exeley INTERNATIONAL JOURNAL ON SMART SENSING AND INTELLIGENT SYSTEMS

Article | DOI: 10.21307/ijssis-2020-021 Issue 1 | Vol. 13 (2020)

# 1/10th scale autonomous vehicle based on convolutional neural network

Avishkar Seth*, Alice James* and Subhas C. Mukhopadhyay

Macquarie University, NSW, 2113, Sydney, Australia.

*E-mails: avishkar.seth@students.mq.edu.au; alice.james@students.mq.edu.au

This paper was edited by Nasrin Afsarimanesh and Anindya Nag.

Received for publication July 26, 2020.

## Abstract

A vehicle capable of using sensors to detect and control its driving actions is called an autonomous vehicle. The development of autonomous vehicles caters to many application areas in the technological advancement of society. This research paper shows a demonstration and implementation of an autonomous vehicle based on a convolutional neural network. The vehicle uses a 1/10th scale RC car as its primary base for the system control with the camera as its primary input. For the computing platform, a Raspberry Pi 4 microprocessor board is used. To enhance the capabilities, the ultrasonic sensor has been implemented in the system as well. The unique aspect of this project is the system design, the CAD modeling, and the track built used to train and test the self-driving capability of the car. The CNN model and the software algorithm also are exclusive to this research project. This research has potential in a variety of application areas in education and also for robotics and autonomous car enthusiasts.

### Keywords

Autonomous vehicle, Convolutional neural network, Raspberry pi 4, Ultrasonic sensor, Camera.

The idea of an autonomous vehicle has always been an exciting and intriguing imagination for scientists and engineers to explore. In recent years, there has been an increasing amount of research and development in the on-road self-driving cars (Bojarski et al., 2016; Zhang and Du, 2019), as well as the autonomous unmanned ground vehicles (AUGV) used in off-road terrains (Man et al., 2018; Patel et al., 2019; Toupet et al., 2019). The research on self-driving cars is mainly focussed on increasing the control capabilities of the car to a level that will require no human intervention. It mainly focusses on path recognition, detecting surrounding vehicles, and transport of passengers from one point to another. The SAE (Society of Automotive Engineers) defines the five autonomy levels of self-driving (Goldfain et al., 2019). The levels 1 to level 5 increase in the control aspect of the vehicle self-driving capability. For example, level 1: cruise control, lane-keeping and assisted breaking (human presence required); level 5: a completely autonomous system in seamless control at all times (no steering wheel or driver seat required). Current systems are capable of achieving level 3 to level 4 autonomy only (Google Waymo) (Balaji et al., 2019).

On the other hand, the AUGV driving concept is different than that of a self-driving car. The primary function of these vehicles is monitoring, environment detection, and navigation in areas or terrains which are difficult for a human to access (Behringer et al., 2004; Patel et al., 2019). These are usually scaled down in size compared to an actual vehicle as they do not require any onboard human presence. The vehicles developed by the DARPA Grand Challenge, the MARS Rover, or the industrial robots used in loading and unloading of goods are some examples of such vehicles (Behringer et al., 2004; Toupet et al., 2019; Sen Gupta et al., 2006). Although the application areas and concepts are different in a self-driving car and an AUGV, the technology for development is quite similar Olgun et al. (2018).

The autonomous vehicle used in this project shows the implementation of hardware and software setup and design of the car. The car is aimed to drive autonomously on a defined track as well as avoid obstacles and apply lane recognition to navigate (Schwartz and Milam, 2008; How et al., 2008). The challenges faced during building this car have been elaborated in this paper.

Further, the research work done in the area of building scaled-down autonomous vehicles is explored here (Paull et al., 2017; Srinivasa et al., 2019; Miao et al., 2012). In the study of Pannu et al. (2015), a method to build a self-driving car is presented using raspberry pi 2. The research mainly focusses on lane detection and obstacle avoidance using a camera and ultrasonic sensor as its input. A similar approach is shown in Bechtel et al. (2018), where the primary focus of research contribution is the CNN model used in the system. This research paper also compared three different embedded computing platforms with their capability in terms of performing real-time computations of the CNN architecture.

Apart from these, the more popular RC car-based autonomous vehicles are the MIT RaceCar (Karaman et al., 2017), the F1/10 (O'Kelly et al., 2019) both using the RC Truck platform with NVIDIA Jetson at its processor. However, these systems have an array of costly sensors escalating the cost above $4,000 AUD. The Amazon DeepRacer (Balaji et al., 2019) is a cloud-based platform that uses end-to-end reinforcement learning with a single monocular camera input and an Intel Atom microprocessor, which is also a commercially available product. The research done in Man et al. (2018) shows a similar approach to building the car using raspberry pi, but mainly uses line follower and ultrasonic sensors to navigate and control the car. In Donkey Car (Roscoe, 2020), a different approach to building the autonomous car was shown using an RC Car chassis. It has a high-level self-driving library written in Python for the implementation of the neural network. To analyze the AI aspect of the research, the paper by NVIDIA DAVE-2 (Bojarski et al., 2016) and ALVINN (Pomerleau, 1989) were the first attempts to build an operational CNN-based model.

Analyzing different research work done, this research paper uses different heterogeneous and open-source material to build the autonomous vehicle (Blaga et al., 2018; Zhang and Mahale, 2018; Tian et al., 2018). It also reports the in-depth details of each component used and its function in the system.

The outline of the paper is organized as follows. The first section introduces the research topic and elaborates the related work done in this research field. The research work is also compared and summarized. The second section elaborates on the methodology of the system's functional requirements and the design approach for hardware and software configuration. The testing performed and the results of the implementation are discussed in the third section of the paper. Lastly, the conclusion and future improvements are described in the fourth section.

## Methodology

The methodology of this research is described in three sub parts. The overview of the complete system and its functional requirements are analyzed before the selection process of the hardware components is complete. The design process and software configuration are given and examined for the self-driving control of the vehicle in the third part.

### Functional requirements

The functional requirements of the system are analyzed and discussed in this section of the paper. Each component and its interfacing with the complete system is a crucial aspect of the vehicle.

1. Embedded computing platform: the primary requirement of the processor is to compute real-time data provided by the sensors. It needs to have sufficient RAM and CPU as performance speed is a crucial aspect of the system. It also needs easy interface ability with the different electronics and sensors used. The software compatibility requirements should also be able to meet the processor board since various machine learning functions will be run on it.
2. Visual input: the vehicle must be able to drive itself based on visual input that is analyzed in real-time. The camera input also needs to have a wide viewing range and pixel quality to decipher the frames (Seelye et al., 2010; Leni, 2017; Wu et al., 2017). Lastly, it needs to be compatible with the microprocessor and its software.
3. The vehicle chassis: the chassis design must be sturdy enough to house the different components on its body while the car drives itself. As the vehicle will be tested in different outdoor environments, it needs stable shock absorbers, large enough wheels, and a four-wheel drive.
4. The servo motor: the servo motor will be used to control the steering angle of the vehicle. It needs to be a three-wire servo that will control its angle by generating PWM signals. It need not have a 360° rotation but it needs good enough torque to steer the vehicle's front-wheel drive with ease.

5. The DC motor: the DC motor is the throttle of the vehicle to move it forward or backward. The motor type can be a brushed and brushless DC motor. However, since the application does not require very high speed and needs more control, a brushed motor will be a suitable choice. Also, it has a lower cost than its brushed counterpart.
6. ESC (electronic speed controller): this unit must be compatible with the DC motor and the battery. The ESC controls the throttle value of the DC motor with PWM signals. It usually also has a BEC (battery eliminating circuit) built-in which helps in giving suitable current and voltage values to the DC motor. It is best to select the battery, DC motor, and ESC together as they need to be interfaced together.
7. Power requirements: the system needs two types of battery power. The first one is to power the DC motor through the ESC circuit that will require either a LiPo or a NiMH battery. The system will require at least 1,000 mAH capacity and sufficient stable voltage range to power the DC motor. The second battery primarily needs to power the microprocessor board and the servo motor. A standard Li-ion based power bank of about 6,000 mAh capacity should be good enough with USB output.
8. Sensors: an autonomous vehicle application can have a range of sensors like LiDAR, IR (infrared), ultrasonic, IMU (inertial measurement unit), stereo camera, etc. (Gui et al., 2017; Sen Gupta et al., 2009) While the potential is endless, a basic autonomous vehicle requires object avoidance as its fundamental requirement. Hence, an ultrasonic sensor with a three-pin configuration is suitable for the system.
9. Servo/DC motor controller: the servo and DC motor need PWM signals to control the steering angle and the throttle values. A compatible PWM servo driver board can be used here which will have three-wire input and multiple channels to control additional motors if required.
10. Miscellaneous: additional components like jumper wires, SD card, joystick, screws, battery chargers, and USB cable will also be required which are discussed in brief later.

## Design approach

### *Vehicle hardware and sensors*

1. Raspberry Pi 4: for the embedded computing platform, the latest release of Raspberry Pi 4 single-board computer (SBC) has been chosen. It has features like 1.5 GHz quad-core ARM Cortex-A72 CPU for faster processing and Wi-Fi, Bluetooth for wireless connectivity. Due to its easy hardware and software configuration features, it has been chosen for this application. The NVIDIA, Intel Atom, or the Google Coral SBCs have also been used in such applications, however, each one has a separate hardware/software setup and cost higher than the Rpi-4.
2. Wide angle monocular camera: for the visual input, different types of cameras were examined. The SainSmart Wide Angle Fish-Eye Camera Lens has been chosen for the system. The features include 5MP, 1080p @30 fps, and Omnivision 5647 sensor in a fixed-focus module with a ribbon connector for Rpi-4.
3. 1/10th scale RC car chassis: for the chassis, the 1/10th scale RC Truck type chassis has been selected. The chassis is assembled using individual components for an electric type 4WD Hobby RC Car (HSP Brand). The details are discussed in the latter part of the design section.
4. E6001 6 kg analog servo: a standard E6001 6 kg Analog 3-wire servo (HSP Brand) has been selected for the system. The servo has a +/−60° rotation angle with 4.8 V power input. The power is supplied from the Li-ion power bank through the PWM driver.
5. RC 550 Brushed DC motor: the HSP brand 2-wire brushed DC motor has been selected for the system. It has a 20,000-rpm capacity with 150 A forward/100 A outward and about 28 km/hr top speed. It is connected to the ESC for control and power.
6. 6 V/3 A BEC ESC: a separate 6 V/3 A BEC ESC has been selected for the system. The ESC is compatible with the motor and battery requirements. The ESC for a brushed motor controls its speed by changing the voltage on and off very rapidly several times per second also known as 'chopping' the voltage. The length of the PWM signal determines the speed or direction (forward/reverse) of the DC motor. The ESC has a connector for battery, DC motor, and also has a switch and a 3-wire cable.
7. Power requirements: a standard 6 Cell NiMH battery of 2,000 mAh capacity has been implemented in the system. Each cell is 1.2 V and there are six such cells making it a 7.2 V battery. A Tamiya type female adapter is used to connect it to the ESC. For the second battery,

a standard 10,000 mAh Li-ion power portable power bank is used with stable 5 V USB output.

8. Ultrasonic HC-SR04 sensor: to avoid obstacles, a distance measuring ultrasonic sensor HC-SR04 module has been used in this system. It has about 2 cm to 30 cm detection range and has a 15° angle on either side of the sensor.
9. PCA 9685 servo driver: for controlling the PWM signals for the throttle and steering values the PCA9685 16-channel 12-Bit PWM Servo driver has been used. The PWM frequency and duty cycle can be tuned to regulate the servo accurately by coding the circuit controller.
10. Miscellaneous: the additional components required to meet the hardware system requirements are as follows. A SanDisk Extreme 128 GB microSD Card is connected to Rpi-4 for memory storage. Further, a PS4 Joystick is implemented to control the vehicle commands in user/train mode.

Figure 1 shows all the individual major components used to build the vehicle.

## Circuit diagram

This section gives the complete circuit diagram implementation of the vehicle components. Firstly, the connections are described, and the interfacing pin configurations are explained thereafter.

Figure 2 shows the complete system circuit diagram developed for the vehicle. The electronics software tool Fritzing has been used to generate the circuit architecture.

1. Raspberry Pi and Camera: the camera is inserted in the Rpi camera slot through a 15-pin ribbon connector which attaches it firmly to the Rpi.
2. Raspberry Pi and PCA9685: the PCA9685 VCC pin have been connected to Rpi pin 1, SDA to pin 3, SCL to pin 5, and GND to pin 6.
3. PCA9685 and servo motor: servo motors 3-pins, VCC GND, and Data to channel 1 of the PCA9685, which has a +5 V supply.
4. PCA9685 and ESC: ESC motors 3-pins, VCC GND, and Data to channel 0 of the PCA9685, which has a +5 V supply.
5. DC Motor and ESC: GND and VCC pin connection is done using the 2-pin connector.

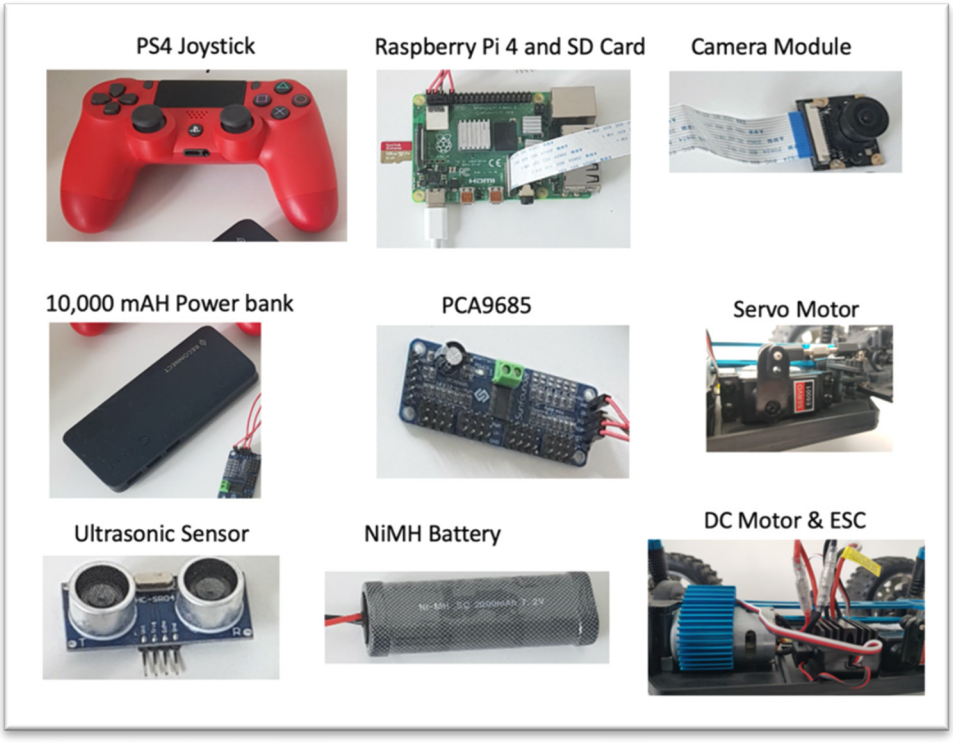



Figure 1: Picture of the individual vehicle hardware components used in the system.

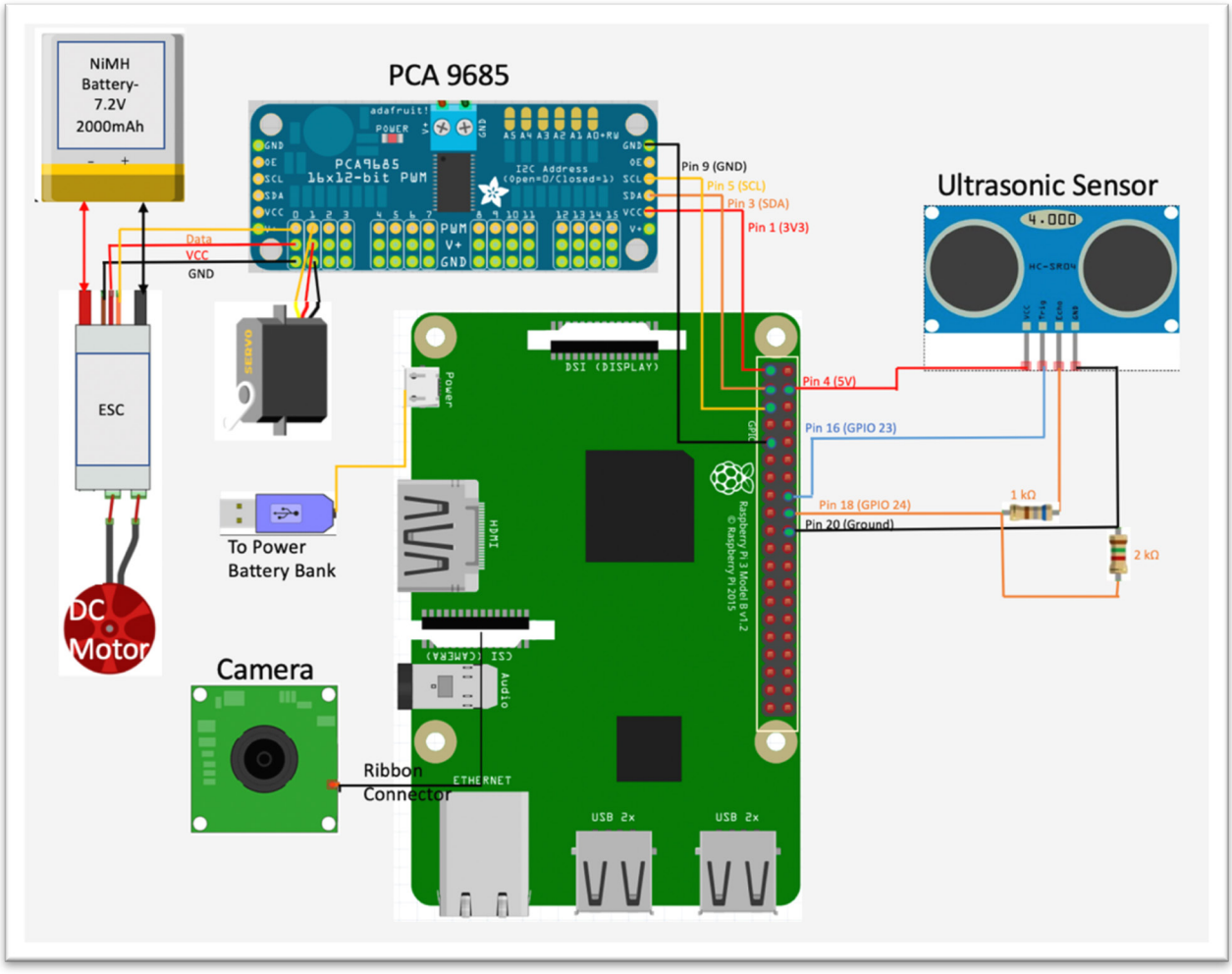



Figure 2: Complete system circuit diagram.

6. NiMH and ESC: two pin GND and VCC pin connection via Tamiya connector.
7. Ultrasonic sensor and Raspberry Pi: the TRIG pin of the sensor connects to pin 16 (GPIO23) of the Rpi. A 1 kΩ and 2 kΩ resistor is used in series combination to step down the voltage of the ECHO pin connected to the pin 18 of Rpi. Lastly, the +5 V and GND connections are provided to the sensor via pin 4 and pin 20, respectively.

## CAD design

The following section elaborates on the CAD designs used to build the vehicle base plate for housing the components. The first design is done for a base plate to house the raspberry pi, power bank, ultrasonic sensor, and PCA9695 servo driver. The base plate has been custom designed for this application. It is possible to use different materials and techniques to build the base plate like 3D printing or drilling. While these techniques will work, the fastest and most efficient one was to use a laser cutting technique Nag et al. (2018). In this system, an acrylic sheet of 4 mm thickness has been laser cut. The 2D CAD model has been made using Autodesk Fusion 360 design tool. Figure 3 shows the CAD model designed using this software. The design has been implemented such that it can be placed on the four corner sections of the chassis and also has sections laser cut to allow the other components to be fixed in place. Some of the components which need to be removed frequently like the power bank are fixed in a temporary manner as shown later.

This design has further been implemented on the transparent acrylic sheet material with

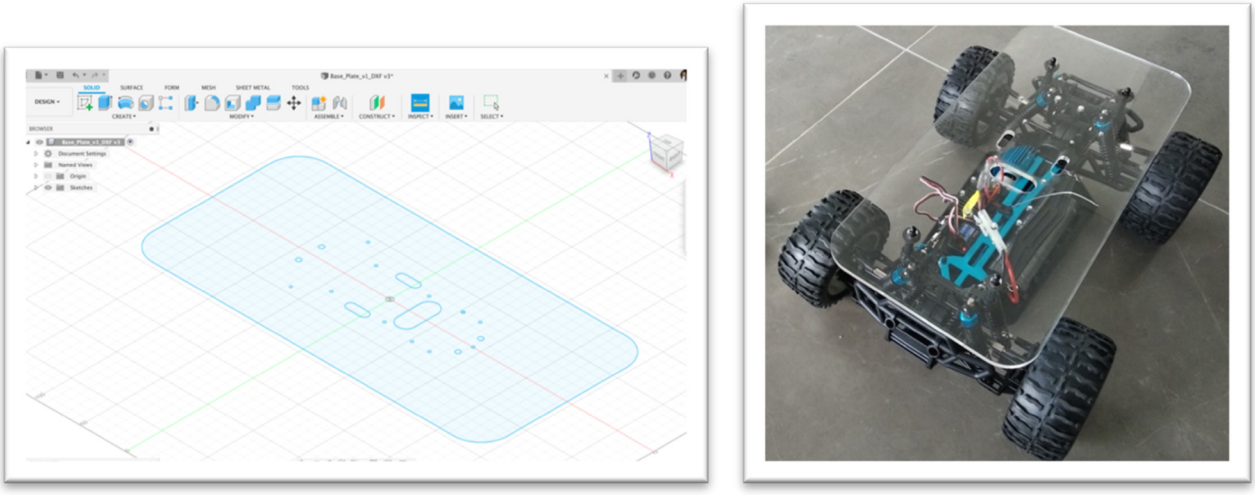


Figure 3: CAD design for the laser cut base plate implemented on Autodesk Fusion 360 (left) and the base plate placed on the vehicle chassis (right).

dimensions length = 360 mm, breadth = 190 mm, and thickness = 4 mm. The laser-cut process took only 15 min, while a similar 3D print this would take about 4 hr to complete.

The next CAD model has been designed to place the camera at a certain height and angle on the base plate body. The camera is to be positioned so that it can have a complete view of the lane markings. The tower-type case has been also made suitable to fix itself permanently with screws to the base plate and the camera as well. To make the case, a 3D CAD design has been made using Autodesk Fusion 360. Figure 4 shows the CAD design model along with its 3D printed model. An 18% infill with PLA material is used to make the physical case. The final dimensions are length = 150 mm, breadth = 85 mm, and height 85 mm.

## Software installation and configuration

In this section of the paper, the software installation and configuration are discussed. The given circuit diagram is used, and the raspberry pi computer software is configured for the self-driving system. The

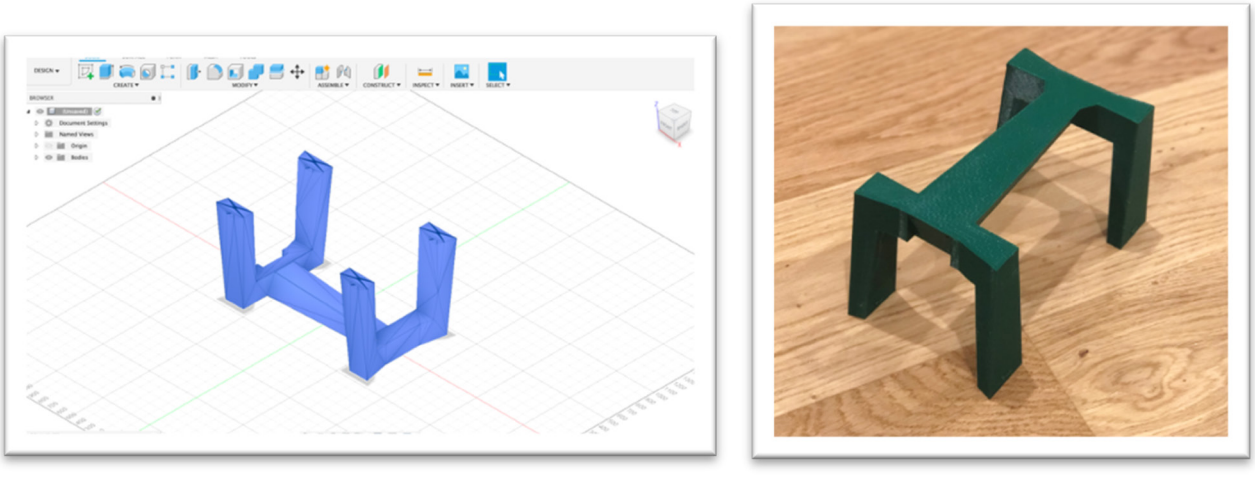


Figure 4: CAD design for the tower type case implemented on Autodesk Fusion 360 (left) and the physical PLA material 3D printed case (right).

microprocessor Rpi and a host Windows computer are used to perform these operations. Figure 5 shows an overview of the software message queue loop of the system. This overview gives the vehicle state in different stages.

The first stage is where the vehicle is in 'user mode' or 'training mode'. As can be seen, the first function is performed by the camera visual input where it captures images and saves them to the SD card. The control here is completely performed by the user through a joystick/web interface designed to communicate wirelessly with the onboard Rpi. In this mode, the vehicle is 'trained' by the user to drive on the track and take images that are saved on the memory card. The quality of images in the dataset is dependent on how well the driver can drive the vehicle on the path without any errors. The dataset contains images of the track with a timestamp and adjacent steering and throttle angles given for the motors by the controller while driving.

In the next stage, these data are trained in a CNN (convolutional neural network) model on the host PC. The generated machine learning model is then transferred to the raspberry pi. This stage is called the 'autopilot mode or 'test mode' where the vehicle is tested to run without any human intervention or control on the same track. This time, the camera takes images, and the CNN model analyses these images in real-time and also outputs steering and throttle angle values. These values are sent from the Rpi to the motors in the form of PWM signals of varying duty cycle (on/off) period through the PCA9685 driver. The test data are also stored in the memory card and later can be analyzed with the training data. Here, the vehicle makes the best attempt to 'replicate' the user driving, hence the autopilot efficiency relies on the training datasets. This technique is also called supervised learning or behavioral cloning.

## *Host PC installation*

In this section, the software installation for the training computer is provided as shown in Figure 6. Before the Rpi is configured, the software has to run through the PC. In this project, a Windows Laptop Nvidia GeForce GTX 1050 Ti GPU is used for this purpose. It is better to have a system with a powerful GPU to reduce training times. The software language used primarily for the configuration is Python. It is a widely used programming language for machine learning and AI applications.

Firstly, the open source Python distribution platform 'Anaconda' is installed on the computer. The latest python 3.7 version is installed here.

To install the next software packages, the anaconda prompt terminal is used. Next, install the

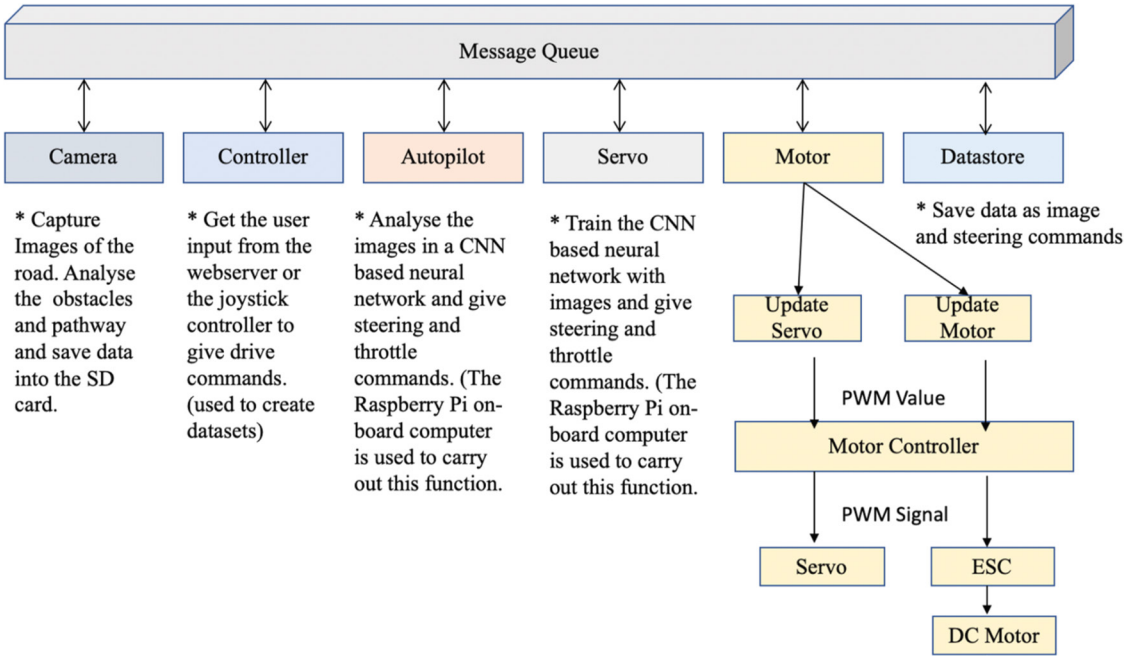



Figure 5: The overview of the software message queue of the system (vehicle state).

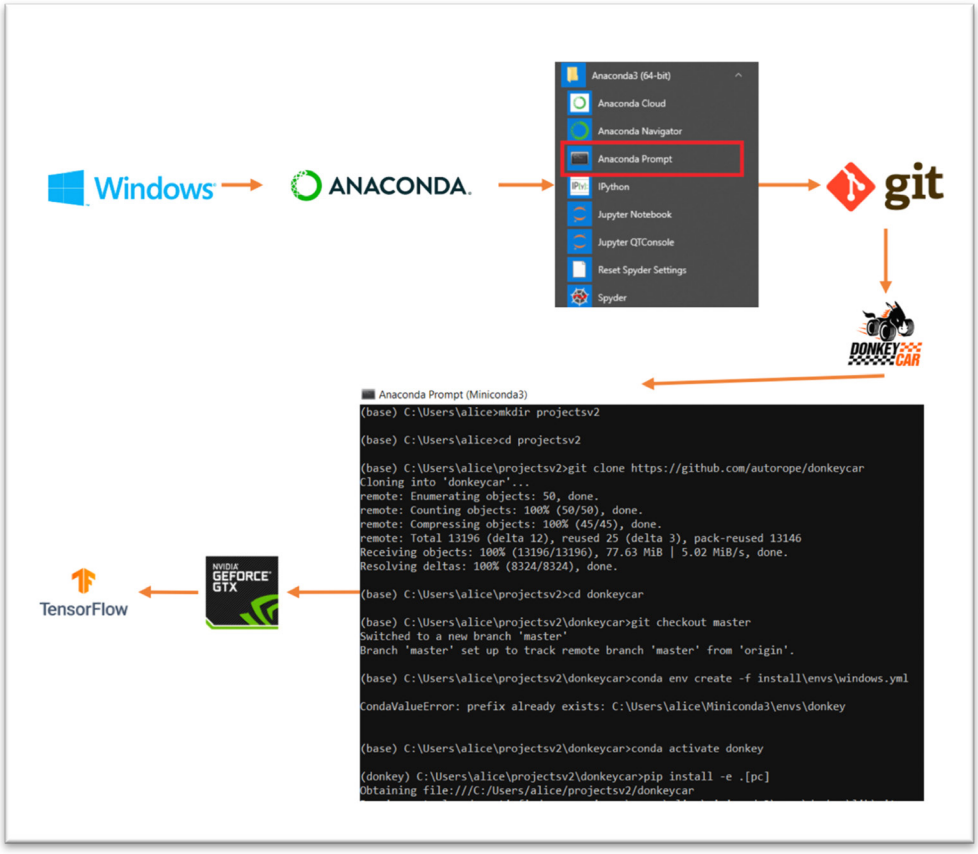



Figure 6: Windows host PC installation steps.

version control system 'Git' to manage the project and files on the computer. Next, the open-source code from various online sources is downloaded and managed in a separate folder on the laptop. Mainly, the GitHub repository from Tawn Kramer's 'Donkey Car' (Roscoe, 2020) is used to build and configure the installation and software process. Next, the TensorFlow which is an open-source software library is installed with version 1.13.1. It also provides support to run 'Keras', the open-source deep-learning library to train the machine learning models. The majority of the software programming done in this project uses open-source material to configure the self-driving capability of the vehicle. However, the programs used have been configured and updated for this specific application and algorithm discussed later.

## *Raspberry Pi setup*

The next step is the configuration of the Rpi on board the vehicle. Figure 7 shows the eight steps in the process of configuring the Rpi with the help of the laptop commands. These steps are to be performed only once to permanently configure the Rpi board.

Initially, the micro SD card is connected to the laptop and the latest Raspbian Buster OS is flashed.

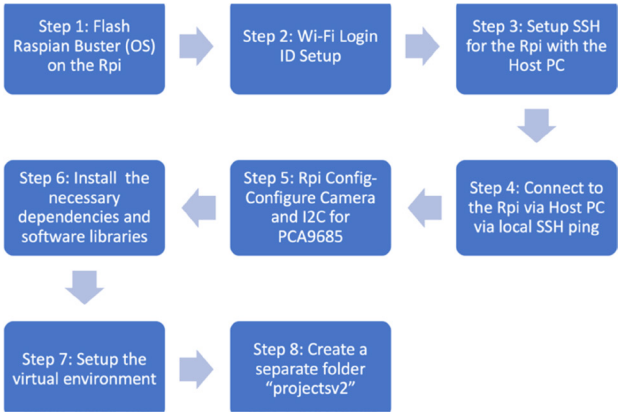



Figure 7: Software configuration steps for the Raspberry Pi 4 setup.

Next, the Rpi needs to be enabled to perform operations from commands given by the laptop. For that, the Wi-Fi login and ID/password is saved in the memory card as a '.conf' file and used by the Rpi to connect to Wi-Fi on the first boot. Next, the Secure Shell (SSH) is set up by adding a file named 'ssh' in the root directory of the memory card. Now, this memory card is inserted in the Rpi and is ready to connect to the laptop via Wi-Fi. The connection uses the Host PC connection via local SSH ping. Also, the update and upgrade functions are performed from the terminal. Next, the camera and I2C for the Servo driver PCA9685 are configured with the 'sudo raspi-config' command. In the configuration, the default password, the hostname is changed and the 'I2C' and 'Camera' are enabled in the 'Interfacing Options'. Also, in 'Advanced Options' the 'Expand Filesystem' is used to use the complete SD card memory storage space. Next, the necessary dependencies and software libraries for python, i.e. version, dev, pip, virtual environment, camera, numpy, pandas, gpio, i2c-tools, avahi-utils joystick, etc. are installed using the 'sudo' command. Next, the virtual environment is activated. Also, the Open CV for a real-time computer vision application is installed. Lastly, the files are managed, and a separate folder called 'projectsv2' is created.

## Interfacing Rpi and sub-components

A new file myconfig.py has been created for the vehicle's main loop. This code contains all the necessary program and each sub-component class is used in this python script to execute it in real-time onboard the Rpi.

### *Camera*

The camera connection is shown earlier in the circuit diagram. After enabling the camera interfacing option in the raspi-config, the camera can start functioning. The camera setup is done in a new file created named camera.py where a new class Base Camera is defined. This file is the main program in augvconfig.py that will be executed by the system while in training mode. To exit and save the file: press 'Ctrl + Fn + X' and then 'Y' and then press 'Enter' while in the 'nano ~/mycar/myconfig.py'. The first step is to define the camera parameters and dimensions. The selected dimensions for the camera are width = 160, height = 120, and depth = 3. The depth is 3 for RGB color (Red Blue Green) if 1 is selected it is for mono. The height and width are in pixels. Other executable conditions like horizontal/vertical flip, the frame rate is also defined here. An fps of 20 is selected for this project. Figure 8 shows a screenshot of the camera code configuration and a sample image of 160 × 120 pixels taken from the dataset.

### *Ultrasonic sensor*

The ultrasonic sensor is connected as shown in Figure 2 circuit diagram. It has a viewing range of 30° and a 300 mm distance object detection range. The sensor generates short high-frequency sound waves at regular intervals (8 pulses) through the TRIG signal at 40 kHz. The ECHO pin switches to logic 1 till the sound does not echo back from the object. The time period for it to return back is the width of the ECHO pulse remaining logic high. Once the time is known, the distance can be calculated using

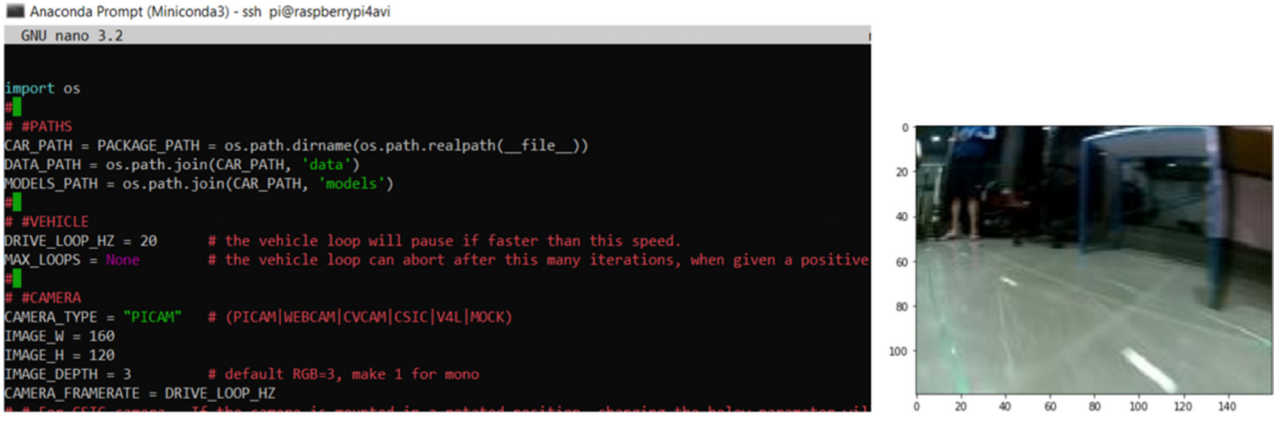



Figure 8: Screenshot of the camera code configuration and sample 160 × 120 pixels image from the vehicle on board camera.

distance = speed × time. A new class hc_04sensor is written in the ultrasonic.py part which is added to the main code of augv.py. The open-source library 'Bluetin-Echo 0.2.0' is used to configure the ultrasonic sensor logic in the vehicle's main loop. This sensor placed on the front section of the car continuously measures the distance and keeps track of any obstacles. If an obstacle is detected it stops the main loop completely, otherwise it does not interfere with the vehicle state which uses the camera visual input to give drive commands.

### *PCA9685 servo driver*

The servo driver used is connected to the I2C port of the Rpi and is enabled as mentioned earlier. The commands 'sudo apt-get install -y i2c-tools' and 'sudo i2cdetect -y 0' are used here to complete the setup. This gives the default address of the 16 channel PCA9685 board as 0×40 (binary 1000000) shown in the screen capture in Figure 9.

This address 0×40 is used in the vehicle augvconfig.py file. Further, the steering channel is selected at 1 and throttle as 0, referring to the circuit diagram.

### *Web interface and joystick control*

For this project, both the web interface and joystick setup have been performed. The PS4 joystick uses the Bluetooth connection with the Rpi to communicate and send commands. A separate python script joystick.py is created to define the class and commands for the ps4 buttons. The ds4drv is a Sony DualShock 4 user space driver for Linux which is installed using 'sudo pip install ds4drv' on the Rpi. After completing the setup and assigning buttons and axis controls to the program, the joystick can be used to control the vehicle.

The web interface is written using HTML and python languages to control the vehicle. Figure 10 shows the screen capture of the web interface connected through the IP address localhost.

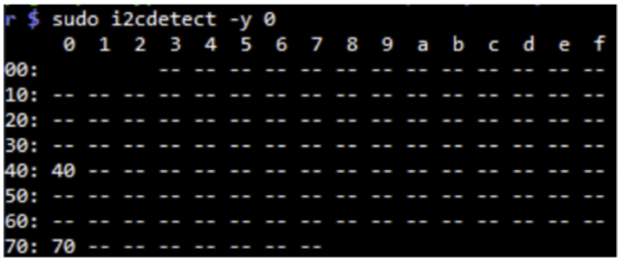



Figure 9: Screen capture of the I2C detected on the Rpi terminal.

## System testing and results

This section of the paper discusses the results of the final system assembly and testing of vehicle autonomy. The section is divided into five individual subsections as follows.

### Final vehicle system

Here, the final vehicle system is assembled with its chassis, motors and ESC, complete circuit diagram, the 3D printed, and laser cut models and positioning of the camera and ultrasonic sensor. The following provides a step by step process that has been implemented for the vehicle.

Firstly, the DC motor, Servo Motor, and ESC is attached to the vehicle 1/10th scale chassis. The chassis comes prebuilt with holes and sections for the motors and ESC to be attached. The DC motor is attached to the gearbox on the chassis. Also, the wheels had to be attached separately to the chassis to complete the base. These components were fixed permanently with a screw. However, the NiMH battery is placed on the chassis and fixed with R-clips, which makes it easy to remove and attach for charging. Next, the laser cut acrylic sheet is assembled with the Rpi and PCA9685 which has been attached to it via cable ties. The power bank, breadboard, and ultrasonic sensor have been hot glued to the laser cut base to avoid them from falling at high speeds. Next, the wiring has been done according to the circuit diagram shown earlier. The heatsink is also added to the Rpi here to avoid overheating. The tower case has been printed with indents to fasten screws to the base through the four corners on which it stands on. Lastly, the camera connects to the front section of the tower case with screws. The breadboard has been used here to allow further prototyping of the system with additional sensors if required in future upgrades. Finally, the acrylic plate is placed on the chassis and fixed using R-Clip as shown. Figure 11 shows the complete assembly of the vehicle hardware.

### Circuit testing

Before the vehicle is tested to run in user/autopilot mode, the circuit has to be tested with the software configurations and necessary calibrations are made.

Firstly, the DC motor connected to the ESC is calibrated with the Rpi. The primary aim of this testing is to find a suitable speed and subsequent PWM

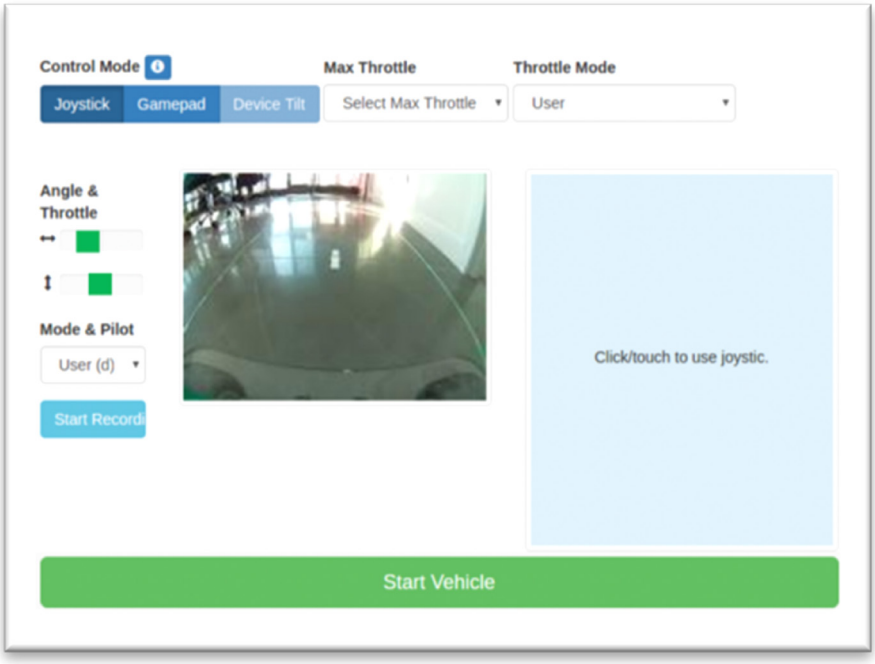



Figure 10: The Localhost web interface (screen capture) to control the vehicle.

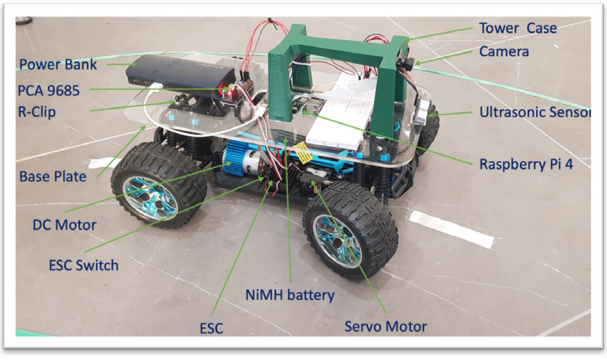



Figure 11: The complete assembly of the vehicle hardware system.

value to control the throttle rate. Also, the forward, reverse, and stop functions have been calibrated with the servo driver and Rpi. As mentioned earlier, the channel 0 of the PWM board is the throttle channel, and channel 1 is for steering. Figure 12 shows the code snippet program of the motor calibration used in the myconfig.py file.

The calibration is crucial to avoid the car having too much speed and hitting a wall or going off track. The value of the PWM control signals pulse range from 1,000 to 2,000 microseconds. The width of the pulse determines the speed and direction of the motor rotation. Hence, the value 1,500 points to the center for the motor, i.e. stop signal. To find these

```
# #STEERING
STEERING_CHANNEL = 1            #channel on the 9685 pwm board 0-15
STEERING_LEFT_PWM = 415         #pwm value for full left steering
STEERING_RIGHT_PWM = 295        #pwm value for full right steering
#
# #THROTTLE
THROTTLE_CHANNEL = 0            #channel on the 9685 pwm board 0-15
THROTTLE_FORWARD_PWM = 390      #pwm value for max forward throttle
THROTTLE_STOPPED_PWM = 351      #pwm value for no movement
THROTTLE_REVERSE_PWM = 300      #pwm value for max reverse throttle
```

Figure 12: The code snippet program of the motor calibration.

values given in Figure 12, different values are given via the Rpi starting from 370 and going +/–10. The reverse calibration is tricky and takes a while to figure out since it requires the 'reverse' value, then the 'stop' value, and then again 'reverse'.

Similarly, the steering values are calibrated where the control signal sends one PWM pulse at 60 times/sec rate. The width of this pulse decides the amount of right/left the motor turns. The steering servo results in its extreme left at 415 value and extreme right at 295 PWM value. The servo can go +/–60° in either direction corresponding to +1/–1 as shown in Figure 13. The floating point values between +1/–1 are used to save the angular control values in the datasets shown in Figure 14.

## Track building

The following section describes the custom track built to train and test the vehicle. Building a custom track is a challenging task and getting it to work requires multiple iterations. Firstly, a large enough space is required to build the track as the car has a turning radius defined by the servo motor angle. The selected track has been built indoors in an apartment type space. Bright color ribbons have been used to make the track lanes and center markings and they are stuck to the floor using tape. Green color has been used for the inner and outer lane and white color ribbons for the broken center line markings. Figure 15 shows the custom-built indoor track design for the car and its implementation.

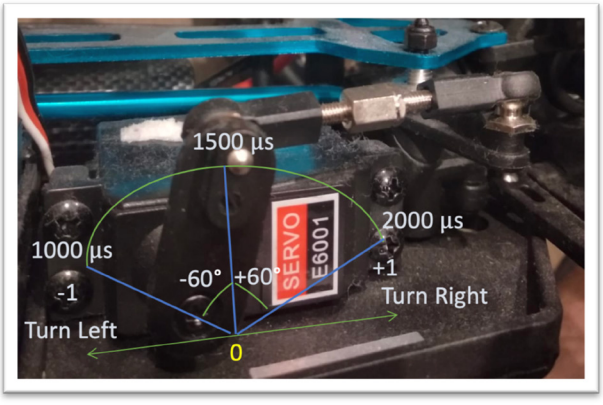



Figure 13: The steering servo and PWM working principle.

| PWM Value | PWM Pulse Width (microseconds) | Steering Angle (in Radians) | Steering Angle (in Degrees) |
|---|---|---|---|
| 415 | 2000 | -1.000 | -57.290 |
| 405 | 1900 | -0.834 | -47.780 |
| 395 | 1800 | -0.668 | -38.270 |
| 385 | 1725 | -0.502 | -28.760 |
| 375 | 1650 | -0.336 | -19.249 |
| 365 | 1575 | -0.170 | -9.739 |
| 355 | 1500 | 0.000 | 0.000 |
| 345 | 1425 | 0.170 | 9.739 |
| 335 | 1350 | 0.336 | 19.249 |
| 325 | 1275 | 0.502 | 28.760 |
| 315 | 1200 | 0.668 | 38.270 |
| 305 | 1100 | 0.834 | 47.780 |
| 295 | 1000 | 1.000 | 57.290 |

Figure 14: Table generated for steering Angle and PWM equivalent values.

The track dimensions are 15.2 meters (4.6+4.6+3+3) for the outside marking and 11.6 meters (3.8+3.8+2+2) for the inner marking. Thus, this provides sufficient space for the 1/10th scale car to move with a smooth turning space. Further, designing the track indoors is useful to allow frequent testing and adjustments.

## Train/test mode

Car training and testing is split into three sections.

### *Train/user mode*

The car is driven on this track by the user using a joystick/web interface for about 15 to 20 laps. These training data are saved on the memory card containing several such images and '. json' text files as shown in Figure 16. This dataset is then transferred to the laptop via SSH protocol using 'FileZilla' software.

### *Host PC training*

Training the dataset on the Keras model requires larger training times (couple of hours) for larger datasets. This dataset is trained on the host laptop using a custom CNN model with the following parameters:

| No. | Convolution filters | Strides | FC layers | Parameters | Loss |
|---|---|---|---|---|---|
| 1 | 12×3×3, 18×3×3, 24×3×3, 36×3×3 | 3,2,2,1 | 900, 246, 32 | 1,100k | **0.098543** |

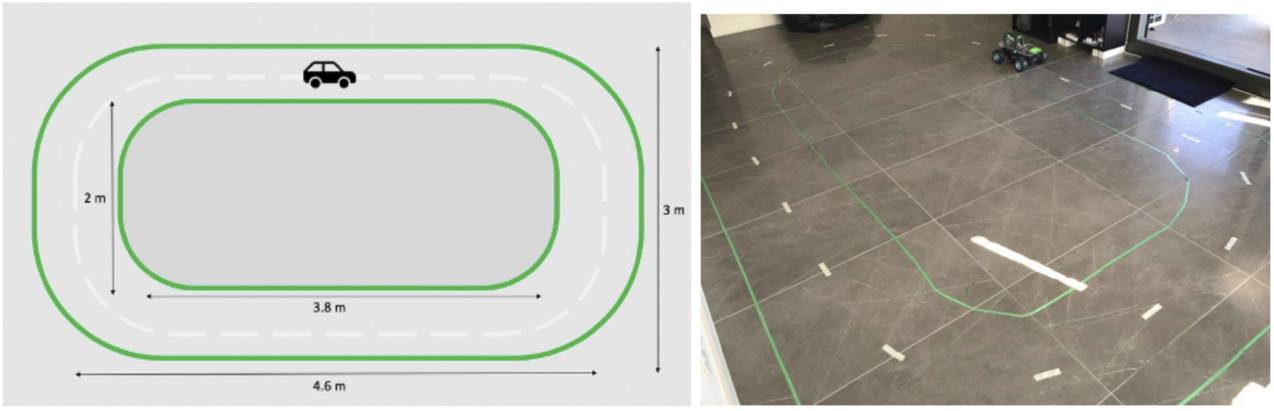



Figure 15: The custom-built indoor track design and its implementation.


Figure 16: A sample training dataset from the track training.

A batch size of 128 is used with Keras linear type model. The model has a validation loss of 0.0.98543 which is suitable for the autopilot. The model is saved as a '.h5' file which then has been transferred back to the Rpi for running it in real-time. Figure 17 shows the model loss of the training data. It can be seen at about 55 epochs; the training stops as the loss starts increasing.

### *Test/autopilot mode*

In this mode, the trained model is used by the Rpi to run on the track in a continuous loop as shown in Figure 15. The following flowchart algorithm shown in Figure 18 is used by the autopilot mode. For this application, the throttle speed is kept at a constant value and only steering values are predicted by the autopilot.

The vehicle loop begins with initializing the camera and ultrasonic sensors and running the python scripts for the car defined in manage.py. Once the car is switched on, it first checks if there is any obstacle within 10 cm in front of the car. If no, the car starts moving and the camera inputs the images at 20fps. These images are then analyzed by the CNN model provided and output a predicted value of steering angle between –1 and +1. The value is then set by the actuator and the servo motor turns or remains constant. In the actuation, the vehicle state

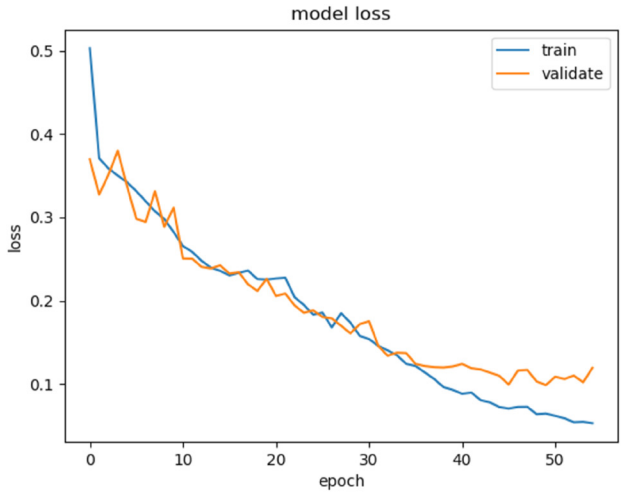



Figure 17: Training data model loss graph.

also checks for any user inputs given to increase or decrease the throttle. If the car detects any obstacle in 10 cm range, it immediately sets the actuation throttle value to 0 and waits until the obstacle is removed.

## Steering angle analysis

The training dataset contained about 30,960 records of images and '. json' file records. This dataset is analyzed here as shown in the first graph using Jupyter Notebook and Python programming language. The graph plot (A) in Figure 19 shows that the most common frequency of angle was about '0', i.e. go straight. The car is driven on the track in a clockwise direction due to which the frequency of the right turn is quite larger than the left turns. In fact, the only time left-turning angles are used is when the car needs to align itself with the straight lane marking of the track.

The second graph plot (B) shows the autopilot dataset analyzed after the car is run for the same number of laps as the training mode. The graph plots are almost identical proving the autopilot is successful in behaving as the driver with minimal errors.

## Conclusion

In this paper, a novel guided method to build a 1/10th scale autonomous car using a custom CNN model has been presented. A thorough understanding of the related work that resulted in finding a development gap in the hardware design approach in building a self-driving car that has been addressed and justified. Each functional requirement has been identified and the suitable hardware components and sensors have been used. The vehicle is capable of smart sensing and uses a camera and ultrasonic sensors effectively to perform autonomy as well as to detect obstacles on the track.

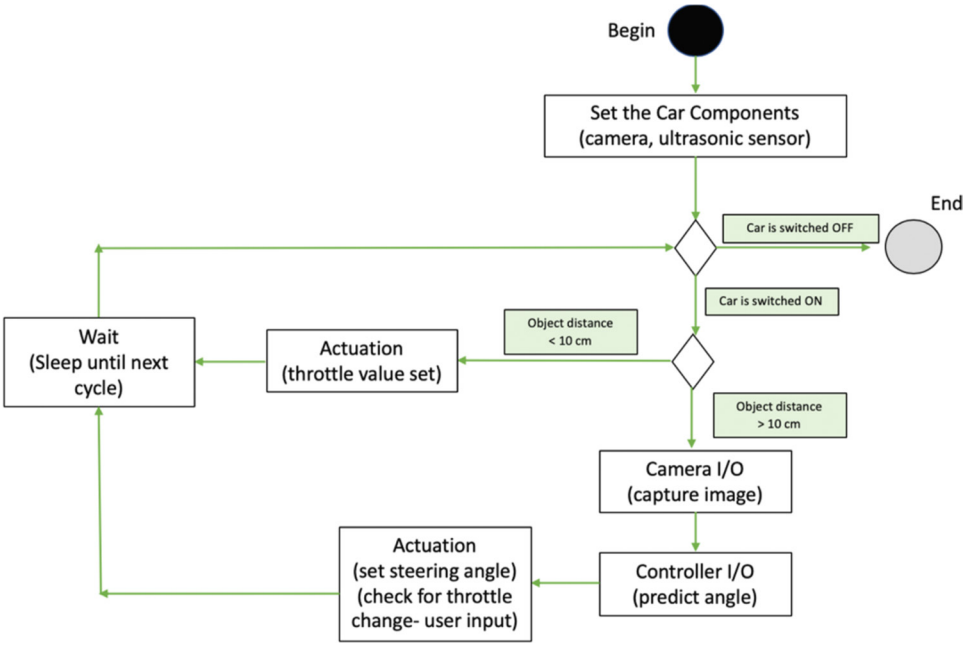
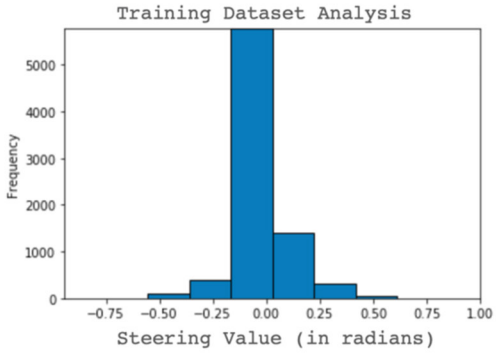



Figure 18: The car algorithm flowchart.

A Training Steering Angle Histogram Graph Plot

```
In [21]:
ugv_track1['steering'].plot.hist(edgecolor='k').autoscale(enable=True, axis='bot
h', tight=True)
```




B Autopilot Steering Angle Histogram Graph Plot

```
In [22]:
ugv_track1['test_steering'].plot.hist(edgecolor='k').autoscale(enable=True, axis
='both', tight=True)
```

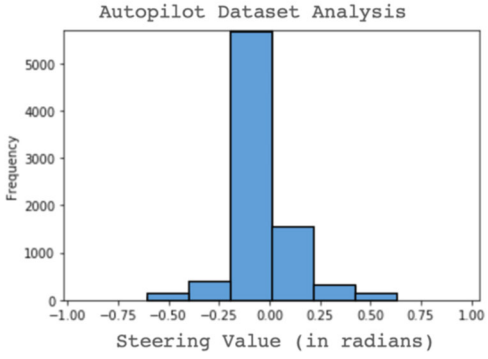



Figure 19: Training and autopilot steering angle histogram graph plot analysis.

Further, the training and autopilot performance given the custom CNN model is proven to be effective according to the low validation loss and identical steering angle analysis for the trained system. The vehicle has various application areas in the military requiring ground support missions, in the survey and inspection of industrial facilities. Also, its fundamental units can be useful in developing assistive autonomous robots for visually impaired individuals.

To improve this system in future, the following points are looked upon:

1. The system hardware could be improved further by experimenting with different SBCs such as Google Coral Board, NVIDIA Jetson, and Intel ATOM with the hardware electronics.
2. With further developments, this intelligent system can be enhanced by adding functionalities such as IMU, GPS, and LiDAR sensors. The GPS sensor enables the information for the absolute position of the vehicle, while the IMU sensor allows vehicle orientation in 3D space. The LiDAR sensor helps in simultaneous localization and mapping (SLAM) technology which is widely used in autonomous systems.
3. The software improvements would include training and testing the vehicle in a virtual environment using a simulator, that would enable further testing and analysis of vehicle performance

without the need to physically test it in different environments.
4. Further, the training models could be tested with different augmentation techniques allowing the vehicle to maneuver and navigate in unchartered terrains.
5. Lastly, the artificial neural network has the potential to improve using different Keras models with TensorFlow backend. It could allow the autonomous throttle control as well as enhancing the moving speed of the vehicle.

## Literature Cited


Balaji, B., Mallya, S., Genc, S., Gupta, S., Dirac, L., Khare, V., Roy, G., Sun, T., Tao, Y., Townsend, B., et al. 2019. DeepRacer: educational autonomous racing platform for experimentation with Sim2Real reinforcement learning. arXiv preprint arXiv:1911.01562.

Bechtel, M. G., Mcellhiney, E., Kim, M. and Yun, H. 2018. DeepPicar: a low-cost deep neural network-based autonomous car. 2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), Hakodate, pp. 11–21, doi: 10.1109/RTCSA.2018.00011.

Behringer, R. et al. 2004. The DARPA grand challenge – development of an autonomous vehicle. IEEE Intelligent Vehicles Symposium, 2004, Parma, pp. 226-231, doi: 10.1109/IVS.2004.1336386.

Blaga, B., Deac, M., Al-doori, R. W. Y., Negru, M. and Dănescu, R. 2018. Miniature autonomous vehicle development on Raspberry Pi. 2018 IEEE 14th International Conference on Intelligent Computer Communication and Processing (ICCP), Cluj-Napoca, pp. 229–236, doi: 10.1109/ICCP.2018.8516589.

Bojarski, M., Testa, D. D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J. and Zieba, K. 2016. End to end learning for self-driving cars. arXiv:1604.07316, April, available at: http://arxiv.org/abs/1604.07316.

Goldfain, B., Drews, P., You, C., Barulic, M., Velev, O., Tsiotras, P. and Rehg, J. M. 2019. Autorally: an open platform for aggressive autonomous driving. *IEEE Control Systems Magazine* 39(1): 26–55.

Gui, P., Tang, L. and Mukhopadhyay, S. 2017. Tree pruning robot tilting control using fuzzy logic. Proceedings of the 2017 Eleventh International Conference on Sensing Technology (ICST), 978-1-5090-6526-4/17/$31.00 © 2017, pp. 153–157.

How, J. P., Behihke, B., Frank, A., Dale, D. and Vian, J. 2008. Real-time indoor autonomous vehicle test environment. *IEEE Control Systems Magazine* 28(2): 51–64, doi: 10.1109/MCS.2007.914691.

Karaman, S. et al. 2017. Project-based, collaborative, algorithmic robotics for high school students: Programming self-driving race cars at MIT. 2017 IEEE Integrated STEM Education Conference (ISEC), Princeton, NJ, pp. 195–203, doi: 10.1109/ISECon.2017.7910242.

Leni, A. E. S. 2017. Instance vehicle monitoring and tracking with internet of things using arduino. *International Journal on Smart Sensing and Intelligent Systems* 10: 123–136.

Man, C. K. Y. L. L., Koonjul, Y. and Nagowah, L. 2018. A low cost autonomous unmanned ground vehicle. *Future Computing and Informatics Journal* 3(2): 304–320.

Miao, X., Li, S. and Shen, H. 2012. On-board lane detection system for intelligent vehicle based on monocular vision. *International Journal of Smart Sensing and Intelligent System* 5(4): 957–972.

Nag, A., Menzies, B. and Mukhopadhyay, S. C. 2018. Performance analysis of flexible printed sensors for robotic arm applications. *Sensors and Actuators A: Physical* 276: 226–236.

O'Kelly, M., Sukhil, V., Abbas, H., Harkins, J., Kao, C., Pant, Y., Mangharam, R., Agarwal, D., Behl, M., Burgio, P. and Bertogna, M. 2019. F1/10: an open-source autonomous cyber-physical platform. ArXiv, abs/1901.08567.

Olgun, M. C., Baytar, Z., Akpolat, K. M. and Sahingoz, O. K. 2018. Autonomous vehicle control for lane and vehicle tracking by using deep learning via vision. 2018 6th International Conference on Control Engineering & Information Technology (CEIT), Istanbul, pp. 1–7, doi: 10.1109/CEIT.2018.8751764.

Pannu, G., Ansari, M. and Gupta, P. 2015. design and implementation of autonomous car using Raspberry Pi'. *International Journal of Computer Applications* 113(9): 22–29.

Patel, N. et al. 2019. A deep learning gated architecture for UGV navigation robust to sensor failures. *Robotics and Autonomous Systems* 116: 80–97.

Paull, L., Tani, J., Ahn, H., Alonso-Mora, J., Carlone, L., Cap, M., Chen, Y. F., Choi, C., Dusek, J., Fang, Y. et al. 2017. Duckietown: an open, inexpensive and flexible platform for autonomy education and research. 2017 IEEE International Conference on Robotics and Automation (ICRA), IEEE, pp. 1497–1504.

Pomerleau, D. A. 1989. Alvinn, an autonomous land vehicle in a neural network. In Advances in Neural Information Processing Systems (NIPS). Technical report, Carnegie Mellon University, Computer Science Department, Pittsburgh, PA, pp. 305–313.

Roscoe, W. 2020. Donkey Car: an opensource DIY self driving platform for small scale cars, available at: http://www.donkeycar.com (accessed June 17, 2020).

Schwartz, J. D. and Milam, M. 2008. On-line path planning for an autonomous vehicle in an obstacle filled environment. 2008 47th IEEE Conference on Decision and Control, IEEE, pp. 2806–2813.

Seelye, M., Gupta, G. S., Seelye, J. and Mukhopadhyay, S. C. 2010. Camera-in-hand robotic system for remote monitoring of plant growth in a

laboratory. Proceedings of 2010 IEEE I2MTC Conference, Austin, TX, May 4-6, pp. 809–814.

Sen Gupta, G., Mukhopadhyay, S. C. and Finnie, M. 2009. Wi-Fi based control of a robotic arm with remote vision. Proceedings of 2009 IEEE I2MTC Conference, Singapore, May 7, pp. 557–562.

Sen Gupta, G., Mukhopadhyay, S. C., Demidenko, S. and Messom, C. H. 2006. Master-slave control of a teleoperated anthropomorphic robotic arm with gripping force sensing. *IEEE Transactions on Instrumentation and Measurement* 55(6): 2136–2145.

Srinivasa, S. S., Lancaster, P., Michalove, J., Schmittle, M., Rockett, C. S. M., Smith, J. R., Choudhury, S., Mavrogiannis, C. and Sadeghi, F. 2019. Mushr: A low-cost, open-source robotic racecar for education and research. arXiv preprint arXiv:1908.08031.

Tian, Y. et al. 2018. Deeptest: automated testing of deep-neural-network-driven autonomous cars. Proceedings of the 40th International Conference on Software Engineering, Gothenburg, May, pp. 303–314.

Toupet, O. et al. 2019. Terrain-adaptive wheel speed control on the Curiosity Mars rover: algorithm and flight results. *Journal of Field Robotics* 37(5): 699–728.

Wu, B. et al. 2017. SqueezeDet: unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, Honolulu, July, pp. 129–137.

Zhang, Q. and Du, T. 2019. Self-driving scale car trained by deep reinforcement learning. arXiv preprint arXiv:1909.03467.

Zhang, W. and Mahale, T. 2018. End to end video segmentation for driving: lane detection for autonomous car. arXiv 2018, arXiv:1812.05914.