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**

March 11, 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 car's 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 determine if detected objects are traffic cones. If the system finds a traffic cone, the car will accelerate towards the cone and run it over. 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 it is a traffic cone. After hitting a traffic cone, the car will stop.

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 is near a traffic cone. 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

- Begin the program with a car that is stationary.
- Query the laser every 1/2 second.
- After receiving data from the laster, identify if there is an object in the car's trajectory.
- When an object is detected in front of the car, determine the car's distance from the object.
- If there is an object 40+ feet in front of the car, or if there is no object detected, the car's maximum speed is 10.
- If there is an object 20-40 feet in front of the car, set the car's the car's maximum speed is 5.
- If there is an object 20- feet in front of the car, and the car is currently moving, the car's maximum speed is 2.
- If there is an object 10- feet in front of the car, and the car is currently moving, it will begin to turn.
- If there is an object 10- feet in front of the car, and the car is not moving, the car will continue to not move.
- When the object is 20 feet in front of the car, the car will randomly determine if it should turn left or right.
- After determining a direction to turn, the car will query the laser and camera to determine if there is an object in that direction.
- If an object is not detected, the car will turn 90 deg. in the chosen direction.
- The car will either start turning 10 feet away from the object in front of it, or it will stop within 5 feet of the object in front of it.
- If an object is detected in the direction the car originally wants to turn, query the laser to determine if there is an object in the other direction (i.e. if the car wanted to turn left, but detects an object to the left, determine if there is an object to the right).
- If an object is detected in both directions the car wishes to turn, the car will decelerate and stop within 5 feet of the object in front of it.
- While turning, the car's max speed remains at 2.
- After turning, the car will proceed to drive in a straight line until it encounters another object.

3.2 A Requirements

- When the car is 25 feet from the object, query the laser and use the data to determine if the object is a traffic cone.
- If the object is not a traffic cone, turn away from the object as outlined in the B requirements.
- If the object is a traffic cone, set max speed to 10 and double acceleration.
- When a traffic cone is hit, set max speed to 0.
- The object identification process will time out when the car is 15 feet from an object - i.e. if a decision about if the object is a traffic cone is not made by the time the car is 15 feet from the object, the controller will treat the object as if it is not a traffic cone.

4 Domain Analysis

In this project, we are going to implement Matlab Simulink and Robotic Operating System (ROS) to develop our self-driving vehicle. The interaction model is shown below by using a sequence model diagram. 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].

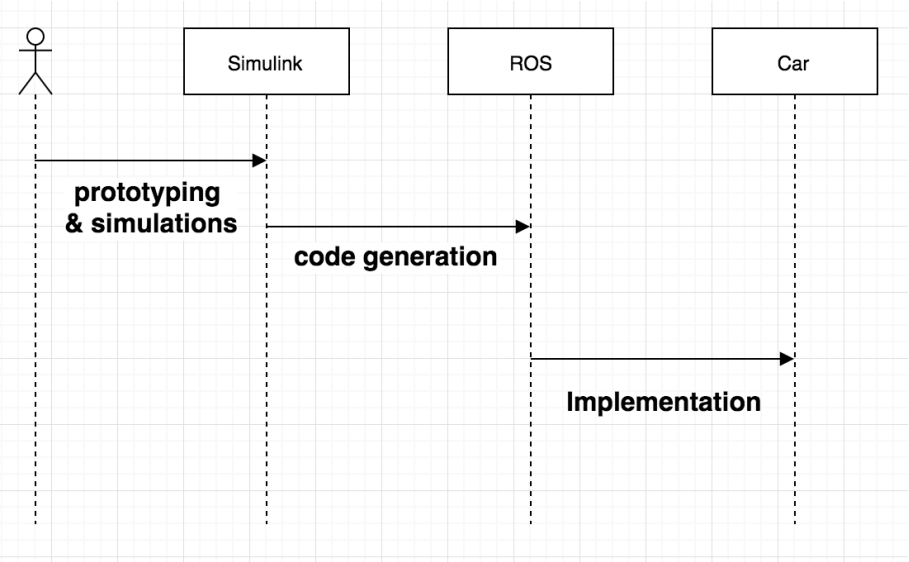

Figure 1. Sequence Diagram

5 Important Algorithms

5.1 Expectation-maximization clustering

EM clustering is used for grouping objects together that are similar to each other and dissimilar to objects that belong to other clusters. It utilizes the EM algorithm, which is an iterative method for finding maximum likelihood (ML) or maximum a posteriori (MAP) estimates of parameters in a statistical model of interest. The model of interest depends on unobserved latent variables. We want to implement this algorithm to classify objects the vehicle meet during the driving process. For example, if the vehicle detect a traffic shape object, it will remember this pattern and if the vehicle meet a traffic cone again, it knows that it met this object before.

5.2 PID controller

A control loop feedback algorithm that is commonly used in industrial control systems for it's feature of robustness. A PID controller continuously calculates an error value (difference between a desired setpoint and a measured variable) and applies a correction with parameters related to proportional, integral, and derivative terms of the error value. We will be using PID control algorithm to make our car speed up and slow down steadily. The reason that we want to implement this algorithm is that we tried to give a speed to vehicle directly before, but the vehicle wrecked sometimes. Imaging you make a static vehicle have a speed suddenly, first that is difficult to achieve unless you have a super sport car, second, it may be harmful to your vehicle and dangerous to others. Therefore, we want to apply PID control algorithm to make our vehicle act smoother.

5.3 OpenCV

An open source library written in C++ that includes programming functions that support tasks targeted for real-time computer vision. Some application areas include: 3D feature toolkits, Human-computer interaction, facial recognition, object identification, etc. We want to use OpenCV library to do the basic image processing things. To make the vehicle be able to remember object pattern, we want to use OpenCV library to analyze the pixel pattern of a certain object and store this pattern.

6 Class Design

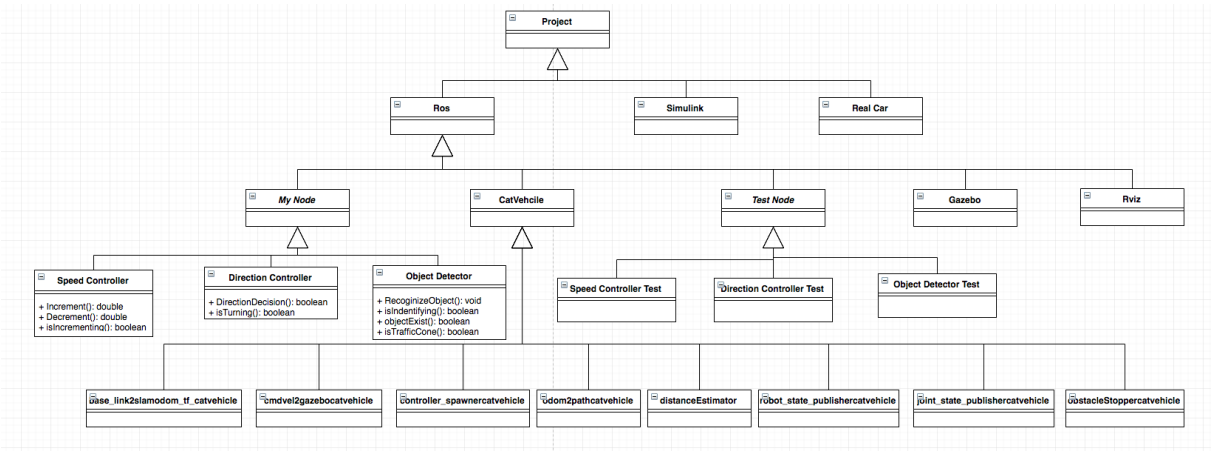

Figure 2. Class Diagram

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 object detection 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. After ensuring our car runs normally with expected speed and direction, we will go ahead to test the object recognition part which is included in requirement A class.

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

- Test case should pass only when original speed of the vehicle is set to 0 and is Identifying() returns true;
- Test case should pass only when query laser twice every second;
- Test case should pass only when speed of the vehicle should no larger than 10 while object is 40+ feet in front of the vehicle;
- Test case should pass only when speed of the vehicle is no larger than 5 while object is 20 - 40 feet in front of the vehicle;
- Test case should pass only when speed of the vehicle is no larger than 2 while object is 0 - 20 feet in front of the vehicle;
- Test case should pass only when speed is no larger than 2 while the vehicle is turning;
- Test case should pass only when speed is incremented at maximum acceleration while detected object is traffic cone;

7.2 Direction Test Cases

- Test case should pass: when object is 0 - 10 feet in front of the vehicle, the vehicle should be in turning state, it means Boolean function isTurning() returns true.
- Test case should pass: when detected object is traffic cone within 25 feet, the vehicle should not be in turning state which means Boolean function isTurning() returns false and .
- Test case should pass: when no object is detected, the vehicle should not be in turning state which means Boolean function isTurning() returns false.
- Test case should pass: when object is 10+ feet in front of the vehicle, the vehicle should not be in turning state, it means Boolean function isTurning() returns true.
- Test case should pass: when isTurning() returns true, turning angle variable should always be equal to 90 degrees.
- Test case should pass: when isTurning() returns false, turning angle variable is always set to 0 degree.
- Test case should pass: when isTurning() returns true and objectExist() returns false.

7.3 Object Recognition Test Cases

- Test case should pass: when laser data received matches our pre-defined traffic pattern, isTrafficCone() returns true and objectExist() returns false.
- Test case should pass: when laser data received does not match our pre-defined traffic pattern, isTrafficCone() returns false and objectExist() returns true;
- Test case should pass: when the car is 15 feet from a object, boolean function isIdentifying() returns false;

8 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. Let's 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.

9 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.

10 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

11 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*