Integration of Machine Learning into an Autonomous Vehicle Simulator

Jacob Nazarenko
Boston University Academy, 1 University Rd., Boston, MA

Aman Jha
Needham High School, 609 Webster St., Needham, MA

Major Assistance Received

We would like to thank Professor Sertac Karaman and the rest of the MIT Beaver Works Summer Institute Staff for teaching us the concepts used in this study, as well as their applications in autonomous driving. This experiment would not have been possible without their dedication to the program, and their endless willingness to help us in our endeavors to explore the area of autonomous systems.

We would also like to thank Professor Russ Tedrake and Professor Nicholas Roy of MIT for advice and demonstration of their robotics simulators. Their ideas and design helped guide us for our own simulator design. Creating our own simulator enabled us to record data from the experiment.

The machine learning aspect of this study was inspired by the teachings of Professor Andrew Ng of Stanford University. The gradient descent algorithm used in our experiment was based on the gradient descent algorithm presented in his machine learning course, available through Coursera: https://www.coursera.org/learn/machine-learning

Note on Experiment Subjects

No human or non-human vertebrate subjects were involved in this experiment.

TABLE OF CONTENTS

***Abstract* –** This project researches the possibility of optimizing the control system of an autonomously driving vehicle with the use of a gradient descent data analysis algorithm. It also investigates the degree to which the vehicle's movement is optimized. A simple teleoperated vehicle simulator was first developed with the Unity3D game engine, and the vehicle's driving control was then made autonomous through the streaming of data back and forth over UDP network sockets between a simulated LIDAR system in Unity3D, and an autonomous driving algorithm running in Python. Consequently, the vehicle can drive through a given 3D environment with a potential field algorithm, data can be collected about the vehicle during its drive, and chosen pairs of variables in this data can then be analyzed with a polynomial fit via a gradient descent algorithm. The output of the gradient descent algorithm can then be reapplied to the vehicle's driving control system, optimizing obstacle avoidance in some cases. The time it takes the vehicle to get from one point to another, along with how well it avoids obstacles, serves as a measure of how well gradient descent optimizes its movement. Although optimization is possible in some cases, the experiment is still ongoing.

I. Introduction

Today, self-driving car manufacturers attempt to create the safest autonomous algorithms to ensure the security of their customers, passengers, and the public. Research done in sensor technology make sensors cheaper and more accurate, which will make self-driving cars available to a much larger population. However, algorithms continue to improve. Although they increase in accuracy and flexibility, human developers cannot account for everything a vehicle will experience on the road. To properly ensure safety, developers construct simulators and examine the efficiency of algorithms through virtual testing.

Public demand and market pressure call for the development of safer cars to be completed faster and more precisely. The best way to accomplish this without compromising quality is through simulation and machine learning. Machine learning is a type of artificial intelligence (AI) that enables computers and robots to learn from previous experiences without being explicitly told directions, as in traditional programming. Machine learning works by sifting through collected data, finding patterns, and tuning output variables accordingly. When a machine learning-based program encounters a familiar pattern or problem, it can predict future developments. A machine learning algorithm is a success if the computer or robot automatically corrects errors and becomes better at its task without outside operator influence. The program designed by the computer is called a model, and feedback from its performance is vital to future improvements.

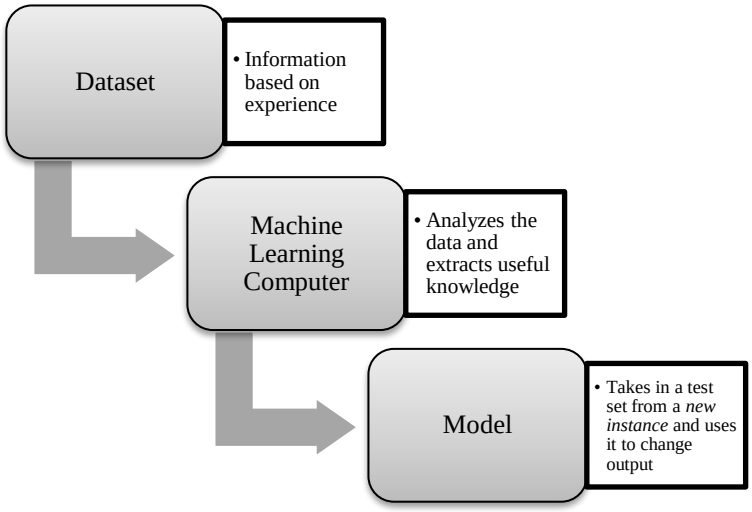

Figure 1. Machine Learning Concept Flow. The diagram above describes the three conceptual steps involved with a successful machine learning algorithm.

The primary goal of the research presented in this paper was to apply a machine learning algorithm to the analysis of an autonomous vehicle's movement in a simulated environment under an already successful movement algorithm, and to use the output model to control the vehicle without any dependence on this movement algorithm. The successful algorithm used to move the vehicle to learn data was a potential field algorithm, one of the earliest methods for basic obstacle avoidance. A potential field algorithm treats a given course like an electromagnetic field. The goal (if the course has an end) is given a positive charge, and the vehicle is given a negative charge. However, any detected obstacles are also given a negative charge. Free area is neutral. Under this system, the vehicle will attempt to generally navigate towards an end goal while avoiding any

obstacles near it. The goal is known as attractive potential, and the obstacles are known as repulsive potential. With these elements combined in a map, the vehicle will choose the path of least resistance, which will almost always lead to the goal in the most efficient path. An alternative way to think about a potential function is to imagine a bowl. The bottom of the bowl represents the goal, and the vehicle is placed at the rim of the bowl. Any obstacles represent dentures sticking out from the floor of the bowl. When the vehicle is released, it will automatically avoid the obstacles using the shortest path available to reach the goal. The objective of a potential function is to avoid collisions and use the least energy possible.

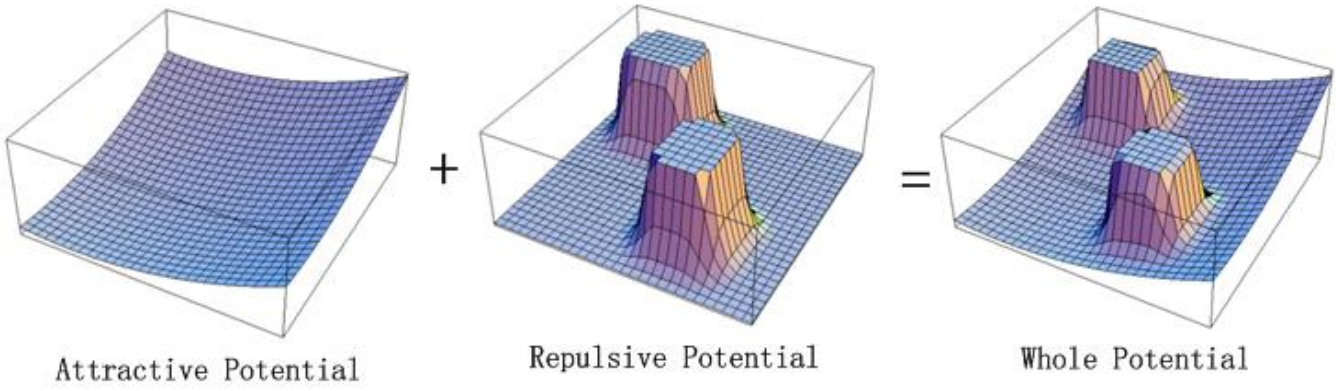

Figure 2. Potential Field Diagram. The diagram above illustrates the concept of attractive and repulsive potential and how in obstacle avoidance, a robot can use the diagram to path-find its way to a goal. The diagram was created using Python Pyplot. [1]

A potential field algorithm can also be used without a final goal, which is the object of the machine learning research presented in this paper. Instead of aligning the goal to a fixed point on a map that the robot can detect, the goal will always remain at a fixed point in front of the vehicle, meaning the robot will move forward and avoid obstacles to continue its path. Obstacles will be sensed and dealt with accordingly; essentially each frame is a new potential field map.

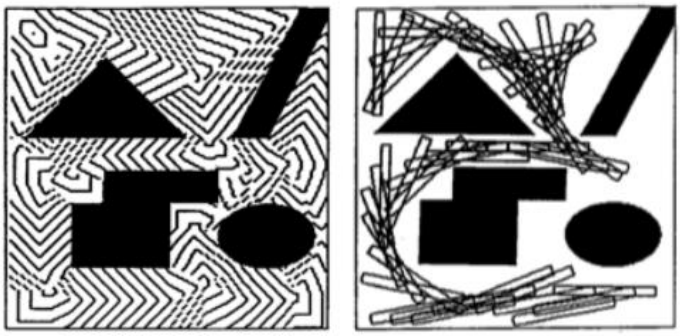

Figure 3. Potential Field Analysis and Results. The above images show the distribution of the potential field for a given set of obstacles (on the left), as well as the estimated trajectories for a robot moving among those obstacles (on the right). [2]

The machine learning functionality applied to the analysis of the vehicle's movement under the potential field algorithm consisted a gradient descent algorithm, which yielded polynomials describing the changes in the speed and steering angle of the vehicle with respect to changes in the environment around it. These polynomials could then be used to control the vehicle without the potential field algorithm. This machine learning functionality could similarly be applied in situations where other obstacle avoidance algorithms (opposed to potential fields, which is just one type of obstacle avoidance algorithm) were originally controlling the vehicle's movement, meaning that the potential field algorithm could be switched out for a different successful movement algorithm, or even teleoperated (human) control. Using other algorithms, however, was outside the scope of this experiment. This experiment showed that a simulated vehicle could 'learn' how to avoid obstacles and move around an environment through potential fields.

II. Materials and Methods

This experiment was conducted with the use of the Unity3D game engine and the Python programming language. All aspects of the 3D scene were created and stored in Unity3D, and synchronized on multiple computers through Unity Collaborate, a cloud storage solution for Unity3D projects. Python was used exclusively for the analytical portion of this experiment, namely for the analysis of the vehicles speed and steering angle data, as well as data about the surrounding environment.

Though the development of the simulator isn't the major part of the research presented by this paper, the simulator was created for the purposes of this research and therefore understanding its methodology is vital to acknowledging the accuracy of the experiment. The vehicle model was designed by Unity. The code for the vehicle was based off real car mechanics – utilizing rear wheel drive, wheel colliders, slip calculation and a center of mass offset to make the vehicle as accurate as possible. The vehicle operated off two driving variables: acceleration and steering angle. The vehicle also used a sensor found on real self-driving cars to imitate the situations found for obstacle avoidance. The yellow bulb on top of the vehicle represents the LIDAR (Light Detection and Ranging) sensor, which is a primary sensor found on autonomous robotics. A LIDAR uses a rapidly oscillating laser to return light back to a receiver and track distance. Since light speed is constant given a single medium, LIDARs are very efficient at gauging distances. Since a LIDAR can operate at hundreds of separate angles per second, it can map entire environments in 2D. In the simulator, the LIDAR was simulated through raycasting. Given that the angle directly in front of the vehicle represents 0, the vehicle would cycle between sampling angles -135 degrees and 135 degrees at 0.25 degree increments, for a total of 1080 mapped distances per frame. This enables the vehicle to generate an environment and recognize obstacles around it. The raycasting system works exactly in the same way a self-driving car's LIDAR operates.

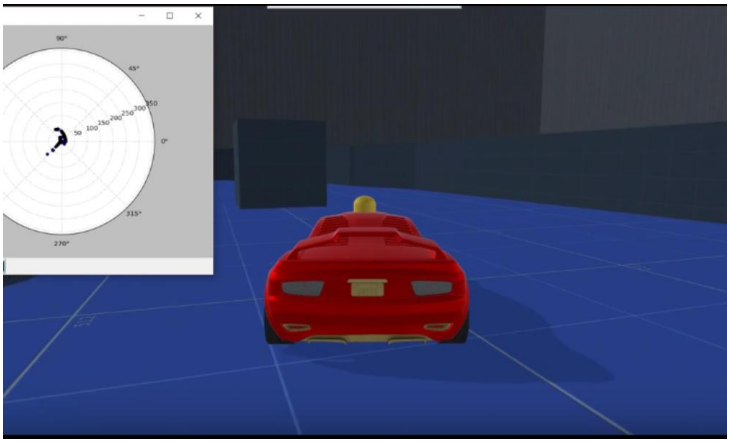

Figure 4. LIDAR Scan Demonstration. The diagram above shows how the LIDAR can scan obstacles and rapidly map changing environments.

The goal of this experiment was to determine if it was possible for a data analysis algorithm such as a gradient descent polynomial fit could mimic the movement of an autonomous vehicle controlled by a potential field algorithm. To determine this possibility, the following algorithm was used:

(1) The user selects two variables from a list through a displayed interface: one to analyze against the vehicles speed and one to analyze against the vehicle's steering angle
(2) The two variables having been recorded, two network sockets are opened between the Python code and Unity C# code – one for receiving data and one for sending data – and a potential fields algorithm running in Python begins sending driving commands to the vehicle based on data that it receives about the surrounding environment from Unity
(3) While the car drives around under the potential field algorithm, the values of the two chosen variables, along with the vehicle's speed and steering angle, are continuously recorded into long arrays by the Python code through two separate network sockets, and are analyzed by a gradient descent algorithm as soon as the Unity scene is deactivated

(4) While the analysis is ongoing, the user must reactivate the Unity scene so that the vehicle is no longer being controlled by the potential field algorithm, and so that when data analysis completes, it may be controlled by the polynomial generated by the data analysis algorithm
(5) Once the analysis completes, the sockets previously used to extract data about the environment around the vehicle and control the vehicle are now used by the Python script to take in the same data, and use the data as an input to the polynomials generated, where the output (speed and steering angle) is sent back to the vehicle over the controlling socket
(6) The vehicle can operate on the generated speed and steering angle polynomials

A visual explanation of this process is included in the following diagram:

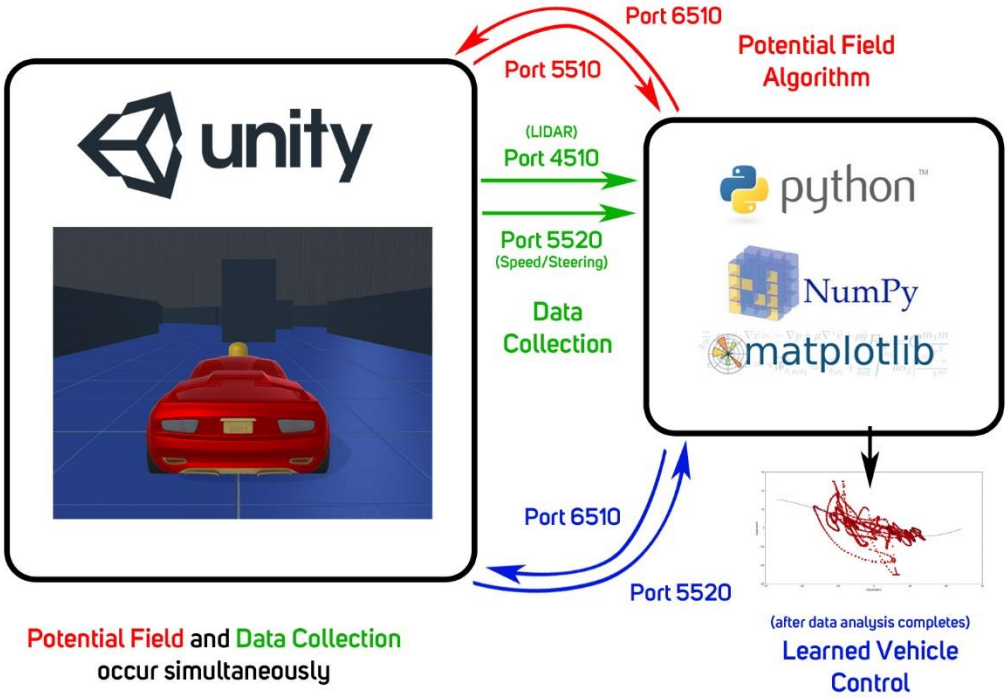

Figure 5. Data Collection and Analysis Algorithm. The diagram above describes the different features of the experiment's setup in terms of the platforms and network ports used for each feature.

For step (1), the primary Python script displays a selection dialog allowing the user to choose the variables they would like to analyze against speed and steering angle, respectively. These variables were chosen with their importance in mind, as each of the variables offered for the two outputs are among the most important in determining their value through a potential field algorithm. The dialogue was programmed with the TkInter library, and stores an array of the user's responses for later use.

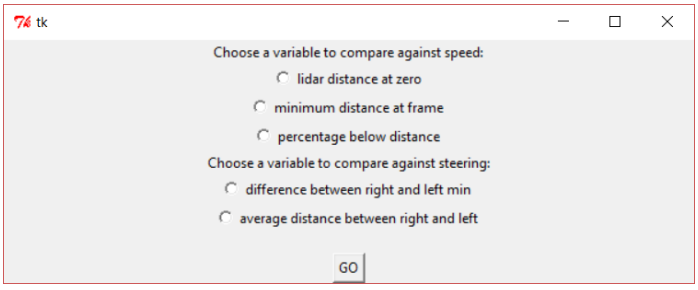

Figure 6. TkInter Dialog Box. The dialog box depicted in this figure collects user responses regarding the variables to compare and analyze against the vehicle's speed and steering angle.

In steps (2) and (3) stated previously, four separate network sockets are opened on the user's machine (localhost): two identical ports, 5510 and 5520, for sending virtual LIDAR data from Unity3D to the Python script; a third port, 6510, for sending speed and steering angle output values from the potential field algorithm running in Python to the vehicle in Unity3D; and a fourth port, 4510, for sending the vehicle's speed and steering angle from Unity3D to the Python script.

Virtual LIDAR data sent from Unity3D is very much like actual LIDAR data. Each message consists of an array of 1080 range values, representing 270 degrees of visibility from the vehicle, and with 4 range values per degree. Every message sent over a network socket, regardless of data type, must be formatted into a packet or series of packets by a formatting function in the corresponding language before being sent. Consequently, it must also be 'deciphered' or reformatted by a reformatting function in the corresponding language receiving the message to be restored to the state in which it was sent. This is true for all data sent over the network in this experiment. The only reason for opening two identical sockets here was the fact that each socket can only have one listener, and the potential field algorithm is a separate Python script from the one that records data for later analysis. Both, however, need to listen to LIDAR data.

Port 6510 is first opened by the potential field algorithm script, and is used for sending speed and steering angle values to the vehicle. A potential field algorithm is a way of calculating the necessary trajectory and speed of an autonomous vehicle based on its surroundings. It repels the vehicle from surrounding obstacles, pushes the vehicle into the largest area of free space in front of it, and slows down or backs up the vehicle if necessary. The vehicle begins to act like a magnet begin repelled from other magnets of like charge (the obstacles) around it. [3]

In the implementation of the potential field algorithm used in this study, the gradient of each range returned from the vehicle's virtual LIDAR was calculated. The potential (U) of each obstacle (or the force that each obstacle would have on the car) was inversely proportional to the distance between the vehicle and the obstacle:

$$U_{obstacle} = \frac{1}{||pos_{car} - pos_{obstacle}||} = \frac{1}{\sqrt{(x_{car} - x_{obstacle})^2 + (y_{car} - y_{obstacle})^2}} \qquad \text{(Eq. 1)}$$

To figure out the direction pointing down the potential surface, and the total influence of nearby obstacles on the steering angle and speed of the vehicle, the gradient was calculated at each surrounding LIDAR data point with respect to the car's coordinates, x_{car} and y_{car}:

$$\frac{\partial U_{obstacle}}{\partial x_{car}} = \frac{\partial}{\partial x_{car}} \frac{1}{\sqrt{(x_{car} - x_{obstacle})^2 + (y_{car} - y_{obstacle})^2}} = -\frac{(x_{car} - x_{obstacle})}{((x_{car} - x_{obstacle})^2 + (y_{car} - y_{obstacle})^2)^{\frac{3}{2}}}$$

And

$$\frac{\partial U_{obstacle}}{\partial y_{car}} = \frac{\partial}{\partial y_{car}} \frac{1}{\sqrt{(x_{car} - x_{obstacle})^2 + (y_{car} - y_{obstacle})^2}} = -\frac{(y_{car} - y_{obstacle})}{((x_{car} - x_{obstacle})^2 + (y_{car} - y_{obstacle})^2)^{\frac{3}{2}}} \qquad \text{(Eq. 2)}$$

Next, to determine the total potential (U_{total}), all gradients were summed with respect to x_{car} and y_{car} separately:

$$\frac{\partial U_{total}}{\partial x_{car}} = \sum_{i=0}^{i=n-1} \frac{\partial U_i}{\partial x_{car}} \quad \textbf{And} \quad \frac{\partial U_{total}}{\partial y_{car}} = \sum_{i=0}^{i=n-1} \frac{\partial U_i}{\partial y_{car}} \qquad \text{(Eq. 3)}$$

To determine the steering angle, only the x-component of the total gradient was used:

$$Steering\ angle = k_{steer} * \frac{\partial U_{total}}{\partial x_{car}} \qquad \text{(Eq. 4)}$$

In eq. 4, k_{steer} is a constant used to set the amount of influence the potential field had on the vehicle's steering angle. In a similar way, the magnitude of the total potential was used to determine the speed of the vehicle, which could be negative based on the magnitude of the potential's y-component, causing the vehicle to go in reverse when it encountered a set of obstacles it could not overcome:

$$Speed = k_{speed} * \left\| \left\langle \frac{\partial U_{total}}{\partial x_{car}}, \frac{\partial U_{total}}{\partial y_{car}} \right\rangle \right\| sign\left(\frac{\partial U_{total}}{\partial y_{car}}\right) \qquad \text{(Eq. 5)}$$

In eq. 5 above, k_{speed} determined the amount of influence that the potential field had on the vehicle's speed. Because eq. 5 above often caused the vehicle to get stuck attempting to move forward and reversing due to speed limitations in several

places, the equation was modified so that it simulated the gradual decay of the vehicle's velocity due to friction, and lightly influenced the vehicle's actual speed with the potential-based speed value:

$$Speed_{current} = \mu * Speed_{current} + \alpha * Speed_{potential} \qquad \text{(Eq. 6)}$$

In the above equation, μ represents a constant close to but less than 1 which simulates the decay of speed over time (a value of 0.95 was used in this case), and α represents a constant close to but greater than 0 which determines how much the potential-based speed value will influence the buildup of speed overall. With this approach, the vehicle handled obstacles much smoother, and therefore functioned in a more predictable manner. To help the vehicle retreat from obstacles, the steering was inverted with a negative constant whenever the overall speed dropped below 0. The adjustment of the constants mentioned here helped fine-tune the algorithm.

All the network sockets that were used in this experiment were programmed to record the end of the potential field control period, and so when the user unclicks the play button in Unity3D, a stop message is sent to the Python script indicating that the stream of data was interrupted, and that it should now begin analyzing the data it received. The two sets of data are then sequentially put through a gradient descent algorithm. A gradient descent algorithm works by attempting to come up with a series of coefficients for a polynomial fit to a set of data in such a way that it minimizes a cost function. This cost function is very closely related to the mean square error function, and has the following form:

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 \qquad \text{(Eq. 7)}$$

In the equation above, m is the number of y values, h_θ is the function the algorithm currently estimates (applied to a given x value represented by $x^{(i)}$), and $y^{(i)}$ is the y value corresponding to that x value. This cost function determines how well a set of estimated coefficients fits a curve. The lower the output of the cost function, the better a set of coefficients works.

To minimize this cost function, the algorithm takes the gradient of the three-dimensional representation of this function, where the different coefficients are the independent variables, and the cost is the dependent variable, and tries to reach the minimum point (or zero) of the gradient by continuously calculating it and subtracting a very small portion of the gradient from the currently estimated set of coefficients (so as not to overshoot the minimum point). This small portion of the gradient of the above cost function has the following form:

$$\alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} \qquad \text{(Eq. 8)}$$

In the expression above, α is the learning rate (which can be between 0.1 and a very negative power of 10), and all the other values are the same as in the cost function. This is the derivative of the cost function (with respect to the set of coefficients), multiplied by the learning rate. After many iterations, the currently estimated set of coefficients can come infinitely close to the minimum point of the cost function, and can be used to compute a new, much lower cost of fitting the improved coefficients to the data. [4]

The coefficients found are the values that would yield the polynomial with the smallest possible cost, and would therefore provide the most accurate polynomial fit to the set of data. The NumPy and Matplotlib libraries were used to program the gradient descent algorithm, and the code displayed each analysis result in a new Matplotlib plot window. An example of a gradient descent result is in the following figure:

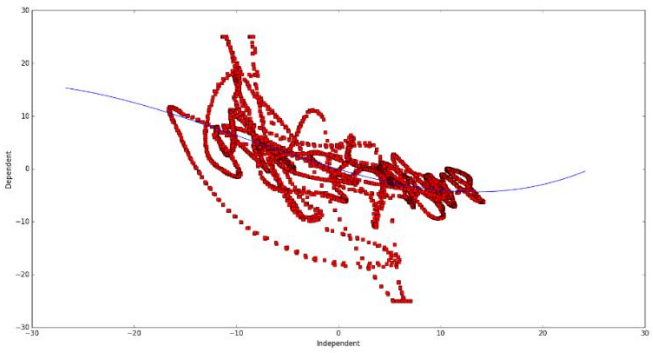

Figure 7. Sample Gradient Descent Output. This image contains a sample graph generated by the gradient descent algorithm. The 'independent' variable in this case could be one of the variables the user chose from the dialog, while the 'dependent' variable in this case could be the steering angle of the vehicle.

In steps (5) and (6) of the algorithm described at the beginning of this section, the coefficients of the polynomial generated by the gradient descent algorithm can be used to turn the input data sent in from the vehicle's virtual LIDAR into output control data (speed and steering angle) for the vehicle, very similarly to the potential field control period described earlier in this section. There are once again two sockets in use: one of which was previously used to receive LIDAR data, 5520, and one which was previously used to send potential field output data to the vehicle, 6510. The vehicle is now able to operate on data that the algorithm has learned from the potential field control period, and results may be noted.

After some experimentation with a non-continuous course, one that had a start point and an end, it became clear that working with a continuous course yielded more consistent results, and a figure-eight course was therefore chosen for this study.

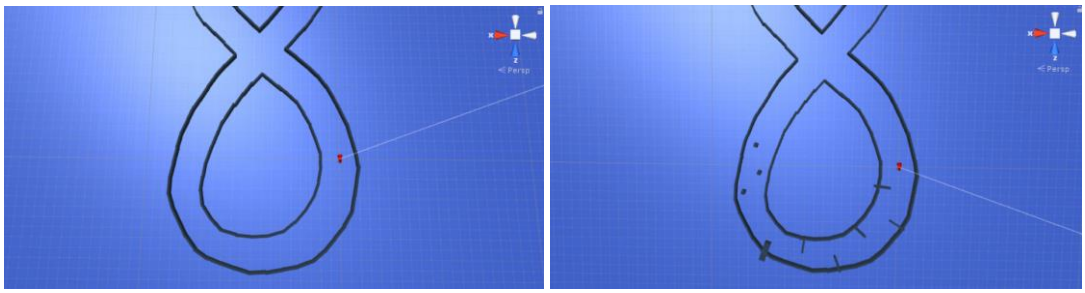

Figure 8. Figure Eight Course Without (Left) and With Obstacles (Right). The car travels around the course repeatedly, learning movement patterns, and eventually navigating with the information it learned.

Once such a set of constants and maximum acceleration cutoff (0.4 m/s^2) for the potential field algorithm that allowed the vehicle to travel all the way around the course at least once even with obstacles, trials commenced. For each trial, the same potential field constants and maximum acceleration of 0.4 m/s^2 were used, and the variables that were analyzed with respect to the vehicle's speed and steering angle were changed out. For each data collection period, the vehicle was controlled through the course approximately three times with the potential field algorithm. This way, it became obvious which variables were the most reliable to use for the vehicle's learning process. If a certain pair of variables was successful, the car would be driven through the course with learned vehicle control until the vehicle did up to 5 laps without stopping.

During each round of testing, the following variables could be used for comparison against speed and steering angle:

- Analysis with respect to **speed**
  - **LIDAR distance at zero:** the distance to the nearest obstacle directly in front of the vehicle
  - **Minimum distance at the current frame:** the nearest distance to any obstacle in any direction from the vehicle
  - **Percentage of points below a certain distance at the current frame:** the percentage of ranges that are below a certain threshold value
- Analysis with respect to **steering angle**
  - **Difference between the minimum range on the right and minimum range on the left:** the difference between the smallest range on the right half of the vehicle and the smallest range on the left half of the vehicle
  - **Difference between the average of ranges on the right and average of ranges on the left:** the difference between the average of ranges on the right half of the vehicle and the ranges on the left half of the vehicle

All combinations of one variable for speed and one variable for steering angle were chosen, yielding a total of six cases that were tested without obstacles, and the same six cases that were then tested with obstacles. Because it was known that both the

speed and the steering angle values should have approximately cubic polynomial distributions,[1] the gradient descent algorithm was set to output third-degree polynomials by default.

III. Results

Although there were no significant results for this experiment at first, further experimentation with the parameters of the algorithms and the vehicle's environment in Unity did reveal some useful findings.

Table 1. Results for Trials without Obstacles. The following is a set of 6 trials during which there were no obstacles on the course, each trial with a different combination of variables analyzed. The maximum acceleration of the vehicle was clamped at 0.4 m/s^2 for both potential field and learned control, and all the gradient descent output polynomials were of the third degree.

Without Obstacles				
Lap time with potential field: 38 secs				
Speed Method	**Angle Method**	**Final Cost of Gradient Descent**	**Laps Completed**	**Lap Time**
LIDAR distance at zero	Difference between min right range and min left range	169.2 (speed); 6.8 (angle)	0	N/A
LIDAR distance at zero	Different between right range avg. and left range avg.	169.56 (speed); 3.56 (angle)	>5	46 secs
Min. distance at current frame	Difference between min right range and min left range	1.77 (speed); 6.88 (angle)	0	N/A
Min. distance at current frame	Different between right range avg. and left range avg.	1.77 (speed); 3.5 (angle)	>5	46 secs
Percentage below threshold distance	Difference between min right range and min left range	0.1 (speed); 7.02 (angle)	0	N/A
Percentage below threshold distance	Different between right range avg. and left range avg.	0.05 (speed); 3.53 (angle)	>5	46 secs

From these results, it is certainly noticeable that one of the steering angle variables used for analysis, namely the difference between the minimum left and right range, wasn't suitable for this study. The other variable was the more reliable one, as the polynomial output managed to control the vehicle in a way that was stable enough for it to continuously perform laps without any issues, regardless of the high final cost for analysis of LIDAR distance with respect to speed in the second trail down. The fact that the former variable mentioned was unsuitable was confirmed by the results of the second round of trials, those with obstacles on the course.

Table 2. Results for Trials with Obstacles. The following is a set of 6 trials during which there were numerous obstacles on the course, each trial with a different combination of variables analyzed. The maximum acceleration of the vehicle was clamped at 0.4 m/s^2 for both potential field and learned control, and all the gradient descent output polynomials were of the third degree.

With Obstacles				
Lap time with potential field: 56 secs				
Speed Method	**Angle Method**	**Final Cost of Gradient Descent**	**Laps Completed**	**Lap Time**
LIDAR distance at zero	Difference between min right range and min left range	128.42 (speed); 82.46 (angle)	0	N/A
LIDAR distance at zero	Different between right range avg. and left range avg.	129.51 (speed); 50.93 (angle)	>5	76 secs
Min. distance at current frame	Difference between min right range and min left range	7.75 (speed); 80.65 (angle)	0	N/A

[1] We knew that the speed and steering angle would increase on one end and decrease on the other end of a given axis, but a cubic distribution, as opposed to a linear distribution, would also include the fact that the vehicle experiences both linear and angular acceleration in its movement.

Min. distance at current frame	Different between right range avg. and left range avg.	7.73 (speed); 48.67 (angle)	1	N/A*
Percentage below threshold distance	Difference between min right range and min left range	1.87 (speed); 82.64 (angle)	0	N/A
Percentage below threshold distance	Different between right range avg. and left range avg.	1.9 (speed); 59.85 (angle)	2	48 secs
*Not enough distance covered to accurately determine time per lap				

Once again, trials where the variable analyzed with respect to steering angle was the difference between the left and right minimum ranges were all unsuccessful. The trails that were successful, however, revealed which variables analyzed with respect to speed were truly reliable, and the tradeoff between timing and distance. Overall, the results observed were affected by many different factors, from starting point, to the exact amount of data captured, to the precise locations and numbers of the different obstacles. Although similar versions could be recreated, it was noted that exact values were very difficult to replicate. Changing one of the factors noted here could have a drastic effect on the output values and on the learned control of the vehicle in each trial, and there were certainly obstacle setups that were too obstructive, and needed to be changed for the vehicle to successfully complete at least a single lap. Nevertheless, these results did show some promise, and could very well be improved even further with more time and deeper investigation.

IV. Discussion and Conclusions

The results of this study gave some key insights about the application of machine learning to an autonomous movement algorithm such as the potential field algorithm used. They revealed which variables were more reliable to analyze, and which variables were more useful in the cases with obstacles. Per the results in Table 1 above, it made very little difference which variable was analyzed with respect to the vehicle's speed. This makes sense, as the vehicle did not come close to any solid surfaces when driving around on a course without obstacles, and therefore the potential field algorithm did not need to vary the vehicle's speed as drastically as it needed to with obstacles. Correspondingly, the vehicle kept a consistent speed, and could navigate once around the course with the potential field algorithm once almost 20 seconds faster than it could with obstacles. As mentioned previously, one of the variables analyzed against the vehicle's steering angle turned out to be unsuitable for this study, and causing the vehicle to turn too rapidly and unpredictably, and preventing it from completing even a single lap around the course. In the three cases where the other variable for angle was used, the final cost output of the gradient descent function was significantly lower, usually by about 20%-40%, and the vehicle's turning was much smoother. In these three cases, the vehicle could continuously navigate the course under learned control, even if the various imperfections of its movement caused it to take about 8 seconds longer per lap.

As shown by the results in Table 2, it was much more difficult to obtain consistent results with obstacles on the course. A large factor that determined how well the vehicle would perform was the precise setup of the obstacles, which needed to be changed several times until the vehicle could make it through enough laps to acquire the necessary amount of learned control data. Once this was set, the vehicle could operate along the course with large fluctuations in speed and steering angle, leading to larger final costs of the gradient descent algorithm in most cases. The one place where the final cost was lower was for the speed data in the first two trials with obstacles. This may have been due to the definition of the variable being analyzed against speed, the LIDAR distance in front of the vehicle, the correlation of which would have been more reasonable when the vehicle encountered obstacles in front of it and slowed down correspondingly. The most successful trials with obstacles included the second trials and the last trial. In the second trial, the vehicle could do laps continuous with occasional close encounters with obstacles, but took a much longer time (76 seconds per lap) than the original potential field control time of 58 seconds per lap due to its sporadic oscillations in both speed in steering angle. Despite these oscillations, the final cost of angular gradient descent was somewhat smaller than that of the last trail. In the last trial, the vehicle showed surprisingly high speed output values, and therefore managed to complete several laps around the course in a much shorter time than was expected. It could only complete several laps, however, due to the somewhat higher cost of its angular gradient descent, indicated by the uncertainty in its turning.

Although the results we received certainly did have some promise, they were also affected by many possible sources of error, both in terms of the general algorithm utilized and in terms of its specific parts. First, because this study made use of UDP network sockets for data transfer, there was always the opportunity for packet loss, although the analysis code running in Python observed a seemingly constant stream of data. This means that there was possibly unnoticeable loss of data that went unaccounted. Next, although the difference between acceleration and speed in this paper is clearly noted, there was a conflict between speed and acceleration values in the structure of the study itself. While the vehicle's control script, quite reasonably, could only accept acceleration values, the script responsible for outputting the vehicles speed data was yielding

the vehicle's actual speed, and therefore the data being analyzed and then used to control the vehicle was the its speed values. Although this did not impact the experiment too heavily, it led to varied results during the gradient descent process for the vehicle's speed. Also, the fact that small changes to course start and end points, the positions of obstacles, and the numbers of obstacles had relatively large impacts on the quality of a trial's results made this experiment especially difficult to conduct. These factors may be a suggestion that the study could be greatly improved if a different autonomous movement algorithm or machine learning algorithm were used. This suggestion, however, could only be confirmed with more time and further investigation.

In conclusion, our results indicated that it was in fact possible to apply a machine learning algorithm such as gradient descent to data coming from a vehicle being controlled by a movement algorithm such as a potential field algorithm, and to control the vehicle successfully with the output of this machine learning analysis. The experiment conducted showed that this could be demonstrated a continuous course both with and without obstacles, if the obstacle setup does not prevent the vehicle from completing enough laps to obtain a reasonable amount of data. In the future, this study could be expanded in numerous ways, one of which could include the testing of other types of vehicles, along with more options for variables that could be analyzed against output control variables.

V. Applications to Future Research

Self-driving cars and autonomous systems have become an area of major development for computer science researchers and vehicle manufacturers across the world. As algorithms improve, it's important to recognize the importance of both simulated testing and the incorporation of machine learning into obstacle avoidance. Through machine learning, scientists and researchers can cut the time needed to perfect values and algorithms by letting the machine tune its own variables in a simulated environment, without having to spend large sums of money on testing equipment.

This study showed how machine learning can enhance potential field based obstacle avoidance and tune its variables over time. Similarly, machine learning can be applied to all sorts of obstacle avoidance algorithms, as well as simultaneous localization and mapping (SLAM) algorithms.

Applications of this research can be used for different vehicles, such as drone flight. For example, the DJI Phantom 4 quadcopter is one of the only commercially available drones with obstacle avoidance. Smart algorithms designed with machine learning tuning in mind can enable other companies and manufacturers to provide drones that can avoid obstacles and stay safe during flight. The simulator in this study can be easily modified to emulate many different types of vehicles, and can take any type of autonomous movement algorithm as input.

References

1. Wang T. Collision-Free Path Planning using Potential Field Method for Highly Redundant Manipulators [Internet] 2016 [Cited February 2016]. Available from taylorwang.wordpress.com/2012/04/06/collision-free-path-planning-using-potential-field-method-for-highly-redundant-manipulators/

2. Barraquand J, Langlois B, Latombe J. Numerical potential field techniques for robot path planning. IEEE Transactions on Systems, Man, and Cybernetics. 1992; 22(2):224-241.

3. Karaman S. Basics of Autonomy: Planning and Control. Presentation; 2016; Massachusetts Institute of Technology.

4. Ng, A. Y. (2016). Programming Exercise 1: Linear Regression [pdf doc]. Retrieved from course available at: https://www.coursera.org/learn/machine-learning