FINAL PROJECT REPORT

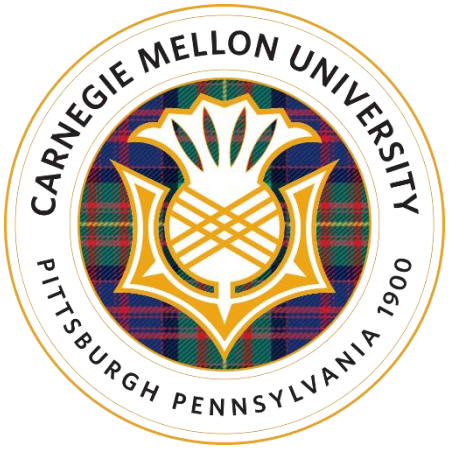
AUTO-PARK FOR SOCIAL ROBOTS

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
Carnegie Mellon University

(Sponsored by United Technologies Research Center)



1. Abstract

Auto-Park for Social Robots is an autonomous and collaborative parking system for self-driving vehicles in an indoor parking lot. This report summarizes the progress made by Team Daedalus as part of the Master of Science in Robotic Systems Development project "Auto-Park for Social Robots" during the academic year 2015/16 at Carnegie Mellon University.

Through this project the team has tried to develop decentralized systems for collaboration between autonomous cars and implement effective planning and scheduling strategies for multiple vehicles. The team aims to tackle the use-case of autonomous parking in parking garages by implementing the collaborative strategies on physical robots and complex planning techniques in software simulation.

The team has identified and fulfilled key functional and nonfunctional requirements of the project as per the performance metrics specified by the team in agreement with their sponsor United Technologies Research Center. In this report, detailed subsystem descriptions and their testing results are documented following which project management, risks and mitigation strategies are discussed. Having gained valuable experience from testing, the team has identified the key lessons learned from this project.

The project was successful in accomplishing the goal of autonomous parking and returning of vehicles, navigating collaboratively in a parking lot- all with the press of a button on a user's smartphone app. The simulation environment was used to park vehicles in a large parking lot with hundreds of vehicles based on approaches like multiple-heuristic goal assignment and goal assignment using the multi-armed-bandit approach. The end-to-end system developed during this project is a contribution towards better and safer self-driving vehicles.

Table of Contents

2. Project 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.

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 two distinct systems:

1) **A physical system** which 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 main focus is on establishing a robust system for effective collaboration between vehicles.

2) **A simulation system** which helps in showcasing the potential benefits of implementing such a collaborative system at a large scale in terms of saving time for the user. Various optimization techniques are employed to efficiently plan and schedule the parking of large number of vehicles with collaboration and data sharing between vehicles as a central theme.

3. Use case

Black Friday, the day after Thanksgiving, is one of the nation's largest shopping days. Customers line up, and sometimes camp, outside stores hours before they open in hopes of getting the best deals on the latest releases. So many people attend that it causes a 63% increase in parking lot traffic (Figure 3.1) (INRIX, 2015). Parking lots are already incredibly dangerous, causing 20% of vehicular accidents. With the increase in traffic and the palpable stress, these numbers can increase dramatically.

Kris has been eyeing the latest game system for a while now and has been waiting for the Black Friday discount to purchase it. The stores open at midnight, so Kris arrives at 11:30 PM. Without the use of the AutoPark system, Kris is forced to search for a parking spot without assistance. Just after entering the lot, another driver cuts Kris off and takes her spot. A few minutes later, Kris is fooled by a small Fiat parked too far forwards in a spot. The next free spot has been blocked by a shopping cart (Figure 3.2). Kris is then nearly hit by a car reversing from a spot. Finally, at 12:17 AM, Kris finds a spot far away from the entrance of the mall and begins the long

walk to the entrance. 47 minutes after driving into the parking lot, she finally enters the store and begins to fight her way towards the gaming section.

Figure 3.1

Figure 3.2

Had Kris used the Virtual Valet app with the AutoPark system installed in her car, she could have entered the parking lot at 12 AM, pulled up to the entrance of the mall, pressed "Park" on the Virtual Valet app (Figure 3.3) and entered the mall a mere 3 minutes after entering the lot. While Kris was entering the mall, the AutoPark system on her vehicle would initiate communication with the other AutoPark vehicles in the parking lot. The vehicles would send Kris' car information regarding what spots are free and the current path of any vehicle moving in the parking lot (Figure 3.4). Kris' car would use that information to find a free spot closest to where it will be picking Kris up (Figure 3.5). It will also factor in the other vehicles moving in the lot to avoid creating too much traffic in one area and congesting the lot (Figure 3.6). Once it has decided on both a spot and a path, it will relay this information to other vehicles in the parking lot (Figure 3.7) so future vehicles can also make informed decisions.

Figure 3.3

Figure 3.4

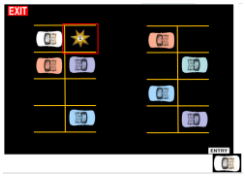


Figure 3.5

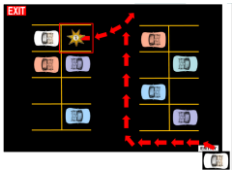


Figure 3.6

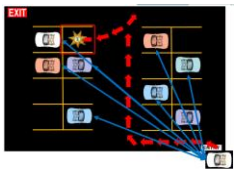


Figure 3.7

Similarly, when Kris is ready to leave, she can simply press "Return" on the app. The app will notify her when the car is ready for pickup, and she can promptly enter her car and leave the lot within 5 minutes. Without the use of the AutoPark system, Kris would be leaving the mall at a much later time, caused by the earlier delay in finding a parking spot. She would then have to remember where she parked her car, and walk the length of the parking lot, weighed down by her purchases. Finally, once she has entered her car, she would have to wait for a break in traffic to exit her parking spot, and then slowly make her way out of the lot, battling traffic along the way. The entire procedure would take approximately 25 minutes. Clearly, the AutoPark system saves the user time and stress and creates a safer parking system for everyone.

4. 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.

As with any project, the requirements have changed several times over the year and the final requirements are shown in the tables 1-4.

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 parking spot related data with other vehicles.	Establish communication with other vehicles within the 10mx10m test area. 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	Follow optimal route to exit.	The vehicle will maintain a velocity between 0 and 10 cm/sec.	Mobile Platform, Software
MF.5	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.6	Exit parking spot	Exit the spot within 2 attempts without collision.	Mobile Platform
MF.7	Sense obstacles in the environment.	Avoid obstacles between 10-50 cm high and 10-120 cm wide.	Mobile Platform, Perception
MF.8	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.9	Stop in the event of an emergency	Stop within 3 seconds of an emergency (obstacle or internal vehicle error).	Mobile Platform, Perception

MF.10	Software simulation to showcase effective planning and scheduling of cars inside a parking garage	Performs significantly better in terms of average park time, pause time and exit time for over 100 vehicles	Planning and visualization

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.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 Vehicle takes no more than 45 seconds to back into spot	Mobile Platform

5. Functional Architecture

5.1 Physical System

The system is represented in the functional architecture, as seen in Figure 5.1. The inputs to the system are a pre-loaded 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" statuses 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" status 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.

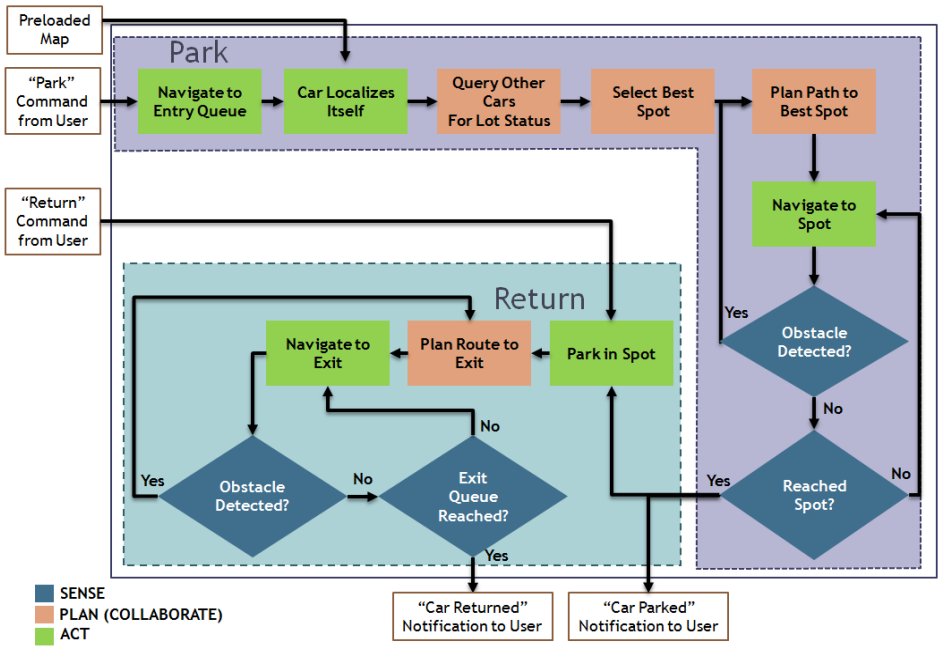


Figure 5.1- Functional Architecture

5.2 Simulation System

The functional architecture for the simulation system is depicted in figure 5.2.

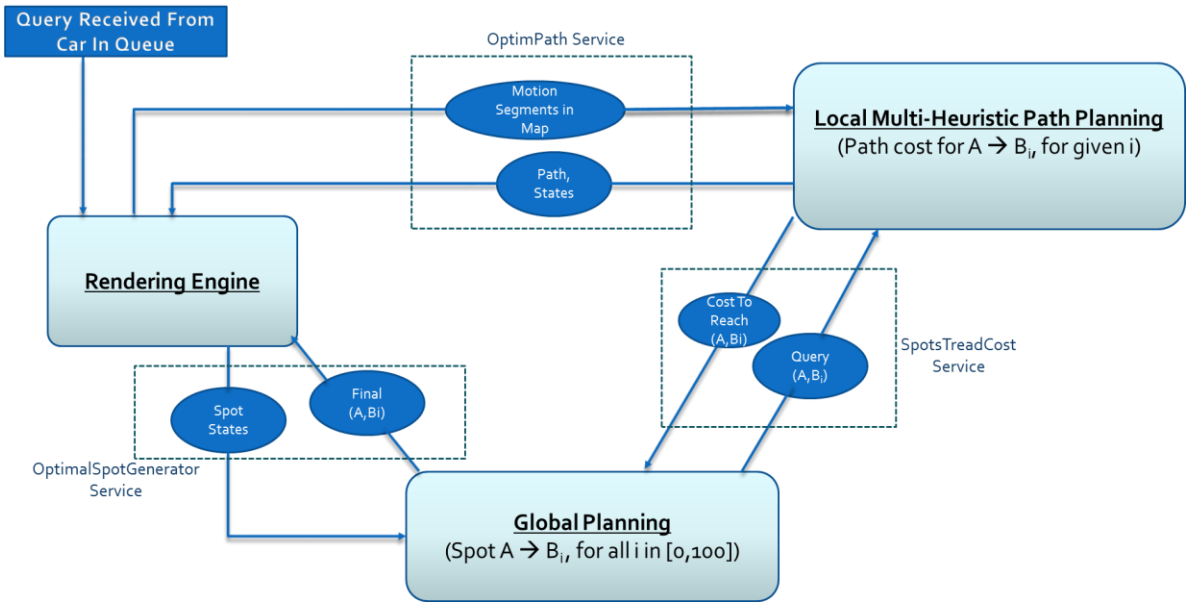


Figure 5.2 Overview of Simulation Functional Architecture

The simulation system is initiated whenever a vehicle enters the parking lot in the visualization. The new car is added to a queue and the rendering engine queries the global planner

regarding which spot should be designated to the new entrant. The global planner receives information regarding the current state of the all the spots in the lot and uses this information to update the cost of each spot in the parking lot.

After this update, the global planner queries the local planner regarding costs of specific spots. This query is needed to take into account the cost to reach a spot given the current condition of the parking lot. The local planner returns the cost associated with the spot and the global planner uses this to find the best spot in the parking lot. This information is relayed to the rendering engine.

The rendering engine, having received the spot the car in the queue should park at, queries the local planner for a path to reach that spot. The local planner returns a path to the rendering engine which is used to park the car in the designated spot. When a car queries the simulation system, the rendering engine queries the local planner for a path from the current location of the car to the exit.

6. System-level trade studies

6.1 Single Board Computer (SBC)

A powerful single board computer is a fundamental requirement to implement autonomy and create a collaborative network between various robots in a system. The processing unit needs to handle Ubuntu 14.04, ROS navigation and planning algorithms, a vision system, and networking with other platforms all running in parallel. To perform these tasks, it is important to choose a single board computer, which has capable enough hardware, in terms of processing power, RAM, etc., and supports other peripherals, such as a Kinect, XBees etc. For these reasons, performance, ease of integration, and support of peripherals have the highest weights in the trade study. Cost is an important factor because the system will require multiple units of these single board computers [3]. The comparison is depicted in table 5.

Table 5 – Single Board Computer Trade Study (Point Scale: 0-10)

Criteria	Weights	Raspberry Pi 2	Edison	BeagleBone Black Rev C	Odroid XU-4	Asus N3150IC
Performance	25	6	10	4	10	10
Cost	10	10	4	7	6	8
Documentation	15	9	5	7	7	8
Ease of Integration	20	8	4	8	8	9

Peripheral	20	5	7	5	7	10
Availability	10	10	10	10	10	10
Total		7.5	6.9	6.4	8.2	9.3

6.2. Point-to-Point Communication

One of the core technologies that the project is intended to showcase is point-to-point communication between multiple vehicles. Additionally, the project will demonstrate how the vehicles can utilize shared data to collaborate in an effective manner. The network technology used to carry out this operation will have a huge impact on the implementation technique and overall performance of the system. Eliminating a central server is a non-negotiable requirement of the user, which is why cloud computing and wireless distributed computing are the two point-to-point communication options. Since this network will be supporting collaboration between real time systems, low latency is a major requirement. Owing to the short timeline of the project, ease of implementation will play a critical role in getting the system up and running within the given timeframe [4] [5]. The comparison is depicted in table 6.

Table 6 – Point-to-Point Communication Trade Study (Point Scale: 0-10)

Criteria	Weights	Cloud Computing	Wireless Distributed Computing
Range	15	10	7
Reliability	15	10	8
Bandwidth	5	10	8
Latency	25	6	9
Cost	5	8	8
Ease of Implementation	25	7	9
Scalability	10	8	6
Total		8.0	8.2

6.3. Communication Hardware

To implement the point-to-point communication system, each mobile platform needed a hardware to establish and join a network. The hardware that were considered were point-to-point WiFi routers like the Bullet 2HP[6] and the XBee [7] routers following the Zigbee or the Digimesh protocol. The key features desired in the communication units were - support for mesh networking,

good indoor range, high bandwidth and robustness. All hardware that operated in the unlicensed frequency range were considered while selection. The comparison is depicted in table 7.

Table 7 – Communication Hardware Trade Study (Point Scale: 0-10)

Criteria	Weights	XBee Pro 900	XBee Pro	Bullet 2HP
Low Frequency	5	10	7	7
Mesh Networking	40	10	10	5
Range	20	7	3	5
Reliability	20	5	5	9
Bandwidth	5	5	5	9
Form Factor	10	7	7	3
Total		7.85	6.9	5.9

The XBee Pro 900 proved to be better than the WiFi based Bullet 2HP. One of the key advantages it possessed, in addition to a longer indoor range and low frequency, was the inbuilt mesh networking feature. Hence, the XBee Pro 900 was selected as the communication hardware for the project.

6.4. Mobile Platforms

The mobile platforms are the most critical component of the project. The mobile platforms are a central dependency for the project as every subsystem would be integrated with the platforms in terms of hardware and software. The team therefore considered the option of creating a platform from scratch to better suit the requirements or buy an off-the-shelf platform. The trade study is depicted in table 8.

Table 7 – Mobile Platform Trade Study (Point Scale: 0-10)

Criteria	Weight	Custom Platform	Oculus Prime	Pioneer	Husky
Time To First Use	15	5	8	9	6
Cost	25	6	8	7	1
Non-Holonomic Steer	10	9	5	5	5

ROS Support	20	1	9	7	9
Support Documentation	10	1	8	7	8
Form Factor	10	8	8	8	5
Hardware	10	7	8	5	7
Total		4.95	7.9	7.3	5.85

The trade study showed that the Oculus Prime was a superior platform in terms of the support it has form the ROS community and the support documentation provided from the manufacturer. Hence, the team decided to go ahead with the Oculus Prime platform.

7. Cyber-Physical Architecture

The system can be divided into 5 main subsystems, as seen in Figure 7.1. The hardware and software interaction between these subsystems is detailed below.

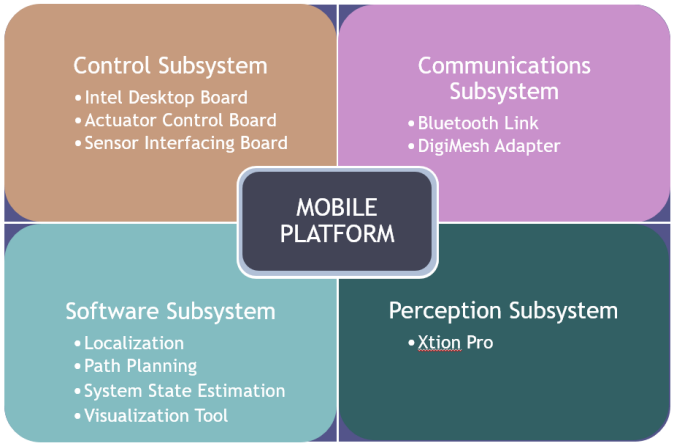


Figure 7.1: System Overview

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 an Intel Desktop Board which is running Xubuntu 14.04 and ROS Indigo. The platform also has a depth sensor (Xtion Pro) for mapping, localization and navigation. 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 7.2).

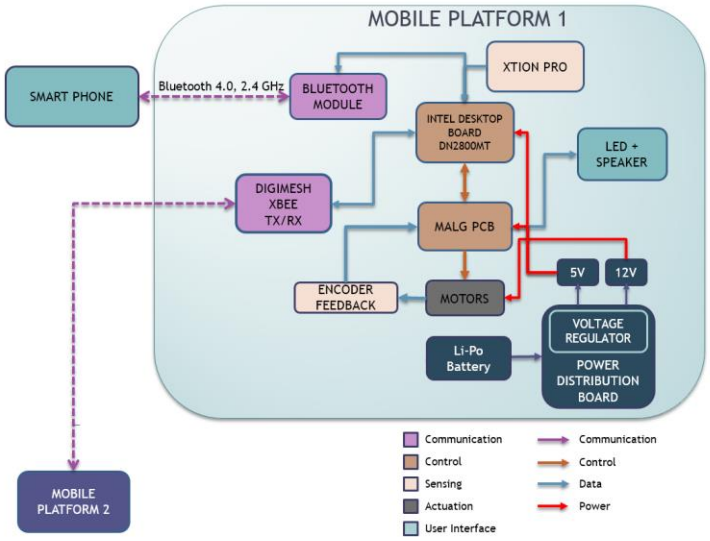


Figure 7.2: Hardware Architecture

7.2 Software Architecture

The software architecture (Figure 7.3) 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.

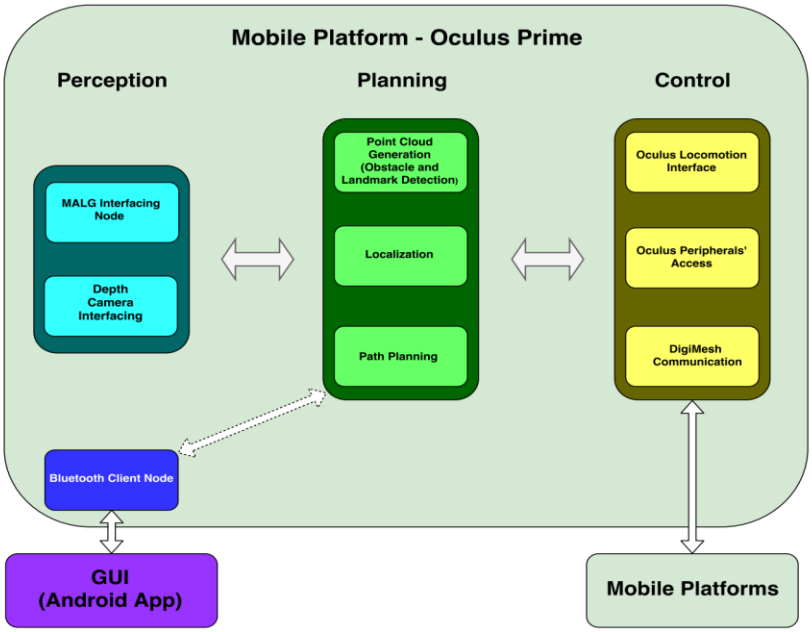


Figure 7.3: Software Architecture

8. System Description and Evaluation

8.1 Subsystem Description

An overview of the current subsystems of the project is discussed in this section. 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.

8.1.1 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 8.1 and 8.2.

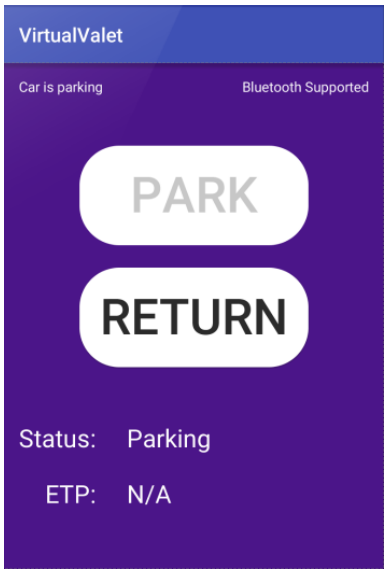


Figure 8.1: Vehicle is Parking

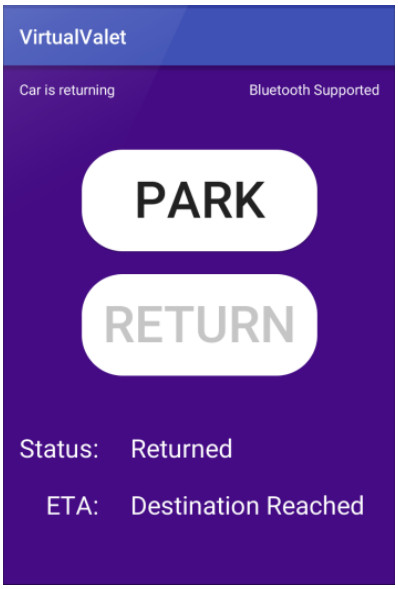


Figure 8.2: Vehicle has Returned

- 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.

8.1.2 Mobile Platform

The Oculus Prime platform, depicted in Figure 8.3, 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 8.3: Oculus Prime Platform

The Oculus Prime 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.

8.1.3 Single Board Computer

The single board computer runs the entire communication and planning subsystem. As per the current routine, the system 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 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, publishing this data back to the decision unit so that it can be sent back to the app through the Mobile App node.

8.1.4 Mapping, Localization and Navigation:

ROS gmapping, which uses a Rao-Blackwellized particle filter based SLAM algorithm, is used to map the entire parking lot structure. Following the mapping, the map is refined using bitmap editor. ROS AMCL (Adaptive Monte-Carlo Localization) is used to localize in the parking lot using wheel encoders and gyro for odometry and depth data for particle filtering. For path planning the DWA (dynamic window allocation) planner is used. The planner first finds a feasible trajectory to the goal location and then uses the shooting-method to shoot out different control actions which are evaluated using value iteration on different criterion.

The locomotion node communicates with the Oculus Prime server running on the SBC via TELNET. Its functions include feeding waypoints to the platform which are translated into platform control commands using all the different ROS packages stated above.

8.1.5 Collaboration and Communication:

The communication 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 8.4.

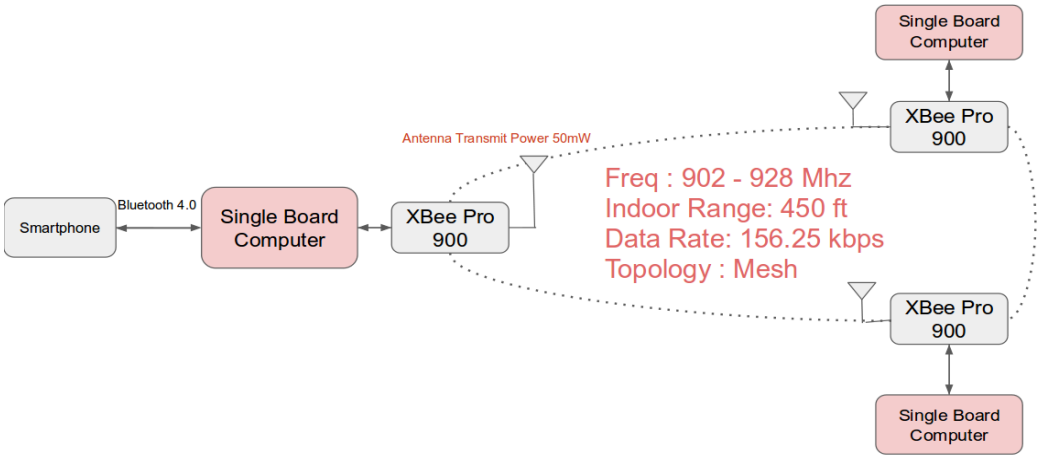


Figure 8.4: Collaboration Architecture

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 a list of all the vehicles and their status within the lot (in queue, parking, parked, returning, or returned) and a list of the waypoints taken by each moving vehicle.

The collaboration node is used to choose the most optimal spot and the path to that spot. It is sent a list of spot options and a list of the waypoints being followed by the currently moving vehicles. The spot options list can either be all the free spots in the parking lot or the pre-selected spot. A spot is pre-selected in two instances - when the vehicle is exiting or when a virtual is added, because it is added to a user-selected spot. When the spot is pre-selected, the collaboration runs an A* planner to find the best path to the spot. The spot and path are returned from the collaboration node. When there is a list of spots, the A* planner is run on the top 10 spots in the list, where the top 10 is determined by the 10 spots closest to the exit. The A* planner returns the path and the resulting f-score of the path. The spot and path with the lowest f-score are returned from the collaboration node. In either case, the path is returned in the form of waypoints for the vehicle to travel to.

The A* planner is a modified version of the traditional A* in three respects. First, there is an added corner cost. This is because the platform does not have accurate turns and thus should make as few turns as possible. The second modification is to the g-cost. Instead of using a traditional distance map with the same cost in each square, the distance map is overlaid with the

paths of any other vehicle moving in the parking lot. This discourages the planner from choosing overlapping paths. There were several different ways the cost of a path could be added to the distance path. For example, it could be a simple "1" or "2" added, or the cost of the path could be higher at the entrance and lower at the spot, or vice versa. After testing several scenarios, it was determined that adding a "2" along the path creates the best avoidance, without causing the vehicle to take long or winding paths. The third modification to the A* planner is that the heuristic is not the distance between the current location and the goal, but the distance between the current location and the parking lot exit. This causes a higher overall f-score for the spots farther away from the exit and a lower overall f-score for the spots closest to the exit, thereby encouraging a spot close to the exit to be chosen.

8.1.6 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, paths, 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 10x10 grid and overlaying different shapes on top to signify the origin, destination, and current location of the platform.

In Figure 8.5, the vehicles are depicted by their number. Parking vehicles are depicted as a number in their intended parking spot, colored lines going from the entrance to the spot, and the same vehicle number in the entrance queue in a colored circle matching the color of the path (example: vehicle 6). Parked vehicles are shown with a number and colored circle in their spot (example: vehicle 2). Returning vehicles are shown a number in their parking spot, colored lines going from the spot to the exit , and the same vehicle number in the exit queue in a colored circle matching the color of the path (example: vehicle 1).

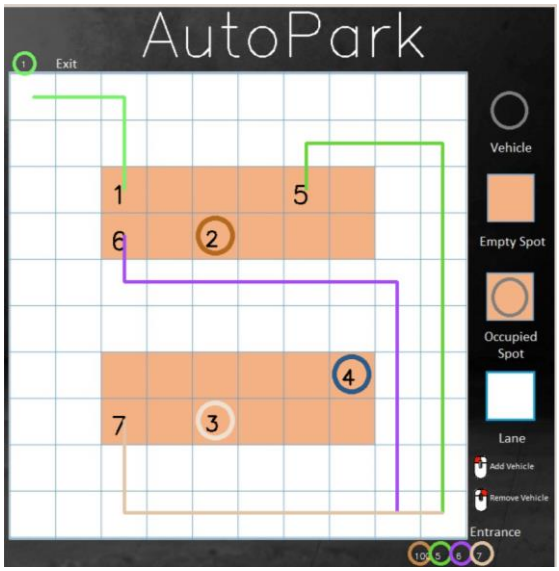


Figure 8.5: Occupancy Map GUI

The GUI is also capable of adding and removing virtual vehicles from the parking lot. This enables the creation of more complicated scenarios to enable more thorough testing. When a virtual vehicle is added, the path to the spot is calculated and broadcast over the communication system.

8.1.7. Simulation Subsystem

The main purpose of the simulation subsystem is to apply different planning and scheduling methods to prove the effectiveness of collaboration in selecting optimal parking policies for more than a hundred vehicles arriving and exiting at random instances of time. The simulation subsystem was implemented using ROS and the detailed simulation architecture is shown in figure 8.6.

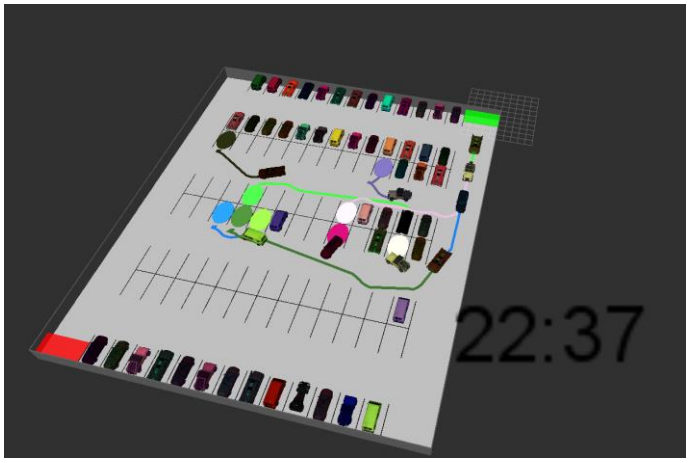


Figure 8.6 Simulation System in Action

The simulator was divided into four major sections whose details are specified below:

Global Planner: This planner is responsible to rank and assign costs to parking spots based on multiple heuristics. The decision to assign a spot to the car is made by this unit. This information is then relayed to the simulation engine to be processed and rendered in the environment. This planner takes into account the static costs base of a spot in the parking lot (distance to exit[9], nearby occupied spots) as well as the dynamic costs associated with selecting a spot (cost to navigate to the spot, cars in queue and time of the day).

Local Planner: It is a custom Astar lattice based planner which returns a path and path cost for a query from either the rendering engine or global planner. The planner takes into account the distance from obstacles while planning and uses an ackerman steer motion model for motion primitives. The planner relays a dynamic cost to the global planner associated with navigating to a spot. It relays a path to the rendering engine between the queried points.

mabPlanner: In order to generate a globally optimal policy for parking vehicles, a trade-off between exploration and exploitation is required. Also, the heuristic based planner does not take into account the random process of arrival and exit of vehicles. Hence, a multi-armed bandit approach was employed which uses UCB1 arm pulls as a one step look-ahead policy. To implement this, the parking lot is divided into four different areas and each area is evaluated on the actual average time to park and exit in the area . Whenever a car enters the parking lot, it is assigned a one-step lookahead based on a UCB1 policy after which the default heuristic policy is

followed in that area. When the car parks or exits, the statistics for that area are updated. Since the distribution is non-stationary, a discounted approach is employed. This online planning helps to significantly improve the park, pause and exit times.

Rendering Engine: It accepts the data from other nodes to render an environment showing the state of all the vehicles in the parking lot. RVIZ markers are used for the visualization of different vehicles and their paths. It also updates the other systems with the real-time state of the lot so that planning can take the dynamic environment into account. The rendering engine also run a Numerical Optimization to switch places of cars based on a tradeoff between cost minimization and reward maximization. The cost in this case is the addition to entropy of the system and the reward is reduction in return times.

8.2. Testing

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

8.2.1 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.

8.2.2 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.

8.2.3 Mobile Platform Locomotion

The following tests were performed on Oculus Prime to ensure robustness of the Oculus Prime Server:

- 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.

8.2.4 User Interface and Communication Integration

The following validation tests were performed on User Interface to ensure it works well with the Communication Subsystem:

- The user interface displays the data communicated by the XBees. Specifically, it shows the state of each vehicle in the network.
- Virtual vehicles added through the interface become a part of the network.
- Path computed by the A* planner gets properly overlayed in the User Interface

8.2.5 Waypoint Navigation Accuracy

The following tests were performed to generate metrics for platform's localization and navigation performance:

- Give waypoints through the web interface to go from a fixed spot A to fixed spot B and observe the planned and executed trajectory.
- Give waypoints through the Python script to go from a fixed spot A to fixed spot B and observe the platform's motion.
- Measure the inaccuracies, if any, between the desired and actual destination.

8.2.6 Simulation Environment

The performance of our planner and the greedy first-spot approach (Greedy 1) and closest spot to exit (Greedy 2) were compared in the simulation environment. In our planner, the heuristics only and multi-armed bandit approach were tested. The approaches were compared based on three criteria- average time to park, average time to exit and the average pause time of each vehicle. The average pause time denotes the time a vehicle stops inside the parking lot due to congestion. Minimizing each of these times reflects a good planning strategy, as can be seen in figure 8.7.

Approach	Average Parking Time (seconds)	Average Pause Time (seconds)	Average Return Time (seconds)
Greedy 1	33.52	23.57	28.64
Greedy 2	36.85	25.72	31.92
Heuristics Only	23.31	17.17	26.62
Multi Armed Bandit Approach	12.5	15.37	15.6

Figure 8.7: Comparative Results

8.2.7 Final System Integration

The final system was tested using all systems, as seen in Figure 8.7: Full System Integration. The app was used to initiate the park sequence on the first vehicle. A secondary XBee connected to the UI informed the vehicle of all available spots. The vehicle interpreted this information to choose the most optimal spot. This information was communicated over XBees and reflected in the UI. While traveling to the destination, an obstacle was detected and avoided. The vehicle continued to the original destination. While the first vehicle was parking, the second vehicle received the park command from the app. Via XBee, it received information regarding the free parking spots and the path being taken by the first vehicle. The second vehicle chose a spot and a path to the spot that avoided the first vehicle. This was also communicated and shown in the UI. When each vehicle reached its destination, it communicated this information over XBees and the UI showed the new status of the vehicle. The process was repeated for the return sequence.

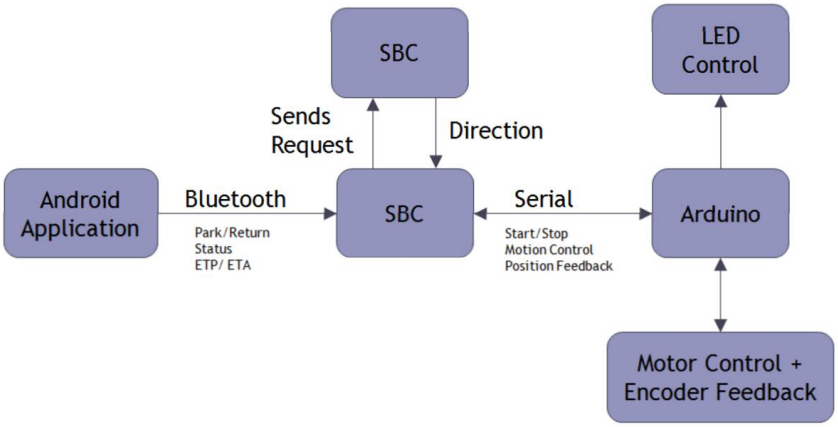


Figure 8.8: Full System Integration

8.3. Performance Evaluation against the Spring Validation Experiment (SVE)

8.3.1. SVE Demo 1

Table 8: SVE Demo 1 Tasks and Performance

Task	Success Criteria	Performance in SVE
"Park" command sent to the first mobile platform using the Android app	LED Blinks to indicate command received, UI shows updated vehicle destination and path	Task completed successfully at SVE and SVE Encore
First vehicle begins locomotion towards optimal spot upon receiving the command	Vehicle moves forward	Task completed successfully at SVE and SVE Encore
"Park" command sent to the second mobile platform using the Android app	LED Blinks to indicate command received, UI shows updated vehicle destination and path	Task completed successfully at SVE and SVE Encore
Second vehicle begins locomotion towards spot near entrance while avoiding path of first vehicle upon receiving the command	Vehicle moves forward	Task completed successfully at SVE and SVE Encore
First vehicle parks in intended spot	Vehicle parks without colliding with infrastructure. Avoids any obstacles encountered. App and UI update vehicle status.	Task completed successfully at SVE and SVE Encore
Second vehicle parks in intended spot	Vehicle parks without colliding with infrastructure. Avoids any obstacles encountered. App and UI update vehicle status.	Task completed successfully at SVE and SVE Encore
"Return" command sent to the first mobile platform via the Android app	Vehicle exits spot, plans and follows path to exit without colliding with infrastructure.	Task completed successfully at SVE and SVE Encore. Vehicle brushed against wall.
"Return" command sent to the second mobile platform via the Android app	Vehicle exits spot, plans and follows path to exit without colliding with infrastructure.	Task completed successfully at SVE and SVE Encore.

8.3.2. SVE Demo 2

Table 9: SVE Demo 2 Tasks and Performance

Task	Success Criteria	Performance in SVE
Launch file shall be started to run the simulation on Greedy and Autopark approach	Autopark planner should perform better than greedy approach in terms of time i.e Average time to park in of the Autopark planner is lesser than the greedy approach.	Task completed successfully at SVE and SVE Encore
A random pattern of arrival and departure times for the vehicles will be created.	Markers should appear and disappear at the appropriate time.	Task completed successfully at SVE and SVE Encore
RViz will display the environment and start rendering vehicles as per the data being received from the rendering engine.	Markers shouldn't overlap with each other.	Task completed successfully at SVE and SVE Encore

8.4 Performance Evaluation with respect to Function and Performance Requirements

- Functional Requirements Validated at SVE
  - **Receive "Park" and "Return" commands from user via smartphone app (MF.1)**
    - The requirement was successfully validated at SVE by demonstrating ROS framework to accept commands from smartphone app via Bluetooth. Upon receiving "Park" or "Return" command successfully, an indicator LED blinks and the UI and App show the updated status of the vehicle.
  - **Share parking spots states with other vehicles (MF.2)**
    - Bidirectional communication with multiple XBees in a scalable network with mesh topology was demonstrated to work successfully between two mobile platforms and one laptop acting as the UI.
  - **Navigate autonomously through parking lot (MF.3)**
    - Vehicles navigated throughout parking lot without human interference.
  - **Follow optimal route to exit (MF.4)**
    - Vehicles planned and executed the shortest path between their parking spot and the exit.
  - **Park inside a parking spot (MF.5)**
    - Vehicles parked completely within a designated parking spot.
  - **Exit parking spot (MF.6)**
    - Vehicles exited parking spot without human interference.
  - **Sense obstacles in the environment (MF.7)**
    - Obstacle was detected and avoided.
  - **Avoid infrastructure (MF.8)**
    - Vehicle detected infrastructure and lightly brushed against it in one occasion due to faulty localization.
  - **Stop in the event of an emergency (MF.9)**

        - Vehicle avoided detected obstacle and had no need to stop.
    - **Simulation to show collaborative planning and scheduling of over 100 cars (MF.10)**
        - Notable improvement was observed in Park, Return and Pause times of the vehicles
    - **Identify optimal parking spot (DF.1)**
        - Optimal spot was identified each time.
    - **Follow optimal route to spot (DF.3)**
        - Vehicles planned direct routes and avoided other vehicles
    - **Avoid other vehicles (DF.4)**
        - Vehicles did not collide and followed minimally overlapping paths to ensure there were no congested areas.
- Performance Requirements Validated at SVE
    - **95% of messages are received**
        - Requirement validated successfully as all Bluetooth messages were successfully sent and received.
    - **Establish communication with other vehicles within the 10mx10m test area. 90% of messages will be received and 80% correctly parsed**
        - All vehicles were able to send, receive, and parse 100% of the messages sent between the three XBees.
    - **100% of navigation will be autonomous**
        - All navigation was performed autonomously.
    - **Exit parking lot within 120 seconds of receiving command**
        - Vehicles exited parking lot within 30 seconds of receiving command.
    - **Park 100% within a parking spot within 2 attempts**
        - Vehicles exited parking spot on first attempt.
    - **Exit the spot within 2 attempts without collision**
        - Vehicles did not collide while exiting the parking lot
    - **Detect obstacles between 10-50 cm high and 20-120 cm wide**
        - Requirement validated successfully in SVE by mobile platform avoiding detected obstacle.
    - **Maintain a distance of 10-15 cm between vehicle and infrastructure**
        - Vehicles largely maintained this distance, although came closer on some instances. This was due to the size of the parking spots and faulty localization.
    - **Stop within 20 cm of a static obstacle**
        - Requirement is no longer valid. However, vehicle planned path around parking lot, which is a more appropriate requirement.
    - **Distance between spot and exit is shortest amongst all available spots**
        - Requirement was validated by vehicles parking in most optimal spot
    - **Park in spot within 120 seconds of receiving command**
        - Vehicles parked within 60 seconds of receiving command.
    - **Vehicle maintains at least 60.96 cm (2 ft.) between the front of one moving vehicle and the back of another moving vehicle**
        - Vehicles planned paths to avoid one another.

8.5 System Strength and Weaknesses

8.5.1. Strong Points

One of the greatest strengths of the system is that the subsystems are largely modular and can be applied to various parking lots and scenarios. Listed below are the subsystems and advantages:

- Communication System is decentralized and does not alter the infrastructure in any way.
- Simulation System is scalable to any larger parking lot of varied dimensions.
- Owing to the modular software architecture, the global planner and local planner are replaceable with any other planners that interface with the rendering engine in the same way.
- Planning in simulation explores the multi-armed-bandit approach instead of weight tuning.
- Android app is functional and enables easy user interaction with system.
- The team has modular obstacle detection capabilities developed and ready to be integrated as an alternative to ROS based obstacle avoidance.
- Communication subsystem is robust and has sufficient indoor range.
- Oculus Prime mobile platform provides versatile capabilities and its ROS packages aid in developing the localization and planning subsystem.
- The mock parking lot created easy to assemble and its dimensions remain consistent with the map.

8.5.2. Weak Points

Listed below are the subsystems and disadvantages:

- The system has many single points of failure, specifically in terms of hardware like the power distribution board and the actuator control board. Moreover, the dependency on Oculus Prime custom hardware affected timelines in case a replacement part was needed immediately as the manufacturer is based in Canada, which further increases lead times.
- The vision system on the platform is not robust which further affects the localization of the platform. The symmetric nature of the parking lot further worsens this problem.
- The planner on the platform currently relies on the Dynamic Window Approach which causes the platform to get stuck when navigating tight spots.
- The accumulation of the odometry errors on the platform results in the platform getting progressively more lost as it traverses in the parking lot.
- Communication subsystem isn't an exact implementation of VANET due to which its performance under heavy traffic can't be discerned.
- Simulation environment fails to capture the true dynamics of a vehicle due to which it might exaggerate the results.
- Relying on a depth camera to generate laser scan adds unnecessary computation load as we have to process more data.
- Oculus Prime doesn't have Ackerman Steering due to which our planners and actual locomotion of the platform are disjoint and need an additional layer of interfacing.

9. Project management

9.1 Critical Path/Schedule

The major system development milestones that need to be attained throughout the project were:

1. Implement a collaborative network to work for multiple platforms
2. Implement a robust navigation subsystem involving localization, path planning and obstacle avoidance for multiple platforms.
3. Implement an entire simulation system to show effective planning and scheduling of over 100 vehicles in a parking garage.
4. Integration of all the subsystems

Fall Semester Schedule:

The schedule for fall semester is detailed in the table 10 below:

Table 10: Fall Semester Schedule

Timeline	Goals	Milestones
Progress Review 1: 22-Oct	Literature studies for various subsystems complete, app GUI complete and bluetooth interface with SBC tested	
Progress Review 2: 29-Oct	Actuator control board and SBC integrated	
Progress Review 3	Collaboration Software and hardware setup and obstacle avoidance software developed	Sensor control board tested
Progress Review 4	Vision system integrated and map tested	
Fall Validation Experiment and Demonstration	30-Nov	
11/30 - 12/3	Dedicated integration and testing	
Progress Review 5	3-Dec	FVE demo complete
Progress Review 6	10-Dec	

Spring Semester Schedule

Table 11: Spring Schedule

Timeline	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	
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	

The project was completed successfully as per the intended schedule.

9.2 Budget

Table 12: Project Budget

Part	Part Number	Quantity	Cost/Unit	Total Cost
Oculus Prime (Mobile Platform)*	N/A	1 purchased, 1 from MRSD, 1 from sponsor	$499.00	$499.0
ODROID-XU4	N/A	3	$74.00	$222.00
MinnowBoard Max*	N/A	2	$147.00	$294.00
Arduino Nano v3.0	N/A	3	$15.38	$46.14
Infrared Sensor	Sharp GP2Y0A21YK	3	Inventory	Inventory
Asus Xtion Pro	N/A	1	$159.99	$159.99
Miscellaneous (mounts, electronics, memory, cables)	N/A	N/A	N/A	$625.99
DigiMesh XBee Pro *	N/A	5	$99.00	$594.00
MALG PCB	XAX-053	3	$55.35	$166.05
AmazonBasics 7 Port USB Hub	HU2W70E1	2	$15.79	$31.58
AmazonBasics 4 Port USB Hub	HU3641V1	2	$16.98	$33.96
Parallax Laser Range Finder	RB-Plx-257	1	$99	$99
Intel DN2800MT Marshalltown Thin Mini-ITX Motherboard - BLKDN2800MT	Intel DN2800MT	1	$145	$145
Power Lipo3S PCB	XAX-063	3	$48.64	$145.92
Adata 64GB SSD	N/A	1	$52.66	$52.66
CC Bec Pro- Castle Creations	N/A	3	$20.27	$60.81
Whitelabel Bluetooth 4.0 USB Dongle Adapter	N/A	4	$12.99	$51.96
Miscellaneous Electronics	N/A	N/A	$500	$500
Miscellaneous Hardware	N/A	N/A	$150	$150
Total Budget:				$3878.06
				*big ticket items

Evaluation of Budgeting Process:

The budgeting process was carried out very smoothly with parts ordered in a timely manner with sufficient extras. We managed to keep the budget within the maximum amount of $4000 including all big ticket items. One of the major successes were the backups of the platforms and Oculus Prime spare parts which turned out to be extremely critical towards the end of the project. One of the major failures was the large number of different SBCs ordered due to inefficient trade studies on our part at the beginning of the semester.

9.3. Risk Management

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 13: Project Risks

ID	Description	Owner	Risk Analysis	Area of Impact	Handling Strategy	Status
1	No Mobile Platform	Mohak	N/A	Technical, Schedule, Cost	Check Status with sponsor regularly, Set final deadline for platforms to arrive. Email list of MPs to sponsor. Meet with sponsor. Purchase test platform.	Closed
2	Inadequate Mobile Platform	Shivam	3x3	Technical	Review platform needs Only select adequate platform	Closed
3	Unsuitable Smartphone Interface	Dorothy	3x2	Technical Programmatic	Frequent and extensive testing	Closed

4	Subsystem Incompatibility	Pranav	5x2	Technical Schedule Cost Programmatic	Research on ROS to ensure compatibility. Software architecture to ease integration. Carry out low level cross compatibility tests.	Closed
5	Too Many Requirements	Shivam	4x2	Schedule Programmatic	Trimmed requirements Began weekly sprints Created Kanban cards	Closed
6	Inaccurate Parking Lot and Obstacles	Richa	1x1	Programmatic	Analyze parking lots IRL. Scale to match mobile platform	Closed
7	ROS Related Issues	Pranav	5x1	Technical	Arduino/ROS testing. Android/ROS testing. ROS-ROS serial communication.	Closed
8	SBC and Platform Incompatibility	Mohak	N/A	Technical Schedule Cost Programmatic	Use MinnowBoard Max borrowed from Team C	Closed
9	MinnowBoard Max Processing Power Limitations	Mohak	5x5	Technical Programmatic	Test SBC thoroughly to investigate other possible problems. Research alternative, more powerful SBCs	Closed
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	Closed
11	Mobile Platforms don't arrive on time	Shivam	4x5	Technical,Schedule,Cost	Check Status with sponsor regularly, Talk to Prof. Dolan regarding additional platform.	Closed
12	Data latency in subsystems	Team	4x4	Technical	Early and Extensive Testing	Closed
13	Interference in Bluetooth and multiple Xbees	Dorothy, Richa	4x3	Technical	Early identification and fixing requirements accordingly	Closed
14	Components arriving late	Team	3x5	Schedule, Cost, Programmatic	Order many extras, ordering well in advance, contingency plans and immediate action	Closed

15	Dependence on Oculus Prime ROS packages	Pranav	3x3	Schedule Technical	Alter system design accordingly as soon as loopholes or dead-ends are identified	Closed
16	Dependency of localization on mapping capabilities and parking lot complexity	Mohak	3x2	Programmatic, Technical	Keep design of parking lot as simple and “feature rich” as possible	Closed
17	Multiple points of hardware related failure in perception and communication	Shivam, Richa	3x4	Programmatic Technical	Incremental integration and testing during development	Closed
18	Inconsistencies in communication and planning logic	Richa	4x1	Technical	Incremental integration and testing during development	Closed
19	Navigation Issues in Parking Lot	Shivam, Pranav	4x3	Technical	Extensive Testing, Modification of Nav. Stack	Closed

In order to efficiently track the risks, a risk manager was appointed from within the team in both semesters and each risk was assigned a risk owner. The high likelihood and consequence risks such the mobile platforms not arriving on time, inadequate processing power and navigation issues in the parking lot were actively mitigated by the whole team.

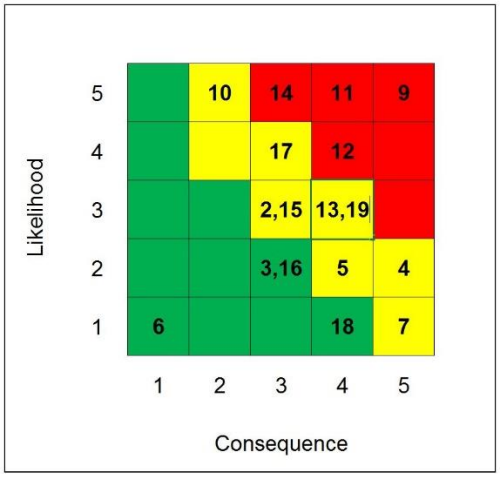


Figure 9.1 Risk IDs Charted with Consequence and Likelihood Levels

10. Conclusions

The project was largely successful in demonstrating collaborative parking between mobile platforms in the physical world and the advantages of planned parking in simulation. The team was able to meet most of its requirement according to the set performance metrics. The team created a new way to implement the A* algorithm to avoid creating congestion while maintaining optimal placement of vehicles. Additionally, the team was able to formulate a novel variant of the classical 'Exploration vs Exploitation' problem that has been extensively discussed in the Multi-Armed Bandit approach.

10.1 Lessons Learned

The team faced numerous challenges during the course of the project: both technical as well as logistical. The project was great learning experience for the team and the critical lessons learned are highlighted below:

Technical Lessons Learned

10.1.1 Sensor Placement

Sensor placement plays a critical role in any system, especially if you intend to localize your robot using the sensor; case in point being the placement of the depth camera on the robot. The range limitations and the sensor-cone of the depth camera should be taken into mind when selecting the sensor and as well while placing it on the robot. This is even more critical while running the ROS AMCL (Adaptive Monte-Carlo Localization) as the localization relies on a 2D projection of a 3D world.

10.1.2 Generating Plans in Tight Spaces

The DWA (Dynamic Window Approach) planner essentially performs a forward simulation from the robot's current state to predict what would happen if a sampled velocity were applied for some period of time. In our case, the DWA planner could not find a valid trajectory due to the proximity of the walls to the platform. The team learned that generating plans in tight spaces would not work always and the team decided to code its own recovery maneuvers before generating plans from the DWA.

10.1.3 Destructive Testing

This is the most critical lesson that the team learned the hard way- the importance of testing your code and hardware for possible failure scenarios. This involves testing code with expected as well as erroneous data for prolonged periods of time. Testing the mobile platforms for various failure cases such as inaccurate localization and planning-failures also proved to be helpful in the long run.

Project Management Lessons Learned

10.1.4 Procuring Backups

The importance of having backups ready to replace faulty components cannot be stressed enough. The team invested in backups throughout the project and this paid off in the long run as the failed components were swiftly replaced during testing. However, ordering backups of components that are useless in the long run can prove to be detrimental to the financial planning of the team

10.1.5 Identify Critical Components To The Project

Certain components, like the mobile platforms, are critical to the execution and delivery of any project. It is absolutely essential to identify these components and take a call on them as early as possible. Following up on these components should not only be the responsibility of the project manager but collective responsibility rests on the team. The team faced further delays in receiving one of its platforms due to a misunderstanding in ordering which makes it even more necessary for the team to actively track such components.

10.1.6 The 'Knock-it-off' Point

One of the most difficult decisions in project management is taking the call on 'knock-it-off' points- the time when the team would stop pursuing a strategy. This lesson addresses the problem of what to do when a really great feature might not be delivered on time. Continuing aimlessly without taking a call to not integrate the feature might cause unprecedented delays in integrating the entire system and prove to be detrimental in the long run.

10.2 Future Work

The team successfully demonstrated the collaborative communication of the AutoPark system. To continue this work, there are a few approaches:

10.2.1 Time-Base A*

The current implementation of our modified A* algorithm is state-based. That is, it assumes that the vehicles in motion will stay in motion while the vehicle is executing its path and the stationary vehicles will stay stationary while the vehicle is executing its path. A less naive approach would be to incorporate the real-time updates mentioned in section 10.2.1 to plan a route for the vehicle that will not create congestion at the time that the vehicle reaches the area. This will create a more accurate planning system that enables a real-time approach to the parking problem.

10.2.2 Map Sharing

The system has currently only been tested in a known parking lot with a pre-loaded map. To enhance the system, map sharing would enable the AutoPark system to work in any parking lot. When a vehicle enters a lot that is already occupied by AutoPark-enabled vehicles, it should receive physical details of parking lot through the communication channel. The first vehicle in the lot can obtain the map from the cloud through the secure, built-in update system of future autonomous vehicles.

Following the successful results from the simulation environment, the team aims to continue working on developing on its multi-armed bandit approach. One of the many challenges in implementing the multi-armed bandit approach to the planned-parking problem include the following:

10.2.3 Delay etween Action and Reward

The 'reward' term in the multi-armed bandit approach is generated after the vehicle has parked in a spot and the planner learns the parking time for the vehicle. However, since the time the 'action' of sending the vehicle to a spot is taken, many more new vehicles enter the lot and are assigned spots. This delay in taking an action and receiving a reward is not covered as part of the standard UCB1 algorithm. To overcome this, we tried to implement a discounted approach to the UCB1 algorithm. However, in the future we aim to implement a variant where we generate an 'estimated reward' and update in later when the actual reward arrives.

10.2.4 Q Learning

The parking problem will in the future be formulated as a Q-Learning problem where the current heuristics in the global planner will be treated as the features of the state of the parking lot and temporal difference learning will be used to learn the weights on the features. The simulation will be run for a long period of time and at each time step the reward obtained will be treated as a sample of the Q-value of the particular state-action pair. When our agent has experienced enough amount of episodes, the most optimal weights will be learnt for the parking lot and the planner will converge on an optimal policy for spot allocation. Using approaches such as epsilon-greedy and UCB, we will be able to appropriately trade-off between exploration and exploitation to arrive at a globally optimal policy for the parking lot.

10.2.5 Policy Search

Another method that we will be employing is policy search to learn the best heuristic weights. Unlike Q-Learning which learns weights taking the actual Q values as samples, we will try to directly learn policies that predict the best Q values. Policy search evaluates the current policy (weights) for a long time on the parking lot. The evaluation criterion will be the average park and return times experienced in the parking lot. After we get the performance estimates, hill climbing will be done to optimize the weight on the heuristic values.

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.
3. "Comparison of Single-board Computers." *Wikipedia*. Wikimedia Foundation, 1 Oct. 2015. Web. 25 Sept. 2015. https://en.wikipedia.org/wiki/Comparison_of_single-board_computers
4. "Cloud Robotics." Wikipedia. Wikimedia Foundation, 23 Sept. 2015. Web. 25 Sept. 2015. https://en.wikipedia.org/wiki/Cloud_robotics
5. "Distributed Computing." Wikipedia. Wikimedia Foundation, 28 Aug. 2015. Web. 25 Sept. 2015. https://en.wikipedia.org/wiki/Distributed_computing
6. "Bullet 2HP Review" http://www.cruisersforum.com/forums/f13/internet-ubiquiti-bullet-any-good-59477.html
7. "XBee." *XBee*. XBee. Web. 25 Sept. 2015. http://xbee.wikispaces.com/
8. "Oculus Prime Details" http://www.xaxxon.com/oculusprime/details
9. Search Based Planning Laboratory www.sbpl.net
10. “INRIX Predicts Increase of Thanksgiving and Black Friday Traffic for Malls, Airports and Cities” http://inrix.com/press/2015-thanksgiving-traffic-forecastiving-and-black-friday-traffic-for-malls-airports-and-cities