Autonomous Car Graduation Project
First Term Report

Under Supervision of Dr. Mohammed Fouad
Dr. Ahmed Kabeel

Written by Ibrahim Izz El-Deen
2-2018

General Overview

Autonomy and Automotive Embedded Systems are the current trend in the car industry. Look around to see how Electric Powertrains have overtaken traditional Internal Combustion Engines, and how smart systems within cars have risen in new models every year, resulting in more convenient and reliable automated driving systems.

Thus, clever engineers are in need, not just in electrical specializations, but also in software domain. Therefore, this project was mainly proposed to help produce the kind of engineers that have the basic understanding of electric self-driving cars technology and how to choose the right hardware components, develop the right software, simulate a self-driving car and implement it on real road scenarios, using Computer Vision, Machine Learning and Robot Operating System (ROS).

Our Approach is very simple, we do our research, learn, simulate and then implement. We might not reach our final goal by 100% but at least we guarantee a good learning process in our way through.

The idea of this project relies heavily on the concept of *Behavioral Cloning* That is when you do something, in our case, manually drive a car, record your data, feed these data to your machine learning model to train it to deal with different scenarios and then release your model on the road to see how well it autonomously drive and navigate its way.

We use a bunch of technologies and hardware components to do so. In the following pages, we will detail some of them.

Software

The need for software is crucial. Software makes your machine function in more complicated, robust and creative ways, in our case: to self-drive and navigate. We use a bunch of software technologies, libraries and frameworks in order to simulate self-driving.

Let's first understand that our car needs to sense and percept its environment, mainly using a camera module. The car sees its surroundings, also called "Detection", Recognizes different objects, also called "Classification", and make decisions based on that perception.

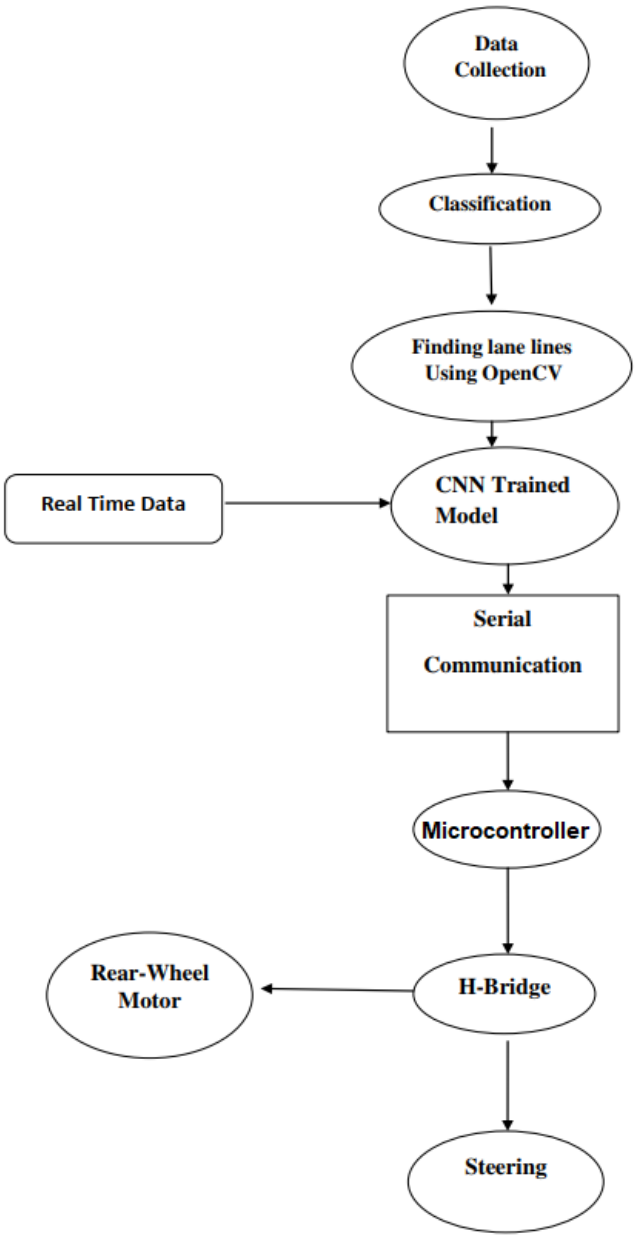
Decisions include following lane lines or trying to stay within road lanes, stopping when intercepted by a pedestrian or obeying a road symbol like "STOP" or "Turn Left".

Self-driving car made in this project is able to navigate the track by making prediction using the trained data set with the help of CNN model.

First, Data are gathered via video streaming through a Camera using OpenCV. After the collection of data, the video is segmented into frames and classified into classes. This classified data is converted into required format using computer vision algorithms, and after pre-processing, the data is feed to a CNN model for training.

CNNs is used for extracting the features from the images and learn through these features by updating the bias and weights of the perceptron. The trained model then takes the input images from live camera and predicts which direction

to choose or stop. The trained model after prediction generates a string, and through serial communication, the string is sent to a microcontroller. Finally, the microcontroller processes the code according to the string received from the trained model and sends control signals to the ESC to drive motors of the car to move or stop according to the prediction.



Beginning with lane lines, there are number of *Image Processing* operations required to detect lane lines using *OpenCV* Library as follows:

1. Grayscale

Processing gray scale image is easier than RGB image because RGB image has three planes one for each red, green and blue along with different pixel intensities for each plane at same position.

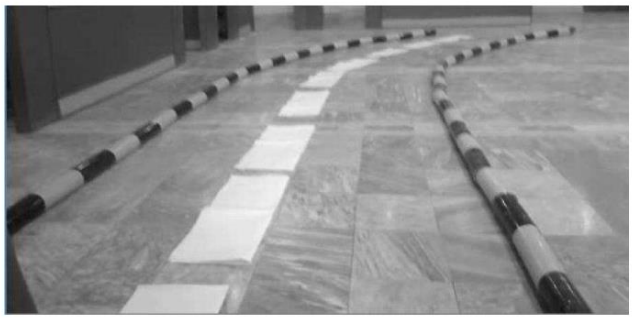


Figure 5.3: Grayscale

2. Gaussian Blurring

This technique is used to reduce noise and for smoothing of image. Reducing noise and smoothing is done by Gaussian filter. Blurring is necessary before canny edge detection otherwise, edge would not be detected causing noise points detection as well. Gaussian blur is applied on the gray scale image.

3. Canny Edge Detection

This technique is used for detecting edges in our image. Canny edge detection method is applied on blurred image.

Blurred image is fed to canny edge detection so as to ensure accurate edges detection with no noise factor as shown in figure 5.5

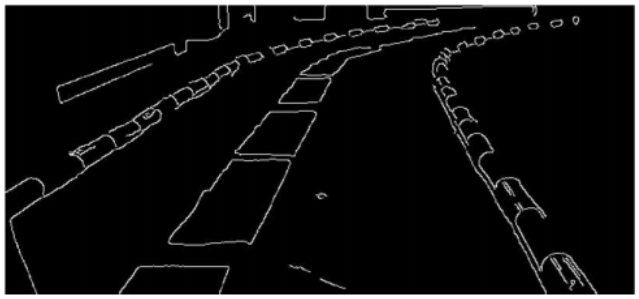


Figure 5.5 Canny Edge Detection

4. Hough Transform

Hough transform is a feature extraction technique used in image processing. The purpose of this technique is to find the points on the image with same threshold and create lines on them. This technique has ability to join the points having gaps on the image and same features. See Figure 5.8.

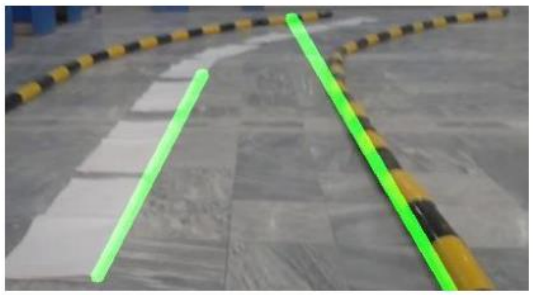


Figure 5.8(b): Display lines

All these Image Processing Techniques we do it using the following code written in Python and OpenCV

```
import cv2
import numpy as np

def make_points(image, line):
    slope, intercept = line
    y1 = int(image.shape[0])# bottom of the image
    y2 = int(y1*3/5)         # slightly lower than the middle
    x1 = int((y1 - intercept)/slope)
    x2 = int((y2 - intercept)/slope)
    return [[x1, y1, x2, y2]]

def average_slope_intercept(image, lines):
    left_fit    = []
    right_fit   = []
    if lines is None:
        return None
    for line in lines:
        for x1, y1, x2, y2 in line:
            fit = np.polyfit((x1,x2), (y1,y2), 1)
            slope = fit[0]
            intercept = fit[1]
            if slope < 0: # y is reversed in image
                left_fit.append((slope, intercept))
            else:
                right_fit.append((slope, intercept))
    # add more weight to longer lines
    left_fit_average  = np.average(left_fit, axis=0)
    right_fit_average = np.average(right_fit, axis=0)
    left_line  = make_points(image, left_fit_average)
    right_line = make_points(image, right_fit_average)
    averaged_lines = [left_line, right_line]
    return averaged_lines

def canny(img):
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    kernel = 5
    blur = cv2.GaussianBlur(gray,(kernel, kernel),0)
    canny = cv2.Canny(gray, 50, 150)
    return canny

def display_lines(img,lines):
    line_image = np.zeros_like(img)
    if lines is not None:
        for line in lines:
            for x1, y1, x2, y2 in line:
                cv2.line(line_image,(x1,y1),(x2,y2),(255,0,0),10)
    return line_image

def region_of_interest(canny):
    height = canny.shape[0]
    width = canny.shape[1]
    mask = np.zeros_like(canny)

    triangle = np.array([[
    (200, height),
    (550, 250),
    (1100, height),]], np.int32)

    cv2.fillPoly(mask, triangle, 255)
    masked_image = cv2.bitwise_and(canny, mask)
    return masked_image
```

```
# image = cv2.imread('test_image.jpg')
# lane_image = np.copy(image)
# lane_canny = canny(lane_image)
# cropped_canny = region_of_interest(lane_canny)
# lines = cv2.HoughLinesP(cropped_canny, 2, np.pi/180, 100, np.array([]),
minLineLength=40,maxLineGap=5)
# averaged_lines = average_slope_intercept(image, lines)
# line_image = display_lines(lane_image, averaged_lines)
# combo_image = cv2.addWeighted(lane_image, 0.8, line_image, 1, 0)
#
cap = cv2.VideoCapture("test2.mp4")
while(cap.isOpened()):
    _, frame = cap.read()
    canny_image = canny(frame)
    cropped_canny = region_of_interest(canny_image)
    lines = cv2.HoughLinesP(cropped_canny, 2, np.pi/180, 100, np.array([]),
minLineLength=40,maxLineGap=5)
    averaged_lines = average_slope_intercept(frame, lines)
    line_image = display_lines(frame, averaged_lines)
    combo_image = cv2.addWeighted(frame, 0.8, line_image, 1, 1)
    cv2.imshow("result", combo_image)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

A **Convolutional neural network** is a class of deep neural networks. Convolutional neural networks contains multilayer perceptron and are capable to understand complex patterns.

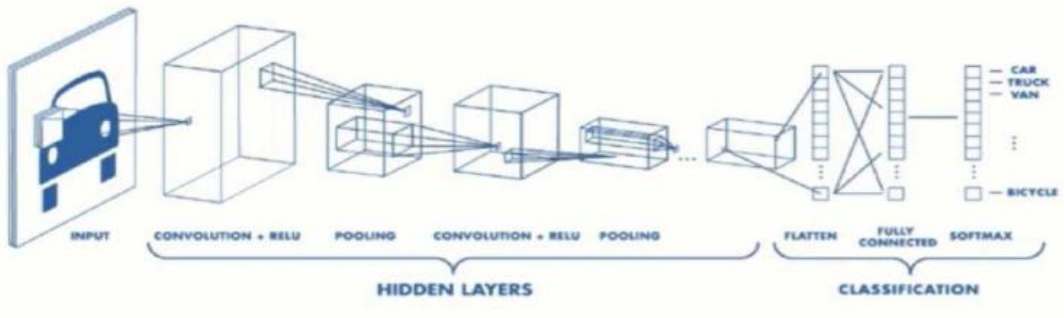


Figure 3.6: Convolutional Neural Network [3]

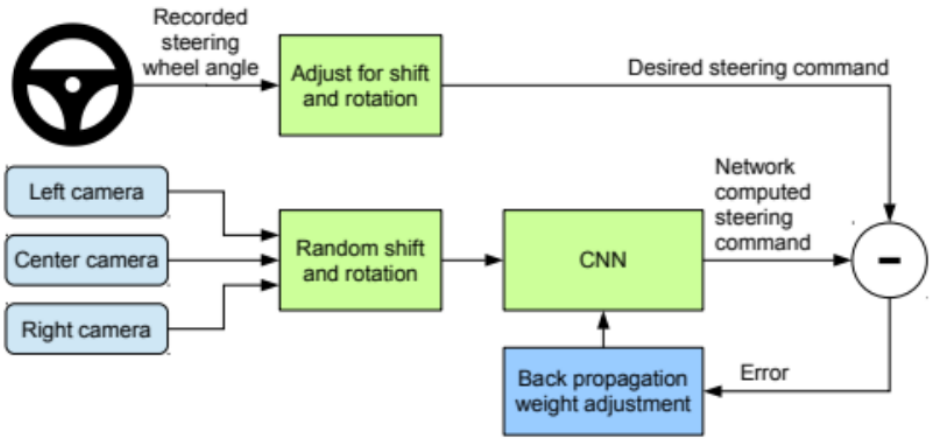
Supervised learning is used for training of the data. This data is classified then trained using CNN sequential model. CNN model used for the training of the data contains Number of

hidden layers. CNN is used for extracting the features from the images and learn through these features by updating the bias and weights of the perceptron.

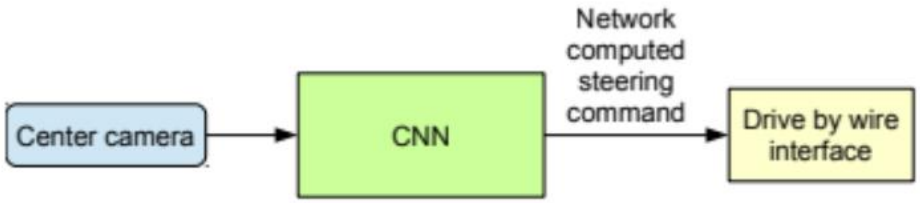
A training data set consists of inputs. Each input corresponds to some label to train the model. The CNN learns from this data, update the bias and weights accordingly and come up with a trained model. Test images are the new data that the neural network has never seen before.

In our case, the training set consists of frames as input, and corresponding steering angle and throttle speed as output. And the mission is to try to predict these outputs for completely new set of data to make our car drive itself on new roads.

Images are fed into a CNN, which then computes a proposed steering command. The proposed command is compared to the desired command for that image and the weights of the CNN are adjusted to bring the CNN output closer to the desired output.



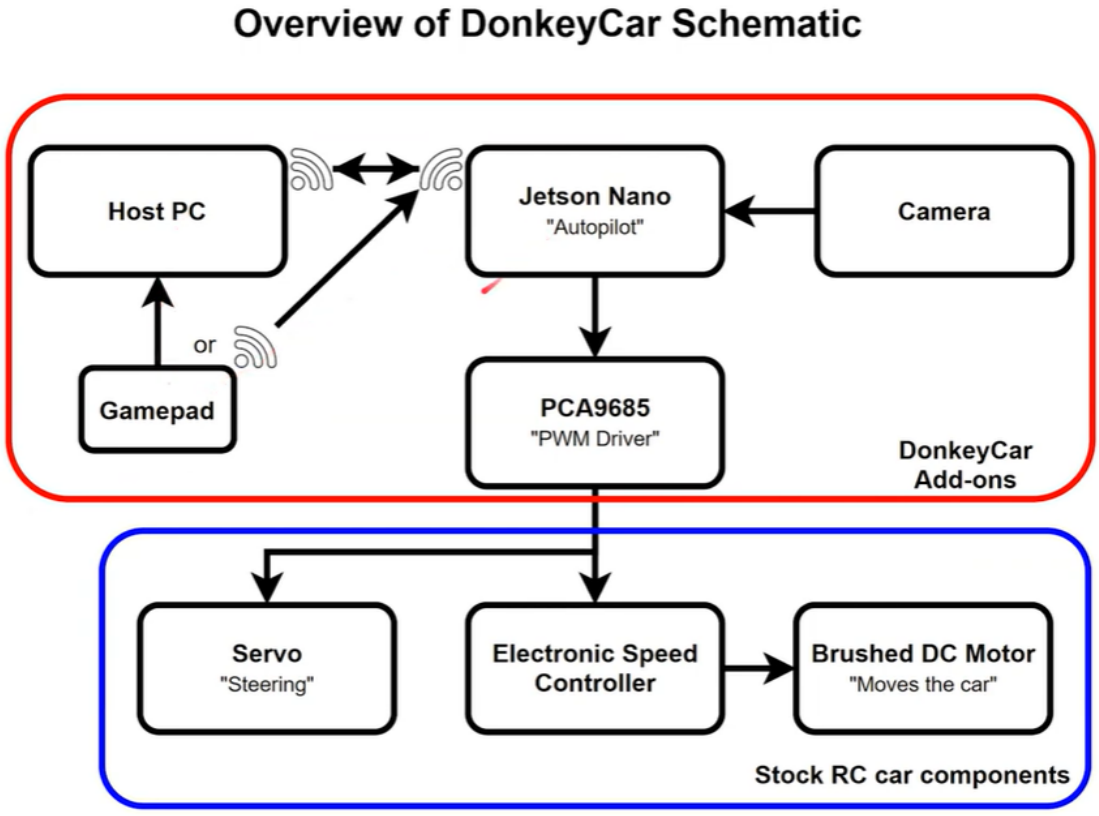
Once trained, the network can generate steering from the video images of a single center camera. This configuration as shown.



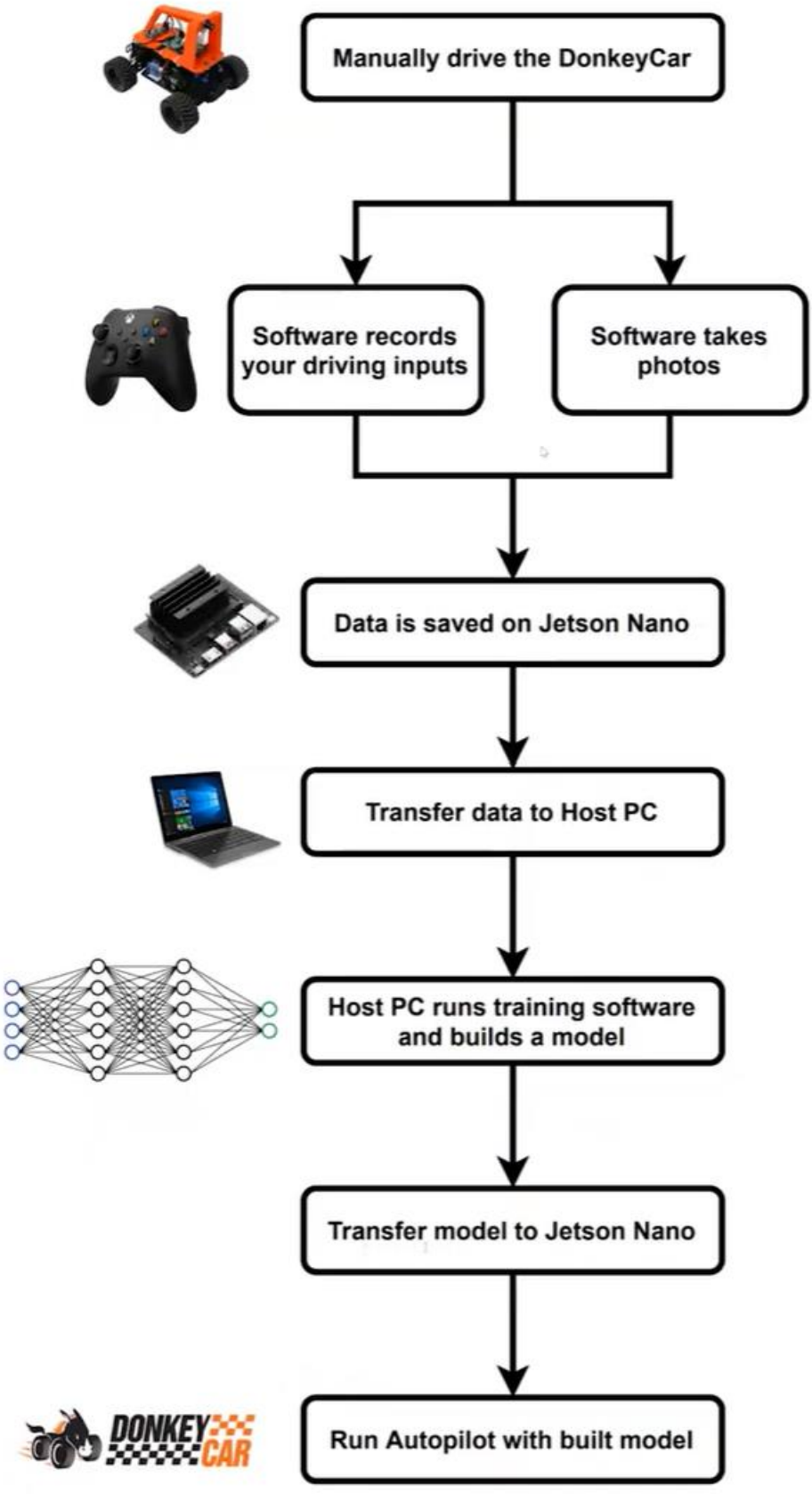
In this behavioral cloning stage, we use *Self-Driving Simulators* like the one based on Unity game engine produced by Udacity in its self-driving nanodegrees, and another one based on Keras and Tensorflow called DonkeyCar Simulator produced by a project platform with the same weird name.

Implementation

Let's now shade light on the hardware part, where we actually implement behavioral cloning on a track model or, in a more complicated scenario, on roads! The backbone reference project is The Donkey Car Project, starting with a few diagrams to lustrate the idea, here is an overview:



Note that the *Jetson Nano* is our main controller and processing board. We will discuss each vital component in detail soon.


Manually drive the DonkeyCar
Software records your driving inputs
Software takes photos
Data is saved on Jetson Nano
Transfer data to Host PC
Host PC runs training software and builds a model
Transfer model to Jetson Nano
Run Autopilot with built model
DONKEY CAR

Purchased Components

- **Jetson Nano B01 4GB Developer Kit**

The main processing unit in our project. It's used to preprocess images, train the neural network and predict the output by eventually running the model in a real-time scenario. It runs Linux Ubuntu (Jetpack) Operating System, and must interface with the camera and PWM Controller later on.

- **Raspberry Pi Camera V2.1 8MP**

The main sensor in our project used to percept its surroundings. The camera is 8MP and has an angle of 160 degrees connected by a ribbon cable to the CSI port on the board. It's capable of 3280 x 2464 pixel static images, and supports 1080p30, 720p60 and 640x480p90 video.

- **Kyosho DMT VE-R 4WD RC Car**

The -almost- perfect choice for our project. It has a 2400KV ORION VORTEX R10 brushless motor, a PWM ESC, a ready Servo Motor and a great metal chassis. Both the throttle motor and the servo will be programmed using a PWM controller linked to the board.

- **3S 3300mAh 11.1V LiPo Battery**

Choosing the right battery to our car is crucial. LiPo Batteries have the advantage all the way over NiMH ones.

The weight/power ratio in LiPo batteries is significantly better. LiPo batteries are noticeably lighter and they can store the same amount or more energy relative to their capacity than NiMH batteries. The power output of LiPo batteries is greater in quality and quantity. The power output of LiPo batteries is steady throughout the discharge, whereas the power output of NiMH batteries starts to decrease soon after charging because of higher discharge rate of the battery type.

Therefore, with a LiPo battery with the same capacity as a NiMH battery a longer drive time and better performance can be achieved.

- **IMAX B6 LiPo Balance Charger**

Choosing the right charger is not less important, too. This charger accurately balances and charges Lithium Polymer, LiFe, NiCd and NiMH batteries. It will handle LiPo/LiFe up to 6S and NiMH/NiCd up to 15S and shows individual cell voltage during charge with real-time updates throughout the charge cycle.

Battery balancing and battery redistribution refer to techniques that improve the available capacity of a battery pack with multiple cells (usually in series) and increase each cell's longevity.

The intuitive menu system means charging and cycling is an easy process and can be done quickly and accurately in the field, by using a 12V input either such as a car battery, or at home with a 12V power supply.

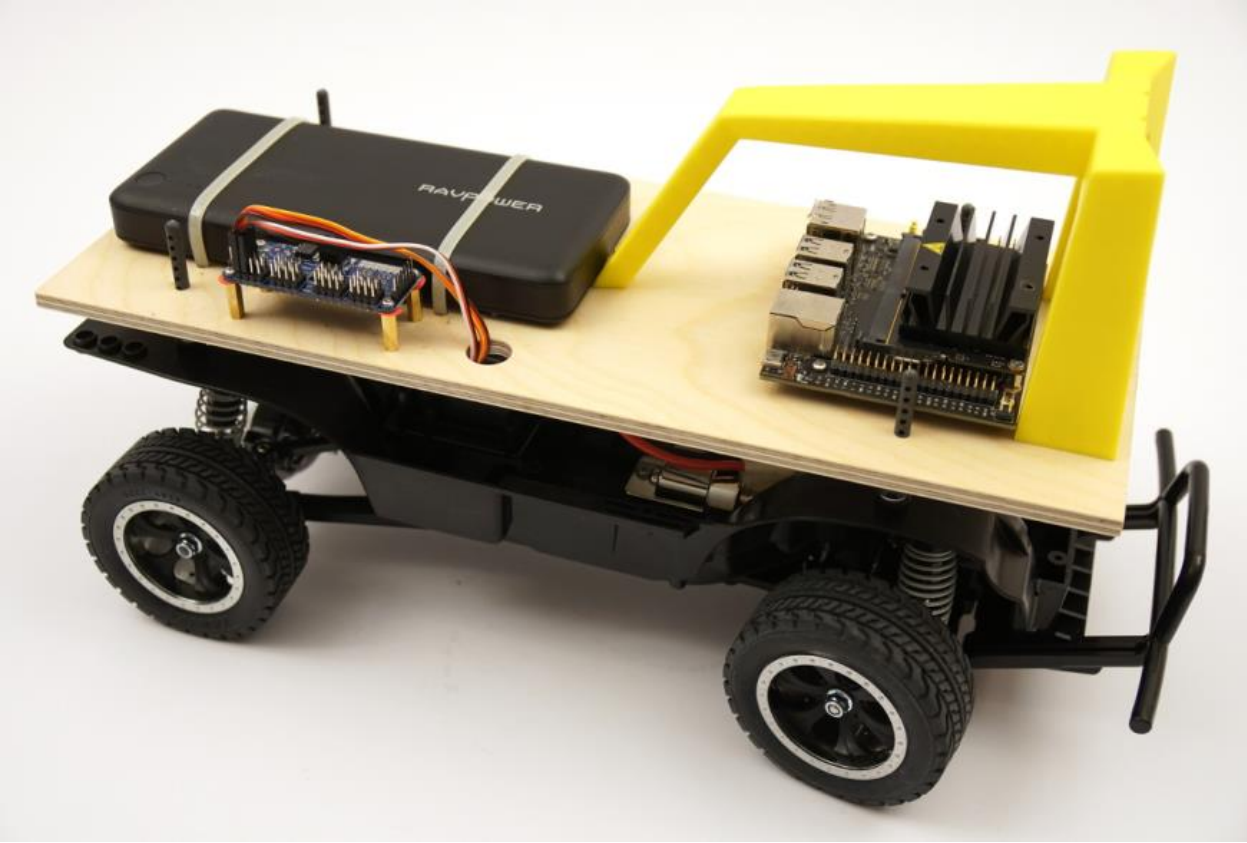
- **Kinect V1**

Can be used for SLAM in the future. SLAM (simultaneous localization and mapping) is a technique for creating a map of environment and determining robot position at the same time. It is widely used in robotics.

While moving, current measurements and localization are changing, in order to create map it is necessary to merge measurements from previous positions.

All this can be achieved using Robot Operating System (ROS). In addition, an autonomous robot car can be simulated using Gazebo package in the ROS 2 Framework.

The final product should look like this one here:



However, we are yet to purchase other vital components including the SSD, Power Bank and the PWM Controller.

Project Timeline and Future Steps

Month	Outcome
10	• Learnt Python Programming • Learnt about Neural Networks
11	• Learnt more about Neural Networks • Learnt about Numpy and Pandas • Learnt Basic Linux Ubuntu Commands
12	• Purchased The RC Car, The Jetson Nano, The Pi Camera, LiPo Battery, IMAX Charger and other cables and adapters • Learnt about OpenCV and how to detect lane lines using Hough Transform Method
1	• To Learn about Keras and use it in multi-classification of different road symbols
2	• To Implement Behavioral Cloning by Simulating a self-driving car on Udacity Simulator • To Learn ROS 2 and Gazebo Simulator
3	• To Simulate a self-driving car using ROS 2 and make it reach a goal position via GPS • To add more complex functions to this simulated model including Lane Assist, Cruise Control, T-Junction Navigation and Crossing Intersections
4	Upcoming
5	
6	

References

- https://www.duckietown.org/
- https://docs.donkeycar.com/
- https://www.eurorc.com/page/69/differences-between-nimh-and-lipo-batteries#:~:text=The%20advantages%20of%20lithium%20batteries,their%20capacity%20than%20NiMH%20batteries
- https://link.springer.com/referenceworkentry/10.1007/978-0-387-30164-8_69#:~:text=Behavioral%20cloning%20is%20a%20method,input%20to%20a%20learning%20program
- https://www.udemy.com/course/applied-deep-learningtm-the-complete-self-driving-car-course/
- https://www.udemy.com/course/ros2-self-driving-car-with-deep-learning-and-computer-vision/learn/lecture/33120242?start=0#content
- https://custom-build-robots.com/top-story-de/donkey-car-e-book/14046
- https://www.youtube.com/watch?v=8TtNsaQVcsw

Team Members

اسم الطالب	الكود
منة أحمد دعبس	277-2018
أحمد محمود موافي	42-2018
محمد مصطفى إسماعيل	241-2018
محمد السيد أبو سمرة	231-2018
عبد الرحمن علاء الصعيدي	162-2018
محمد محمود سليمان	203-2017
هدير البطال	298-2018
حسين عبد الحليم بدير	104-2018
إبراهيم السيد يوسف عز الدين	2-2018

You can download a PDF copy of this document here: