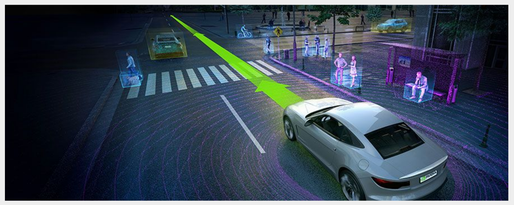
Autonomous Navigation and Decision-Making Process Using Machine Learning and Deep Learning

Jeongyong(Chris) Yang

Background Information



Autonomous cars are self-driving cars that do not require drivers. They rely on sensors to detect objects surrounding them such as other vehicles, pedestrians, and signs. After detection, cars need to make wise decisions according to each unique cases. For example, it should learn to stop once sensors detect red light and turn only when an appropriate signal is given. In addition, it needs to acknowledge all the different scenarios that can occur on roads and learn how to behave in each of those cases. For instance, even when people walk on roads when they are not supposed to, cars cannot simply hit them and cause accidents. Because safety is always the priority, the car should stop and wait until they pass the road. Once sensors detect people and its location, the cars need to 'think' whether the locations of people and car have a possibility to cause an accident and behave accordingly.

It is not only autonomous cars that require difficult decision-making process in navigational situations, but there are other cases where machines should analyze the situation and behave accordingly to each of them by prioritizing its actions. When people are transporting goods from one place to another, identifying the fastest and most efficient route is important. It can be simulated as a "virtual race" between autonomous cars since racing requires cars to finish the race fastest. In addition, when a building is on fire, detecting people and helping people escape from the building is very essential.

There are many ways to improve the current situation. Better sensors can be invented to detect surrounding objects better, calculate the shape of the object, discover the identity of object, and identify the distance apart from the sensor. There are many variables to calculate in a world with machines and sensors can solve part of the problem by detecting objects and analyzing them better.

Another approach in solving this problem is through by efficient decision-making. This can be done by coding through online library sources, such as the OpenCV and DeepLearning4J. By coding, vehicles or other machines can behave in certain ways in different situations. Coding is the basis to every machine learning situations. For example, when cars detect other cars trying to come into its lane, it has to slow down for the other car to enter the lane. In a race, once walls are detected, cars need to avoid hitting on them by curving from few miles before it.

"OpenCV is a software toolkit for processing real-time image and video, as well as providing analytics, and machine learning capabilities." OpenCV is one of the library sources that people use to access many advanced computer vision algorithms used for image and video processing in 2D and 3D. OpenCV has many range of applications in computer vision applications such as medical imaging and character recognition that can help classify skin lesions and detection of skin melanomas. OpenCV with processor and cameras are powerful class of computer vision.

Machine Learning is involved during this process. Specifically, deep learning is essential in solving this problem. Deep learning is a name people use for stacked neural networks, networks that are composed of many layers. These are made of nodes, a place where computation happens, loosely patterned on neuron in human brain. Node combines the input of the data which measures the weights and assigning significance to inputs for task the algorithm is trying to learn. These input-weight produces are summed, which is passed to an activation function. The activation function then prioritizes the variables according to the weight and produces the output. Deep learning networks are distinguished from more commonplace single-hidden-layer neural networks by their depth. The hidden-layers are hidden part of deep learning where it takes the input values and produces the output. It is similar to the method or function in computer science. If there are more than three layers, it is identified as deep learning. In deep-learning networks, the layer of nodes can train on different features based on previous layer's output. This whole concept is similar to the pugh chart in engineering, where it looks at all the different variables and scores each possibilities according to how well it takes care of the variables. In machine learning, the concept will be done by creating a concept similar to the flowchart. It will form numerous layers according to the importance of it and will behave differently in each situation. One of the problems it solves best is processing and gathering world's unlabeled data and finding similarities and anomalies in the data that people cannot recognize. For example, deep learning can take millions of images and cluster them, can cluster text and classify them, and can perform automatic feature extraction without people's intervention.

In other words, through machine learning and deep learning, computers can learn from themselves based on existing data and experience. It allows them to learn from millions of data. For example, in a super-mario game, it will try to beat the game by trying many different methods. Once the previous method fails, it will learn its failure from the previous trial and behave differently, after realizing the previous method was not optimal. Eventually, it will end up beating the game without any problem.

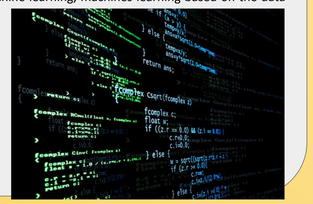
One of the difficulties of this project is that there are unlimited ways in the processing itself. One simple example is during a race. In a race, the goal is to complete the track from the start to the destination within the least amount of time without bumping into walls or obstacles. There are millions of different ways to complete a race. Considering the speed of every cars are same, the cars that travel the least distance will win the race. The cars will have to consider every possible variables, such as the location it should curve and the velocity, will be very important in their decision making process that will allow them to distinguish between the winning car and the losing cars.

Goal & Objectives

The main goal of my project is to design, build, and test a sophisticated code that will allow the autonomous vehicle to go through complicated mazes. Instead of driving with real vehicles, an alternative lego will complete the maze with the code and information it learned throughout few trials. In order to achieve this goal, a well-written code that causes no error has to be written. Instead of writing actual code, this portion of the project will be done with writing pseudocodes. Constant debugging and experimenting will be necessary in order to improve the pseudocode. Second, a maze has to be constructed in order for the autonomous vehicle to run. The maze will be drawn on a paper and with the pseudocode, the vehicle will conduct each movement. Finally, after the code has been checked over few trials with different types of mazes, the vehicle will gather information about its location and its velocity(both x and y directions) and record it. In addition, it will collect information about the location of the walls, based on the distance from the vehicle at every step. With this information, it will predict a possible map and draw the map so that the next vehicles can follow with the information given. This part of the project will require machine learning, machines learning based on the data given.

The objectives of this project are:

1. Build a sophisticated computer program that will safely allow the vehicle arrive to the destination without any problems or crashes
2. Gather information based on the vehicle and map that perfectly arrives to the destination from its initial point and use that information to draw and keep track of the map.



Abstract

Autonomous vehicles are self-driving cars that do not require human drivers. They use sensors that are attached to the vehicle as their vision to detect their environment. After the vehicle detects other objects or signals, computer programming (coding) allows them to react to the situations adaptively. Even though the sensors do not need to be improved, the millions of situations the cars can face on roads create difficulties for people to build a sophisticated computer programming that makes the autonomous vehicles completely safe on roads.

First, I decided to build an algorithm pseudocode to help resolve this problem. During the process, I built mazes and followed the instructions based on the algorithm manually to check whether the algorithm is effective. I mainly used three different models for my mazes, each with different difficulty level to ensure that the algorithm works every time. Then, I decided to record the information(velocity and displacement for both x and y directions) about the vehicle on the map so that the following vehicles can get a picture of the map automatically. However, if the subsequent vehicle detects a different or an altered map with its sensors, the new information will also be recorded on the map. Finally, the final vehicle will follow the path set by the first vehicle, but the map will guide the car with the most efficient path after completely learning and optimizing the possible paths.

The Engineering Design Cycle

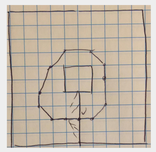
Brainstorming Solutions	Since there are always many good possibilities for solving a problem, this step includes looking and finding all the possible approaches so that the best solution could be found eventually.
Optimizing Best Solution	The goal of this process is to find the best solution based on the ideas collected by the previous stage.
Building a prototype	By building a prototype will allow me to test and examine how the solution will turn out or work.
Experimenting the solution	The solution needs to be tested in reality and checked whether if it works or whether there are ways to improve the existing solutions.
Redesigning the solution	Usually, the design cycle or process includes multiple trials or iterations of the model or solution and redesign or remodel them.

Method Part 1 - Collect Ideas for Building Computer Program Using Mazes

Initial Rules:

1. Draw all the Possible Routes the vehicle can move to (except moving backwards since it will be both unnecessary and inefficient)
2. Move to the Direction where the vehicle can move the furthest out among the possibilities that were given from the first rule.
3. After each move, repeat steps one and two until destination was reached.

Diagram(Maze) One (10 by 10 Square with 2 by 2 Square as Wall):

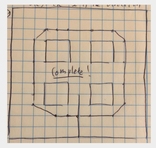


The first maze was a 10 by 10 square with a 2 by 2 square at the middle, which acts like a wall, and the vehicle had to use the rules to start from a location and arrive back to the location after oscillating around the 2 by 2 square.

While the vehicle was traveling through the first maze, there were no needs to alter the existing rules, but it was necessary to add few additional rules.

4. While the longest route is chosen from steps one and two, if the longest move will cause the vehicle to crash onto a wall, the vehicle should only move to the direction it would not end up crashing on the wall.
5. When there are two identical longest distances, the vehicle should move to the direction that is at the middle of the two longest distances, no matter how long the middle distance is.

Diagram(Maze) Two (12 by 12 Square with four 2 by 2 Square as Wall):



After the vehicle went through the maze, there were no problems within the vehicle and the rules. However, the four blocks (grids) of wall at the initial point caused the vehicle to safely return to the destination. Therefore, I reduced one grid of wall for the next model, which might greatly change the result, allowing me to fix the existing rules or add additional rules.

Method Part 1 - Collect Ideas for Building Computer Program Using Mazes (continued)

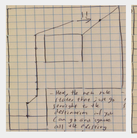
Diagram(Maze) Three (12 by 12 Square with four 2 by 2 Square as Wall):

With one wall (grid) reduced, the existing rules lead to a very different model. In this case, the vehicle does not go to the destination after one oscillation, but moves away from it and goes to the wall. With the existing rules, it will collide with the wall, which should be avoided. Therefore, new rules were necessary to fix this problem.

Correction of Rules

1. Delete Rule 2
2. After calculating all the possible different routes, calculate the distance from each of the routes to the destination and move to the direction that has the minimum distance from the route to the destination.
3. When the vehicle can go towards the destination straight, ignore all the rules and go straight to the destination.

Diagram(Maze) Four(P-model-12 by 12 Square with 6 by 4 Rectangle and 4 by 4 Square as Wall):

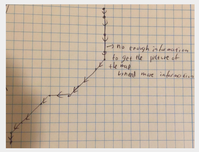


The new rules did not cause any problems throughout both of the experiments using P-model, which meant that it was good enough for me to begin writing the pseudocode. However, I needed to reduce one variable and limit the vehicle's direction possibilities so that it only calculates the routes to 45, 90, 135, 180, 225, 270, 315, and 360 degrees (16 possibilities to 8 possibilities) to make the pseudocode and the model simpler.

Method Part 3 - Using Autonomous Vehicle to Draw Map & Keep Information on the Map

The next part of the project was to keep the information of the vehicle on the map, which will allow the following vehicles to use and apply that information when traveling. Therefore, I included saving the x and y location, with the velocity of x and y at every point. These information would be used to draw the map. I used the P-model used during the first method to conduct this part of the experiment. The starting point and destination was given and with the given pseudocode, the vehicle will arrive at the destination within few steps. When the vehicle arrived at the destination, I checked the information that would be stored.

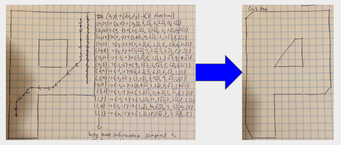
Stage One: Reconstructing P-model (12 by 12 Square with 6 by 4 Rectangle and 4 by 4 Square)



With only the x and y displacement and velocity, it was impossible for the vehicle to assume that the map was P-shaped. There needed to be more information to fully comprehend the map.

Therefore, I added another variable that calculated the distance from the car to the first wall in eight directions, which added one more line of code in the pseudocode (to record the distance from the vehicle to the wall), and the diagram looked like:

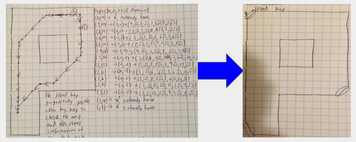
Stage Two: Reconstructing P-model (12 by 12 Square with 6 by 4 Rectangle and 4 by 4 Square)



Even though the vehicle was able to roughly guess the map, it looked slightly different compared to the real map. The three corners were incorrect and the 4 by 4 square at the middle of the P-shape became a rhombus instead.

The three corners being slightly incorrect was very small part of the map and since the vehicle does not even reach those boundaries, adding additional code to fix them was unnecessary. However, the shape of the rhombus needed to be changed either to a square or a shape close to the original map given, which could be possible if the vehicle went to the left direction from the starting point instead of moving downwards. Therefore, I purposefully forced the vehicle to move initially to the left as its first move and conducted the experiment again with the same pseudocode, which was:

Stage Three: Reconstructing P-model (12 by 12 Square with 6 by 4 Rectangle and 4 by 4 Square)



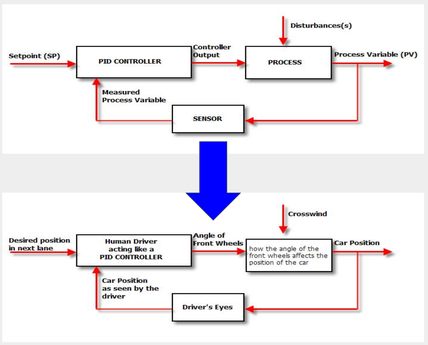
This time, the P-model drawn from the vehicle was very close to the P-model given. Even though the information from both the first and second vehicle was unable to fix the boundaries for three corners, the 4 by 4 square wall was completed and looked exactly as the original model. Therefore, I needed to include few lines of additional code to the pseudocode designed.

Method Part 2 - The Pseudocode

```
public double[] function1() //about when to move or stop
    {
        function3();
        if(d - 1 >= v * v)
        {
            v = v + 1; or v = v + (2)^.5; //depending on diagonal
        }
        else if(d < v + (v-1) + (v-2)...) //until v-value=0
        {
            v = v - 1; or v = v - (2)^.5; //depending on diagonal
        }
        else
        {
            v = v;
        }
        return [x,y,dx,dy, function2()] // eight directions to wall has been added after part three
    }

    public double[] function2() //distance from the vehicle to the walls
    {
        return [8 distance from the vehicle to the walls];
    }

    public double function3() // the rules that we set up
    {
        double[] distances = function2();
        distances = Measure the shortest distance from the destination to all of the eight directions that
were returned from function2();
        double largest = distances[0];
        for(int x = 1; x < distances.length - 1; x++)
        {
            if(distance[x] == distance from vehicle to destination)
            {
                largest = distance[x];
                break; //since the rule says to ignore all the other rules if the vehicle can go straight to
the destination;
            }
            if(largest > distances[x])
            {
                largest = distances[x]; //choose the largest distance out of them
            }
        }
        turn the vehicle to that direction;
        return largest;
    }

    public void function4() //about the map and whether it reached to the destination
    {
        boolean arrive = false;
        double[] information = new int[12];
        while(!arrive)
        {
            information = function1();
            record information on map;
            if(information == null) //that means the vehicle stopped moving, which means it arrived to
the destination or crashed onto the wall
            {
                arrive = true;
            }
        }
    }
```

Future Goals

I decided to expand this project and apply PID into the project. PID stands for Proportional Integral Derivative. PIDs can be used instead of the vehicle keeping all the information. The exact information can be stored within the map itself and guide the vehicles to move in certain directions and velocities, depending on their location the map. PID can be explained in a diagram.



This is an usual diagram for PID. The setpoint is the value that people want the process to be. The controller compares the setpoint with the actual value. If they are identical, the controller does not need to do anything, but if the values are not same, PID needs to make an action to fix this and set them equal to each other. Below is the diagram with the case of PID applied to autonomous vehicles.

Here, the setpoint, which is the goal, is to move the vehicle's position into the next lane and if the setpoint and actual value, the value that human driver will do, are equal, it does not need to do anything but just let it happen. However, if they are different, the controller needs to adjust so that the autonomous vehicle does act like as if it is a human-driver.

Applying PID to my model will allow my project to be more realistic. My setpoint could be arriving to the destination and if it fails to do so, it will indicate that the pseudocode has errors on it and that rules need to be changed. In addition, in my case, the first vehicle that moves to the destination from the initial point records both the displacement and velocity for both x and y coordinates. Since velocity can be used to calculate displacement by integration and used to calculate the acceleration by derivative, applying PID to my model will be very beneficial.