Nathan Lucero

Dr. Kyu Han Koh

Software Engineering

03 December 2019

Autonomous Model Vehicle Final Report

1. Introduction

The topic of self-driving vehicles has always been a topic that grabbed my attention. Big name corporations like Tesla, Google, and Mercedes have already mass produced and sold plenty of their vehicles which are capable of “auto-pilot” driving. My Autonomous Model Vehicle (AMV) was a bit inspired by these companies to see if I can get somewhat of an idea of how both the hardware and software work together. My confidence in assembling my own AMV was directly boosted by a group of STEM students who managed to assemble and code their own model car with its own unique sensors.

My objective was to successfully assemble and code my own AMV to perform two unique modes. Mode one which achieved a line detection functionality by using an infrared module located underneath the car. Mode Two made it possible for the model car to drive in a linear fashion on its own with ultrasonic sensors for obstacle avoidance. The robot car will automatically detect obstacles ahead of its motion with the ultrasonic sensors by sending waves and calculating the distance.

1. Background Research

The book titled “*Getting Started with Arduino”* served as an intermediate manual for the Arduino’s IDE. In chapter three of the book it helped install the actual IDE, the necessary drivers, and explain some working ports on the physical controller which helped during the research stages of the project. Once the IDE was configured, I then proceeded to find some articles to gain some more insight on my project.

I began my research by googling the first autonomous driving vehicle to see if there will be any similarities in my simple AMV. I was then instantly directed to my first article called “*First Results in Robot Road-Following*”. This project was funded by the US department of Defense by a team of engineer sat Carnegie Mellon University. From which they developed and distributed their analysis of their two robots called Terregator and Neptune. They provided algorithms including a stable control scheme for visual surveying.

The Terregator served as their primary road follower having six wheels and multiple cameras. Powered by an ongoing generator it used a 1200 Haud radio link for status display and command inputs. A 10-megahertz microwave link for TV signal from the vehicle to a digitizer. Also, a Mototola 6800 for steering commands and velocity adjustments. A more basic model was called Neptune having only three 3 wheels and two cameras. This model was the first to achieve continuous motion of road following with black electrical tape.

Their primary goal was to keep the vehicle centered on the road as it rolls along at a constant speed. This was done by repeatedly digitizing road images, locating the road edges in the image, calculating the (distance away) from the center line, and steering to realign the vehicle. Their best-found algorithm took sixteen screenshots of the left and right edges, averaged both sides, and kept the best ones that looked like road edges and got rid of those that had no resemblance. Through trial and error they were finally able to eventually find more efficient edge processing techniques and applied them to the Terragator to create a successful line tracker using cameras.

My second article called “*Design of An Arduino-based Smart Car*” which was from another group of engineers located at a University located in China. They managed to construct a toy car controllable via Bluetooth on a Android device and was also capable of obstacle avoidance. Their primary goal was to design a simple, accessible, and user-friendly model vehicle using embedded devices and able to communicate through any Android device. They explain that the Arduino circuit board is designed as an open source electronic prototyping platform providing blueprints and flexible development kits for enthusiastic users who intend to produce their own interactive objects or environments. Assistance of the Arduino IDE made the code easy to upload and compile through a USB transmission line. By touching or pressing on the Android application screen, a manipulator can send the inputted commands to the Arduino microcontroller on the car through Bluetooth and observe the corresponding executions accomplished by the actuators (for example, the motors).

Their first successful mode for controlling the car was operated on from their smartphone via Bluetooth and was used as a remote control. This mode achieved four fundamental functions: Forward, Left, Right, and Backwards by sliding your finger on the application to the desired direction. Also, with the android smartphone this vehicle mode was capable of using Googles Voice search detection algorithm. This algorithm was used to handle the users inputted voice through the smartphones microphone and send those digitized instructions to the Arduino for the vehicle to handle and output the direction said.

Their second successful mode kept the car moving forward until an obstacle appears using ultrasonic sensors within a defined threshold distance. After identifying the obstacle; it will stop and detect distances front-left and front-right directions to determine which offers greater clearance and then proceed in that direction.

The third article I researched was called “*Working model of Self-driving car using convolutional Neural Network, Raspberry Pi, and Arduino*”. This project was coded and assembled by a single person whom studied at a University located in India. His primary goal was to tackle the seriousness of road crashes. Showing that as of 2018:1.3 million people die each year due to road crashes, 3,287 deaths each day, and in India specifically you have a 1 in 37,000 percent chance of being in a road crash, reasons such as drunk driving, texting and driving, and breaking traffic rules are root causes to these crashes.

His method to increase safety in driving was to make a model self-driving vehicle using embedded technology and a simple convolutional neural network algorithm. This convolutional neural network works directly with image processing. The image processing technique he chose has the filter of edge detection. The model proposed, takes an image with the assistance of the Raspberry Pi camera attach on the car. The image captured servers as the inputted image to be compared to as its convolutional neural network filter for edge detection. All images are gray-scaled before passing them on to the neural network for comparison. Once the image is analyzed it then sends a desired instruction to the Arduino. The Arduino receives the digitized analysis and moves the car to a particular direction.

With everything coded and assembled this model was successfully able to drive and detect the edges of a premade track with the help of its hardware and its simple convolutional layer implementation.

1. Methods

The blueprint for the hardware was inspired by an article called “*Design of An Arduino-based Smart Car*”. Their modes included obstacle avoidance and wireless controlling of the car via smartphone. The information from this article revealed that the Arduino was capable of holding different modes in memory. This encouraged me to implement a system that will have obstacle avoidance and line detection as two separate modes. The hardware that I sought after included: gear motors, four wheels, two lithium 18650 batteries w/ switch, a L289N Motor Driver, a sensor, a servo-motor, Arduino circuit board, a chassis.

The L298N Driver is essentially the engine of the vehicle. It is responsible for making the car move and rotate. I found an article online called “*In-Depth: Interface L298N DC Motor Driver Module with Arduino*” which explains that the L298N does this through its electric current sent by the batteries using pulse width modulation for speed control. The **Duty Cycle is by definition** average voltage is proportional to the width of all the. The higher the duty cycle (the wider the pulses are), the greater average voltage being applied to the motor will result in (High Speeds). The motor’s spinning direction can be controlled by changing polarity of its input voltage by using an H-Bridge Circuit. An H-Bridge circuit contains four switches with the motor at the center forming an H-like arrangement. Closing two particular switches at the same time reverses the polarity of the voltage applied to the motor. This causes change a change to the spinning direction of the motor.

The Arduino version Uno played an important role since it essentially served as the brain of the vehicle. This microcontroller also comes with its own integrated environment as a free download from Arduino’s main website. This main website also provides a forum where Arduino enthusiast’s can upload their sketches/programs for the community to re-use. It uses both C/C++ as acceptable programming languages on its IDE. It is also able to manage 32 Kilobytes of Flash Memory and 2 Kilobytes of RAM.

The critical piece of hardware contributing to the vehicle being able to handle obstacle avoidance is the HC-SR04 ultrasonic sensor. According to “*The* *Complete Guide for Ultrasonic Sensor HC-SR04 with Arduino*” this ultrasonic sensor uses sonar to determine the distance to an object. The transmitter trigger pin sends a signal: with a high-frequency and if there is an object or obstacle on its path it will bounce back from which the echo pin will receive it. The time between the transmission and reception of the signal allows us to calculate the distance to an object. If an object is detected then this will activate the Servo Motor in degrees ranging from 0 to 180 degrees.

Another piece of hardware contributing to a desired mode is the Infrared Module located underneath the vehicle for line detection. The line detection sensor works by detecting reflected light coming from its own infrared LED and by measuring that amount of infrared light resisted. How the car will traverse its pre-made track is as follows: (Assuming the car is placed in-directly on top of the line) if the left sensor detects the line then the car will turn left, if right sensor detects the black line then the robot will turn right, if neither sensor detects the black line then the car will continue moving forward. It will keep re-adjusting itself infinitely.

We now have two separate modes for which the vehicle can successfully traverse a pre-made track and also detect nearby obstacles. To access both of these modes I used an infrared remote control. This remote along with other remote controls have special unique codes that are used as ID’s for each corresponding button. Once the button ID codes were found both the left and right buttons were respectively used to store two different modes on the vehicle.

The article “*Working model of Self-driving car using convolutional Neural Network, Raspberry Pi, and Arduino*” provided a simple foundation of how the code should be written. The student who worked on this project provided perfect pictorial representation of how his proposed model will work. For my autonomous model vehicle the proposed method will be as follows: firstly, the Ultrasonic module is a sound sensor which mounted on top of the car and propagates different frequency’s, the received frequencies (by the echo pin hardware) will then be passed onto a detection prediction algorithm, it will then send the outputted result to the Arduino microcontroller, and lastly the Arduino sends a signal to the motors to proceed in the calculated direction.

4. Implementation

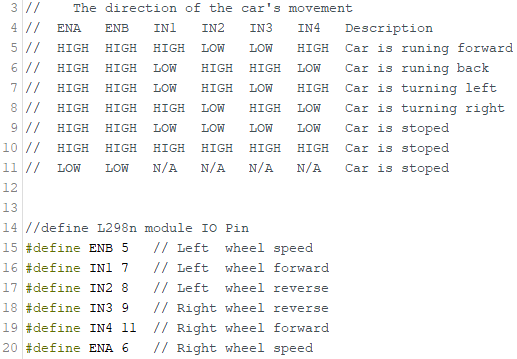
My implementation process began by researching the price of the hardware. Rather than buying each individual piece separately I decided to go on Amazon and purchase a model kit off of a company called, Elgoo. This saved me time since everything would come at once and also surprisingly saved me money (forty dollars cheaper!).

Once the hardware arrived the assembly process then initiated. A YouTube channel by the name of DroneBot Workshop gave some insightful advice on how he managed to overcome some hardware difficulties and also provided a useful assembly tutorial.

Once the car was built, I used the book titled “*Getting Started with Arduino”* to help with configuring Arduino’s IDE on my computer. The book provided sample code which helped make the car move in its fundamental desired directions: forward, left, right, and backwards. Having this sketch compiled on Arduino’s IDE was a huge step in the right direction.

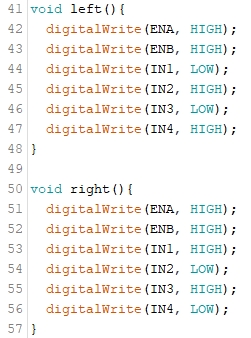
Code derived from: <https://create.arduino.cc/projecthub>

Defining Hardware:



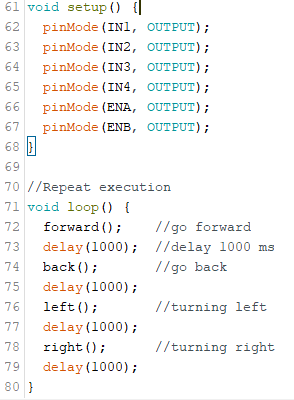
This picture defines the cars pinholes and also gives us a template of which pulse width pattern correlates to which direction the vehicle will move. The defined pins are as follows: ENA pins are used to control speed of Left Motor, ENB pins are used to control speed of Right Motor, IN1 & IN2 pins are used to control spinning direction of Left Motor , IN3 & IN4 pins are used to control spinning direction of Right Motor, OUT1 & OUT2 pins are physically connected to Left Motor, and OUT3 & OUT4 pins are physically connected to Right Motor.

Vehicle Direction Movement:



As we can see we have two different pulse width patterns for our methods called left() and right(). More methods will be created with their respected pulse width pattern for forward(), backward(), and stop().

Sketch Prototype “Move”:

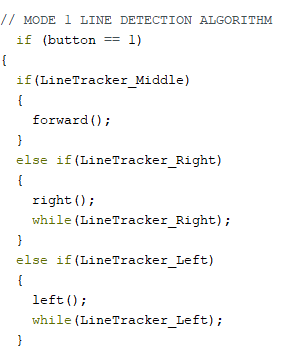


After coding each directions pattern within their defined method’s, we can now move on to the setup method. The setup method will execute first and only once. The pinMode code in the setup method configures the specified pin to behave either as an input or an output. The code will then run and compile under the method loop which as the method states will be the actual code to be looped forever.Delay (1000) means exactly as it reads delay the next instruction for 1000 milliseconds or 1 second.

Now that I knew all the components were properly connected, I then proceeded onto finding some working algorithms for line detection and obstacle avoidance. After finding both algorithms I ran into a situation where I did not know how to combine both modes onto the Arduino. Thankfully, the Elgoo company put in an infrared remote controller. Having this remote allowed me to store two modes on my vehicle.

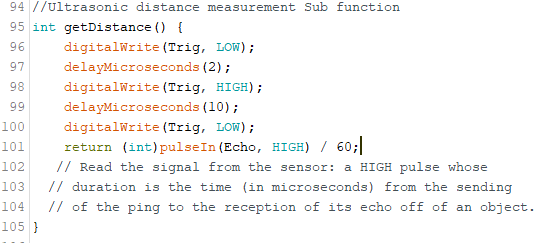
As previously mentioned, infrared remote controllers must have unique ID’s corresponding to each button on the remote. I was able to obtain the left button code and the right button code which was generously given through Elgoo’s website. Once I obtained the codes made a switch case where if the left button was chosen a declared variable “Button” would be set to 1. Where If the Button ==1 then proceed to the line detection code. If the right button was chosen the variable “Button” would be set to 2. Where if button == 2 then proceed to the obstacle avoidance code. If nothing is pressed the car will wait forever.

Line Detection Algorithm:



Under the method loop the code will perform the following: (Assuming the car is placed directly over black electrical tap) If the middle module detects the line then keep going forward, if the right sensor module is triggered make car turn right, and if the left sensor module is triggered make car turn left.

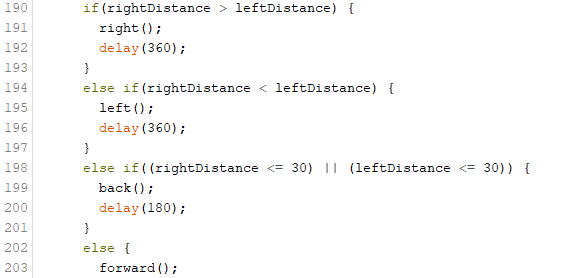
Detection Prediction Algorithm:



This picture shows the algorithm for calculating the distance of a detected obstacle. To reiterate, the “Trig” (short for the trigger pin) propagates an ultrasonic sound and waits for the sound to be bounced off an object, then the sound is received by the echo pin on the HC-SR04 sensor. Since distance equals velocity divided by time, we can follow the algorithm (line 101). Where velocity is the returned integer value received by the echo; pulseIn(Echo,HIGH) waits for the pin to go from a High state and begins timing, then waits for the pin to go to a LOW state, and stops timing. It returns the length of the pulse in microseconds. We now have velocity multiplied by time divided by 60 will give us a distance in centimeters.

Obstacle Avoidance Algorithm:





The algorithm is as follows: we start off (line 168) by setting our servo motor to 90 degrees or straight ahead, delay for half of a second (Middle distance is now calculated and this middle distance will be momentarily stored and compared with left and right distances), If middleDistance is less than or equal to 30 centimeters, an object is found within these parameters. If an object is found the following will happen: the car will stop, delay half of a second, Make the servo motor rotate to 10 degrees (I put it to 10 degrees and not zero because it was hitting a wire), delay for one second, right distance is now calculated, delay another half of a second, set motor to 90 degrees for a smooth transition, delay one second, set motor to 180 degrees, delay one second, and finally the left Distance is not calculated.

Once the left distance is calculated the servo motor and ultrasonic sensor resets to 90 degrees. It then interprets all the stored and proceeds to a direction from which was calculated furthest away and loops infinitely. If nothing was detected then the car will continue to drive straight infinitely as well.

5. Discussion

When I was initially working on this project, I convinced myself before purchasing the kit that the HC-SR04 was a camera. From the articles I read none of them had an in-depth analysis of what this piece of hardware did. I wish that I had camera functionality to demonstrate a more artificial intelligence like project. However, I found this piece of hardware to be really fascinating to learn about since it was something new to me. The science behind sound prorogation and sound capturing for detection prediction analysis is really simple which is probably why plenty of students (WAY younger than me) have already successfully completed.

I found one caveat during my line tracking runs that the track must have smooth turns otherwise the car will either spin in circles or keep going forward until it finds a track. Also, once I found what units for distance were being used it was really easy to calibrate a “good” distance so that the car didn’t smash into a wall or stop really far away.

Problems I encountered were: Sensor detection adjustments, Car speed adjustments (too fast or too slow), Car will not move for an unknown reason, and lastly, Car will drive straight but will not rotate its wheels, and lastly getting two modes to be stored and utilized. I was able to overcome these challenges by understanding more and more about Arduino’s IDE and understanding the syntax being used. Once I was able to translate what was going on it became easy to fix any of the troubles that I listed.

6. Conclusion

After finding out that the HC-SR04 was not a camera I began to second think whether or not my project constituted as a good artificial intelligence example. I then thought back to the beginning of the semester when we were defining what artificial intelligence was. While we do not have a clear-cut definition, we can see something as a form of AI if we look at them as rational agents with sensors and actuators that serve to maximize someone’s benefit. This definition of a rational agent to maximize someone’s benefit was applied to a simple automatic household Roomba which does pretty much the same thing my vehicle does. While my model vehicle does not have complex neural networks, it does have sensors (HC-SR04), actuators (gear motors), and maximizes my benefit by functioning as a partially autonomous model vehicle.

What I learned the most during this project was about convolutional neural networks in image capturing. I was unsuccessful to demonstrate this first-handedly since my sensor was sound capturing and not image capturing. Nevertheless, applying convolutional neural networks to images is a topic that can be further researched on my behalf. What fascinates me most about this topic is how complicated a machine can get when filters such as shapes, people, animals, objects, etc. It therefore becomes hard to computationally predict so many different specifications for a single machine to handle due to the enormous amount of data it will be fed.

If I put this project on my resume and use this project as leverage, I would hope to engage in a conversation about image capturing techniques, sound prorogation hardware to software relationship, and discuss other simple rational agents on the market.

7. References

A. K. Jain, "Working model of Self-driving car using Convolutional Neural Network, Raspberry Pi and Arduino," *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, Coimbatore, 2018, pp. 1630-1635.

doi: 10.1109/ICECA.2018.8474620

Banzi, Massimo, and Michael Shiloh. *Make: Getting Started with Arduino, 3rd Edition*., 2010.

Last Minute Engineers. “In-Depth: Interface L298N DC Motor Driver Module with Arduino.” *Last Minute Engineers*, Last Minute Engineers, 15 Dec. 2018, lastminuteengineers.com/l298n-dc-stepper-driver-arduino-tutorial/.

Pendergast, Robert L., et al. “Complete Guide for Ultrasonic Sensor HC-SR04 with Arduino.” *Random Nerd Tutorials*, 2 Apr. 2019, randomnerdtutorials.com/complete-guide-for-ultrasonic-sensor-hc-sr04/.

Wallace, Richard & Stentz, Anthony & Thorpe, Charles & Maravec, Hans & Whittaker, William & Kanade, Takeo. (1985). “First Results in Robot Road-Following”. 1089-1095.

Zhao Wang, Eng Gee Lim, Weiwei Wang, M. Leach and Ka Lok Man, "Design of an arduino-based smart car," *2014 International SoC Design Conference (ISOCC)*, Jeju, 2014, pp. 175-176.

doi: 10.1109/ISOCC.2014.7087683