DDCAR

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

1 Requirements

1.1 Difficulty

1 - easy; 2 - difficult; 3 - challenging.

1.2 Quantitative Specification

25% - rare, 50% - sometimes, 75% - usually, 100% - always.

1.3 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%

1.4 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%

2 Verification

Requirement 1: DDCAR can avoid all the barriers automatically while running.
Validation 1: Run test “test_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 having an accident.

Requirement 2: DDCAR can automatically stop in the end.
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: DDCAR can detect the number of the objects in the visual field of the vehicle as expected.
Validation 3: Run test “test_barrier_cnt”
Test 3: This unit test need to set the expected number of barriers in parameter "expected_barrier_num". 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 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.

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 was used to generate the world file, from ROS Parameter Sever, and store the name of the detected objects in an array. Then 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.