list of step-by-step testing and analysis for different subsystems:

### 8.3.1 Android App

- Android app is able to establish a reliable serial connection to single board computer via Bluetooth with repeatability.
- Sending and receiving of commands with ROS node was tested to work without failure.
- Testing with garbage commands being sent, like letters or anything outside of the defined protocol, to ensure that an incorrect response was not elicited from the ROS node and/or app.
- Testing with untimely data sent to the app such as sending "Parked" status when it was "Returning" to ensure the system does not fail and an incorrect response is not produced.
- App GUI tested to ensure that disabled buttons cannot be pressed and the status of the vehicle changes as required.

### 8.3.2 Communication

The following tests were performed using two XBee Pro 900 adaptors:

- **Range Test**: 100% of messages received in 15 feet range.
- **Speed and Latency Test**:
  Test – Time stamped 100-element integer list (502 bytes) sent serially.
  Result – Transmission time of 1.8 seconds observed.
- **Establishing Connection**: Connection is established instantaneously without the need for authentication.

### 8.3.3 Obstacle Detection

- IR Sensors
  - Ground truth testing of range data to ensure reliable values.

    - Testing that emergencies are published to emergency node reliably and without fluctuations.
    - Testing that spurious detections do not induce emergency state.
    - Testing with the mobile platform.
- MS Kinect
    - Testing of correct setup of libraries on the single board computer by visualizing point cloud data using RVIZ
    - Cylinder segmentation and detection occurs with repeatability.
    - Emergency messages published on emergency topic when a cylinder is detected in specified range reliably.
    - Emergency message does not fluctuate randomly
    - Spurious detections do not cause emergency message to be published

### 8.3.4 Mobile Platform Locomotion

- Ensuring mobile platform is fully functional on software and hardware end via teleoperation commands sent through HTTP interface.
- Testing of Oculus Prime Telnet API by teleoperation through Telnet command interface.
- Testing whether odometry is functional through Telnet command interface.
- Testing locomotion state machine rigorously via mock data on ROS Publishers

### 8.3.5 Subsystem Integration

Subsystem integration testing was done in two sections. The first subsystems that were integrated were the two communication subsystems: Bluetooth and XBees (Figure 42: App and XBee Subsystem Integration). To begin, the Park button on the Android app was pressed. This sent a command over Bluetooth to the SBC. The SBC sent a destination request from attached XBee to a secondary XBee on a secondary SBC. The secondary XBee responded with an occupancy map, which was sent to the simulated mobile platform. The mobile platform parses the map to determine the appropriate destination. Additionally, a status update was sent over Bluetooth to the app. A response from the mobile platform was simulated to indicate that it had reached its destination, which was also sent to the app in a status update. The sequence was then repeated but in terms of Return instead of Park.

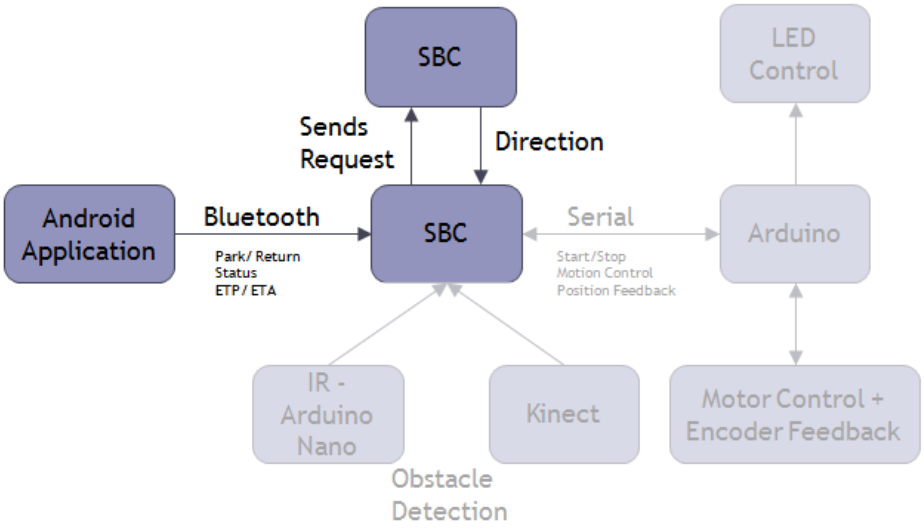


**Figure 42: App and XBee Subsystem Integration**

The second subsystems that were integrated were mobile platform and obstacle avoidance (Figure 43: Mobile Platform and Obstacle Avoidance Subsystem Integration). The mobile platform began locomotion to a fixed location. The Kinect detected an obstacle and notified the platform, which stopped moving. When the obstacle was removed, the Kinect notified the platform, which resumed locomotion towards the original location. The same test was done to test the IRs.

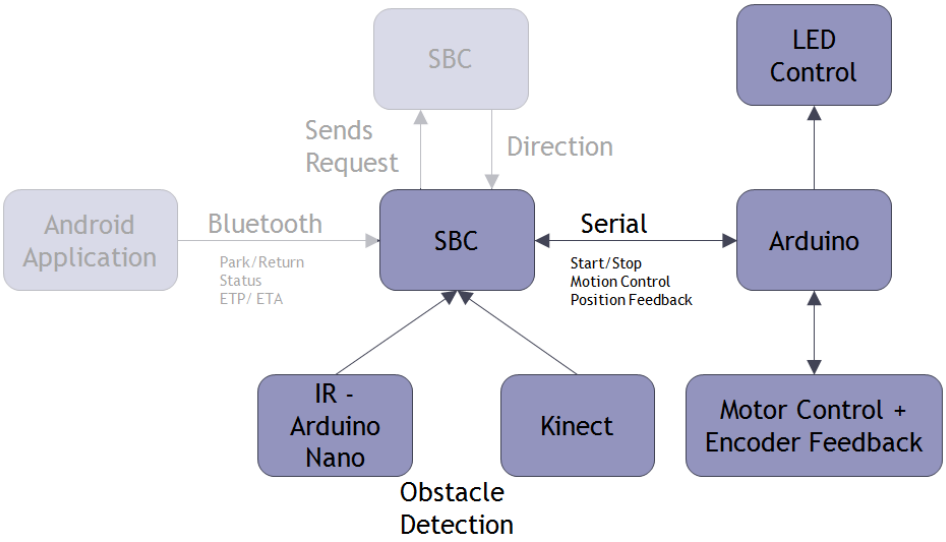


**Figure 43: Mobile Platform and Obstacle Avoidance Subsystem Integration**

### 8.3.6 Final System Integration

The final system was tested using all systems, as seen in Figure 44: Full System Integration. In the same manner described in the communication portion of the subsystem integration (Section 8.3.5), the app was used to initiate the park sequence. A secondary XBee provided an occupancy map to the mobile platform, which parses the map to determine the

appropriate destination. While traveling to the destination, an obstacle was detected, as described in the mobile platform and obstacle detection portion of the subsystem integration (Section 8.3.5). The vehicle stopped until the obstacle was removed, at which time the vehicle continued to the original destination. Once the vehicle reached the destination, the process was repeated for the return sequence.

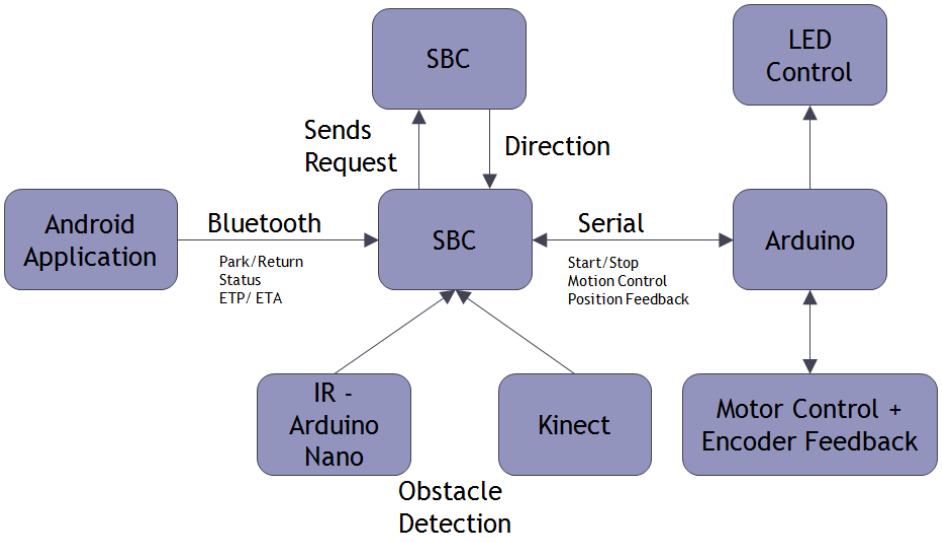


**Figure 44: Full System Integration**

## 8.4 Performance Evaluation against the Fall Validation Experiment (FVE)

### 8.4.1 FVE Demo 1

**Table 6: FVE Demo 1 Tasks and Performance**

| Task | Success Criteria | Performance in FVE |
|---|---|---|
| "Park" command sent to the mobile platform using the Android app | LED Blinks to indicate command received | Task completed successfully at FVE Encore |
| Vehicle begins locomotion upon receiving the command | Vehicle Moves forward | Task completed successfully at FVE Encore |
| Return" command sent to the mobile platform via the Android app | Vehicle Moves backward | Task completed successfully at FVE Encore |

### 8.4.2 FVE Demo 2

**Table 7: FVE Demo 2 Tasks and Performance**

| Task | Success Criteria | Performance in FVE |
|---|---|---|
| "Park" command sent to the mobile platform using the Android app. | LED Blinks to indicate command received | Task completed successfully at FVE Encore |
| Navigation Direction set on second SBC | Platform moves according to set direction | Collaboration using XBees working as a complete subsystem |
| Platform navigates to the second position | Vehicle moves forward/backward | Not demonstrated as it is dependent on XBee collaboration |
| While navigating to the spot, the vehicle will encounter an obstacle and stop within a safe distance of the obstacle | Vehicle does not collide with obstacle | Task completed successfully at FVE Encore |

### 8.4.3 Performance Evaluation with respect to Function and Performance Requirements

- Functional Requirements Validated at FVE
  - **Receive commands from user via smartphone app (MF.1)**
    - The requirement was successfully validated at FVE Encore by demonstrating ROS framework to accept commands from smartphone app via Bluetooth. Upon receiving "Park" or "Return" command successfully, an indicator LED blinks
  - **Share data with other cars (MF.2)**
    - Bidirectional communication with multiple XBees in a scalable network with mesh topology was demonstrated to work successfully as a subsystem.
  - **Sense the environment (static obstacles) (MF.8)**
    - Obstacle detection using infrared sensors and MS Kinect demonstrated successfully in FVE Encore.
  - **Navigate through parking lot (MF.10)**
    - Mobile platform successfully navigates to different locations in the parking lot, stops when an emergency is detected and resumes locomotion when emergency gets cleared.
- Performance Requirements Validated at FVE
  - **90% of messages are received**
    - Requirement validated successfully as the connection is established instantaneously using XBee Pro 900 adapters without the need for authentication.

- **Be able to handle collaboration between 2 vehicles**
  - Requirement validated successfully as bidirectional communication between two different XBees was successfully achieved and network is scalable within 450ft indoor range
- **Detect obstacles within 20 cm of vehicle**
  - Requirement validated successfully in FVE Encore shown by mobile platform stopping when an obstacle is within 20 cm. of the platform.
- **Detect obstacles 10-50 cm high and 10-120 cm wide**.
  - Requirement validated successfully and visualized for detection of cylindrical obstacles using RViz (a 3D visualizer for displaying sensor data and state information from ROS).

## 8.5 Conclusions

### 8.5.1 Strong Points

One of the greatest strengths of the system is that all subsystems have partial functionality and there is a basis for building the project work in the spring semester. It is anticipated that this will speed up implementation and testing. Listed below are the subsystems and advantages:

- Android app is functional and will enable efficient testing
- Serial protocol for communication is defined and tested for long term goals in the multi-vehicle scenario
- Obstacle detection with IR and vision system provides basis for avoidance algorithm
- Oculus Prime mobile platform provides versatile capabilities and its ROS packages will aid in developing the localization and planning subsystem.

### 8.5.2 Weak Points

The system has many single points of failure, specifically in terms of hardware. Moreover, the dependency on Oculus Prime custom hardware; MALG PCB and Power distribution board is a risk and needs proper mitigation plans. Another weak point is that integration of subsystems needed more time than originally anticipated. This lead to only partial subsystem integration shown at the Fall Validation Experiment.

There have been issues selecting the most appropriate Single Board Computer (SBC) for the project. The initially selected Odroid-XU4 was found to be incompatible with the mobile platform. The MinnowBoard Max was then provided by Team C. However, there is still some uncertainty regarding the suitability of its processing power considering the complexity of the system. Elements that need refinement are:

- Bluetooth connection with the mobile app needs to be made less prone to failure
- Collaboration algorithm needs to be tested for a scaled system with multiple vehicles
- Precise locomotion needs to be implemented using odometry techniques

# 9 Project Management

## 9.1 Work Breakdown Structure

The work breakdown structure (WBS) for the fall semester (Figure 45: Fall Work Breakdown Structure) depicts the subsystems that need to be developed, the tasks involved in their accomplishment, and system-level and subsystem-level testing that needs to be conducted. Each subsystem highlighted in the cyber-physical architecture has certain key tasks that need to be accomplished and modules that need to be developed. Further, it is imperative to conduct subsystem testing and validation experiments to ensure seamless integration. Within the structure, green indicates that the task has been complete, yellow indicates that it is in progress, and red indicates that it has not been started.

The majority of the tasks for the fall have been complete. What remains is to have the Kinect working on the MinnowBoard Max. Currently, it works when it is run on a laptop, but not on the SBC. Additionally, the Bluetooth connection between the app and the SBC needs to be perfected in order to smoothly shut down the connection. Currently, the connection does not completely close, leaving the port open. This makes it impossible to reestablish communication on the same port. The XBee network needs to be made scalable to accommodate multiple XBees. The final section of the WBS is Project Management, which contains ongoing tasks that must be worked on throughout the project, such as Risk Management, Weekly Sprints, and updating the Kanban Board.

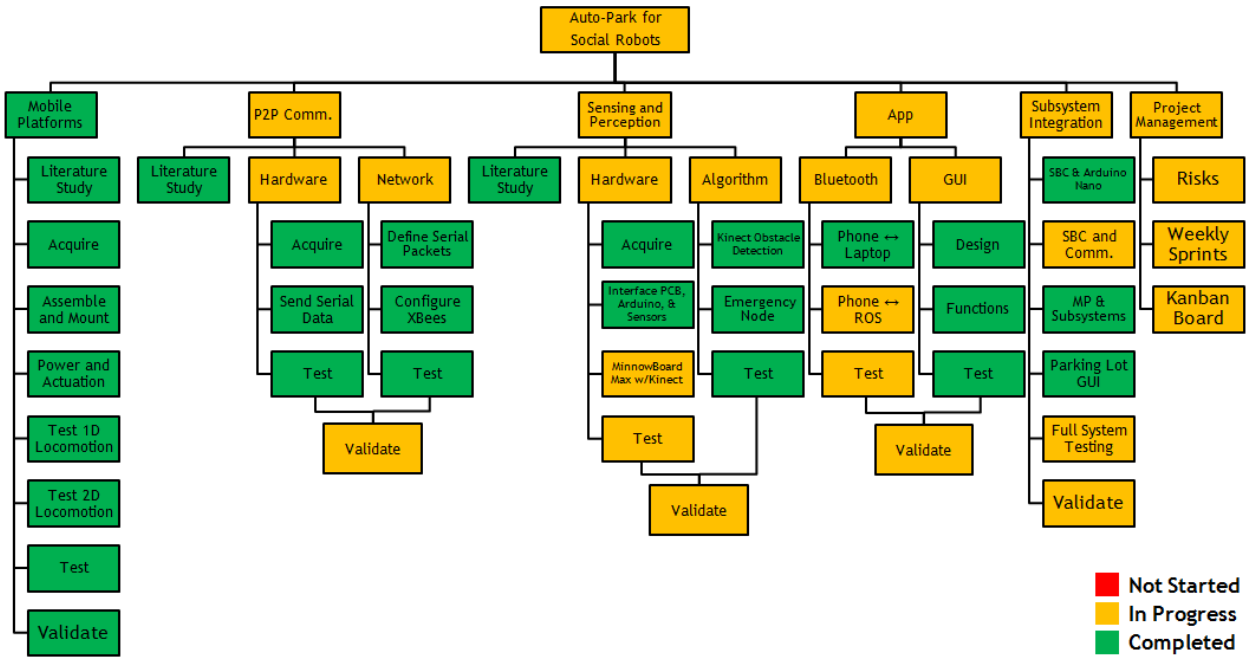


**Figure 45: Fall Work Breakdown Structure**

## 9.2 Schedule

The major system development milestones that need to be attained during the spring semester are:

1. Scale the collaboration subsystem to work for multiple platforms

2. Implement a robust navigation subsystem to move the platform in a precise manner
3. Implement path planning to calculate shortest routes and obstacle avoidance
4. Integration of all the subsystems

Currently, the project is slightly behind schedule as the work for the fall semester has not been completely wrapped up. Once that is complete, the schedule will be back on track and the team will be in a position to resume the work on Navigation, Path Planning and Collaboration subsystems that is planned for the spring semester. The spring schedule can be seen in Table 8.

**Table 8: Spring Schedule**

| Timeline (Spring 2016) | Goals | Milestones |
|---|---|---|
| 01/11 - 01/25 | App is complete in all respects; Repetitive and precise locomotion control; Manufacture or acquire more platforms | Completion of Mobile App Subsystem |
| PR 7: Late January | Multiple partially functional platforms; Fully functional mobile app | |
| 01/25 - 02/08 | Navigation works for multiple waypoints; Literature survey for path planning is complete; XBee Mesh network tested with multiple platforms | |
| PR 8: Mid-February | Mesh Network functional for multiple platforms; Demonstrate robust and accurate navigation of platforms | Completion of Communication/ Collaboration Subsystem |
| 02/08 - 02/22 | Create a graphical user interface to be used for testing and demonstrations; Implement obstacle avoidance | |
| PR 9: Late February | Platform travels in 2D from Point A to Point B while avoiding obstacles; | Completion of Navigation Subsystem |
| 02/22 - 03/07 | Design Mock Parking Lot; Make the GUI fully functional and depict the existing setup; Create a path planning node and integrate with navigation subsystem | |

| Timeline (Spring 2016) | Goals | Milestones |
|---|---|---|
| PR 10: Mid-March | Platform localizes and navigates in mock parking lot; GUI gets updated and shows real-time information | Completion of Path Planning Subsystem |
| 03/07 - 03/21 | Create dummy platforms to collaborate with; Extensively test navigation and path planning; Create a point cloud map of the parking lot; Make active platforms collaborate | |
| 03/21 - 04/04 | Make the platform park in and exit spot; Start integration of all the subsystems | Start integration |
| PR 11: Early April | Single platform collaborates with dummy platforms and parks itself; Multiple active platforms are aware of each other's presence and avoid collision | |
| 04/04 - 04/18 | Testing and integration | Wrap-up integration |
| PR 12: Mid-April | Testing | |

## 9.3 Spring Test Plans

### 9.3.1 Capability Milestones

1. **Sensor**
    a. **Test**: Map a known environment and note the discrepancies between the actual and logged data. Note the resolution of the map.
    **Requirement Validated**: MF.8
    **Milestone(s)**: PR11
2. **Communication**
    a. Parking Spot Matrix
    b. **Test**: Transfer a multidimensional array representing the parking spots in the lot between the single board computers and populate it with relevant data. All the spots should be in the occupied, reserved, or free state. Make some changes to the system and log how that affects matrix. Ensure that the changes made on one board are carried over to the other boards.
    **Requirement(s) Validated**: MF.2, DF.1
    **Milestone(s)**: PR8, PR9
3. **Android App Interface**
    a. **Car to Phone – Vehicle Status and Notification**
        i. **Test**: Press "Park" from the Android app. The vehicle moves towards a parking spot and the app displays the status "Parking". When the vehicle

is parked in a spot, the app displays the status "Parked". Press "Return" from the Android app. The vehicle exits the parking spot and moves towards the exit and the app displays the status "Returning". When the vehicle stopped at the exit, the app displays the status "Returned" and sends a notification to the user.
**Requirement(s) Validated**: MN.1

b. **Android App Prevents User Error**
    i. **Test**: The user can only press "Park" when the vehicle status displays "Waiting" on the app. The user can only press "Return" when the vehicle status displays "Parked" on the app.
    **Requirement(s) Validated**: MN.4, MN.1
    **Milestone(s)**: PR7

4. **Parking Spot – Entering and Exiting**
    a. **Test 1**: Send the car to a spot with both adjacent spots occupied. The car backs into space within two tries. When the car is parked, it is 100% within the parking spot boundaries and within 35º of parallel to both the neighboring cars. Send the car to the exit. The car exits the spot within two attempts without coming into contact with nearby cars or the infrastructure.
    b. **Test 2**: Send the car to a spot with one adjacent spot occupied. The car backs into space within two tries. When the car is parked, it is 100% within the parking spot boundaries and within 35º of parallel to the neighboring car on one side and the parking spot line on the other side. Send the car to the exit. The car exits the spot within two attempts without coming into contact with nearby cars or the infrastructure.
    c. **Test 3**: Send the car to a spot with both adjacent spots unoccupied. The car backs into space within two tries. When the car is parked, it is 100% within the parking spot boundaries and within 35º of parallel to the parking spot boundaries. Send the car to the exit. The car exits the spot within two attempts without coming into contact with nearby cars or the infrastructure.
    d. **Requirement(s) Validated**: MF.6, MF.7, MN.3, MN.4, DN.3
    **Milestone(s)**: PR11
5. **Mapping**
    a. Place markers at specific intervals
        i. **Test**: Place markers at predefined locations. Map the area. Compare their position in the map to their actual physical location and ensure it is correct.
        **Requirement(s) Validated**: MF.8
    b. Ensure map can be loaded into vehicles
        i. **Test**: Upload the map generated to the SBC and visualize with RViz.
        **Requirement(s) Validated**: MF.8
        **Milestone(s)**: PR11
6. **Path Planning – From Entrance to Parking Spot to Exit**
    a. **Test 1**: Place the vehicle at the entrance of the parking lot. Have the vehicle plan a path to a spot and ensure that this is the most optimal path.

b. **Test 2**: Place the vehicle at the entrance of the parking lot. Have the vehicle plan a path to a spot. Then, introduce an obstacle in the path and see how the path gets altered. Ensure that the new path is the most optimal
**Requirement(s) Validated**: MF.4, MF.5, DF.2, DF.3, MN.4, DN.2
**Milestone(s)**: PR9

7. **Obstacle Detection**
    a. **Send and Receive Movement Information**
        i. **Test 1**: Send a Return command to a vehicle. Ensure that the vehicle publishes a message to other vehicles on the network that it is exiting the parking spot. Nearby vehicles that are parking will yield the right of way to the exiting vehicle, allowing it enough space to exit the spot safely and efficiently.
        ii. **Test 2**: When a vehicle is parking, nearby vehicles maintain a safe distance to avoid a collision and give the parking vehicle adequate space to park safely and efficiently.
        **Requirement(s) Validated**: MF.2
    b. **Maintain Safe Driving Distance**
        i. **Test**: Send two vehicles to park in neighboring spots in quick succession. The first vehicle will maintain a constant speed. The second vehicle will maintain a safe distance between itself and the first vehicle. When parking, the second vehicle will stop at a safe distance so as not to interfere with the first vehicle parking.
        **Requirement(s) Validated**: DF.4
        **Milestone(s)**: PR9

### 9.3.2 Spring Validation Experiment

- **Location**: B Floor, Newell-Simon Hall
- **Date**: April 29, 2016
- **Logistics**: Oculus Prime Platform (3), Kinect (3), SBC (6), XBee Pro DigiMesh Adapter (6), USB 4.0 Bluetooth Adaptor (3), Android Phone with Virtual Valet (3), Monitor Screen, Mock Parking Lot
- **Operating Area**: 10m x 10m (open space on B Floor)

| Task | Success Criteria | Requirements Validated |
|---|---|---|
| "Park" command sent to three mobile platforms in succession using three Android phones with the app | LEDs blink to indicate command received | MN.1 |

| Task | Success Criteria | Requirements Validated |
|---|---|---|
| Mobile platforms will enter the parking lot and collaborate with other vehicles to choose the optimal parking spots (Figure 46: Vehicles Localize Within Parking Lot, Figure 47: , and Figure 48: Optimal Spots Identified) | The three spots closest to the exit are chosen | DF.1 |
| Mobile platforms will navigate along optimal paths to the spot (Figure 49: Shortest Paths to Spots Planned) | The paths with the least amount of time are chosen | DF.2, DF.3, DF.4, DN.2 |
| When a vehicle encounters an obstacle, it will plan a path around it and alert other vehicles of the obstacle location (Figure 50: Route Planned Around Obstacle and Figure 51: Route Followed Around Obstacle) | The platforms will not hit obstacles and will park 100% within their designated parking spot | MF.9, DN.2, DN.3 |
| A notification will be sent from each vehicle to the respective user when their vehicle is parked (Figure 52: Vehicles Park and Wait) | Notification is received by three separate phones | MF.6, MN.1 |
| When each user sends the command to return, the robot will exit the parking spot and navigate towards the exit spot along the optimal path (Figure 53: Return Command Received, Shortest Path to Exit Followed) | No collisions occur and mobile platforms exit as quickly as possible | MF.4, MF.5, MF.7, MN.1, MN.3, MN.4, DN.2 |
| When the vehicles reach the exit, they will each send their user a notification stating that it is at the exit | Apps update accordingly and notifications are received on three separate phones | MN.1, MN.4, |

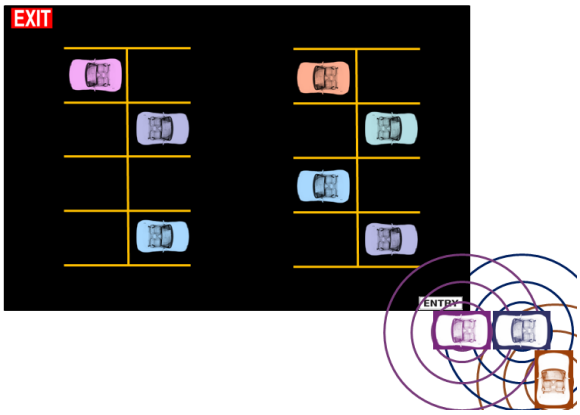


Figure 46: Vehicles Localize Within Parking Lot

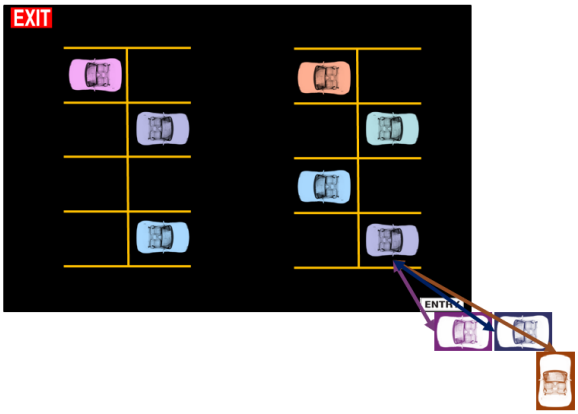


Figure 47: Collaborative Communication Established

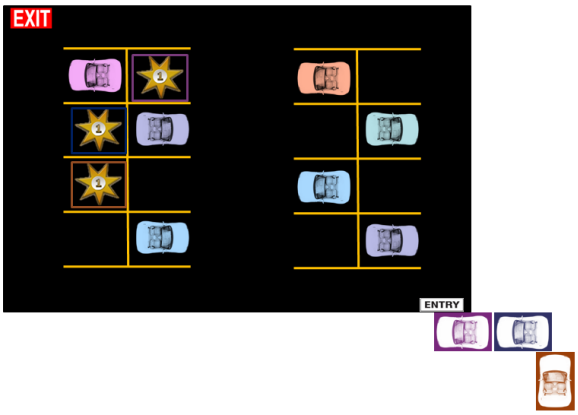


Figure 48: Optimal Spots Identified

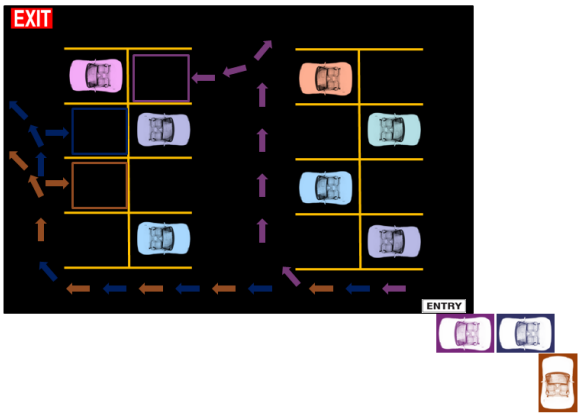


Figure 49: Shortest Paths to Spots Planned

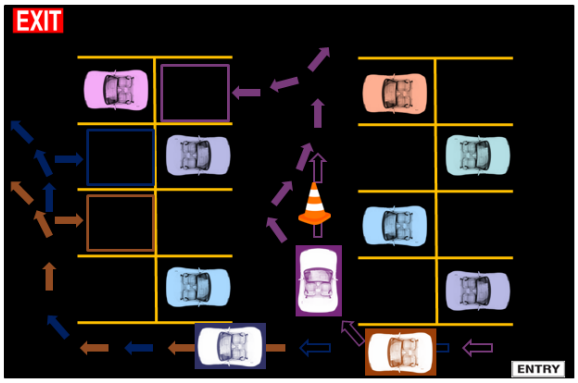


Figure 50: Route Planned Around Obstacle

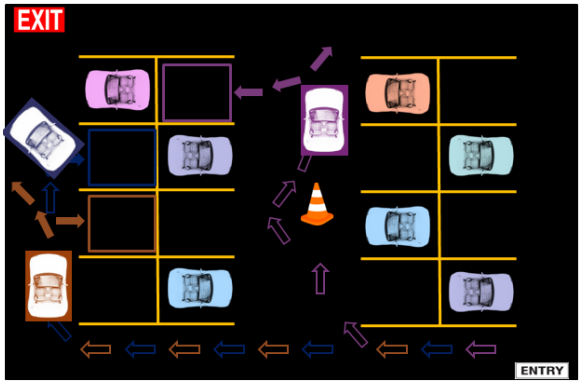


Figure 51: Route Followed Around Obstacle

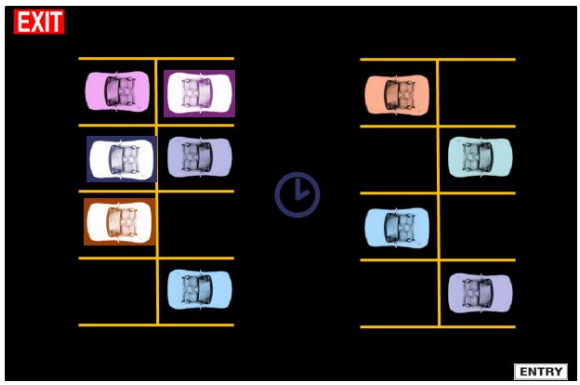


**Figure 52: Vehicles Park and Wait**

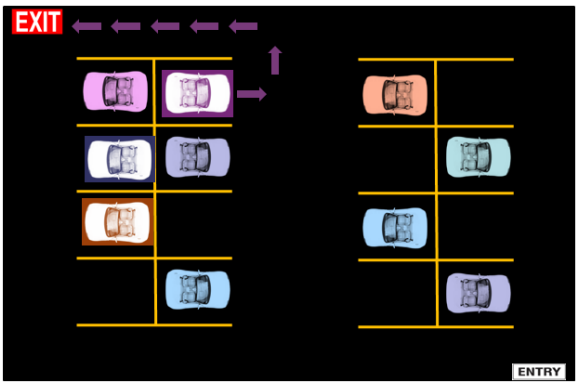


**Figure 53: Return Command Received, Shortest Path to Exit Followed**

Performance Requirements Validated at SVE:

- Maintain a velocity between 0 and 10 cm/sec
- Park 100% within a parking spot within 2 attempts. Be within 35º of parallel with the neighboring vehicles or the lines of the spot, as applicable
- Maintain a distance of 30.48 cm (1 ft.) between vehicle and infrastructure
- Vehicle maintains at least 60.96 cm (2 ft.) between the front of one moving vehicle and the back of another moving vehicle
- Identify optimal spot 98% of the time
- If incorrect spot is chosen, it is ranked within 5% of optimal spot
- Optimal path is chosen 90% of time
- 90% of messages are received
- Exit the spot within 2 attempts without collision
- Will take no more than 45 seconds to exit the parking spot
- The vehicle will arrive at the exit within 90 seconds of receiving the "Return" command
- There will be ZERO changes to the infrastructure
- Budget is within $4000

## 9.4 Budget

**Table 9: Project Budget**

| Part | Part Number | Quantity | Cost/Unit | Current Expense | Total Cost |
|---|---|---|---|---|---|
| Oculus Prime Mobile Platform* | Xaxxon Oculus Prime Kit Version | 3 | $499.00 | $499.00 | $1497.00 |
| ODROID-XU4 | N/A | 3 | $74.00 | $222.00 | $222.00 |
| MinnowBoard Max* | N/A | 3 | $147.00 | $294.00 | $441.00 |
| Arduino Nano v3.0 | N/A | 3 | $15.38 | | $46.14 |
| Infrared Sensor | Sharp GP2Y0A21YK | 9 | $13.95 | | $125.55 |
| MS Kinect Sensor | N/A | 3 | $149.99 | | $448.32 |
| Miscellaneous (mounts, electronics, memory, cables) | N/A | N/A | N/A | $190.99 | $625.99 |
| DigiMesh XBee Pro* | N/A | 6 | $99.00 | $297.00 | $594.00 |
| Total Budget: | | | | $1502.99 (37.57%) | $4000 |
| | | | | | *big ticket items |

The total project budget is $4000. The most expensive items needed for the project are the Oculus Prime Mobile Platforms, the DigiMesh XBee Pro Adapters, and MinnowBoard Max single board computers. Of the total budget, 37.57% has been spent to date.

## 9.5 Risk Management

Since the Preliminary Design Review, updates have been made to risks, listed by their Risk ID below. The complete, updated list of risks can be found in Table 10: Project Risks.

1. **No Mobile Platform**. After waiting for a response from the sponsor, the team finally purchased a mobile platform, the Oculus Prime, without the sponsor. This closed the risk.
2. **Inadequate Mobile Platform**. This risk has been modified to encompass two new areas in which the Oculus Prime may be inadequate. The pre-programmed motions of the Oculus Prime have it turning on its axis. However, the platform should mimic a car in its movements, so this will have to be modified. Additionally, the platform may have some difficulty handling the weight of the devices that have been added to it.
3. **Unsuitable Smartphone Interface**. The status of this risk has changed from "Open" to "In Progress". Because the subsystems are now roughly integrated, the system can be tested through the Virtual Valet app, which causes the app to be tested extensively in a variety of circumstances.
4. **Subsystem Incompatibility**. This risk will stay in progress until the end of the project as subsystem incompatibility poses a risk to the overall robustness and performance capability of the system.
5. **Too Many Requirements**. After an extensive review of the requirements, they have been revised and trimmed as needed to create a more manageable scope for the project. However, this risk remains ongoing.
6. **Inaccurate Parking Lot and Obstacles**. No update has been made to this risk.
7. **ROS Framework**. A new handling strategy has been added to this risk. To ensure timing and message compatibility throughout the ROS nodes, a ROSLaunch file was created that launches all nodes in the correct order.
8. **SBC and Platform Incompatibility**. This risk has been closed. Team C allowed us to use their unneeded MinnowBoard Max, which is 64-bit. The MinnowBoard Max is able to run the Oculus Prime server, which closes this risk.
9. **MinnowBoard Max Processing Power Limitations**. This is a new risk. After integrating the subsystems, the MinnowBoard Max began freezing when running all subsystems. This is most likely due to limitations on the processing power of the MinnowBoard Max. The current plan is to perform in-depth testing to isolate any other potential causes. If no other causes are found, other SBCs with greater processing power will be assessed and purchased.
10. **Closing Bluetooth Port between ROS and App**. This is a new risk. Bluetooth communication occurs between the Virtual Valet app running on an Android phone and the ROS node on the SBC. There has been some difficult with the RFComm ports used to establish the communication. When the app and the node do not shut down gracefully, the port will sometimes stay open, which prevents communication from being opened on that port again. The python script to establish Bluetooth communication has been modified to only connect on one port and to close the port upon receiving "Ctrl+C" or "Ctrl+Z" keystrokes. What remains is to have the app shut down Bluetooth communication gracefully so the port can be reused.

The risk management categories are as follows:

- **ID**: Number used to reference risk
- **Description**: Brief description of the risk
- **Responsible Party**: Indicates who is in charge of handling the risk.

- **Risk Analysis**: The risk analysis has two numbers representing the ranking of the consequence of the risk and the likelihood that the risk will be realized. It is formatted as (Consequence x Likelihood), which can more clearly be seen in Figure 54: Risk IDs Charted with Consequence and Likelihood Levels.
- **Area of Impact**: Technical, Schedule, Cost, Programmatic
- **Handling Strategy**: How the risk will be handled?
- **Status**: Open (no work has been done), In Progress (work is being done to mitigate risk), Closed (no longer a risk)

**Table 10: Project Risks**

| ID | Description | Owner | Risk Analysis | Area of Impact | Handling Strategy | Status |
|---|---|---|---|---|---|---|
| 1 | No Mobile Platform | Mohak | N/A | Technical, Schedule, Cost | Purchased test platform without sponsor | Closed |
| 2 | Inadequate Mobile Platform | Shivam | 3x3 | Technical | Correct turning radius, test weight limits | In Progress |
| 3 | Unsuitable Smartphone Interface | Dorothy | 3x2 | Technical Programmatic | Frequent and extensive testing | In Progress |
| 4 | Subsystem Incompatibility | Pranav | 5x2 | Technical Schedule Cost Programmatic | Research on ROS to ensure compatibility Carry out low-level cross compatibility tests Create an architecture to ease integration | In Progress |
| 5 | Too Many Requirements | Shivam | 4x2 | Schedule Programmatic | Trimmed requirements Began weekly sprints Created Kanban cards | Open |
| 6 | Inaccurate Parking Lot and Obstacles | Richa | 1x1 | Programmatic | Analyze parking lots IRL. Scale to match mobile platform | Open |
| 7 | ROS Related Issues | Pranav | 5x1 | Technical | Arduino↔ROS testing. ROS↔ROS serial communication. Initiate subsystem tests using ROSLaunch files. | In Progress |
| 8 | SBC and Platform Incompatibility | Mohak | N/A | Technical Schedule Cost Programmatic | Use MinnowBoard Max borrowed from Team C | Closed |

| ID | Description | Owner | Risk Analysis | Area of Impact | Handling Strategy | Status |
|---|---|---|---|---|---|---|
| 9 | MinnowBoard Max Processing Power Limitations | Mohak | 5x5 | Technical Programmatic | Test SBC thoroughly to investigate other possible problems. Research alternative, more powerful SBCs | In progress |
| 10 | Closing Bluetooth Port | Dorothy | 2x5 | Technical | Hardcode port number into script. Close port by keystroke on SBC side. Close port when app closes on Android side | In progress |

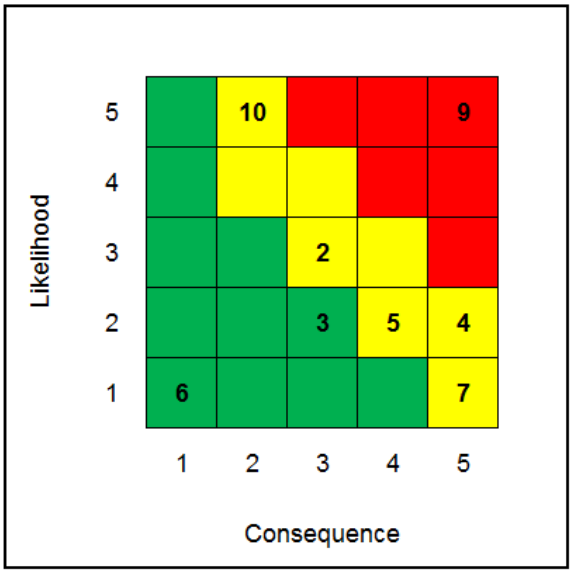


**Figure 54: Risk IDs Charted with Consequence and Likelihood Levels**

# 10 Conclusions

## 10.1 Lessons Learned

- Avoid material and schedule bottlenecks – To avoid bottlenecks and ensure parallel progress in work, dependency in terms of materials and schedule needs to be minimized as much as possible, such that a delay in one task does not hinder progress in another.
- Think ahead and order spares and backups; more backups for high-risk elements – High-risk components in the system need to be identified at an early stage and enough spares and backups should be in place.
- Integrate step-by-step – System integration should be carried out in a step-by-step manner. This provides a way to debug efficiently and ensure subsystem compatibility.

- Rigorously test subsystems – As this is a highly complex system, subsystems should be tested rigorously before starting integration. All possible break cases need to be tested against and it should be ascertained that required performance is being achieved.
- Maintain GitHub repository – The importance of well-organized documentation and version control cannot be emphasized enough.
- Schedule should be based on well-defined metrics – Number of hours to be spent on each task, including buffers, should be used to set the schedule and timeline. Estimates get better with time, but following this approach would ensure realistic goals and avoid schedule slips. Also, schedule slips should be analyzed and corrective measures implemented.
- Risk chart should be updated frequently – Risk chart should be updated on a regular basis to ward off potential high consequence risks ahead of time.

### 10.2 Key Spring Activities

The next step is to scale up the system by acquiring two more Oculus Prime platforms. The SVE will feature three mobile and three stationary vehicles. A mock parking lot will be created and its map generated. Map generation will be a one-time exercise.

Communication subsystem will be scaled up with multiple adapters connected to each other in a mesh topology. Since there is no central server to process the information being exchanged in the network, an algorithm will be developed for real-time handling of this data.

For the navigation subsystem, odometry will be used to facilitate precise vehicle maneuvers required for the project. Obstacle avoidance routine will be incorporated during locomotion and will be tested extensively.

Localization and Planning would be the next key activity. Early in the semester, a literature survey will be done on the potential techniques and algorithms that are suitable for the project in this domain. This will basically involve a localization estimate of the vehicle and updating an occupancy map which will be used for waypoint generation and route planning. Consequently, the best-suited algorithms will be implemented and tested for the accuracy outlined by the performance requirements.

A key goal is to start integration of subsystems as early as possible. A step-by-step approach for integration is the right way to go as it makes debugging much easier and helps in taking corrective measures early on.

## 11 References


1. Stark, John. "Parking Lots." University at Albany, 23 Apr. 2012. Web. 25 Sept. 2015. <http://www.albany.edu/ihi/files/Parking_Lots_Where_Motorists_Become_Pedestrians.pdf>.
2. Fayard, GM. "Work-related Fatal Injuries in Parking Lots, 1993-2002." *National Center for Biotechnology Information*. U.S. National Library of Medicine, 10 Jan. 2008. Web. 25 Sept. 2015. <http://www.ncbi.nlm.nih.gov/pubmed/18325411>.