DDCAR

Tianyang Chen e-mail: (tianyangchen@email.arizona.edu)
Yuanzhengyu Li e-mail: (yuanzyli@email.arizona.edu)
Yawei Ding e-mail: (yaweiding@email.arizona.edu)

ECE573 – Software Engineering Concepts
Spring 2017
Instructor: **Matt Bunting**
March 11, 2017

THE UNIVERSITY OF ARIZONA®

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

DDCAR

1 Executive Summary

1.1 Introduction

In this project, we focus on the control of the vehicle in ROS and realize some function. We plan to create a world map with some objects(barriers). Then, we are going to design a vehicle that can run by itself and avoid barriers. After its exploration, we are able to make a draft map for an unfamiliar environment. Besides, we are able to recognize the objects in the map with the help of machine learning.

1.2 Methodology

Control system and machine learning.

1.3 Goals

Vehicle self-control, avoid barriers.
Create an environment map
Recognize the objects

2. Project Overview

In this project, our team will create ROS software components that controls and runs on a real self-driving car. By using our design and algorithm, the self-driving car could automatically adjust the velocity and direction to avoid barriers. After driving through the real world, the car also could generate a 3D simulation world that mirrors the real world. In our project, we will use the car's front laser sensor to detect and record all the data about the barriers and tracks. Meanwhile, according the data from the laser sensor and our algorithm, the car will change the velocity and direction in real-time by using the car's velocity topic. The last part is object recognized and generating a world file. We will use the source libraries such as OpenCV and image analyze knowledge to do object recognized. First, using label-connected components to detect the number of the barriers. Second, using the data we collect from the laser sensor to calculate the size of each barriers and the proportion between length and width. Third, using the size and shape and our algorithm to recognized barriers. After that, according the coordinate conversion algorithm, we will calculate the position of the barriers in the world file.

3 Analysis and Models

3.1 Requirements

Difficulty:
1 - easy; 2 - difficult, 3 - challenging.
Quantitative Specification:
25% - rare, 50% - sometimes, 75% - usually, 100% - always.

3.1.1 List of B requirements

1. The DDCAR can avoid all the barriers on the track
   Difficulty: 2
   Quantitative Specification: 100%
2. The DDCAR can automatically stop
   Difficulty: 1
   Quantitative Specification: 100%
3. The DDCAR can detect the numbers of the barriers in the visual field of the vehicle
   Difficulty: 2
   Quantitative Specification: 100%
4. The DDCAR can generate a mirror world of real world, which drives through.
   Difficulty: 2
   Quantitative Specification: 100%

3.1.2 List of A requirements

5. The DDCAR can recognize the sizes and the types of the barriers. The types in our project are House, Jersey barrier and Construction cone.

Difficulty: 3
Quantitative Specification: 75%

6. The DDCAR can detect the positions of the barriers as the real world in its visual field
Difficulty: 3
Quantitative Specification: 75%

3.1.3 Dependencies among Requirements

Table 1 Requirement Category and Difficulty

Requirement number	Category	Difficulty
1	B	1
2	B	2
3	B	2
4	B	2
5	A	3
6	A	3

3.2 Domain Analysis

Generally speaking, the vehicle need to build a map model using the data from laser sensor when the vehicle in a totally unfamiliar environment. Then the vehicle can utilize the map to navigate and move. We can use three packages in ROS: move_base package, gmapping package and amcl package.

The structure of the navigation stack setup is shown as following picture:

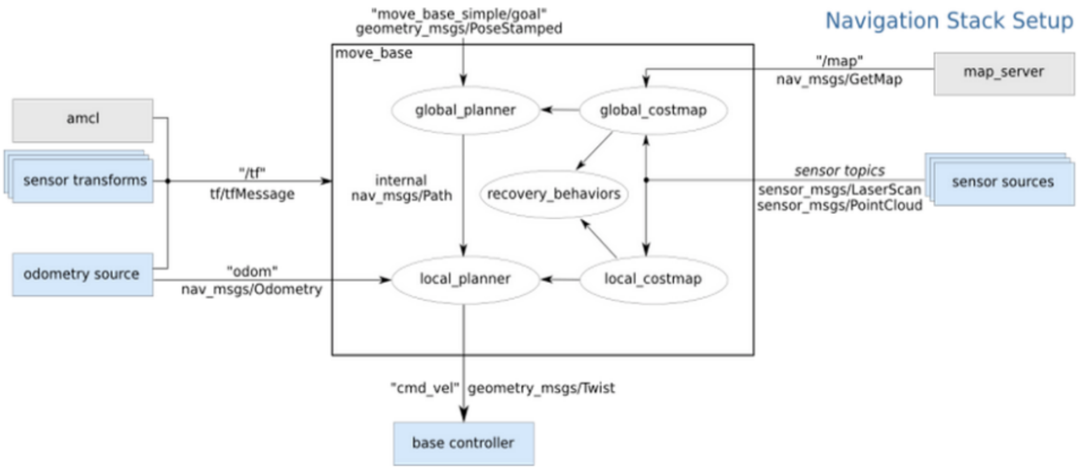

Fig.1 Structure of Navigation Stack

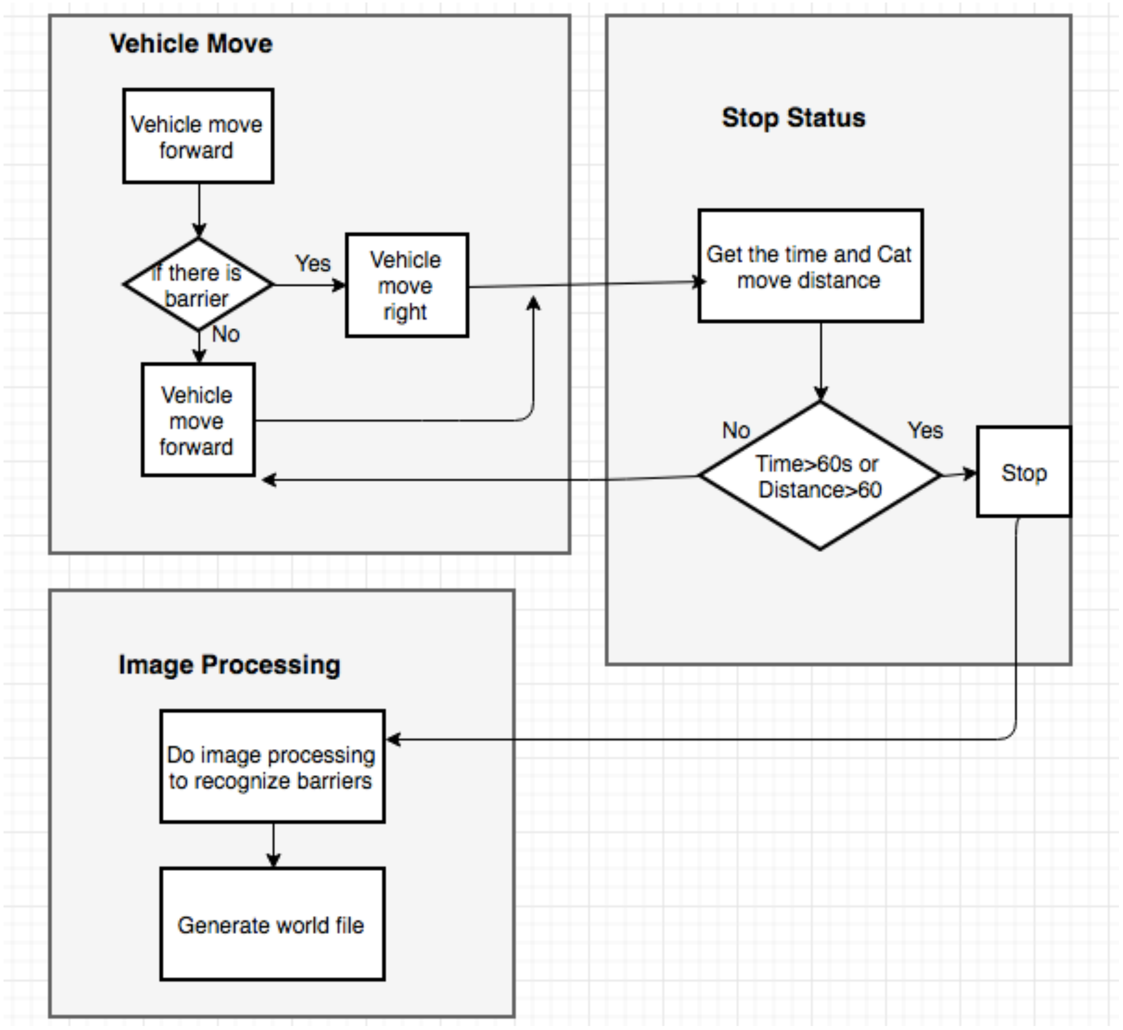

Fig.2 Diagram of interaction

The white box shows the necessary node we should use in ROS. The grey boxes show the node we can choose according to our need in ROS. The blue boxes show the node associating with the vehicle.

The sensor transforms node is used for coordinates transformation. Because the vehicle's center is not exactly the position of the sensor. The sensor sources node is used to input the sensor data. The odometry source is used to input odometry data (tf, nav_msgs/Odometry_message). The base controller node is responsible for encapsulating specific linear speed and angular speed (Twist) and sending them to the vehicle. The map_server node is not necessary when there is no barriers on the ground. But considering the actual situation, we need this node. In ROS, we can use costmap_2d package to build a 2D or 3D map.

3.3 Important Algorithms

1) Avoid barriers:
After getting the data from the front-laser sensor, we calculate the range between the barriers and our car. Using our algorithm to adjust the linear-velocity and angular velocity and also decide in which situation the car go forward, turn left, turn right and backward.
2) Car auto-stop:
Through setting the move time and move distance to make the car stop. This means the car will not stop when meet the barriers, it will find the correct way to running
3) Object detector and recognize:

Using Image Analysis knowledge, by getting the data from the front-laser sensor we will generate a picture, which shows all the data of the barriers and the position for each barrier. First, use image morphology, dilation and erosion, to processing our picture to getter better picture. Second using label connect components algorithm get the number of the barriers. Third, using image segmentation algorithm to recognize and classifier the shapes and types of the barriers.

4 Design and Test

4.1 Class Design

easy_run:

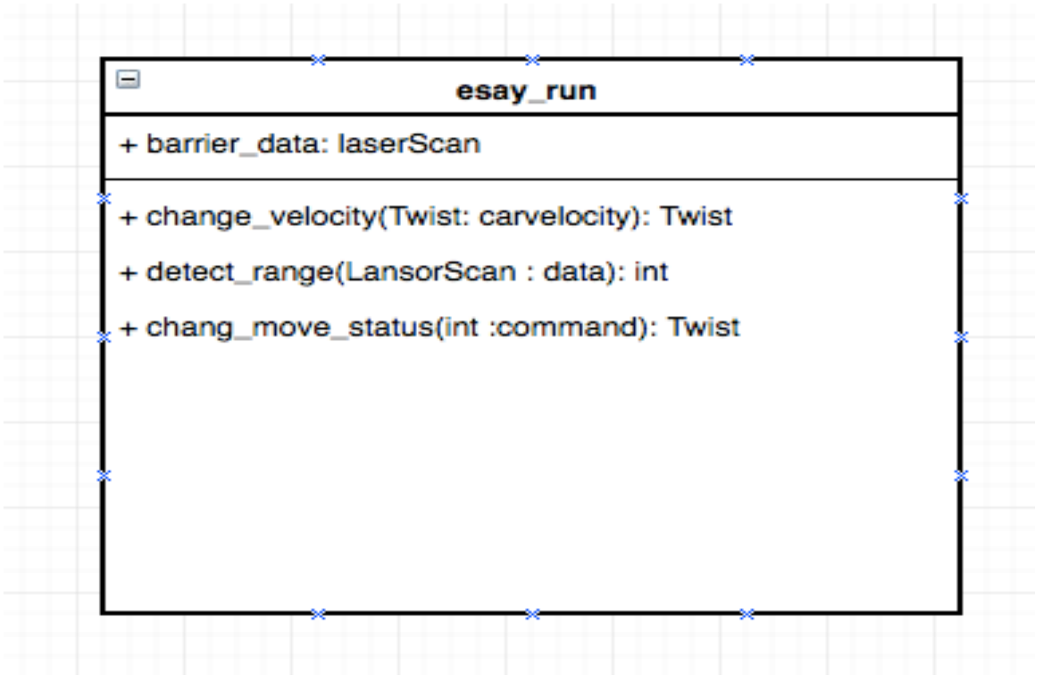
esay_run
+ barrier_data: laserScan
+ change_velocity(Twist: carvelocity): Twist
+ detect_range(LansorScan : data): int
+ chang_move_status(int :command): Twist

Fig.1 Control car's velocity, running direction class

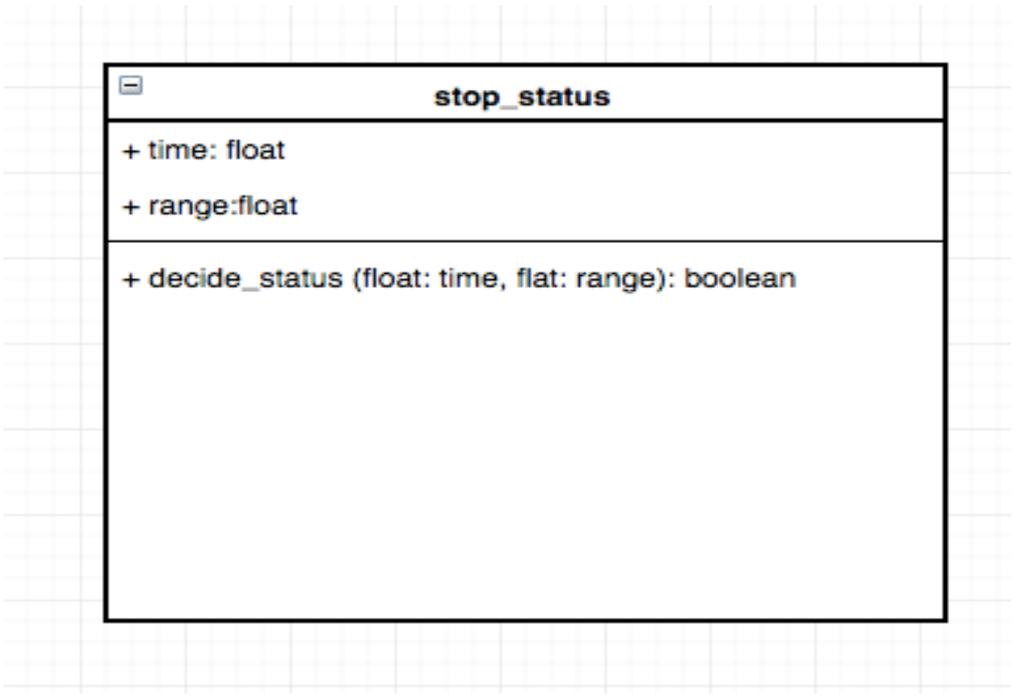

Fig.2 Control car stop class

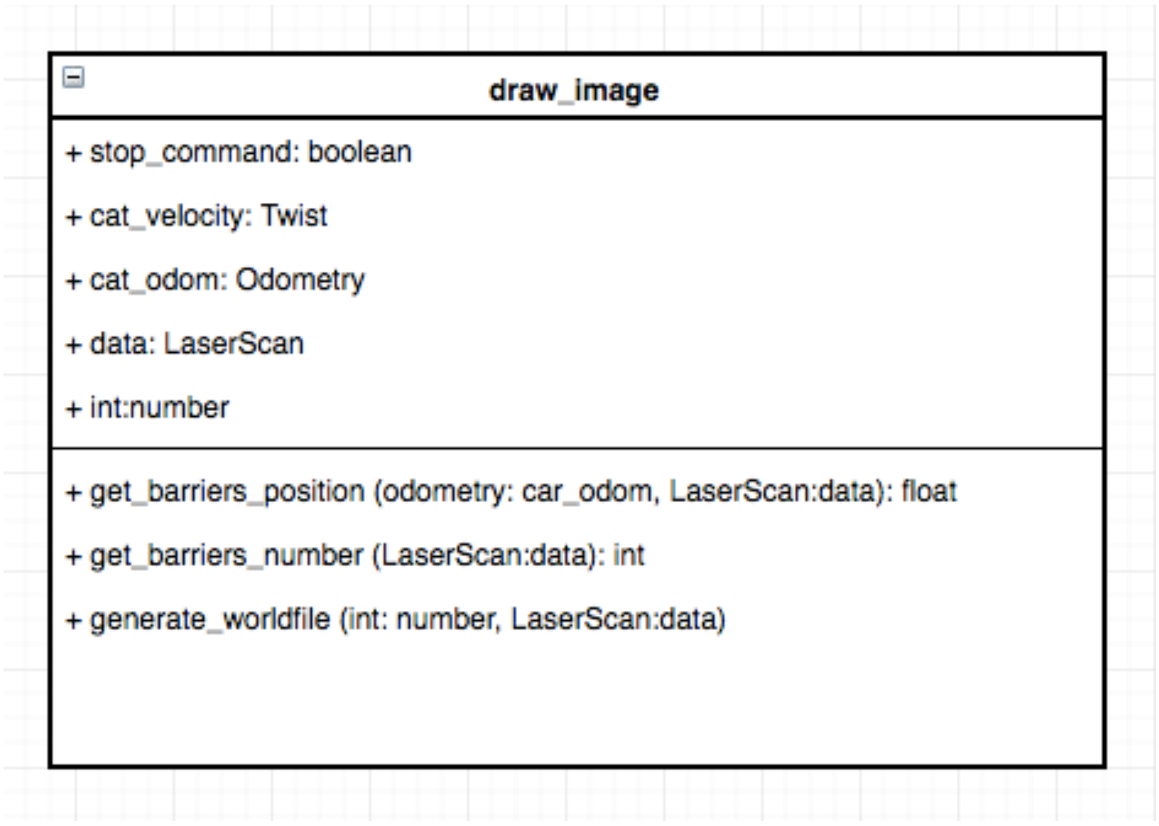
draw_image
+ stop_command: boolean
+ cat_velocity: Twist
+ cat_odom: Odometry
+ data: LaserScan
+ int:number
+ get_barriers_position (odometry: car_odom, LaserScan:data): float
+ get_barriers_number (LaserScan:data): int
+ generate_worldfile (int: number, LaserScan:data)

Fig.3 Generate world file class

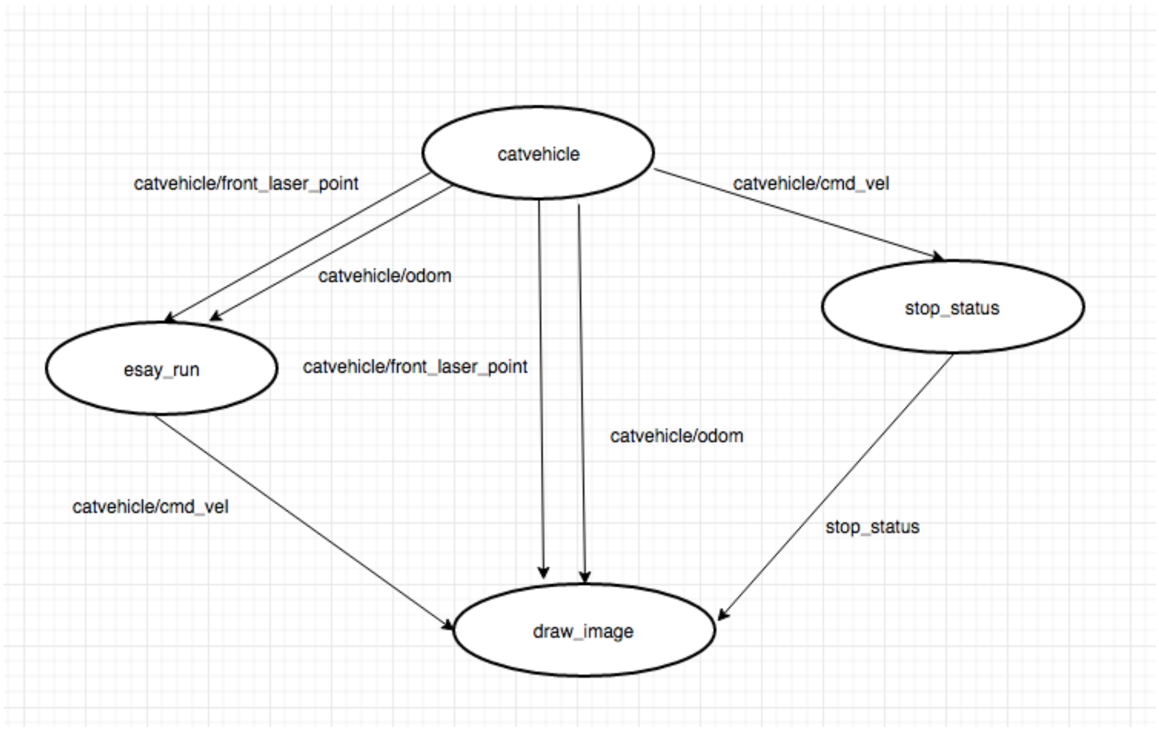

Fig.4 Class Diagram

4.2 Testing Strategy

4.2.1 Tests for B requirements

Requirement 1: avoid all the barriers automatically while running
Validation 1: Run test "test_vehicle_safty"
Test 1: This unit test subscribe to topic "/catvehicle/front_laser_points", and the laser sensor was assembled at the front of the vehicle, which is about 2.4 unit distance from the center of the vehicle. During the running process, the test compute the minimum distance of the points along vehicle heading direction. If at any time this minimum distance is less than 2.6 unit distance, the vehicle was considered as an accident.

Requirement 2: automatically stop
Validation 2: Run test "test_vehicle_stop"
Test 2: This unit test subscribe to topic "/catvehicle/vel". Because the running strategy asks the vehicle to run at most 25 seconds, after 30 seconds this test will compare the velocity with 0, the comparison ensures the velocity is within +/- 0.01 of zero.

Requirement 3: number of the barriers as expected
Validation 3: Run test “test_number_of_barriers”
Test 3: This unit test need to set and expected number of barriers. Our project will publish a topic “/barrier_cnt” with type “Int32” which describes the number of barriers in the visual field of vehicle. If the data of this topic is the same as the expected number, this test was considered as success.

Requirement 4: generate a mirror world
Validation 4: Go to directory ~/.ros
Test 4: If this directory contains a file named “gazebo_output.world”, this requirement is verified.

4.2.2 Test for A requirements

Requirement 5: recognize the type of the barriers
Validation 5: Run test “test_barrier_type”
Test 5: This unit test need to set an array that contains the expected name of the barriers. This test will read the generated file “gazebo_output.world” and store the name of the barriers in an array. And compare this array with the expected array, if more than 75% of the items are the same, this test was considered as success.

Requirement 6: position of the barriers
Validation 6: Run test “test_barrier_position”
Test 6: This unit test need to set an numpy array that contains the coordinates of the center of all the barriers. This test will read the generated file “gazebo_output.world” and store the coordinates of all the barriers, for each barrier, if the coordinate is within +/- 2 unit distance of the expected coordinate, the position of the barrier is considered correct. If more than 75% of the barriers have correct coordinates, this test is passed.

4.3 Integration with Platform

There are three nodes in our project:

1, easy_run: Control the velocity and direction of our car

a) Publish the angular and linear velocity to catvehicle/cmd_vel topic.

b) Subscribe the catvehicle/front-laser_points topic to get the data of barriers.

c) Subscribe the catvehicle/odom topic to get the position of the car.

2,draw_image: Draw the mirror world file.

a) Subscribe from the catvehicle/front-laser_points topic and catvehicle/odom topic to get the data of barriers and car's position to generate picture and do image analysis.

b) Subscribe the stop_status topic to know when the car stop and begin to draw picture.

3, stop_status: Control the car to stop

a) Publish the command to make the car stop.

b) Subscribe from catvehicle/cmd_vel topic to get the angular and linear velocity.

Message types of topics:

1.The message type of catvehicle/cmd_vel topic: geometry_msgs/Twist.

2.The message type of catvehicle/front-laser_points topic: sensor_msgs/LaserScan.msg

3.The message type of catvehicle/odom topic: nav_msgs/Odometry

4. The message type of stop_status: boolean

5 Implementation Plan

5.1 Task allocation and breakdown

Tianyang Chen: Vehicle self-control, avoid barriers.
Requirements:1,2,6
Yuanzhengyu Li: Create an environment map

Requirements:4,6

Yawei Ding: Recognize the objects
Requirements:3,5,6

5.2 Timeline for completion

Requirements (list any requirements have to finished before Alpha): 3/15

Alpha Release with Requirements (list requirements have to finished): 3/25

Requirements (list requirements have to finished): 4/5

Beta Release (With all Requirements): 4/14

Final Release: 5/3

5.3 Global and Shared Tasks

The requirement 6 will be global tasks since it connect each of us work

Testing of the overall project is also a global task

Finally, all the group members will combine their work together and write tests for the project.