Autonomous car with path detection and obstacle avoidance

Team Name: MAL
Short Team Name: MAL

Ajay Satishkumar Amarnath (amarnathajay@email.arizona.edu)
Megha Agarwal (meghaagarwal@email.arizona.edu)
Lenny Lopez (lennylopez17@email.arizona.edu)

ECE573–Software Engineering Concepts
Spring 2017
Instructor: **Matt Bunting**

May 3, 2017

Arizona's First University.

College of Engineering

Department of Electrical & Computer Engineering
Box 210104, Tucson, AZ 85721-0104

Contents

List of Figures

List of Tables

Autonomous car with path detection and obstacle avoidance

1 Executive Summary

Automobiles have been constantly evolving ever since the introduction of the first automobile in 1807. However, the one thing that has remained constant is the need for a driver to operate the automobile. This is the reason that in recent times, there has been a desire to reduce the dependence on drivers and introduce semi-autonomous and autonomous cars. This could be one of the most significant and important changes in the automotive industry. Well implemented autonomous cars have the potential to be extremely safe, well beyond the capacity of the average human driver as well as make journeys a lot more efficient and comfortable. Practically speaking, however, there must be an increase in the system capabilities to be able to handle all the aspects of driving with a minimal margin of error. As these systems gradually improve, we will see the role of the driver eventually become like one of an aircraft pilot. The driver would only set the parameters of the journey like setting the start and end points and leave the control to the computer which would interface with the series of sensors connected and handle the driving aspect. There are many scenarios to which autonomous driving can be applied. We discuss one such scenario below.

Our approach to this problem deals with cars that may be autonomous in the off-road terrain. We take a car that may need to move from point A to point B in a certain off-road area. Taking the example of the typical rocky desert terrain of Southern Arizona, we see that there may be large rocks, protected cacti as well as terrain that cannot be climbed straight on like cliffs. To move in the best possible path from point A to point B, we would need to be aware of the position as well as the size of the obstacles that we encounter.

As we encounter obstacles in the range of the car, we store their dimensions and position and decide on which direction we would need to move. Based on the information that we have at that moment in time, we decide on the direction of the movement of the car using a pathfinding algorithm. Our aim in this project is to find the best possible path between two points in an uneven terrain by traversing through, discovering the obstacles and moving around them.

If a system like this is brought to the market, we will see a great reduction in off-road accidents, greater efficiency of fuel consumption and improved ease of use. This will also make applications like hiker rescue much easier and improve the accessibility of locations.

2 Project Overview

Autonomous navigation in off-road environments presents many new challenges when compared to city-like environments. The lack of highly structured components in the scene

complicates the design of an obstacle detection system. In addition to the geometric description of the scene, terrain typing is also an important component of the perceptual system. Recognizing the different classes of terrain and obstacles enables the path planner to choose the most efficient route toward the desired goal. It is an interesting challenge due to the complexity of the kind of geometric obstacle possibilities of the environment which need to be detected and avoided.
The project will focus on using sensor systems on the catvehicle that do not rely on a typical structural assumption on the scene (such as the presence of a visible ground plane) to detect obstacles and enable the path planner to route the car in a way to avoid those obstacles in the most efficient way. For this project, we would focus on utilizing the sensor data from the catvehicle to develop algorithms to detect a range of typical off-road obstacles in different terrain types. Simultaneously, we will be developing real-time routing algorithms which take in the data available in real-time to make decisions about the best possible route to the destination. The project will be completed over 2 months.

3 Requirements

Here are the 'B' requirements for our project.

1. The simplesolution node detects objects along the car's view. (3)
   Prerequisite: Installation of packages is completed.

2. The simplesolution node examines detections and determines whether they should be combined into a single object. (3)
   Prerequisite: Object detection is completed.

3. The pathfinder node will determine if it is possible to find a route between an initial point and final point and will display a message if it is not possible. (3)
   Prerequisite: Simplesolution node is completed.

4. The pathfinder node will determine the best possible route between an initial and final point based on the point and obstacle coordinates. (3)
   Prerequisite: Simplesolution node is completed.

Here are the A requirements for our project.

1. The pathfinder node will estimate the distance (according to the selected best possible path) required to reach the final point from the initial point. (2)
   Prerequisite: A* algorithm implementation is completed.

2. The time estimator node will estimate the time required to reach the final point from the initial point.(2)
   Prerequisite: A* algorithm implementation is completed.

4 Domain Analysis

The application will utilize obstacle-detection and path finding (repetitive A*) algorithms to provide the best possible obstacle-free path for the users journey. The interaction model of the application is covered in the following use-case and sequence models which show the interactions of behind-the-scenes portions of the design.

4.1 Use Case Summaries

Input initial and final positions: The user will specify the initial and final positions of the catvehicle in order for it to calculate the best possible obstacle-free path between the two positions.
Push Updates: The programming team will perform scheduled tests to eliminate bugs in the code.

4.2 Use Case Description

There are two use cases for this project

Use Case 1

Use Case: Input initial and final positions
Summary: The user will specify the initial and final positions of the catvehicle in order for it to calculate the best possible obstacle-free path between the two positions.
Actors: User
Preconditions: The user knows his/her ending coordinates.
Description: The catvehicle will plan the best possible obstacle-free path for the user from the input coordinates and will update the path every time new obstacles are detected.
Exceptions: No path found.
Postconditions: The catvehicle will wait for new input positions.

Use Case 2

Use Case: Push updates
Summary: The programming team will perform scheduled tests to eliminate bugs in the code.
Actors: Software team
Preconditions: The code has known issues/bugs.
Description: The team will perform scheduled tests to eliminate bugs in the code.
Exceptions: No new bugs found
Postconditions: Continue running tests to look for bugs.

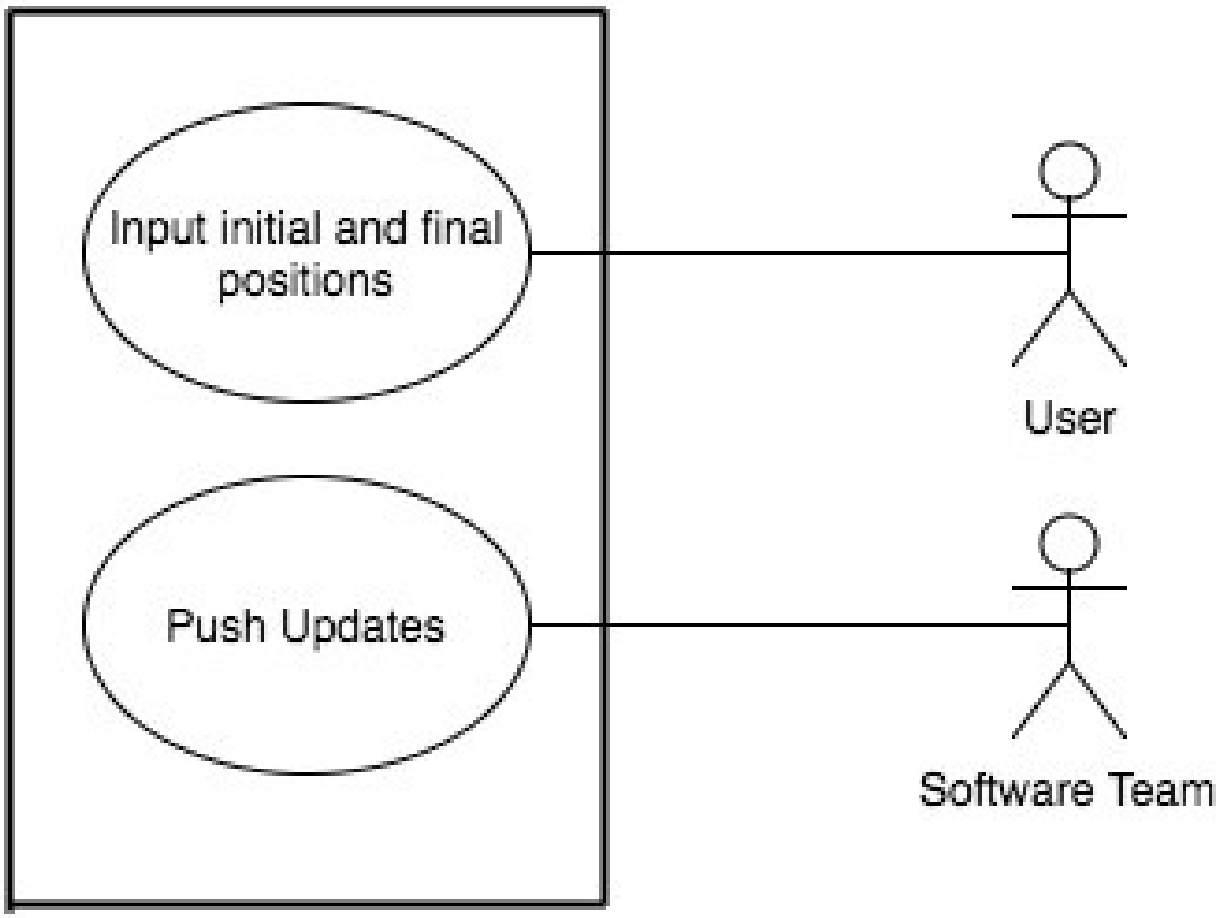

Figure 1. Use Case Diagram for the operation of the autonomous car

4.3 Sequence Models

Sequence models can be used to elaborate the information that is brought out by the use cases.The first sequence diagram shows the sequence of events when a path is not available for the input positions. For example, the final position is inside a house which the car cannot break into. The second sequence diagram shows the sequence of events when an obstacle is detected by the car sensors. Using these diagrams we can visualize the sequence of events that take place when the car is in use.

5 Important Algorithms

First, lets have a look at the problem statement. We are planning to have the car move from point A to point B in the best possible available path using obstacle detection. As we encounter obstacles in the range of the car, we store their dimensions and position and decide on which direction we would need to move. Based on the information that we have at that moment in time, we decide on the direction of the movement of the car. Let us first consider

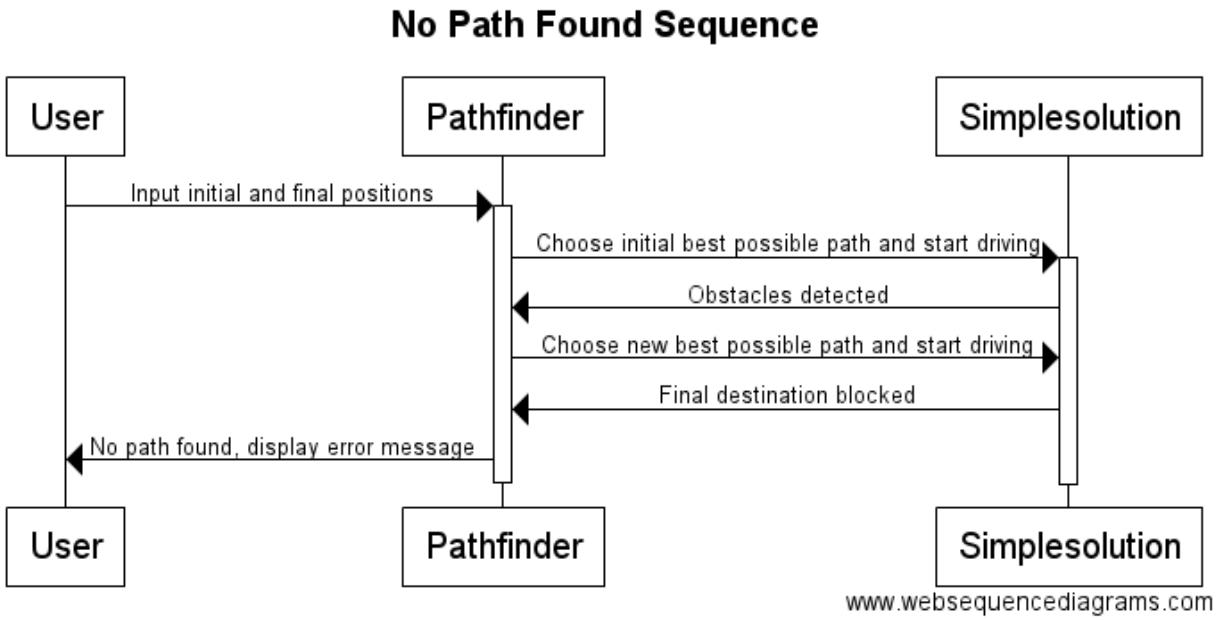

Figure 2. Sequence diagram of the events at the start of the journey. There is no path defined at the start.

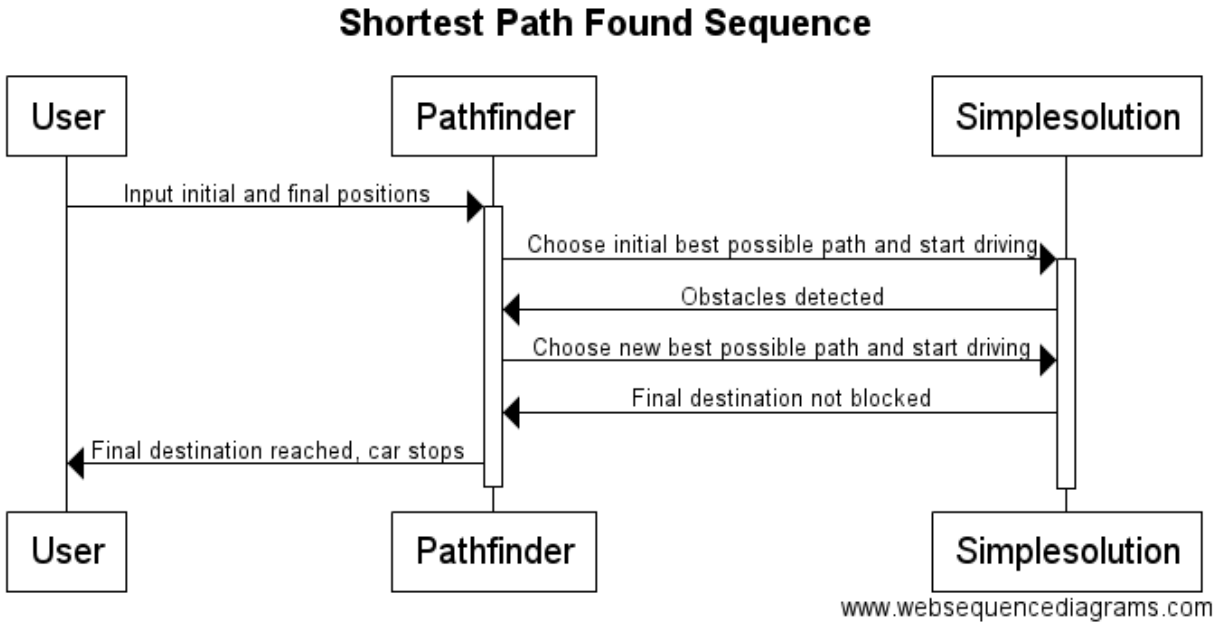

Figure 3. Sequence diagram of the events any time after the car is initialized.

the case of the car moving from point A to point B in a terrain which does not contain any

obstacles. The algorithm, in this case, would be to simply turn the car in the direction of point B and start moving. The algorithms are described in the following subsections.

5.1 Driving Path when there are no Obstacles Present

This algorithm is used when starting the car at the beginning of the journey. The algorithm is given below.

START
get point A and B coordinates
calculate distance and angle of motion
set driving speed
if car not pointed in direction of point B
    set steering angle of the car to turn car towards point B
else
    do not change steering angle
if coordinates of the car are the same as that of point B
    stop car
else
    keep moving
END

Now we see that the above is a very simple example of how a car would move across a flat terrain. However, in real life situations, we see that the terrain will not be a simple flat surface which would dictate that we need to detect and avoid objects that we would encounter as and when we go along. The algorithm for detection of obstacles and the determination of shapes is described below. The shape determination is important as the size, shape and type of obstacle determine the path of the car.

5.2 Obstacle detection algorithm

This algorithm is used to detect the shape, size and type of the object encountered as the car moves along the terrain. The algorithm is given below.

START
Start driving using Algorithm in section 5.1
Use lasers to fire beams around the car
if reflected laser beams received
    mark new obstacle present
    evaluate laser beams for obstacle coordinates, size and shape
    store size, shape and type data for the new obstacle
else
    keep driving
END

5.3 Pathfinding Algorithm

The pathfinding algorithm is the algorithm that is used to calculate the path based on the obstacles created. This algorithm is shown below.

START
if no obstacles are detected
    Use algorithm 1 to start driving
else
    evaluate all obstacle data stored
    find all coordinates to be avoided
    employ search algorithm to find the best path
    correct course
END

This is an important aspect of the car as it controls the overall distance and time taken to move from the initial to the final point. Thus, we need to use a search algorithm that performs well. This will be discussed in the next section.

5.4 A* Search Algorithm

We have seen throughout the years of game development, the A* search algorithm has been the most popular search algorithm used in strategy games like the Age of Empires and Civilization V. We will be using the A* search algorithm in our pathfinding algorithm as it is an efficient performer. The algorithm is given below.

initialize the open list
initialize the closed list
put the starting node on the open list (you can leave its f at zero)

while the open list is not empty

find the node with the least f on the open list, call it "q"
pop q off the open list
generate q's 8 successors and set their parents to q
for each successor
if successor is the goal, stop the search
successor.g = q.g + distance between successor and q
successor.h = distance from goal to successor
successor.f = successor.g + successor.h
if a node with the same position as successor is in the OPEN list
which has a lower f than successor, skip this successor
if a node with the same position as successor is in the CLOSED list
which has a lower f than successor, skip this successor
otherwise, add the node to the open list
end
push q on the closed list
end

Algorithm source - Rajiv Eranki, "Pathfinding using A* (A-Star)", MIT, 2002

We can visualize the working of the A* algorithm with the following figure. The dark gray squares are the open nodes that can be traversed. The light gray squares are the obstacle nodes. The white squares are the explored areas and the black squares denote the calculated path.

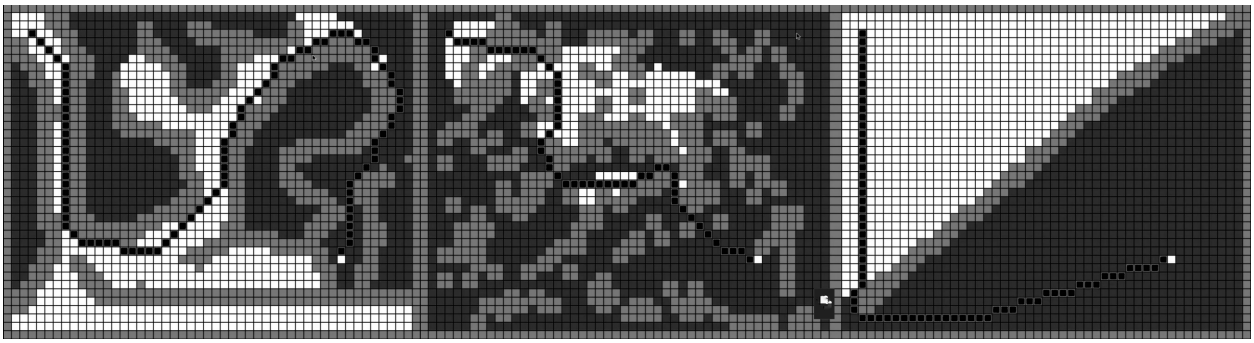

Figure 4. A few examples of the A* algorithm in action. We can see good performance in typical obstacle filled environment shown in the middle.

6 Class Design

For our project, we need 2 nodes. The class diagram of the classes that implement the design is given in figure 5.

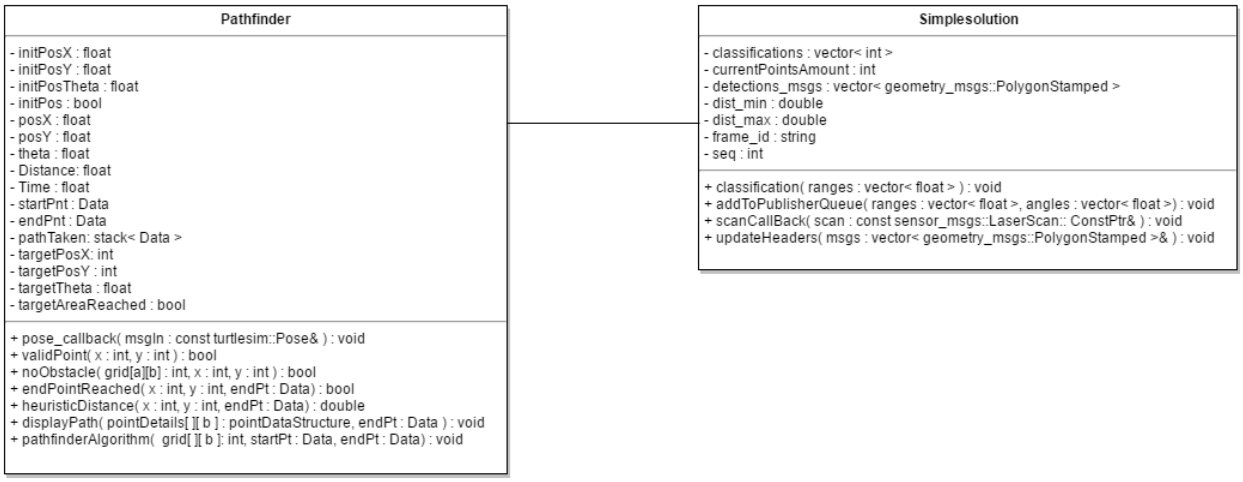

Figure 5. The class diagram for the Autonomous car with path detection and obstacle avoidance project

A brief overview of the 2 classes is given below.
simplesolution: It filters the sensor data from the laser scan along the car's view to detect obstacles. It classifies the obstacles to determine whether it is a single or multiple objects.
PathFinder: Determines best possible possible route from point A to point B with no obstacles. The exception being that no path is found. It assumes that no obstacle is present to determine the initial proposed path. It also estimates the total time for the car to traverse through the proposed path, except when no path is found. Finally, it estimates the total distance in the proposed path, except when no path is found.

7 Testing Strategy

The testing strategy has been ordered with respect to the requirements and can be seen below.

1. **The simplesolution node detects objects along the car's view.**
   The data from the laser scan will be used to test the parameters. We created a world file with a single object of known coordinates and tested the code to check whether the polygon returned contained coordinates within 5 percent of the known coordinates.

2. **The simplesolution node examines detections and determines whether they should be combined into a single object.**
   This will be done to make sure that larger objects are detected as one single object. We again placed a test object of known dimension and coordinates and used the sensors to collect data and tested whether the detection topic showed one object or multiples ones.

3. **The pathfinder node will determine if it is possible to find a route between an initial point and final point and will display a message if it is not possible.**
   This is tested by creating a world file with a single big object, such as a house or gas station, and placing the end coordinate of the algorithm inside the big object. Running the algorithm should show no path found and hence one part of this requirement can be tested. The other part can tested by using the world file created in requirements 1 and 2 in which case the path should be found because the end coordinate is not enclosed by objects.

4. **The pathfinder node will determine the best possible route between an initial and final point based on the point and obstacle coordinates.**
   This is tested by using the world file created in requirements 1 and 2 in which the no path found flag is not high because the end coordinate is not enclosed by objects. As a result, a path is detected and hence this requirement is tested.

5. **The pathfinder node will estimate the distance (according to the selected best possible path) required to reach the final point from the initial point.**
   To test this requirement, we created an empty world file and ran the pathfinder node with the initial and final points and compared the distance outputted with the actual distance between the two points and checked that it was within 5 percent of error.

6. **The time estimator node will estimate the time required to reach the final point from the initial point.**
   To test this requirement, we used the same empty world file as the previous requirement and ran the pathfinder node with the initial and final points and compared the time outputted with the actual time required between the two points and checked that it was within 5 percent of error.

8 Integration with Platform

The platform used to develop the autonomous car will be ROS Indigo. This is a fairly recent release of ROS and has the support for all the tools that we will be using at this time.
The list of packages we will be using within ROS Indigo are

1. **catvehicle:** This will be the car used to run the simulation
2. **obstaclestopper:** catvehicle uses this package to use sensor
3. **gazebo:** Used to visualize the world and path taken
4. **rviz:** Used to visualize sensor data

We will be using the following sensors

1. **The front laser and laserscan:** Used as part of the obstacle detection algorithm. This helps determine the points of obstacles. This also helps the car see if there is an object at its front.

We will be using the results obtained from processing the sensor data to decide the car's (we will be using the catvehicle in this case as an example for the autonomous vehicle) motion requirements. For testing purposes, we decided to give the catvehicle a constant velocity of 1.

9 Task Allocation and Breakdown

Requirement	Team Member	Time to Complete	Test strategy
1: Object detection	Megha, Ajay	2 weeks	Place known test objects some distance away
2: Classification of detections as single or multiple objects	Lenny, Ajay, Megha	3 weeks	Place test objects of known dimensions and quantities.
3: Possibility of path detection	Megha, Ajay, Lenny	2 weeks	Checking co-ordinates of all stored obstacles and look for path.
4: Best possible route determination	Megha, Ajay, Lenny	3 weeks	Verify result of node with a standard run of the algorithm.
5: The best possible route is taken	Megha, Ajay, Lenny	2 weeks	Check actual path taken and test against generated path.
6: Obstacle stays a safe distance away	Ajay	1 weeks	Check distance of path from closest obstacle co-ordinates.
7: Distance Estimation	Lenny, Megha	2 weeks	measure actual distance travelled and compare to generated estimated distance.
8: Time Estimation	Lenny, Megha	2 weeks	measure actual time taken and compare to generated estimated time.

Table 1. Task Allocation and Breakdown Table. This is used to allot the tasks to the team effectively

10 Timeline for Completion

Week	Weekly Project Goal	Project Deadline	Project Deliverables
01/29/2017 - 02/04/2017	Complete Task 1	1/31/2017 (5pm)	Task 1
02/26/2017 - 03/04/2017	Research about potential projects and finalize a project topic with a list of requirements.		
03/05/2017 - 03/11/2017	Complete the project design document and task 2 of the catvehicle challenge.	03/11/2017 (5pm)	Project Design Document, Task 2
03/12/2017 - 03/18/2017	Implement part of the simplesolution node. (B requirements)	03/11/2017 (5pm)	Project Design Document, Task 2
03/26/2017 - 04/01/2017	Complete simplesolution node with testing. Implement part of the pathfinder node. (B requirements)	03/31/2017 (5pm)	Task 3
04/02/2017 - 04/08/2017	Complete part of the pathfinder node implementation with testing. (B requirements)		
04/09/2017 - 04/15/2017	Submit the beta release. (B requirements). Test all the B requirements.	04/14/2017 (5pm)	Beta Release
04/16/2017 - 04/22/2017	Complete the distance and time estimator parts of the pathfinder node with testing (A requirements).	04/21/2017 (5pm)	Requirements Verification
04/23/2017 - 04/29/2017	Test all the 'A' and 'B' requirements.		
04/30/2017 - 05/03/2017	Buffer period for unexpected delays.	05/03/2017 (5pm)	Final Release

Table 2. This is the Timeline table. These are the deadlines by which the different parts of the project outlined must be complete.

11 Global/Shared Tasks and Experience

11.1 Member Qualifications

Ajay: Experience in handling sensor data, a little experience with computer algorithms.
Megha: Strong in computer algorithms.
Lenny: Has taken a variety of projects, so can adapt well to new tasks.

11.2 Justification of task allocation

We can see from the task allocation table that the tasks have been divided according to the strength of the individuals in the team. Since Ajay has experience with handling path finding algorithms, he will be taking the lead in tasks 3, 4 and 5 with help from Lenny due to the complexity of the requirements. We can see the more object detection and classification related portions like task 1 and 2 will have Megha take the lead due to her experience with sensor data and Lenny takes the lead in task 6, 7 and 8.