**Self-Driving Vehicle Using Neural Network and Genetic Algorithm**

Final Report

Students: Daniil Ionov, Paul John Gonzales.

Program: Computer Engineering Technology.

Course: TPJ655 - Technical Project.

Instructor: Benjamin Shefler.

Date: March 19, 2018.

SENECA COLLEGE OF APPLIED ARTS AND TECHNOLOGY

1750 Finch Avenue East, Toronto, ON, M2J 2X5

Tel. 416-491-5050

www.senecacollege.ca

**Table of Content**

Contents

[Executive Summary 3](#_Toc511382034)

[Introduction 4](#_Toc511382035)

[Functional Features of the Product 5](#_Toc511382036)

[System Specifications 5](#_Toc511382037)

[Operating Instructions 6](#_Toc511382038)

[Product Design, Implementation, and Operation of the System 8](#_Toc511382039)

[Block Diagram 8](#_Toc511382040)

[Theory of Operations 9](#_Toc511382041)

[Machine Learning Library 9](#_Toc511382042)

[Unity Simulation 13](#_Toc511382043)

[Python Script 17](#_Toc511382044)

[Hardware Components and Graphical User Interface 18](#_Toc511382045)

[Hardware 18](#_Toc511382046)

[Software 19](#_Toc511382047)

[Maintenance Requirements 22](#_Toc511382048)

[Conclusion and Further Developments 22](#_Toc511382049)

[Appendix 23](#_Toc511382050)

[Appendix A - Electrical Schematics 23](#_Toc511382051)

[Appendix B - Parts List 23](#_Toc511382052)

[Appendix С - Citations and References 25](#_Toc511382053)

[Works Cited 25](#_Toc511382054)

[Appendix D - Contact Information 26](#_Toc511382055)

[Appendix E - CD 27](#_Toc511382056)

[Description of the CD attached 27](#_Toc511382057)

[Appendix F – Simulation Images 27](#_Toc511382058)

[Figure 1. Product Block Diagram. 8](#_Toc511380952)

[Figure 2. Neural Network Structure of the Product. 10](#_Toc511380953)

[Figure 3. Tanh Activation Function. 10](file:///C:\wb\Report.docx#_Toc511380954)

[Figure 4. Uniform Crossover of the Genotype. 11](file:///C:\wb\Report.docx#_Toc511380955)

[Figure 5. Sigmoid Activation Function. 12](file:///C:\wb\Report.docx#_Toc511380956)

[Figure 6. Raspberry Pi 3 Model B. 18](#_Toc511380957)

[Figure 7. Ultrasonic Ranging Module, HC-SR04. 18](#_Toc511380958)

[Figure 8. Main Menu of the Simulation Application. 19](file:///C:\wb\Report.docx#_Toc511380959)

[Figure 9. Initial Setting from Main Menu. 19](file:///C:\wb\Report.docx#_Toc511380960)

[Figure 10. Pause menu of simulation. 20](file:///C:\wb\Report.docx#_Toc511380961)

[Figure 11. Main screen of simulation. 20](file:///C:\wb\Report.docx#_Toc511380962)

[Figure 12. Car Physics Setting from Pause Menu. 21](file:///C:\wb\Report.docx#_Toc511380963)

[Figure 13. Neuroevolution Settings from Pause Menu. 21](file:///C:\wb\Report.docx#_Toc511380964)

[Figure 14. Voltage Divider Circuit for TTL to CMOS Conversion for Ultrasonic Ranging Modules (HC-SR04). 23](file:///C:\wb\Report.docx#_Toc511380965)

[Figure 15. Parts List. 24](#_Toc511380966)

[Figure 16. Contact Information. 26](#_Toc511380967)

[Figure 17. AutoCAD Drawing of the Physical Car. 27](#_Toc511380968)

[Figure 18. Designed Unity 2D Car. 28](#_Toc511380969)

[Figure 19. Designed Map Created in Adobe Photoshop. 28](file:///C:\wb\Report.docx#_Toc511380970)

# Executive Summary

This report will provide readers the opportunity to learn about neural network algorithms demonstrated by real use cases. The product will be demonstrated by the team that has created the project and will be shown in Seneca College Newnham Campus. The project solves a variety of different problems such as showing people that neural networks are not as complex as it seems and how it could be implemented to solve real life problems. These problems in particular are how self-driving cars will prevent car crashes and ease the amount of work done by the driver to prevent drowsy driving. The extensive research and development of this project will contain many different theories that the team has devised will best suit the needs of the remote control car hardware. This includes how the hardware will mimic the simulated car in the Unity3D software.

# 

# Introduction

A dilemma in the modern world are car accidents that occur from drowsiness or being under the influence. The National Highway Traffic Safety Administration from the United States, says that drowsy driving was responsible for 72,000 crashes, 44,000 injuries and 800 deaths in 2013. The purpose of this project is to prevent these types of accidents from happening by demonstrating a way of implementing artificial intelligence. The project’s artificial intelligence will be approached and tested by showcasing a remote controlled car as a representation of a real vehicle. This is a great way of presenting a neural networks capabilities as a real life use case by showing how automation could save a great deal of lives in the near future.

It is important for everyday processes to be automated to innovate the development of society. Smart systems are gradually taking part in our life to make everything easier and effortless. An important aspect of a smart system is the ability to learn and adjust itself for unexpected conditions such as change in environment or movements. Machine learning is widely used in everyday life for a means of assistance in modern smartphones, self-driving vehicles, search engines, visual aid, and more.

This product will demonstrate how several types of sciences can be used together to implement automated and/or improve any activity that involves human interaction. These sciences include the use of artificial neural networks and evolutionary algorithm, which are both inspired by neuroscience and the evolution theory but are showcased using mathematics. It is an effective way to implement machine learning in complex processes. The project will augment an existing remote controlled vehicle so that it will be able to drive automatically and avoid obstacles without any human help and a means of being able to override the artificial intelligence for the human to take over the driving.

# 

# Functional Features of the Product

Product requirements:

* Ability to drive itself and avoid obstacle:
  + Turn left or right on incoming obstacles.
* Operation mode to be manually driven from smartphone application,
  + Turn left or right,
  + Move forward or backwards,
  + Switch back to auto-pilot mode.
* Visual feedback on car’s Power-On Self-Testing (POST):
  + The color of the LED indicates different stages of startup process.
* Cloud service connectivity:
  + Simulation application trains then sends the neural networks,
  + Raspberry Pi requests neural networks from Cloud Service and obtains the latest neural network.
* Real-time simulation of machine learning running on server.
  + Evolutionary algorithm to train neural networks to avoid obstacles.
* External power supply.

# System Specifications

Product specifications:

* 6 Ultrasonic Ranging Sensor HC-SR04:
  + Range: 2 cm - 100 cm.
* 1 RGB LED’s,
  + Forward voltage: 2 V.
* Raspberry Pi 3 Model B microcontroller,
* Three-wheeled smart car for Raspberry Pi from Freenove:
  + Velocity range: -10 cm/s - 10 cm/s,
  + Instantaneous torque: 30 degrees from forward position (left or right).
  + Powered up by external power supply.
* Google Drive Service,
* Custom made C# library for machine learning,
* Custom made smartphone application:
  + Built for Android OS,
  + Created in Unity3D game development application,
  + Written in C#.
* Custom made simulation application:
  + Created in Unity3D game development application,
  + Written in C#.
* Custom made car firmware:
  + Written in Python,
  + Interfaced with Raspberry Pi 3.
* Bluetooth connectivity from smartphone to Raspberry Pi,
* Wi-Fi connectivity to cloud service.
* External Power Supply:
  + 8650 3.7 V Lithium-Ion batteries (x2).

# Operating Instructions

**IMPORTANT:** Simulation must be ran for at least 6 hours before being able to use the autopilot system of the remote controlled car.

**REMINDER:** The following recommended simulation settings were found by repeated testing of the simulation over the course of the weeks.

**HOW TO RUN THE SIMULATION EXECUTABLE:**

* The simulation executable file is only available for Windows operating system. Linux and Mac is not supported.
* Before starting the simulation, understand your computer system settings to fully optimize your simulation experience.
* When the executable file is launched, a configuration window will open prompting for many different graphic settings.
* Any screen resolution or windowed mode is fine.
* ‘Fastest’ graphics quality settings is recommended as the simulation does not focus on having great graphics but needs to be ran smoothly as it utilizes the most of the computer’s CPU usage .
* The user may change the input settings for their own comfortability.

**USING THE SIMULATION SOFTWARE:**

**WARNING!:** Simulation may become unstable if too many cars are spawned and the user’s computer CPU is not able to handle the amount of spawned cars.

1. **Press the Start Simulation button in the main menu:**
   1. A prompt window will pop up asking for the user to enter three prompts.
      1. The most optimal number of population is **64**.
      2. Enter the neural layer info exactly as **7 15 5 2** including the spaces (7 input nodes, 15 hidden nodes, 5 hidden nodes, and 2 output nodes).
      3. Enter a mutation rate of **0.5**. (Numbers cannot exceed greater than 0.5)
   2. User may tick the enable File IO toggle to save the best brains of the species and used for the car. When the toggle is ticked, two properties will become visible.
      1. Copy and paste the folder path of where to save the best brains of the car on to the enter file path textbox.
      2. Toggle load old population **only if you already have a saved population file.**
2. **User may now run the simulation using the default settings of the car physics and AI settings.**
3. **Toggle the minimap to view the rest of the track by pressing the Enable minimap toggle on the top right of the screen.**
4. **Speed up the simulation by moving the slider on the right of the screen.**
5. **Press the escape key or click the pause button to open the pause menu.**
   1. Press the resume button to continue the simulation.
   2. Press the Change AI Settings button to change the settings of the neural network.
      1. Choose any settings to change by ticking the toggle boxes or entering numbers on the textboxes.
      2. Press the apply button to save the new settings.
   3. Press the Change Car Settings button to change the physics of the car.
      1. Choose any settings to change by ticking the toggle boxes or entering numbers on the textboxes.
      2. Press the apply button to save the new settings.
   4. Press the Quit button to reset the simulation back to default settings and go back to the main menu screen.
6. **When simulation is complete, save the best brain and population files onto the google drive for the car to download the new neural network automatically.**

**USING THE REMOTE CONTROLLED CAR:**

**REMINDER:** The bluetooth remote controller app is only available on android apk.

**CAUTION!:** Make sure the car is on a flat ground without any water around to damage the circuit of the car.

1. **Press the two buttons on the top of the enclosure of the car.**
2. **Wait until the postcode of the RGB LED is blue.**
3. **Open the Bluetooth controller app.**
   1. When the application asks to enable bluetooth press yes.
4. **Press the Scan for Bluetooth Devices button**
   1. Wait until the Scanning… Please Wait… label has disappeared for the discovery to finish. (Takes around 15-25 seconds)
   2. The bluetooth devices that were scanned will be loaded onto the dropdown list in the middle of the screen.
   3. User may press the Load paired devices button if the selfdrivingrpi device has already been paired with the local android device.
5. **Choose the selfdrivingrpi device in the dropdown menu.**
6. **Wait until the Status label updates that the connection has been made with the selfdrivingrpi device.**
7. **User may now control the speed of the car using the vertical joystick on the left and control the direction of the car using the horizontal joystick on the right.**
8. **For the user to enable the autopilot mode, toggle the turn on autopilot mode box.**
9. **To stop using the car, press the Stop connection button on the app and the two buttons on the top of the car’s enclosure.**

# 

# Product Design, Implementation, and Operation of the System

## Block Diagram

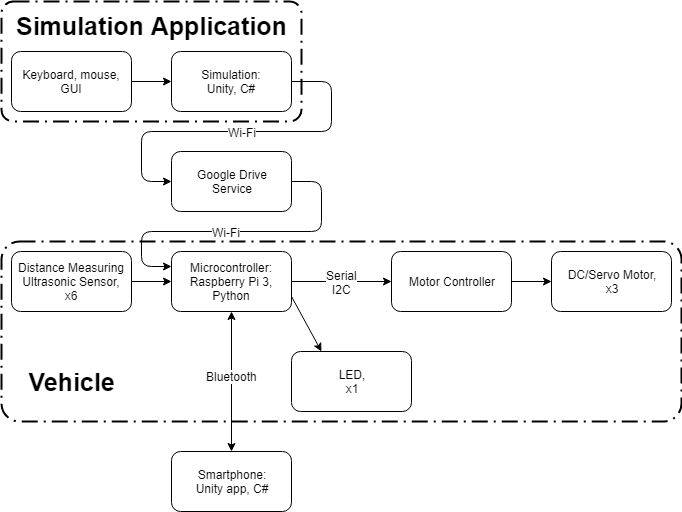


Figure . Product Block Diagram.

## Theory of Operations

The remote-controlled car was constructed with six ultrasonic sensors, servo and DC motors, and RGB LED. All these components are designed to be controlled by a Raspberry Pi and the commands are devised in python. A real-time simulation is running simultaneously with the remote controlled car to continuously update the best performing species (neural network) to the Google Drive service. When the Raspberry Pi retrieves the data, it is then able to automate the car related to the best performing data in the simulation.

A mobile app on Android is created so the user is able to choose between auto-driving or manual mode. The application contains two controllers for the user to use in manual mode.

### Machine Learning Library

Machine learning based systems are easy to scale up in complexity. For example, this product uses six ultrasonic ranging modules to augment a vehicle. A brute force approach to develop firmware for the vehicle could be a time-consuming process; however, machine learning requires much less human resources to be part of the development of firmware. Custom-made machine learning library is used to develop optimal logic for driving and obstacle avoidance. The advantage of such approach is that the number of sensors could be easily increased (modern smart vehicles use up to 12 ultrasonic ranging modules as well as radar systems) to meet the desired level of awareness of surroundings.

The main machine learning algorithm is neuroevolution, and it is based on real theory of evolution proposed by Charles Darwin. Neuroevolution uses two main branches of machine learning: neural networks and evolutionary algorithm.Neural networks are virtual representation of real neuron systems, such as brain.

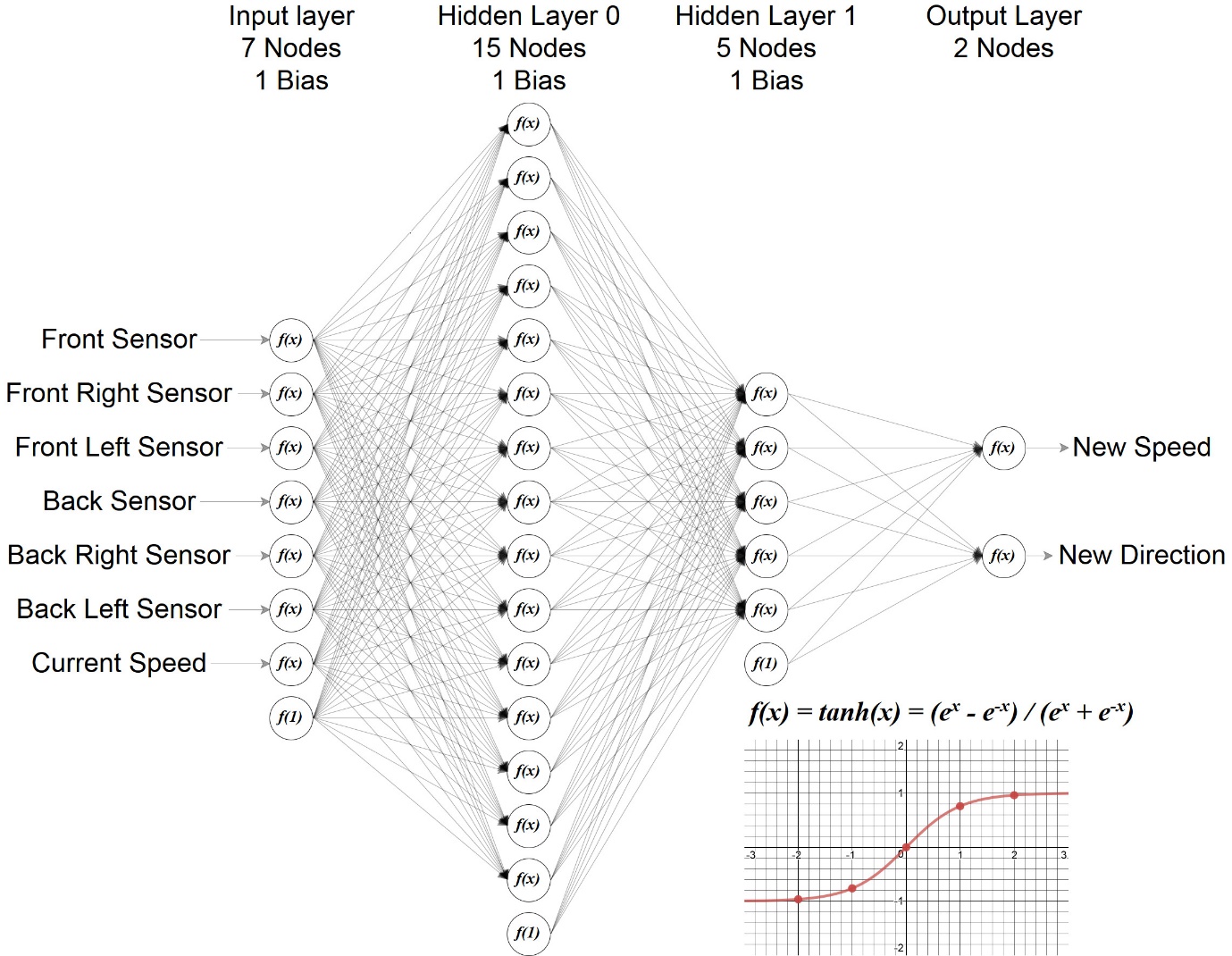
Typical neuron network consists of neurons (nodes) and weights (connections between neurons, stored as floating point number) as shown in Figure 2.

Figure . Neural Network Structure of the Product.

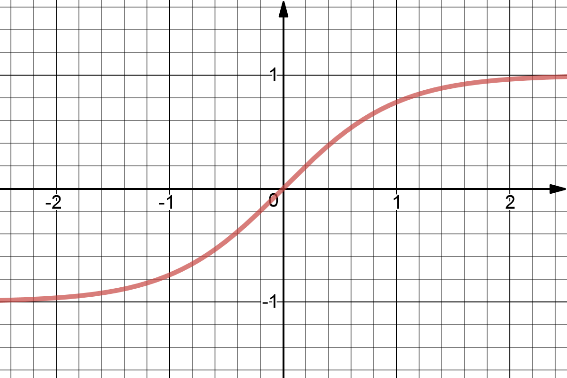
Neural networks are structured by layers of nodes, where information from one layer is passed to another through corresponding connections. Neural networks consist of input layer, hidden layers, and output layer (the hidden layer is any layer between input and output layers). When the information is passed from one layer to another, the value of the node is passed to non-linear activation function, which is used to suppress node’s value to stay in suitable range for neural network to operate. The Tanh function is used because it handles positive and negative values, as shown in Figure 3, which is the best suited for simulation due to the speed of the car being positive or negative (mapped to be in range from -1.0 to 1.0). The value from activation function is then multiplied by the value of corresponding connection/weight and added to the next node. Neural network can be described as complex mathematical functions. Neural networks are written in C# programming language and stored as part of Machine Learning Library. Neural networks are written in object-oriented manner, where individual neural network is defined by neuron layers, which in turn are defined by two-dimensional arrays, or matrices, representing weights/connections.

Figure . Tanh Activation Function.

Evolutionary algorithm is based on theory of evolution, where the main concept is “survival of fittest”. This machine learning library met all main criteria of evolutionary algorithm:

* Initial genotype diversity,
* Ability to pass information from one generation to another,
* Defined fitness function.

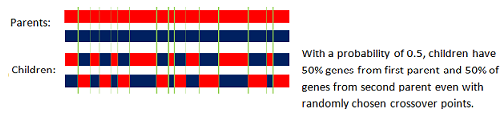
The evolutionary algorithm uses concept of population of species which are subjected to natural selection. In this machine learning library, the initial population consist of randomly generated neural networks, with the size of population specified by the user. Evolutionary algorithm uses generations to define a population in the time frame. The generation is defined in the simulation by the maximum time allowed for cars to finish one lap or until all cars are crashed. At the end of each generation, all cars are subjected to natural selection and breeding algorithm. During the breeding, algorithm continuously chooses two parent species based on their performance and generates two more child species/neural networks to fill the new population for the next generation. When the two neural networks are chosen, their matrices of weights (two-dimensional arrays) are converted into 1 list of weights representing genome (DNA) of this species. Genotypes of the parent species are then subjected to uniform crossover algorithm, which mixes genotypes of parent species to produce child species, where each parent gene has 50% chance of occurring in child genome.

Figure . Uniform Crossover of the Genotype.

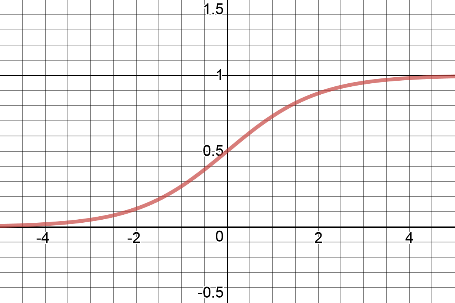
Once children genotypes are created, they are subjected to process of Gaussian mutation, which randomly changes each gene with the probability indicated by the user from the application. The breeding and crossover algorithms allows population to pass its information to the next generation, and mutation algorithm allow population’ genome to diverse when generation passes. The fitness function used in simulation is defined by distance travelled by each car, which could be determined with built-in function from Unity3D game engine. However, to emphasize the importance of being fast, the distance traveled is multiplied by inversed sigmoid function, which is shown in Figure 5, with the value of time spent by this car to travel such distance. This allowed cars to focus on their velocity as well as travelling the most distance. At the end of generation, during the breeding algorithm, the cars are selected randomly based on their fitness value (higher fitness value - higher the chance of being selected for breeding), which means that the car that traveled the most has advantage over the car that traveled less. Once new population is formed and the size reached the size indicated by user, the cycle repeats: new cars are tested in the simulated track and breeding cycle starts. Evolutionary algorithm is general purpose machine learning algorithm and best suited for applications in this product. The machine learning library is written in object-oriented way using classes. The Population class has property named Members, which consists of Species objects representing current generation. Also, Population class has property BestSpecies which stores the best performing species from previous generation (used in simulation as the green car). The Species class in derived from NeuralNetwork class, and has some additional properties, such as Fitness and DNA. The NeuralNetwork class has function called Guess(), which is used to determine new values for speed and direction of the car based on sensor data, and it is called each frame of the simulation by every car.

Figure . Sigmoid Activation Function.

## Unity Simulation

The Simulation of the car is represented in the Unity Game Engine 2018 Beta. The reason the 2018 beta had to be used is that it supports .NET Framework of 4.5+ which will help fulfill the requirements for the implementation of the machine learning library. Additionally, game engine libraries were a great resource for the simulation of the car as it gave many different options of creating the car physics and designing the car to match the real car as accurate as possible. To be able to find the best accuracy between the hardware car and simulated car, the car was measured meticulously using AutoCAD. Using the observations of the measured values of the angle and size of the car from AutoCAD (Figure 17), those values were then ported onto the unity game engine and designed to match the same width and height of the hardware car. Comparatively, the sensor placements were also very important to match the same angle of the hardware so careful measurements had to be made to be able to match the sensors from the enclosure of the real car. Without accurate measurements or placements of the sensors, the sensor data sent to the Raspberry Pi would have become obsolete.

After the values were ported properly to Unity Game Engine, the car size and width is designed using Unity’s transform feature and car sprites were downloaded from the asset store to design the look of the simulated car (Figure 18).Moreover, the car’s sensor placements were created using line renderers that contained collision detection through the Raycast function of Unity scripting API library. The line renderer had the same length and starting origin as the raycast, and the raycast acted as the range of the sensors. The raycast was able to detect any game objects in the simulation and find the fraction of the distance between the starting origin of the ray and the point of the rays detection of the collision (fraction values mapped from a range of -1.0 to 1.0). For the raycast to properly determine which game object to detect, layers had to be created to identify the boundaries of the track. An algorithm were created for the sensors to change colours dynamically depending how far the raycast collision was detected from the boundaries of the track. The algorithm multiplied the fraction value from the raycast by 255 to keep the green value and subtracted by 255 to keep the red value. For instance, depending how deep the raycast has hit the collider such as 25% of the range of the sensor, it would then be 25% red and 75% green.

The track of the simulation was created using Adobe Photoshop and was turned into a map by adding it as a background of the unity project in the Track scene (Figure 19). The black part of the track is the road and any other spaces of the track that does not contain the road is created as the boundary of the track. Boundaries were made using the polygon collider tool from the Unity editor, the track had to two different boundaries, one of the inner boundary and another for the outer boundary. The boundaries were created into a layer named “Boundary”, and the raycast was made to only detect layers named “Boundary” so that no other game objects would interfere with the collision detection. The points of the polygon collider tool of the track were carefully plotted to make sure the visual aspect of the boundary was exactly the same as the collider. The track was designed for the car to learn how to turn on complex turns thus the reason for the sinusoidal road at the very beginning of the track. After the sinusoidal road, there are points of the track where the car has to learn how to slow down and take a hard turn. The track was changed multiple times depending on the observations of each simulated tests so the car learns each aspect of a driving vehicle. Additionally, the road had to be as skinny as possible for the raycast to detect the boundaries as often as possible so the sensor data were accurate to the sensors of the real car.

The physics of the car was created by making the car game object as a 2D rigidbody. When a game object inherits the rigidbody component, it is able to be used to simulate the physics. The scripting API of rigibody makes the car able to have applied forces to it and control its position using simulated physics. Any use of the Rigidbody2D API must be used in the FixedUpdate() function of the script so that the physics runs in a fixed frame and the physics is applied every frame. The physics engine is able to handle how the car moves and acts towards a collision. For example, in the simulation, an added feature had been created so that when the car’s rigidbody collides with the boundary, it turns red and is now ‘dead.’ When the car is dead, there is a rigidbody constraint function used so that it freezes the physics and could never be moved until the car has been respawned. There are different forces applied to the car such as the torque force, speed force, or angular velocity. Using these forces the car is able to be controlled fully and customized to fit the requirements of the real car.

The “Self Driving Car” Unity Project contains many different scripts and artwork for the whole project to work. The scripts are used to control many different aspects of the project. Some of these scripts are named CameraFollow, Car2DController, GUIManager, LevelManager, MainMenu, PauseMenu, Respawn, SpeciesController, and TimeScale. The artworks are created in Photoshop and are cut into small 2D sprites to fit in as the graphical interface of different menus or buttons. There is also an implementation of file io and it saves the sensor data on to a .txt and .bin file so users are able to load in the old population of cars with the same data whenever they want to.

    The **CameraFollow** script was used to control the main camera of the scene. It was made to follow the farthest travelled car during runtime and will switch between different cars depending on who had covered the most distance. The camera script uses vectors in a 2D space and follows the cars position using its transform in the x and y axis.

    The **Car2DController** script was used to control the rigidbody and sensors of each spawned car. This script is where the forces are applied to the rigidbody and how the raycast is applied to the car. This script also measures the time alive and travelled distance of the car. As well as, creating a new instance of each angle of the sensors and draws the lines on the side of the cars.

    The **GUIManager** script controls the graphical user interface of the ‘Track’ scene. Some of these interfaces may include the updated text of each option on the simulation such as the AI settings and progress settings interfaces. These statistical updates are on a fixed frame using the Update() function of the script. The AI settings panel are static properties from the **LevelManager** script as it has been initialized the MainMenu scene. Values do not carry over to the next scenes, henceforth, static properties had to be created. The GUIManager script also controls the minimap feature of the simulation by creating a function of toggling the Minimap camera on or off depending on the bool value of the ‘Enable minimap’ toggle.

    The **PauseMenu** script controls the pause menu interface of the ‘Track’ scene. Most of the script uses the UnityEngine.UI library and TMPro libraries for the downloaded free TextMesh Pro asset. The script contains public properties that are needed to be initialized in the unity editor for the game object to be used within the script. Some of these properties may include the textboxes, panels, buttons, and toggles. The user interface in the unity editor contains button click events and changed visibility of the panels depending on the users inputs. Whenever a user clicks different options, visibility will change and game object actives will be set to false.  The PauseMenu script also accesses the LevelManager and SpeciesController scripts to be able to change the values of the settings that user decides to change. If the car settings are changed, every car in the simulation that was spawned must change and must loop to each car and change its values. Each settings that are able to change are put into a try catch statement just in case the values do not change and the program will close.

    The **Respawn** script controls the respawn event of the car but also shows how the car should react if it collides into the boundary. The script is used when every car dies in the simulation or when the time limit reach its end. When cars respawn, it resets the cars physics and position back to zero. It also uses the OnCollisionEnter2D function and only occurs when the gameobject hits the tag named ‘Boundary’. In the event that it does hit the boundary, the script will freeze the rigidbody of the car and changes the color of the car to red.

    The **SpeciesController** script implements the machine learning library and controls the behaviour of how each car should drive by itself. The script contains the initialization of each of the neural network settings for the car and uses the Car2DController script to access the physics of the car. Some of these settings may be the mutation rate, population, type of crossover, and fitness function. The cars are also created in the species controller and instantiated onto the scene and designed to ignore its own layer collision so that each car won’t be able to hit each other. SpeciesController also assigns the brain onto each car and could now be used for the neural network. The neural network will be able to react depending on the sensory data and speed of the car. The FileIO feature is also called in the species controller and writes and saves the file whenever 50 generations have passed.

    The **MainMenu** script initializes the basic settings of the neural network such as the layer info or mutation rate so users are not bombarded with configuring the advanced settings of the neural network. The script almost acts the same as the PauseMenu script as it contains public properties of textboxes, toggles, and panels and has to be initialized within the unity editor to be used within the script. If the properties are initialized, the properties would be customizable using the scripting API.

**The Bluetooth Controller Mobile Application**

    The Bluetooth remote control app is created using the Unity Game Engine and implements a bluetooth library from Unity Store created by TechTweaking. The bluetooth library is able to make the device running the library act as a client or server. The library mainly supports android devices and connection to any type of microcontroller which in this case is the Raspberry Pi 3. The app also uses many different types of scripts to control the joystick, bluetooth and graphical user interface. These scripts are called; HorizontalJoystick, VerticalJoystick, GUIManager, Controller, and BTLibraryController.

    The bluetooth library used by the mobile app is mainly used to act as a bluetooth client and does not use any functions as a bluetooth server. The only way to use the bluetooth is for the user to connect to the Raspberry Pi 3 bluetooth as a client. There are many different functions and events that are used from the Bluetooth library and could be accessed using the BluetoothAdapter class. The BluetoothAdapter class is used to handle the local devices’ bluetooth adapter such as the being able to connect to the bluetooth server or raising any events when the device is connected or disconnected. Another class used in this library is the BluetoothDevice class. This class could access the local android devices’ info such as mac address, device name, or signal strength. On the other hand, it could also access the public info of a scanned bluetooth device.

    The **HorizontalJoystick** and **VerticalJoystick** script is used to control the animation and position of the joystick. The Joystick scripts inherits from classes such as Monobehaviour, IDragHandler, IPointerUpHandler, and IPoinbterDownHandler. These classes are from the UnityEngine library and also uses UnityEngine.UI and UnityEngine.EventSystems. The drag, pointer up, pointer down handlers are inherited so that the animation of the joystick could be created depending on the actions of the user. From these inherited classes, the app is able to use OnDrag, OnPointerUp, and OnPointerDown functions. The OnDrag function occurs when the user drags the image of the joystick. In this function, the animation is created so that when the user is dragging the image, the image will move along the users dragged position. The animation of the joystick is created by the RectTransformUtility function by the UnityEngine.UI library. The OnPointerDown function casts its pointer event data and sends it to the OnDrag function and only occurs when the user is clicked onto the joystick. And the OnPointerUp function occurs when the user lets go of the joystick. When the joystick is let go, it will move the joystick image back to the zero position and moves the Vector3D back to zero. To be able to access the output of the joystick, vectors are used depending on the position of the joystick and are mapped from a range of -1.0 to 1.0. When the horizontal joystick is moved, the script will output a vector in the x axis from -1 to 1. When the vertical joystick is moved, the script will output a vector in the y axis from -1 to 1. These outputs are sent to the **BtLibraryController** script.

    The **GUIManager** script is used to control the graphical user interface of the mobile application. The script enables for customization of public properties such as dropdown menu, scroll bar, buttons, toggle buttons and labels. In the unity editor, game objects are initialized into the public properties and are then editable using the script. Depending on the events of the **BTLibraryController** script it will change the properties. For example, if the scrollbar value is changed, it will add delay of values sent to the Raspberry Pi 3 using the timescale function. Or if a toggle is clicked on the mobile app it will change the labels of the status.

    The **BtLibraryController** script handles all of the bluetooth events of the mobile app. The BluetoothAdapter class at the start of the script, is used to ask the user to enable its bluetooth on the device. There are created event handlers that are raised whenever an event happens between the BluetoothAdapter and local device. For example, some of these events are; when the device is discovered, it raises an event handler that is used to enter info onto the BluetoothDevice class and and adds the info onto a list. When discovery is finished, it raises an event handler that is used to add the list of devices discovered onto the dropdown menu. When the devices are listed on the dropdown menu the user is able to choose which device to connect to, the device class is then used to connect to the device chosen in the dropdown menu. The scan button raises an event that will search the area for bluetooth device up to a range of 50 meters and usually takes around 15 to 25 seconds long. If the user moves the joysticks in manual mode it will send an output of ascii encoded text in the format of “ mode:value “, the format is inbetween spaces so that when the Raspberry Pi receives the ascii encoded text in bytes, it will be able to use index splicing to separate each sent value. The mode part of the format is either ‘Vertical‘ or ‘Horizontal’ and the value sent is either from ‘-1’ to ‘1’. If the user toggles to make it automatic, it will continously send an ASCII encoded text ‘ AUTOMATIC ‘. Each frame it will send the ‘ AUTOMATIC ‘ text so that the car firmware understands it is on autopilot mode.

### Python Script

The Species class from Machine Learning library (written in C#) is ported and refactored to be used in Python script from Raspberry Pi. The Species object is used to control the physical car with the neural network trained from simulation. To obtain the latest neural network from Google Drive service, the RClone command line program was installed and used on Raspberry Pi. As for the Bluetooth library, python-bluez module was installed and used in main python script. Once main script starts, the car goes into Power-On Self-Testing (POST) procedure:

* Yellow light of LED indicates successful loading of latest neural network from Google Drive,
* Purple light of LED indicates failure during loading of latest neural network,
* Light-Blue light of LED indicated that local neural networks has been successfully loaded and car is ready to be connected to Bluetooth device,
* Red light of LED indicates that local file could not be loaded and the main script is shutting down.

The main script uses Bluetooth library to enable Bluetooth server on Raspberry Pi using RFCOMM protocol and waits for incoming client connection. Once connection is established, server receives two types of data from the client: mode of operation and manual settings. Since Raspberry Pi receives encoded data, this data must be first decoded using UTF-8. The mode of operation is determined by the received data string, which can be “ MANUAL “ or “ AUTOMATIC “. When “ AUTOMATIC “ is received, the car goes into auto-pilot mode, where neural network controls the car and LED is set to green. When the car is in manual mode, it receives instructions in form of “ MODE:value “, where mode can be “HORIZONTAL” or “VERTICAL”, and value is the user controlled direction or speed respectively. If the car loses connection to client, it goes into waiting mode, where LED is set to bright blue. During the waiting stage, car does not move and waits for new connection. The ultrasonic ranging modules work using two transducers each, one of them being used to send sound wave, and other to receive it. The distance to the nearest object is calculated using formula, where *v* is speed of sound in the air, *t* is total time taken to reach he object and bounce back (the division by two occurred because sound traveled twice of the distance in total), and *d* is distance to the object in meters. When all sensor data has being obtained, main script uses Guess function of Species object to predict optimal speed and direction based on the sensor data and current speed of the car. Motor driver controller is interfaced with Raspberry Pi using I2C serial communication protocol. The main python script uses smbus library to send commands to motor controller (the commands are provided by the manufacturer of the base of the car). The motor driver controls two DC motors as the main moving power, and two servo motors to control direction of movement and position of the front ultrasonic sensor.

## Hardware Components and Graphical User Interface

### Hardware

This product uses Raspberry Pi 3 Model B as main micro computing unit. The specifications of Raspberry Pi 3 Model B, shown in Figure 6, are the following:

* CPU: Quad Core 1.2GHz Broadcom BCM2837 64bit,
* Memory: 1GB RAM,
* Network: BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board,
* IO: 40-pin extended GPIO, 4 USB 2 ports, 4 Pole stereo output and composite video port, Full size HDMI.

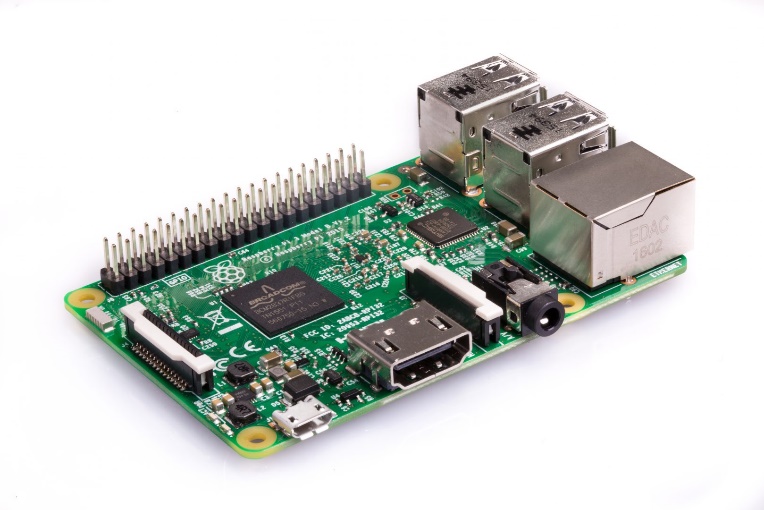


Figure . Raspberry Pi 3 Model B.

Raspberry Pi is interfaced with 6 ultrasonic ranging modules (HC-SR04, Figure 7), motor driver controller and RGB LED. Ultrasonic ranging modules are operated at CMOS voltage logic level, which is 0.0-5.0V; however, Raspberry Pi is operated at TTL voltages (0.0-3.3V). The simple voltage divider circuit was constructed to maintain compatibility of HC-SR04 output pin and Raspberry Pi’s input pin, which is shown on Figure 14.

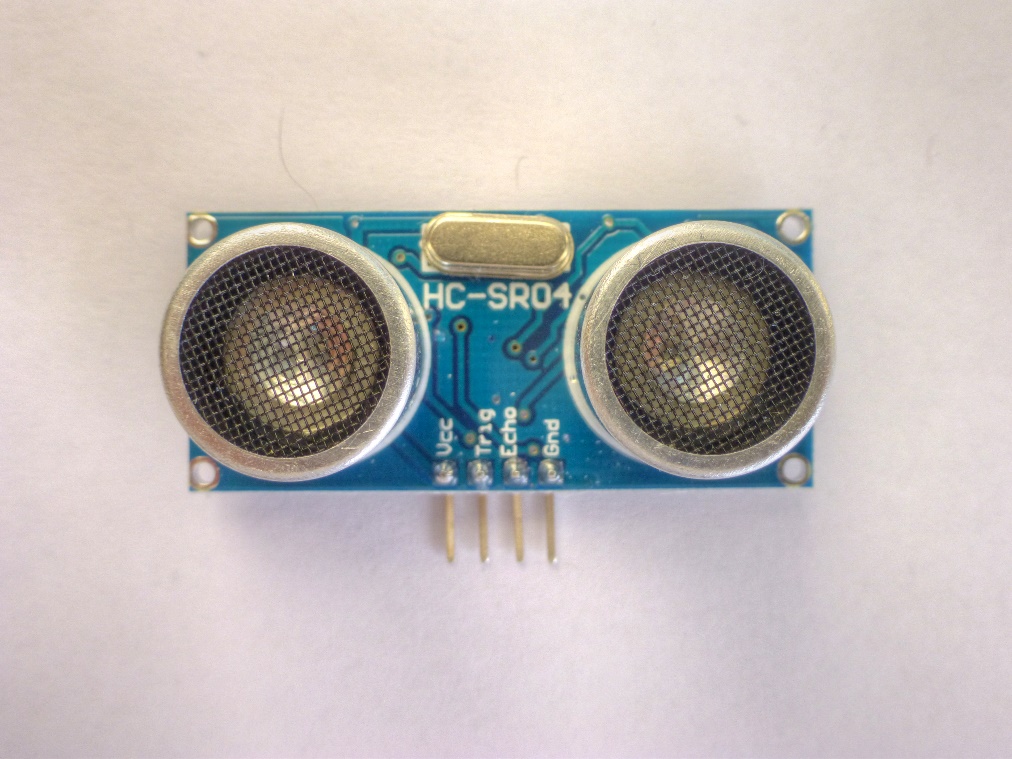


Figure . Ultrasonic Ranging Module, HC-SR04.

### Software



Figure . Main Menu of the Simulation Application.

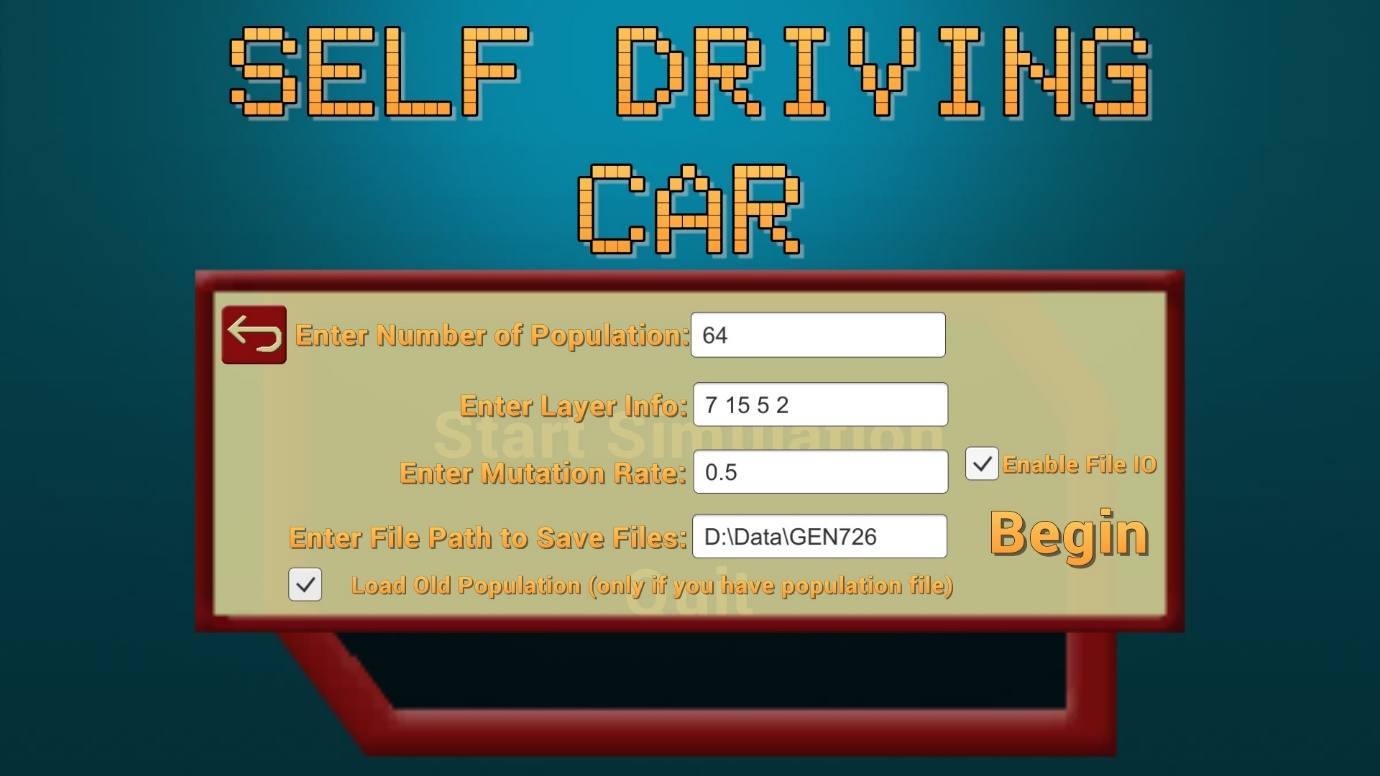


Figure . Initial Setting from Main Menu.

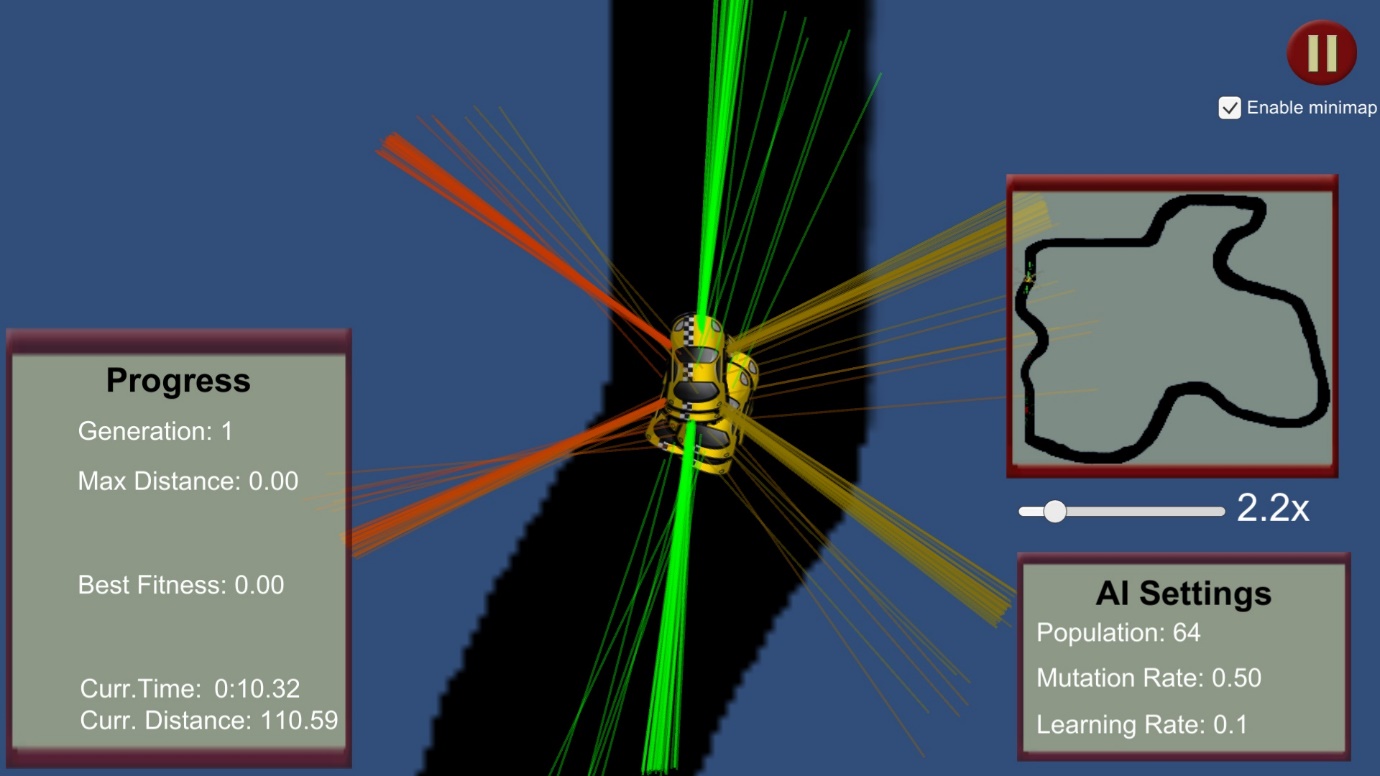
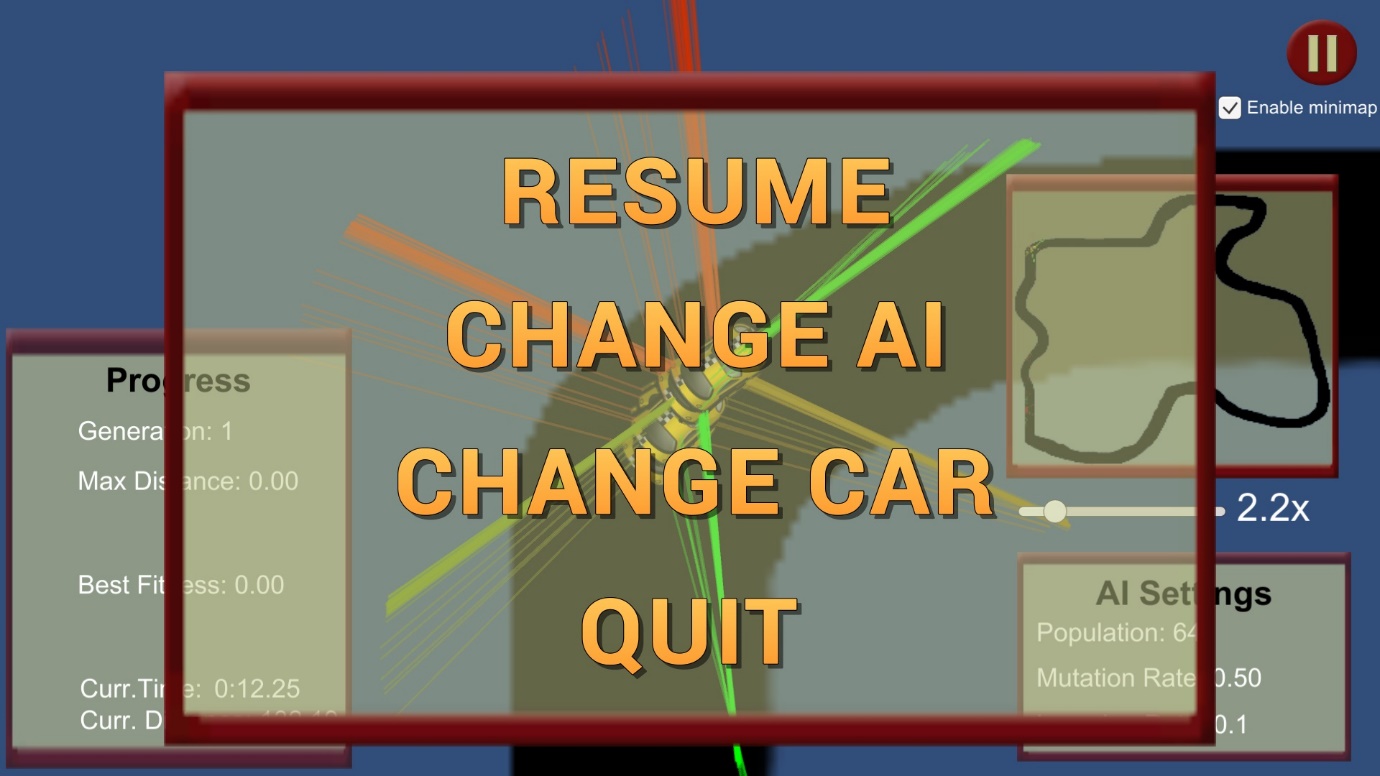


Figure . Pause menu of simulation.

Figure . Main screen of simulation.



Figure . Car Physics Setting from Pause Menu.

Figure . Neuroevolution Settings from Pause Menu.

# Maintenance Requirements

1. Make sure no wet substance is near the circuit of the car.
2. Car must be on a flat surface before being driven for maximum optimal driving performance of the autopilot system.
3. Make sure that the buttons on the top of the enclosure is connected to the load and ctrl button of the circuit.
4. Enclosure is safely in place.
5. Restart bluetooth controller app if the bluetooth does not connect to the device.

# Conclusion and Further Developments

To conclude the product, developing real use cases with artificial intelligence could help mold a safer and efficient future. If neural network algorithms continue evolving in the rate it is now, it will definitely surpass human intelligence in all tasks. It is already defeating humans in games and data analysis.

This product is planned for further developments by using the already made genetic algorithm to find the best location and number of sensors placed on the car in a 3D space rather than 2D. The Machine Learning library is planned to include convolutional neural networks, which can be used for pattern recognitions and image processing. With the addition of camera, using convolutional neural network it will be possible to recognize road signs, such as speed limits, and upcoming obstacles. A recursive modules are planned to be added to the neural network, allow neural networks to have a long and/or short tern memory. With the further developments, the user will have full control over the Machine Learning library from the simulation application. This product is planned to have enclosure for outdoor and/or child use. The product will have a database, where all users will be able to share trained neural networks and contact developers. A GPS module is planned to be used so that the neural network can learn a real map rather than the simulation software’s map. The software of this product is planned to be cross-platform and support Microsoft Windows, MacOS, Linux, Android and IOS.

# Appendix

## Appendix A - Electrical Schematics

Figure . Voltage Divider Circuit for TTL to CMOS Conversion for Ultrasonic Ranging Modules (HC-SR04).

## Appendix B - Parts List

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Part # | Model | Component Description | Supplier | Cost/Unit (CAD) | Qnty | SubTotal (CAD) |
| 1 | Raspberry Pi 3 Model B | Main microcontroller | Raspberry Pi Foundation,  https://www.raspberrypi.org/products/raspberry-pi-3-model-b/ | $48.99 | 1 | $48.99 |
| 2 | Three-wheeled smart car | Three-wheeled smart car for Raspberry Pi from Freenove | Freenove,  http://www.freenove.com/index.html# | $99.99 | 1 | $148.98 |
| 3 | SDSQUAR-016G-GN6MA | Sandisk Ultra 16GB Micro SDHC UHS-I Card with Adapter | Sandisk,  https://www.sandisk.com/ | $12.90 | 1 | $161.88 |
| 4 | HC-SR04 | Ultrasonic Ranging Sensor (pack of 6) | Elegoo | $13.21 | 1 | $175.09 |
| 5 | Bluetooth library | Android & Microcontrollers / Bluetooth | Tech Tweaking | $20.00 | 1 | $195.09 |

Figure . Parts List.

## 

## Appendix С - Citations and References

### Works Cited

“CDC Features.” *Centers for Disease Control and Prevention*, Centers for Disease Control and Prevention, 7 Nov. 2017, [www.cdc.gov/features/dsdrowsydriving/index.html](http://www.cdc.gov/features/dsdrowsydriving/index.html).

“Raspberry Pi 3 Model B.” *Raspberry Pi Foundation*, www.raspberrypi.org/products/raspberry-pi-3-model-b/.

“Uniform Crossover.” *Wikipedia*, en.wikipedia.org/wiki/Crossover\_(genetic\_algorithm)#/media/File:UniformCrossover.png.

## Appendix D - Contact Information

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Student name | Student ID | email | Phone number | Address |
| Daniil Ionov | 018297150 | dionov@myseneca.ca | (416)-577-3727 | 62 Bedle Av., Toronto, Ontario, Canada. |
| Paul John Gonzales | 022317150 | pjgonzales@myseneca.ca | (647)-929-0997 | 58 Santamonica Blvd, Scarborough, Ontario, Canada |

Figure . Contact Information.

## Appendix E - CD

### Description of the CD attached

The CD contains all code and documents relative to this product.

## Appendix F – Simulation Images

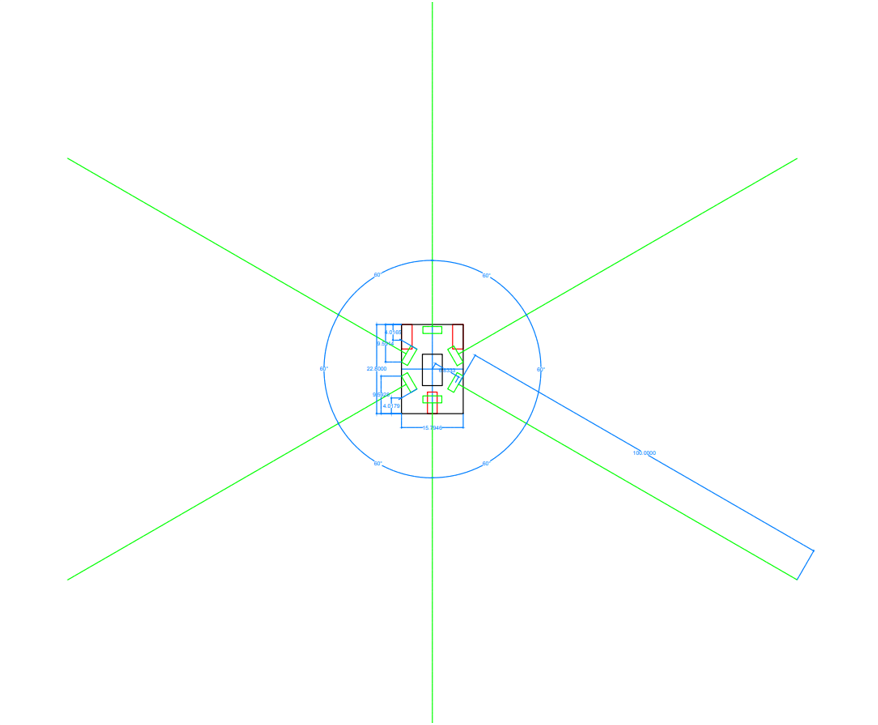


Figure . AutoCAD Drawing of the Physical Car.

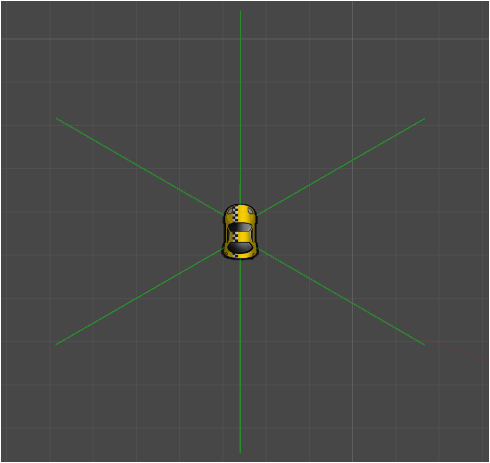


Figure . Designed Unity 2D Car.

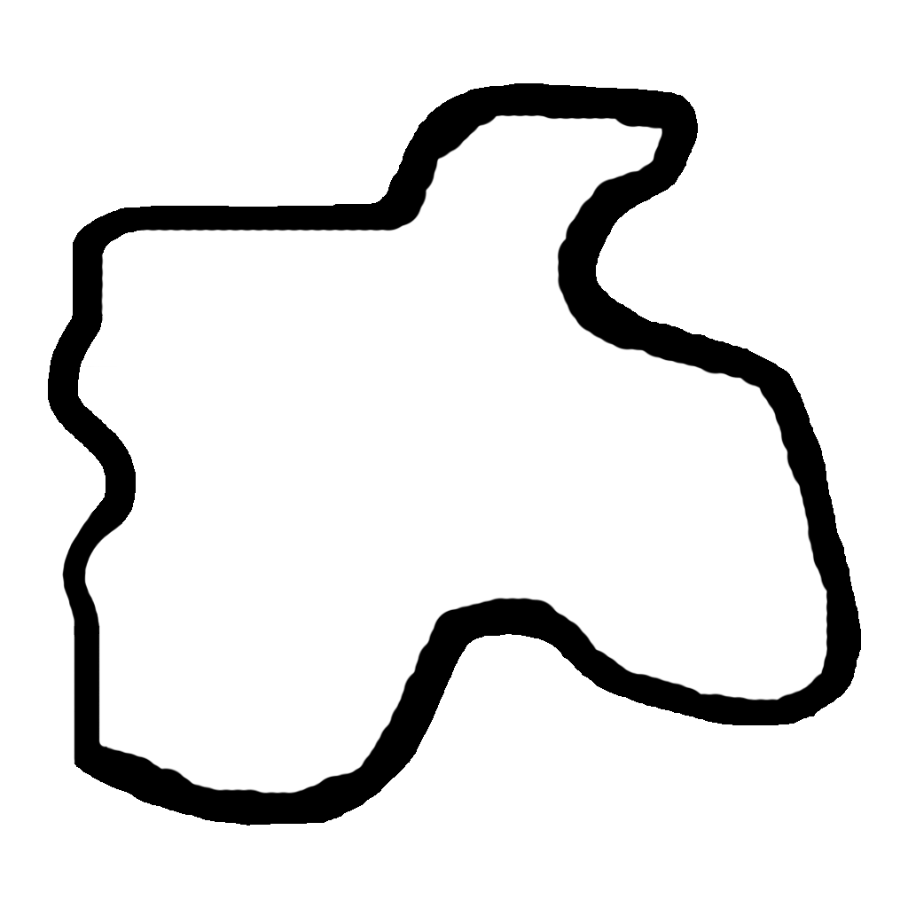


Figure . Designed Map Created in Adobe Photoshop.