KFC AUTONOMOUS CAR

Keeley Criswell(kcriswell@email.arizona.edu)
Frederick Yen(fyen@email.ariona.edu)
Ju Pan(pjokk722@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

KFC AUTONOMOUS CAR

1 Executive Summary

We will design a system to control an autonomous car. Our system will detect objects, and control the cars navigation to avoid hitting those object. The car will be able to decelerate and turn away from the object safely. If the system determines that the car cannot turn away safely, it will bring the car to a stop.

In addition, our system will be able to detect an exit of the world with wall around. If the system finds a exit, the car stops using normal algorithm but escape through the exit to the outside world. To detect and analyze objects, we will use the laser that is mounted on the vehicle. This laser tells us the distance from the nearest object. By analyzing the data from the car, we will be able to determine the shape of the object, and, therefore determine if the object the car detect is an exit.

2 Project Overview

Autonomous car research has gained popularity in recent years, with companies such as Tesla and Google working on their own versions. Because it would mitigate the potential for human error, autonomous vehicles show promise for reducing automobile fatalities. However, the autonomous vehicle problem is an immensely complex one, and no one is close to having an affordable solution that meets safety requirements. Therefore, additional research on controller designs is vital to furthering the progress on autonomous cars.

Our contribution to the overall problem of designing an autonomous car is a small one. Our controller will only have the ability to make a small fraction of the decisions an autonomous car controller must make. However, when considering the safety of human lives, each decision that a controller makes is important. It is easier to accurately test the functionality of complex systems by analyzing small portions of them at a time. Therefore, in order to design a complex system, it is vital to first design a system that is capable of performing a subset of the requirements of the end goal.

In the case of an autonomous car, researchers first must develop a car controller with minimal functionality. Our project will do just that - we will focus on the sub-problems of detecting some potential crashes – when an object is directly in front of a car and when the car is about to turn – and basic object identification – determining if the car meets a exit. In full-scale autonomous car systems, the controller must be able to identify various potential scenarios and objects and react accordingly.

3 Requirements

3.1 B Requirements

B.1 Speed of car while turning should no larger than 1m/s.

B.2 When the object the car detects is 30+ meters away from it, the linear velocity of the car should no larger than 4.

B.3 When the object the car detects is 20-30 meters away from it, the linear velocity of the car should no larger than 2.

B.4 When the object the car detects is 10-20 meters away from it, the linear velocity of the car should no larger than 1.

B.5 The absolute safe distance between the car and the object it detects is 30 meters.

B.6 The quite safe distance between the car and the object it detects is 20 meters.

B.7 Distance of vehicle when starting to turn should be exactly 10 meters.

B.8 The angular speed of the car while detected as safe (not within range of an object) is 0.

B.9 The angular speed of car when turning left should be equal or greater than 3.

B.10 The angular speed of car when turning right should be equal or less than -3.

3.2 A Requirements

A.1 The threshold for the car to decide that there is an exit on right-hand side is equal to 79.9 meters.

A.2 The threshold for the car to decide that there is an exit on left-hand side is equal to 79.9 meters.

A.3 The car will turn right only if it detects a distance greater than 79.9 meters meters 40 times in a row.

A.4 The car will turn left only if it detects a distance greater than 79.9 meters 40 times in a row.

A.5 When the exit is detected, the angular speed when the car begins to turn left is 0.2.

A.6 When the exit is detected, the angular speed when the car begins to turn right is -0.2 .

A.7 The car will stop when it is a distance of 55 meters from the starting point.

4 Domain Analysis

In this project, we are going to implement Matlab Simulink and Robotic Operating System (ROS) to develop our self-driving vehicle. We will produce the primary models in Simulink and use the ROS environment to run simulations and test our requirements. We will specifically use Gazebo and Rviz, which are simulation tools provided by ROS, for simulation and testing. Once we get the expected result in Simulink, we will generate the C++ code, and test our model in the ROS workspace without using Simulink. After this, we will possibly transplant our ROS code to a real vehicle and do the experiment to see if it works in a real-world environment.

Simulink support for Robot Operating System (ROS) enables users to create Simulink models that work with a ROS network. ROS is a communication layer that allows different components of a robot system to exchange information in the form of messages. A component sends a message by publishing it to a particular topic, such as "/odometry". Other components receive the message by subscribing to that topic[1].

Simulink support for ROS includes a library of Simulink blocks for sending and receiving messages for a designated topic. When one simulates the model, Simulink connects to a ROS network, which can be running on the same machine as Simulink or on a remote system. Once this connection is established, Simulink exchanges messages with the ROS network until the simulation is terminated. If Simulink Coder is installed, you can also generate C++ code for a standalone ROS component, or node, from the Simulink model[1].

The following sequence model diagram shows how different nodes communicate with each other. Node **catvehicle** mainly receives speed information from speedController, node **speedController** receives turning information and object distance information and sends linear speed information, node **directionController** receives laser information and object distance information and sends turning direction information, node **stopAfterDistance** receives car's coordination information and sends stop signal, node objectDetector receives laser information and sends object informaiton.

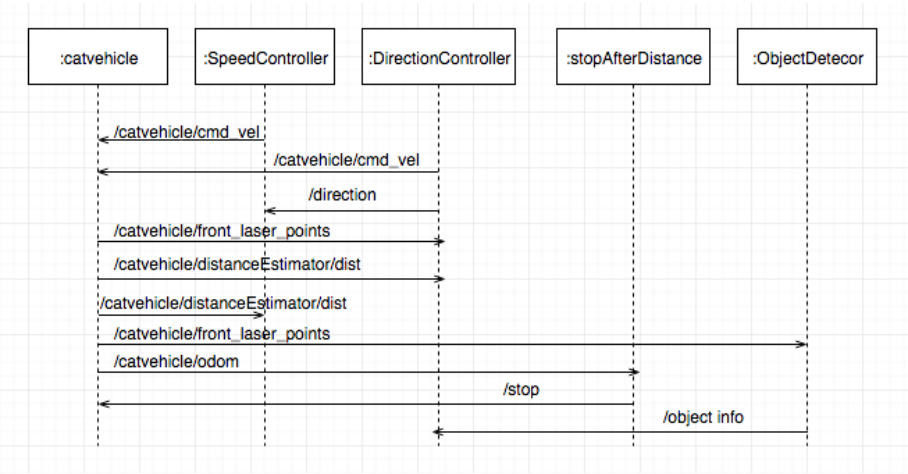

Figure 1. Sequence Diagram

5 Important Algorithms

Direction controlling pseudo code:

```
if(no exit found)
    if(object dist <= 10)
        if(left side has fewer objects)
            turn left
        else
            turn right
    else
        go straight
else
    if(exit is on the right)
        turn right
    else
        turn left
end
```

Speed Controlling pseudo code:

```
if(not turning)
    if(object distance > 30)
        speed = 4
    else if(object distance > 20 && object distance <= 30)
        speed = 2
    else if(object distance > 10 && object distance <= 20)
        speed = 1
else
    speed = 1
end
```

Exit detection pseudo code:

```
if(front_laser_point/range contains more than 40 "80.0" on the right)
    turn right
else if(front_laser_point/range contains more than 40 "80.0" on the left)
    turn left
else
    follow original speed and direction controlling algorithm
end
```

For direction controlling, basically we have 2 parts. The first part is when there is no exit detected. In this situation, the car starts to turn when the distance between itself and the object in front of it is no larger than 10. The turning direction depends on which direction has fewer objects or which direction's objects are further. To achieve this, we need no analyze the "Ranges" array from laser information. There are 180 values in ranges array in total, if the average value of point1 to point90 is less than the average value of point91 to point180, the car chooses to turn right, and vice versa. The second part is when the exit is detected. How does the car decide the object is an exit? We also analyze the "Ranges" array, through experiment, we observer that there are more than forty "80.0" values in point1-point90 group, there should be an exit on the car's right-hand side and if there are more than forty "80.0" values in point91-point180 group, there should be an exit on the car's left-hand side. Notice that if the car detects an exit, it will not keep detecting other objects.

The car controls its speed depends on the distance between itself and the nearest object it detects. Different distance corresponds to different speed. And if the car is turning, the linear speed will stay at a fixed value.

6 Class Design

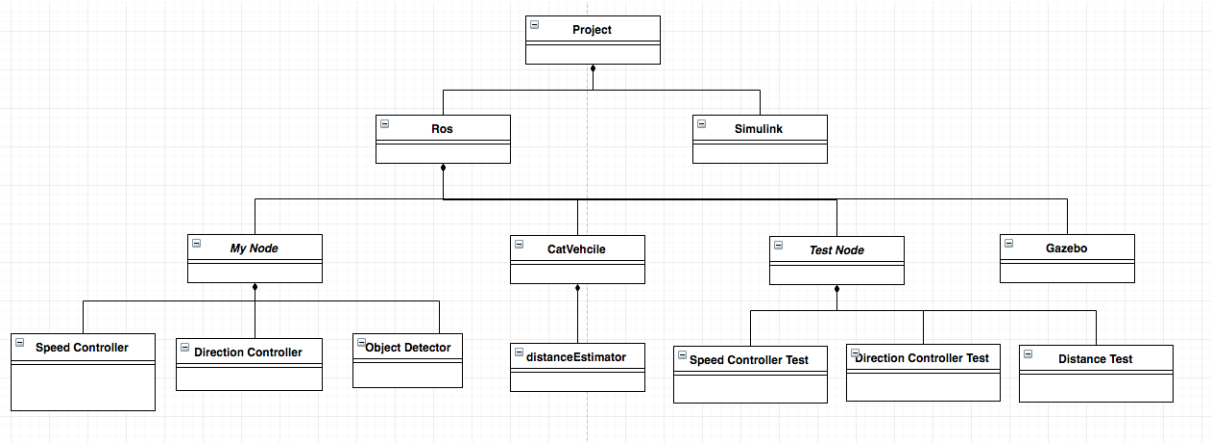

Figure 2. Class Diagram

Since all nodes were developed by Simulink, there should be no class in these nodes. But to present the structure of the whole system, we are still using a class diagram.

7 Testing Strategy

We will implement gtest which is provided by Google to do unit test for our project requirement. Basically, tests are divided into three parts speed test, direction test and distance test. We have come up with a bunch of requirements and designed test for each requirement correspondingly. At the meantime, we will also implement Regression Testing technique. If we find any bugs in our program, we will write new test cases to exhibit these bugs and run tests whenever we the code changes.

We will first test speed and direction simultaneously. We will first do the test for normal case which does not have an exit, after that we will test our application under having exit situation.

We listed basic test cases designed for all requirements we have. Some of the tests are pretty high level, we will break down these high level test into specific code level tests in the future.

7.1 Speed Test Cases

- Comparing the value of /speedTurn topic we subscribed with 1. The comparison ensures the speed of the car while turning is no larger than 1.

- Comparing the value of /speedThirtyPlus topic we subscribed with 4. The comparison ensures the speed of the car while turning is no larger than 4.
- Comparing the value of /speedTwentyToThirty topic we subscribed with 2. The comparison ensures the speed of the car while turning is no larger than 2.
- Comparing the value of /speedTenToTwenty topic we subscribed with 1. The comparison ensures the speed of the car while turning is no larger than 1.

7.2 Direction Test Cases

- Comparing the value of /turnAngleSafe topic we subscribed with 0. Then comparison ensures the angle of the car while detected as safe is 0 degrees.
- Comparing the value of /turnLeftAngleZeroToTen topic we subscribed with 3. The comparison ensures the angular speed of the car when turning left should be equal or greater than 3 degrees.
- Comparing the value of /turnRightAngleZeroToTen topic we subscribed with -3. The comparison ensures the angular speed of the car when turning right should be equal or less than -3.
- Comparing the value of /forceTurnRightRange topic we subscribed with 79.9. The comparison ensures the threshold for the car to decide that there is an exit on right-hand side is equal to 79.9 meters.
- Comparing the value of /forceTurnLeftRange topic we subscribed with 79.9. The comparison ensures the threshold for the car to decide that there is an exit on left-hand side is equal to 79.9 meters.
- Comparing the value of the subscribed topic /forceTurnRightCounter with 40 to be sure that the system counts 40 iterations of distances greater than 79.9 meters before the car begins its turn.
- Comparing the value of the subscribed topic /forceTurnLeftCounter with 40 to be sure that the system counts 40 iterations of distances greater than 79.9 meters before the car begins its turn.
- Comparing the value of the subscribed topic /exitTurningLeftAngle with 0.2 to be sure that when the exit is detected on the left-hand side, the angular speed of the car is 0.2.
- Comparing the value of the subscribed topic /exitTurningRightAngle with 0.2 to be sure that when the exit is detected on the right-hand side, the angular speed of the car is 0.2.

7.3 Distance Test Cases

- Comparing the value of /absSafeDist topic we subscribed with 30. The absolute safe distance between the car and the object it detects should be equal to 30 meters.
- Comparing the value of /quiteSafeDist topic we subscribed with 20. The quite safe distance between the car and the object it detects should be equal to 20 meters.
- Comparing the value of /turnDist topic we subscribed with 10. The comparison ensures the distance of the car starting to turn is equal to 10 meters.
- Comparing the value of the subscribed topic /stopDistance to 55 to be sure that the car has stopped 55 meters from its starting point.

8 Requirement Verification

```
Requirement B.1:  Speed of car while turning should no larger than 1m/
    s
Validation B.1: Run Test B.1
Test B.1: Executes unit tests of speedController node.
ros::Subscriber speedTurn_sub = nh.subscribe("/speedTurn", 100, &
   CallbackHandler::callback, &mCallbackHandler1);
EXPECT_LE(mCallbackHandler1.result,1);
Upon completion, compares value of    /speedTurn  topic we subscribed
   with 1. The comparison ensures the speed of the car while turning is no
    larger than 1.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Requirement B.2:  When the object the car detects is 30+ meters away
   from it, the linear velocity of the car should no larger than 4
Validation B.2: Run Test B.2
Test B.2: Executes unit tests of speedController node.
ros::Subscriber speedThirtyPlus_sub = nh.subscribe("/speedThirtyPlus",
   100, &CallbackHandler::callback, &mCallbackHandler2);
EXPECT_LE(mCallbackHandler2.result,4);
Upon completion, compares value of    /speedThirtyPlus  topic we
   subscribed with 4. The comparison ensures the speed of the car while
   turning is no larger than 4.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Requirement B.3:  When the object the car detects is 20-30 meters away
   from it, the linear velocity of the car should no larger than 2
Validation B.3: Run Test B.3
Test B.3: Executes unit tests of speedController node.
ros::Subscriber speedTwentyToThirty_sub = nh.subscribe("/
   speedTwentyToThirty", 100, &CallbackHandler::callback, &
   mCallbackHandler3);
EXPECT_LE(mCallbackHandler3.result,2);
Upon completion, compares value of    /speedTwentyToThirty  topic we
   subscribed with 2. The comparison ensures the speed of the car while
   turning is no larger than 2.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Requirement B.4:  When the object the car detects is 10-20 meters away
   from it, the linear velocity of the car should no larger than 1
```

```
Validation B.4: Run Test B.4
Test B.4: Executes unit tests of speedController node.
ros::Subscriber speedTenToTwenty_sub = nh.subscribe("/speedTenToTwenty",
   100, &CallbackHandler::callback, &mCallbackHandler4);
EXPECT_LE(mCallbackHandler4.result,1);
Upon completion, compares value of    /speedTenToTwenty  topic we
   subscribed with 1. The comparison ensures the speed of the car while
   turning is no larger than 1.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Requirement B.5:   The absolute safe distance between the car and the
   object it detects is 30 meters
Validation B.5: Run Test B.5
Test B.5: Executes unit tests of speedController node.
ros::Subscriber absSafeDist_sub = nh.subscribe("/absSafeDist", 100, &
   CallbackHandler::callback, &mCallbackHandler1);
EXPECT_EQ(mCallbackHandler1.result,30);
Upon completion, compares value of    /absSafeDist  topic we subscribed
   with 30. The absolute safe distance between the car and the object it
   detects should be equal to 30 meters.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Requirement B.6:   The quite safe distance between the car and the object
    it detects is 20 meters
Validation B.6: Run Test B.6
Test B.5: Executes unit tests of directionController node.
ros::Subscriber quiteSafeDist_sub = nh.subscribe("/quiteSafeDist", 100, &
   CallbackHandler::callback, &mCallbackHandler2);
EXPECT_EQ(mCallbackHandler2.result,20);
Upon completion, compares value of    /quiteSafeDist  topic we
   subscribed with 20. The quite safe distance between the car and the
   object it detects should be equal to 20 meters.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Requirement B.7:  Distance of vehicle when starting to turn should be
   exactly 10 meters
Validation B.7: Run Test B.7
Test B.7: Executes unit tests of directionController node. ros::Subscriber
    turnDist_sub = nh.subscribe("/turnDist", 100, &CallbackHandler::
   callback, &mCallbackHandler3); EXPECT_EQ(mCallbackHandler3.result,10);
Upon completion, compares value of    /turnDist  topic we subscribed
   with 10. The comparison ensures the distance of the car starting to
   turn is equal to 10 meters.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Requirement B.8:   The angle of the car while detected as safe (not
   within range of an object) is 0 degrees
Validation B.8: Run Test B.8
Test B.8: Executes unit tests of directionController node. ros::Subscriber
    turnAngleSafe_sub = nh.subscribe("/turnAngleSafe", 100, &
   CallbackHandler::callback, &mCallbackHandler5); EXPECT_EQ(
   mCallbackHandler5.result,0);
Upon completion, compares value of    /turnAngleSafe  topic we
   subscribed with 0. The comparison ensures the angle of the car while
   detected as safe is 0 degrees.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Requirement B.9:   The turning angle of car when turning left should be
   equal or greater than 3 degrees
```

```
Validation B.9: Run Test B.9
Test B.9: Executes unit tests of directionController node. ros::Subscriber
    turnLeftAngleZeroToTen_sub = nh.subscribe("/turnLeftAngleZeroToTen",
   100, &CallbackHandler::callback, &mCallbackHandler6); EXPECT_GE(
   mCallbackHandler6.result,3);
Upon completion, compares value of    /turnLeftAngleZeroToTen  topic we
   subscribed with 3. The comparison ensures the turning angle of car when
    turning left should be equal or greater than 3 degrees.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Requirement B.10:   The turning angle of car when turning right should be
    equal or less than -3 degrees
Validation B.10: Run Test B.10
Test B.10: Executes unit tests of directionController node. ros::
   Subscriber turnRightAngleZeroToTen_sub = nh.subscribe("/
   turnRightAngleZeroToTen", 100, &CallbackHandler::callback, &
   mCallbackHandler7); EXPECT_LE(mCallbackHandler7.result,-3);
Upon completion, compares value of    /turnRightAngleZeroToTen  topic we
    subscribed with -3. The comparison ensures the The turning angle of
   car when turning right should be equal or less than -3 degrees.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Requirement A.1:   The threshold for the car to decide that there is an
   exit on right-hand side is equal to 79.9 meters
Validation A.1: Run Test A.1
Test A.1: Executes unit tests of directionController node. ros::Subscriber
    forceTurnRightRange_sub = nh.subscribe("/forceTurnRightRange", 100, &
   CallbackHandler::callback, &mCallbackHandler1); EXPECT_EQ(
   mCallbackHandler1.result,79.9);
Upon completion, compares value of    /forceTurnRightRange  topic we
   subscribed with 79.9. The comparison ensures the threshold for the car
   to decide that there is an exit on right-hand side is equal to 79.9
   meters.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Requirement A.2:   The car will decide to turn left if it detects a
   distance of 79.9 meters or greater on its left side
Validation A.2: Run test A.2
Test A.2: Executes unit tests of directionController node.
ros::Subscriber forceTurnLeftRange_sub = nh.subscribe("/forceTurnLeftRange
   ", 100, &CallbackHandler::callback, &mCallbackHandler2);
EXPECT_EQ(mCallbackHandler2.result,79.9);
Compares the value of the subscribed topic    /forceTurnLeftRange  with
   79.9. This way, we are able to verify that the car detects a distance
   greater than 79.9 meters when it is in the vicinity of an exit.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Requirement A3:   The car will turn right only if it detects a distance
   greater than 79.9 meters meters 40 times in a row
Validation A.3: Run test A.3
Test A.3: Executes unit tests of directionController node.
ros::Subscriber forceTurnRightCounter_sub = nh.subscribe("/
   forceTurnRightCounter", 100, &CallbackHandler::callback, &
   mCallbackHandler3);
EXPECT_GE(mCallbackHandler3.result,40);
Compares the value of the subscribed topic    /forceTurnRightCounter
   with 40 to be sure that the system counts 40 iterations of distances
   greater than 79.9 meters before the car begins its turn.
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Requirement A4:   The car will turn left only if it detects a distance
   greater than 79.9 meters 40 times in a row
Validation A.4: Run test A.4
Test A.4: Executes unit tests of directionController node.
ros::Subscriber forceTurnLeftCounter_sub = nh.subscribe("/
   forceTurnLeftCounter", 100, &CallbackHandler::callback, &
   mCallbackHandler4);
EXPECT_GE(mCallbackHandler4.result,40);
Compares the value of the subscribed topic    /forceTurnLeftCounter
   with 40 to be sure that the system counts 40 iterations of distances
   greater than 79.9 meters before the car begins its turn.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Requirement A5:   The angle of the wheels when the car begins to turn
   left is .2 degrees
Validation A.5: Run test A.5
Test A.5: Executes unit tests of directionController node.
ros::Subscriber exitTurningLeftAngle_sub = nh.subscribe("/
   exitTurningLeftAngle", 100, &CallbackHandler::callback, &
   mCallbackHandler8);
EXPECT_EQ(mCallbackHandler8.result,0.2);
Compares the value of the subscribed topic    /exitTurningLeftAngle with
   .2 to be sure that the car turns at an angle of .2 degrees.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Requirement A6:   The angle of the wheels when the car begins to turn
   right is -.2 degrees
Validation A.6: Run test A.6
Test A.6: Executes unit tests of directionController node.
ros::Subscriber exitTurningRightAngle_sub = nh.subscribe("/
   exitTurningRightAngle", 100, &CallbackHandler::callback, &
   mCallbackHandler9);
EXPECT_EQ(mCallbackHandler9.result,-0.2);
Compares the value of the subscribed topic    /exitTurningRightAngle with
   .2 to be sure that the car turns at an angle of .2 degrees.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Requirement A7:   The car will stop when it is a distance of 55 meters
   from the starting point.
Validation A.7: Run test A.7
Test A.7: Executes unit tests of directionController node.
ros::Subscriber stopDistance_sub = nh.subscribe("/stopDistance", 100, &
   CallbackHandler::callback, &mCallbackHandler11);
EXPECT_EQ(mCallbackHandler11.result,55);
Compares the value of the subscribed topic    /stopDistance  to 55 to be
    sure that the car has stopped 55 meters from its starting point.
```

9 Integration with Platform

We have four kinds of sensors on vehicle, two lidars on the car, a spinning one on the top of the car and the front lidar points, a right camera and a left camera. Basically, we want to use two lidars to detect object in front of the vehicle and use two side cameras to determine which direction should the vehicle turn to. For example, if the lidar detects that there is a object in front of the vehicle within the the dangerous distance that we set, the vehicle

have to change the direction. We make the car itself decide which direction it will turn to by implementing a random generator. Lets say the car decide to turn right, however the right camera detects that there is an object on the right side of the vehicle within dangerous distance, the vehicle now wants to turn left. If left camera does not detect anything within dangerous distance, the vehicle will turn left, otherwise, the vehicle will stop moving.

Here are the brief introduction for each sensor which are provided by Cat Vehicle Testbed website.

- **Velodyne HDL- 64e High definition Lidar**

  This sensor is able provide ample information about the surrounding environment of the laser. The sensor is able to synchronize its data with time pulses. The sensor can be used for vehicle navigation, three dimensional mapping, surveying, and industrial automation. Unlike the SICK laser this sensor is able to get a 360o field of view with a frame rate of 5 Hz to 15 Hz, which gives about 1.3 million points per second to the output. Velodyne provides a pre-programmed GPS receiver.

- **SICK Laser Measurement System - LMS291- Firmware for outdoor applications**

  Utilizing its radial field of vision, provided through infrared laser beams, the sensor is able to perform two-dimensional scans. It is a high accuracy non-contact measurement system. The sensor does not require any additional set up materials such as reflectors or position marks. Here is a brief list of some of the tasks that this laser can perform which are relevant to this project:

  - Determining the volumes or contours of bulk materials
  - Determining the volumes of objects (measuring packages, pallets, containers)
  - Determining the position of objects and classifying them
  - Collision prevention for vehicles or cranes
  - Controlling docking processes (positioning)

  One of the reasons that we use SICK laser is because of its features which provide for non-contact optical measurement, even over longer distances. When driving around with an autonomous vehicles objects such as people or other cars may move at high speeds, hence the laser also provides for rapid scanning times.

- **Bumblebee XB3 Stereo Vision digital Camera System**

  This is a sensor with three lenses which enable stereo vision(vision analogous to our eyes). These three lenses provide for a more accurate vision of the objects. The camera includes two main SDK packages: FlyCapture SDK and Triclops SDK. The FlyCapture SDK is used for image acquisition control while the Triclops SDK provides for image rectification and stereo processing. One of the broad applications for which this camera can be utilized for is to extract 3D information in order to acquire data such as positions of objects in two images.

- **PGR FlyCapture**

  This sensor comes with an SDK which works with the cameras to provide a unified API and upon further investigation, there are only 3 function calls required to get an image.

10 Task Allocation and Breakdown

- Fred: Object recognition test cases, object detector node, simulink.
- Ju: Speed test cases, speed controller node, simulink.
- Keeley: Direction test cases, direction controller, simulink.

11 Timeline for Completion

	Date						
Description	03/11	03/15	03/22	03/31	04/14	04/27	05/03
Understand required algorithm and code	x	x					
Node development (Alpha)		x					
Implement test cases		x	x				
Revise design and testing (Beta)				x	x		
Final release					x	x	
Final presentation and report							x

Table 1. Timeline for Completion

12 Global/Shared Tasks and Experience

Fred is assigned tasks regarding to object detection due to his experience with machine learning and unsupervised data clustering. Ju is responsible for PID related control algorithms since he had project experience with control theory and autonomous vehicles. Keeley is responsible for handling the turning decisions and turning algorithms - she has a background in Mathematics and experience programming with object-oriented languages.

References

[1] Mathworks, "Get Started with ROS in Simulink"[Online]: *https://www.mathworks.com/help/robotics/examples/get-started-with-ros-in-simulink.html?requestedDomain=www.mathworks.com*