Final Report - 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**
May 2nd, 2017

THE UNIVERSITY OF ARIZONA®

Arizona's First University.

College of Engineering

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

Contents

1 Executive Summary……………………………………………3

2 Project Overview………………………………………………3

3 Analysis and Models…………………………………………...4
3.1 Requirements…………………………………………………...4
3.2 Domain Analysis………………………………………………5
3.3 Important Algorithms…………………………………………..6

4 Design and Test………………………………………………...7
4.1 Class Design……………………………………………………7
4.2 Testing Strategy and Requirements Verification………………....9
4.3 Integration with Platform………………………………………11

5 Implementation Plan…………………………………………...12
5.1 Task allocation and breakdown…..………………………………12
5.2 Global tasks and experience………………………..…………..12
5.3 Timeline for completion…………………………..……………12

List of Figures

1 Diagram of interaction
2 Control car's velocity, running direction class
3 Control car stop class
4 Generate mirror world class
5 The Interaction of Classes Diagram

List of Tables

1 Dependencies among requirements

1 Executive Summary

1.1 The idea

We plan to make a software system to control Catvehicle in ROS, which is a self-driving car that can avoid all the barriers automatically. Besides, the vehicle can recognize the objects in its visual field and generate a 3D simulation world that mirrors the real world.

1.2 Methodology

Firstly, we need to publish and subscribe the velocity and position of the vehicle. Secondly, we can get the data from the laser sensor in the vehicle. Thirdly, according to all the information and data we get, we need to implement an algorithm to control the vehicle to avoid the barriers. Fourthly, we can generate an image, which records the position and contour of each barrier and use image analysis to recognize the objects. Finally, we can generate a world file, which is a 3D simulation world that mirrors the world that our car has driven through.

1.3 Goals

Vehicle self-control, avoid barriers.
Recognize the objects.
Create a 3D mirror world

2. Project Overview

In this project, our team created some ROS software components. We can run this system on a real self-driving car and 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. Meanwhile, according to the data from laser sensor and using our algorithm, the car will control the velocity and direction in real-time by using "/catvehicle/cmd_vel" topic. The other part is to detect objects and generate a world file. We will use open source libraries such as OpenCV and image analysis knowledge to do object recognition. First, we will generate an image which shows the position and contour of all the objects. Using connected components labeling to detect the number of the barriers. Second, using the data collected from the laser sensor to calculate the size of each barrier and the proportion between length and width. Our algorithm will estimate the object type by this proportion. After that, according to 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. DDCAR can avoid all the barriers automatically while running.
   Difficulty: 2
   Quantitative Specification: 100%
2. DDCAR can automatically stop in the end.
   Difficulty: 1
   Quantitative Specification: 100%
3. DDCAR can detect the number of the objects in the visual field of the vehicle as expected.
   Difficulty: 2
   Quantitative Specification: 100%
4. DDCAR can generate a mirror world of real world.
   Difficulty: 2
   Quantitative Specification: 100%

3.1.2 List of A requirements

5. DDCAR can recognize the type of the objects. These types in our project are House, Jersey barrier and Construction cone.
   Difficulty: 3
   Quantitative Specification: 75%
6. DDCAR can detect the position of the objects in its visual field as the real world.

Difficulty: 3
Quantitative Specification: 75%

3.1.3 Mapping of Requirements

When the program started, our program can control the velocity of the vehicle to avoid all the barrier on the map (Req1). After the travel finished, the vehicle would stop automatically (Req2). After 5 seconds since the car stopped, we would generate a world file (Req4). However, In order to generate a correct world file. Our program must detect the correct position of barriers, the correct numbers of barriers and the correct types of barriers (Req3, 5, 6).

3.1.4 Dependencies among Requirements

Table 1 - Dependencies among requirements

Requirement	Requirements dependent on	Description
3	1, 2	To test how many barriers in the map, the vehicle must avoid all the barriers and could stop in the end.
4	2	To test whether the project could generate a world file, the vehicle should stop first.
5, 6	1, 2, 3, 4	To test the type and position of objects, the vehicle should avoid all the barriers and detect the number of objects as expected.

3.2 Domain Analysis

Generally speaking, the vehicle need to build a map model using the data from front laser sensor when the vehicle is in a totally unfamiliar environment. In future work, the vehicle can utilize the map to navigate and move. When the vehicle is moving, if the distance between the car and barrier is less than 10m and the barrier is in the 50 degrees visible field of car, the car will turn left. Because of inertia, if the barrier is within 4m of the vehicle, the vehicle will move backward. In other situation, the vehicle still move forward. Moreover, if the running time is longer than 25s or the distance between the current point and the start point is longer than 30m, the vehicle will stop. After 5s since the vehicle stopped, the system will begin to generate a world file.

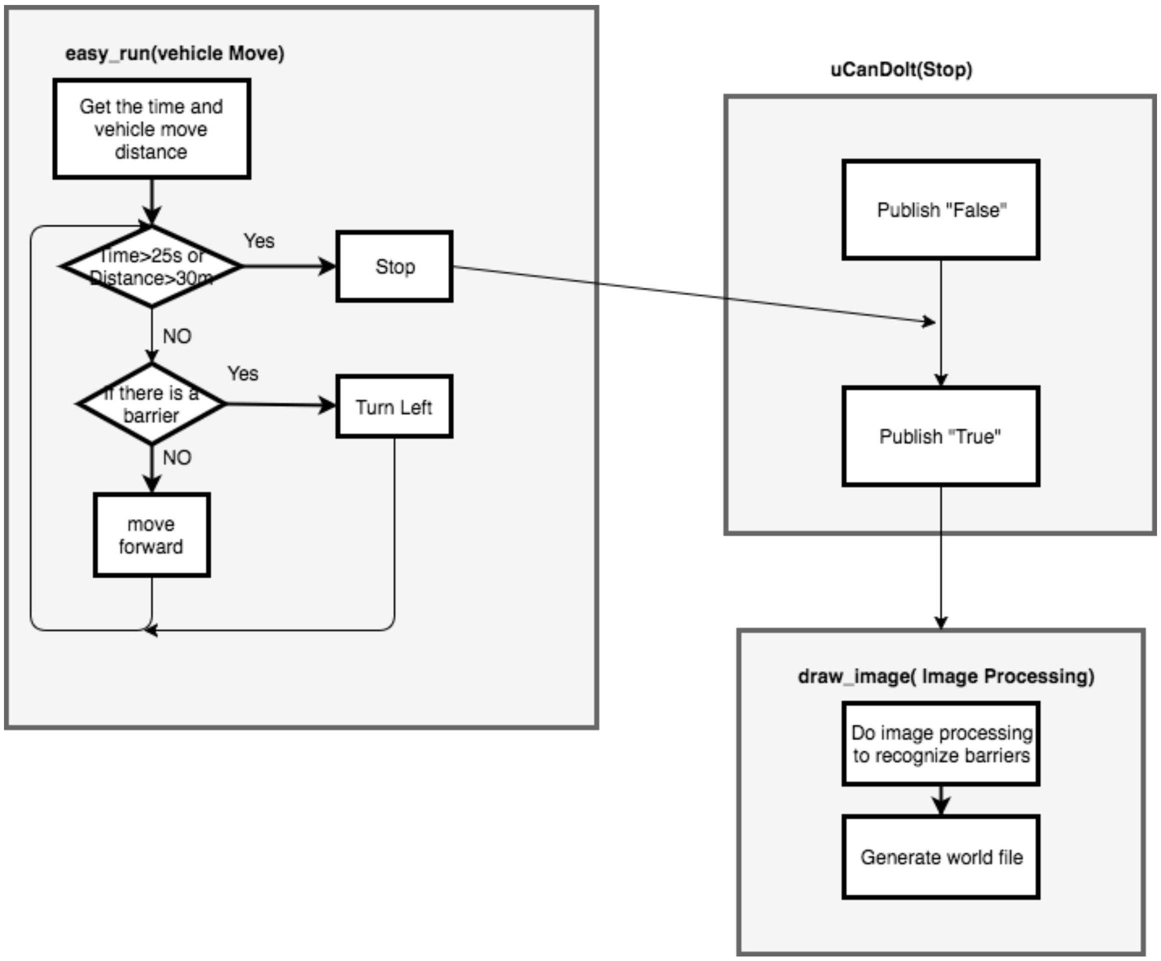

Fig.1 Diagram of interaction

3.3 Important Algorithms

1) Avoid barriers:
After getting the data from the front laser sensor, we calculate the distance between the barriers and the vehicle. We will implement our algorithm to decide whether the vehicle should move forward, turn left or move backward, and adjust the linear-velocity and angular velocity.
2) Automatically stop:
By setting the upper limit time and upper limit distance to make the car stop, if the vehicle exceed any of these limits, the vehicle will stop.
3) Object detection and recognition:

Using Image Analysis knowledge, by getting the data from the front laser sensor, we will generate an image, which shows all the data of the objects. First, we use image morphology, dilation and erosion, to processing our image to getter a better image. Second, we use connect component labeling algorithm to get the number of the barriers. Third, we use image segmentation algorithm to recognize the type and positon of the all objects.

4 Design and Test

4.1 Class Design

There are three main classes of project:

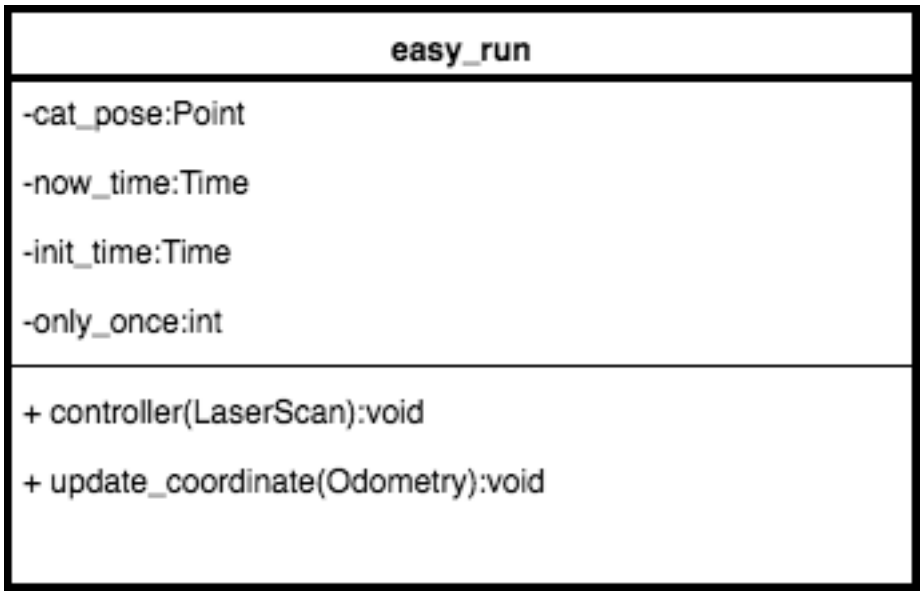

Fig.2 Control car’s velocity, running direction class

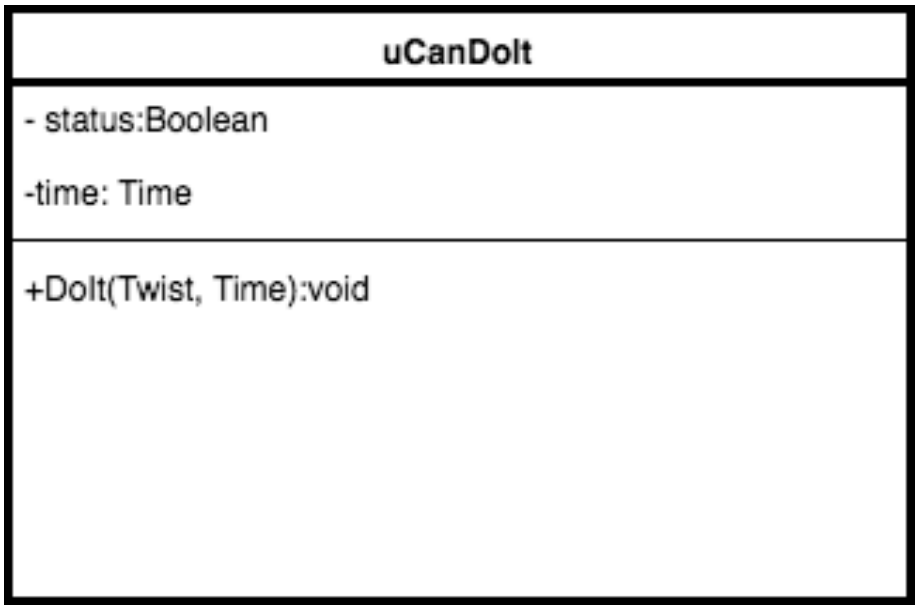

Fig.3 Control car stop class

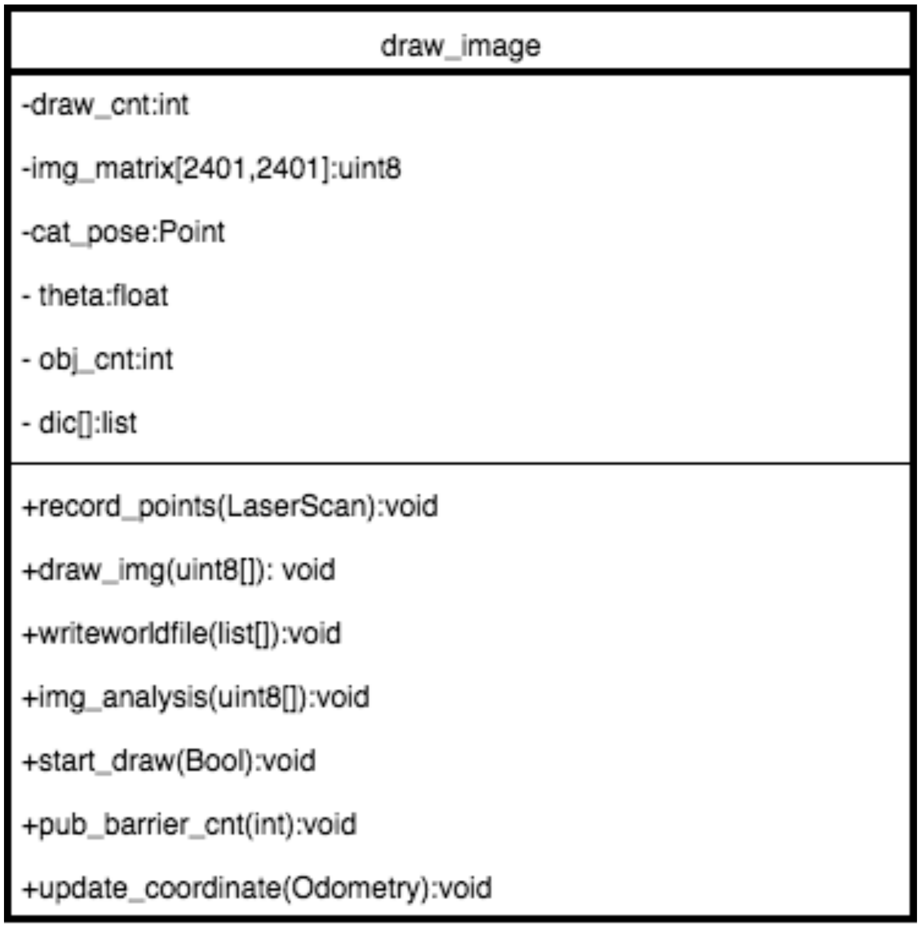
draw_image
-draw_cnt:int
-img_matrix[2401,2401]:uint8
-cat_pose:Point
- theta:float
- obj_cnt:int
- dic[]:list
+record_points(LaserScan):void
+draw_img(uint8[]): void
+writeworldfile(list[]):void
+img_analysis(uint8[]):void
+start_draw(Bool):void
+pub_barrier_cnt(int):void
+update_coordinate(Odometry):void

Fig.4 Generate mirror world class

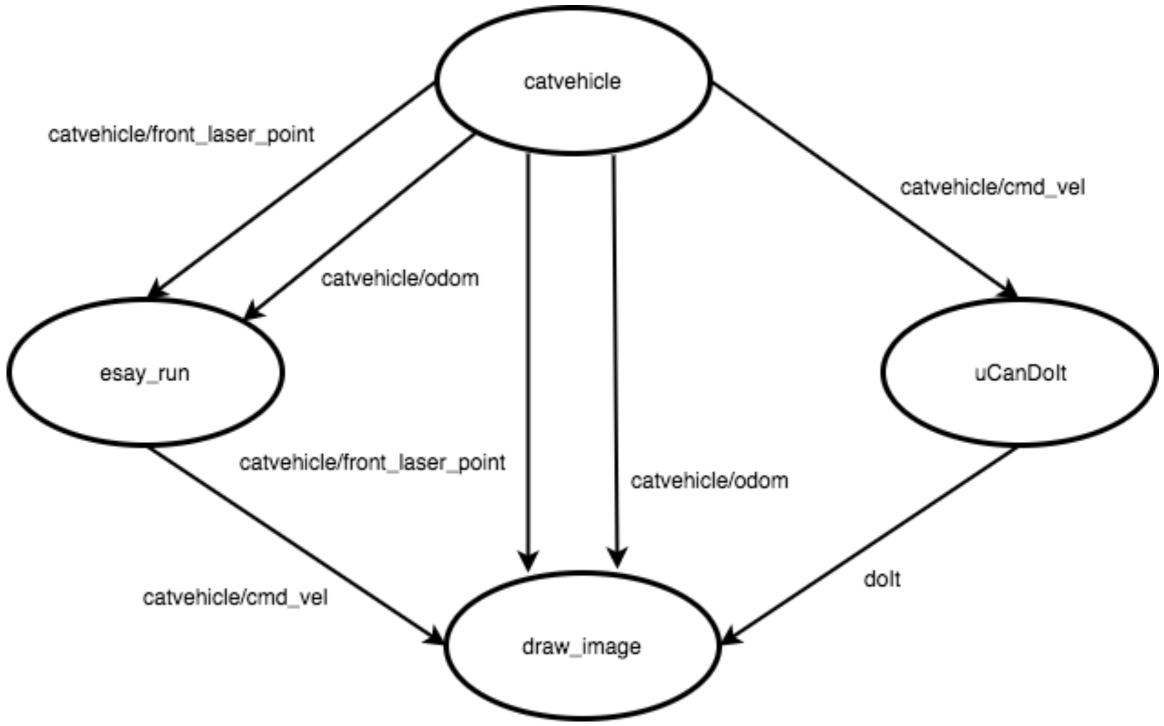

Fig.5 The Interaction of Classes Diagram

4.2 Testing Strategy and Requirements Verification

4.2.1 Tests for B requirements

Requirement 1: DDCAR can avoid all the barriers automatically while running.
Validation 1: Run test "test_safty"
Test 1: This unit test subscribes 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 computes the minimum distance of the points along vehicle's heading direction. If in any time this minimum distance is less than 2.6 unit distance, the vehicle would be considered as having an accident.

Requirement 2: DDCAR can automatically stop in the end.
Validation 2: Run test "test_vehicle_stop"
Test 2: This unit test subscribes 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: DDCAR can detect the number of the objects in the visual field of the vehicle as expected.
Validation 3: Run test “test_number_of_barriers”
Test 3: This unit test need to set the expected number of barriers in parameter "expected_barrier_num". In the project, we will publish a topic “/barrier_cnt” with type “Int32” which describes the number of barriers the vehicle detected. If the number of the barriers this topic given is the same as the number of barriers in the world file, which we use to test. This test was considered as passed.

Requirement 4: DDCAR can generate a mirror world of real 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: DDCAR can recognize the type of the objects. These types in our project are House, Jersey barrier and Construction cone.
Validation 5: Run test “test_object_type”
Test 5: This test need to set an array “expected_objects” that contains the expected name of the objects before running the test. This test will fetch the parameter “detected_objects”, which is used to generate the world file, from ROS Parameter Sever, and store the name of the detected objects in an array. Then we compare this array with the expected array. If more than 75% of the objects are the same, this test was considered as passed.

Requirement 6: DDCAR can detect the position of the objects in its visual field as the real world.
Validation 6: Run test “test_object_pose”
Test 6: This unit test need to set an array “expected_objects” that contains the coordinates of the center of all the objects. This test will fetch the

parameter “detected_objects”, which was used to generate the world file, from ROS Parameter Sever, and store the coordinates of all the objects. For each object, if the coordinate is within +/- 2 unit distance of the expected coordinate, the position of the object is considered correct. If more than 75% of the objects 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 topic “catvehicle/cmd_vel”.

b) Subscribe to topic “catvehicle/front_laser_points” to get the data of barriers.

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

2. draw_image: Image analysis and generate world file.

a) Subscribe to topics “catvehicle/front_laser_points” and “catvehicle/odom” to get the data of barriers and update vehicle’s coordinate to generate image and implement image analysis.

b) Subscribe to topic “/doIt” to get the status of the vehicle and begin to analyze the image.

3. uCanDoIt: publish the status of the vehicle

a) Publish Boolean data, which denotes the status of the vehicle.

b) Subscribe to topic “catvehicle/vel” to get the linear velocity.

Message types of topics:

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

2.The message type of “catvehicle/front_laser_points”: sensor_msgs/LaserScan

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

4. The message type of “/doIt”: 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

5.3 Global and Shared Tasks

The requirement 6 is a global task since it connect each of our work

Test of the whole project is also a global task.

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

5.2 Timeline for completion

Alpha Release: 3/25

The vehicle can detect objects and represent them by polygons.

Beta Release: 4/14

The vehicle can run all by it self and generate a world file, some requirements have been verified.

Final Release: 5/3

The project is ready to release, all tests are passed, all requirements are verified.