Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

2022-2023

PROJECT TITLE
Video-Trained Autonomous Car
PROJECT ADVISOR
Prof. Dr. Bahadır Kürşat Güntürk
TEAM MEMBERS
Fatma Büşra Yaman - 64190002 Taha Yunus Hamamcıoğlu - 64190006

Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

Project Code
Project Title: Video-Trained Autonomous Car
Project Advisor: Prof. Dr. Bahadır Kürşat Güntürk
Project Team Members: Fatma Büşra Yaman, Taha Yunus Hamamcıoğlu
Sponsor Company (if any) :

BUDGET (TL)	PROPOSED	CONSENTED
IMU FUNDING	14.600	15.070
SPONSOR COMPANY FUNDING	-	-
TOTAL	14.600	15.070

PROJECT PLAN	PROPOSED	CONSENTED
PROJECT PLAN Duration in Weeks	28 Weeks	28 Weeks
STARTING DATE		

Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

Project Code

PROJECT ADVISOR	**DEPARTMENT CHAIR**
Name: Bahadır Kürşat Güntürk	**Name: Mehmet Kemal Özdemir**
Contact Information: E-mail : bkgunturk@medipol.edu.tr	**Contact Information:** E-mail : mkozdemir@medipol.edu.tr
Signature: 	**Signature:**

TEAM MEMBER	**TEAM MEMBER**
Name: Fatma Büşra Yaman	**Name: Taha Yunus Hamamcıoğlu**
Contact Information: Tel : 5387694738 E-mail : fatma.yaman@std.medipol.edu.tr	**Contact Information:** Tel : 5303401240 E-mail : taha.hamamcioglu@std.medipol.edu.tr
Signature: 	**Signature:** 

Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

Project Title: Video-trained Autonomous Car

Project Advisor: Prof. Dr. Bahadır Kürşat Güntürk

Team Members: Fatma Büşra Yaman, Taha Yunus Hamamcıoğlu

Project Group Title:

PROJECT OVERVIEW/SUMMARY/ABSTRACT

The focus of this project is applying the latest software technologies to hardware by building an autonomous car that will be trained using video data. The autonomous car is capable of making decisions using deep learning models in tandem with the Intel Realsense camera that is capable of extracting the depth information as well as colour information. The car is trained and tested on modular roads that can be rearranged to create diversified paths. For the training, unique models were created that separates us from the rest.

The project timeline includes 5 work packages. The project progress includes the reports, presentations, and the literature survey. Hardware package includes the design and culmination of the remote-controlled car using the electronic equipment and handling the communication. Dataset preparation is the third package responsible for the creation of the modular roads and the train and test datasets. Deep learning package is for the development of our own dataset alongside the train, test, inference. Finally, the optimization is for increasing the speed and accuracy of the models.

First three work packages were successfully completed by reviewing 15 articles, having a car that can finish 5 consecutive laps on the modular roads, and having a dataset of 40 thousand frames in total respectively. For the fourth package the project aimed to achieve a classification accuracy of 95 percent, a line keeping rate of at least 96%, real time inference, and an avoidance rate of at least 98%. Also, the majority of the elements should be on the diagonal part i.e. the true positive section of the confusion matrix. Last package was accomplished by the fastening of inference and trying a pretrained model in tandem.

Some problems in the hardware section that we were not familiar with delayed the project progression, however, by persevering and handling the other familiar sections simultaneously, we were able to finish the project on time successfully.

Keywords: autonomous car, video trained car, deep learning, CNN, remote controlled car, modular roads, dataset creation

Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

1. OBJECTIVE OF THE PROJECT:

An autonomous car is a type of car that can navigate around all by itself using AI algorithms for the purposes of decision making. To infer on the situation on the outside world, the received data from the cameras and other types of data from the sensors are utilized. To create a reliable dataset, engineers collect a wide array of videos that record various occurrences on the real world. Then they label each frame and feed them into the training pipeline to obtain a model that tells the car how to behave in certain situations. After the end-to-end training of the AI vehicle is completed and properly tested, it is ready to drive autonomously on real roads without the help of a human driver. Our project takes these video trained AI and autonomous navigation concepts and applies them to a remote controlled toy car with self-created roads. Our objectives are as follows:

Our objectives are as follows:

1) Developing a finished autonomous car system that can drive around paths of our modular roads.

2) Creating a dataset of that consists of many different paths by using our modular roads. These roads are puzzle-like in design with interlocking parts and repeating sections so that they can be substituted swiftly and create a complete path.

3) Achieving performance of at least 25 FPS, that corresponds to real-time, on the Jetson Nano platform to make sure that the car can respond quickly to changing stimulus.

4) Ensuring that the car follows the road with utmost accuracy and reliability.

5) Developing the ability to avoid or bypass multiple moving toy cars and pedestrians. Therefore, demonstrating the autonomous car's ability to react to changing road conditions instead of just predictable road detection.

2. LITERATURE REVIEW:

2.1. Evaluating Deep Learning Algorithms for Steering an Autonomous Vehicle

The project that this paper is written for has very similar objectives and methods with our project. The aim of the developers was to build an autonomous car which can drives itself in an indoor environment without crashing obstacles. The equipment used in this project is Raspberry Pi Model B+, a microcontroller and a camera. Dataset was generated by developers by driving the car manually by a remote controller. Dataset consisted of the frames captured by the camera and corresponding commands sent by the remote controller. A virtual machine was used for training which is Amazon EC2. Pretrained models were preferred which were Xception, VGG16 and MobileNet. The neural network was not embedded to the car. They used a different computer to locate the neural network. When the car captures a frame, it was sent to this computer to be classified, then the output which was the steering angle is sent back to the car. Two types of evaluation was performed in this project. In quantitative evaluation, accuracy of the model was tested. Best accuracy was obtained by an Xception model which was 81,19%. In qualitative evaluation, they recorded amount of collisions and see if the car can actualize the outputs of the neural network and they reached an average crash avoidance rate of 78.57. [1]

2.2. Real-Time Self-Driving Car Navigation Using Deep Neural Network

The paper is proposing a project that is aiming to build an autonomous car controlled by a convolutional neural network that can predict the correct steering angles by the real-time RGB frames that the car's camera captures. In the project, car was built by the using the following hardware: Raspberry Pi 3 Model B , a camera and a RC car, Arduino Uno. Data collection method was done by driving the car manually. While driving the car, RGB frames from the camera were recorded with the steering angle commands sent by the developers at the same time. When the dataset that consisting of RGB frames which were labeled by corresponding steering angle command were generated, it was enlarged by data augmentation. After that a convolutional neural network was built by using TensorFlow and Keras libraries of Python programming language and it was trained by the dataset consisting of more than 35.000 instances. Neural network performed perfectly by giving an accuracy rate 89% on the validation data. Next step was testing the car on an actual road. Despite the camera latency, car performed perfectly on the road as well. [2]

2.3. End to End Learning based Self-Driving using JacintoNet

This paper concentrates on a technique called end-to-end learning and uses this technique to build an autonomous car in a virtual environment. The technique is deeply explained in the paper and implemented by using a network called JacintoNet. In the project, data was gathered by driving the car in the virtual environment for a very long time. Driving was done in various conditions. Aim was to obtain the data which includes desired steering angle and speed for the images that the car's camera captured while driving. At the end of the data gathering process, more than 200.000 samples were used to train the network. The performance was observed by comparing the path of the car that is followed during testing with the ground-truth path. The network performed perfectly. Then, it was compared with one of the most popular networks used to deal with image inputs. Their performance was very similar. The project shows that networks that are not complex like the one that is used in this project is doing a great job on end-to-end learning technique. [3]

2.4. Design and Implementation of Autonomous Car using Raspberry Pi

Ways to build a self-driving car by combining most of the popular detection algorithms were discussed and implemented in this paper. The car was built on top of a Raspberry Pi chip. The main idea was building the car that can be driven by a remote controller and to be fully controlled or to be assisted by latest autonomous technologies. A mobile application was preferred to drive the car. Prevention of the car from crashing obstacles while driving was provided by a ultrasonic sensor. Lane detection was done by combining and getting the best features of two commonly used lane detection algorithm which are future based and model based. Image processing was done by taking every detail into consideration to increase the efficiency. For navigating the self-driving car, manual driving was performed in various conditions. Prevention from obstacles and navigation parts allowed the car to be fully autonomous. The algorithms were successfully implemented to the car and the target was achieved. A neural network was not used in this project but very wise approaches on processing of image inputs that also can be used in our project broadened our horizons. [4]

2.5. End to End Learning for Self-Driving Cars

This paper was distinguished by its way of implementing end-to-end learning method to real cars. Purpose of this project was building a convolutional neural network that can predict the correct steering angle directly from the visual inputs obtained by the camera located to the front of a car. With the followed method, intermediate steps like lane following, object detection

were skipped and the learning was only done by the raw image inputs. To be able to gather the training data and match the captured images with correct steering angle inputs, car was driven by a human driver on different roads in different cities. After the data was gathered, the neural network was built which is consisting of 9 layers. Human driver's steering angles were considered as ground truth values and the aim was minimizing the mean squared error between neural network's steering predictions and ground truth values. Evaluation of this network was done in a simulation before taking the car to real world. In the simulation, network was tested by the recorded videos of real world scenarios. Thanks to simulation, developers had the chance to find out what will happen if the neural network was driving the car. When it was decided that the neural network's performance was pretty satisfying. The neural network was embedded to a real car, with the help of advanced equipment that is used in this project. The car drove itself perfectly on the roads of US. [5]

2.6. Comparison of Machine Learning Algorithm's on Self-Driving Car Navigation using Nvidia Jetson Nano

Comparison of three machine learning algorithms on an autonomous car was aimed in this paper. These algorithms were SVM, ANN-ML and CNN-LSTM. To compare their performance an autonomous car was built with the equipment of a Jetson Nano, a microcontroller and a camera. Data which were image inputs were gathered by driving the car manually to train the models. A mini road was designed to test the car with stated algorithms. Algorithms were tested for three different scenarios and in three different car speeds. Obstacles were also included in two of the scenarios. At the end it was observed that CNN-LSTM algorithm was giving the best result in all speeds of each scenarios. [6]

2.7. Real-Time Control Using Convolutional Neural Network for Self-Driving Cars

This project was aiming to building an autonomous car by end-to end approach. Car was built on top of a Jetson Nano processor. Data gathering was performed by manual driving. Gathered data includes the RGB images and corresponding steering angles. MobileNet was used in convolutional architecture. After training of the neural network, the model was tested to see if it can predict the correct steering angles when different RGB images of roads were given. Before testing the model by embedding it into the car, it was tested in a computer environment. When it was seen that the accuracy was good enough, the car was tested on the road. The obtained test accuracy on the road was approximately 85%. [7]

2.8. A Convolutional Neural Network Approach Towards Self-Driving Cars

An autonomous vehicle was designed in this project. Purpose was having the car predicting the desired steering angles directly from camera inputs. A convolutional neural network was used for this goal. The network was trained on the dataset by NVDIA and Udacity. These datasets were including road frames and correct steering angles. For processing the dataset, advantages of Robotic Operating System were used. Car was built by using these main components: Rasberry Pi 3B, a web camera, ultrasonic sensors. Ultrasonic sensors were used to avoid obstacles. After the network was trained, it was tested on a dataset provided from University of Cambridge on a simulator. Developers obtained an autonomy of 86% approximately. [8]

2.9. Deep learning algorithm for autonomous driving using GoogLeNet

The paper compares the best 3 CNN's from competitions that are about feature extraction to find the best model that works for autonomous driving. According to results, that model turns out to be GoogLeNet. Based on this model, the researchers propose a deep learning algorithm named as GoogLenet for Autonomous Driving (GLAD). [9]

2.10. Autonomous Car Driving Using Deep Learning

Most other research uses complex models which make it more expensive to achieve good results. This paper goes in the opposite direction and uses simple models like vanilla UNet/FCN which makes it less costly but still yields good results. Their model works on real cars after getting trained using CARLA driving simulation. [10]

2.11. Autonomous car using CNN deep learning algorithm

This article uses deep learning algorithms of CNN to that learns the suitable output for driving inputs to obtain an autonomous navigation system. Dataset is created on an open simulation system that is the teaching grounds to the AI. The model gives the steering commands after receiving the centre camera inputs. [11]

2.12. DeepPicar: A Low-Cost Deep Neural Network-Based Autonomous Car

DeepPicar is a small replica of the autonomous car DAVE-2 that uses deep CNNs that take the input frames from the camera, feeds it to the model, and obtains the steering angles. The replica car also uses the same architecture but it is a lower cost to train and test to car since it is a toy car that can drive around simple small-size roads. They show that Raspberry Pi can handle real time control of end-to-end deep learning models with its limited computing capabilities. [12]

2.13. End-to-end Learning of Driving Models from Large-scale Video Datasets

This paper aims to create a robust perception-action model. They teach the motion model using large resources of video data and create an end-to-end deep learning model. Their model incorporates FCN-LSTM architectures which is compatibly trained with the video dataset they acquired. This results in a model that can predict which action to take in which situation on real roads with real cars. [13]

2.14. Deep Learning Techniques for Obstacle Detection and Avoidance in Driverless Cars

This article is written by the intent to add to the smart cities by turning the cars into IoT devices that can drive autonomously. The experiments work with a small RC Raspberry Pi car. They command the car from their phones during the training section to create the dataset like our plan. Then they used a CNN architecture to obtain the final results that were around 89%. [14]

2.15. Multi-task deep learning with optical flow features for self-driving cars

The article offers to use optical flow to acquire better accuracy with autonomous cars. “The flow predictor, as a self-supervised deep network, models the underlying scene structure from consecutive frames and generates the optical flow. The controller, as a supervised multi-task deep network, predicts both steer angle and speed.” They also predict and make use of depth information. [15]

3. ORIGINALITY:

In this project, traditional ways were not followed. Originality of the project starts from generating the dataset and applies to each step. Overall progress will be briefly explained here to give a better idea of originality, all the methods will also be explained in detail in part-4.

To generate the original dataset, car was driven manually by a remote controller on the roads that were be designed by the project team. While driving, frames that the car was facing were recorded by the camera. Simultaneously, corresponding commands which were sent from the remote controller were also recorded. These commands were the speed values of left motors and right motors of the car which allows the car to both adjust speed and steering angle. Frames were labeled by matching these speed values and corresponding frames. As a result, a dataset was obtained which tells us which commands are needed to be followed in all possible scenarios that the car may come across. Also, the depth information of each frame was included in this dataset. Depth information of each frame were easily obtained because the camera that was used in the project is Intel Real Sense Camera with Depth Vision. This camera gave us the depth information of each image. This depth information was matched to the corresponding RGB frames and were used to avoid our car from obstacles.

Our way of gathering data can rarely be seen in existing projects but depth information implementation to this method forms a unique blend that was not used in any other projects before. Usually, autonomous car projects are following very similar strategies. It is very common to see an autonomous car that is following lanes and behaving in accordance. In this project, the car was designed in such a way that it will be trained by its manual experiences. Huge online datasets are being used to train the car but in this project the dataset is original. Usually, a lidar sensor is used to avoid obstacles by depth information, but instead of using a lidar, Intel Real Sense Camera was used which gave us the chance to obtain RGB frames and depth information form the same source. This approach of the project team increased the efficiency and enhanced the originality. Furthermore, unlike existing projects that are using pretrained models, project team built their unique neural networks. These neural networks were trained by using the all the mentioned information that forms the dataset. After the training was completed, project team obtained the neural networks which were able to decide on the desired steering angle and speed when an RGB frame and its depth information is given. And by using this neural network, the car had the ability to drive itself safely in all possible scenarios.

Another originality of this project was the modular roads that were built for training and testing purposes. These roads were designed like a giant puzzle. So that by combining different pieces, project team were able to train and test the car on many different roads.

4. SCOPE OF THE PROJECT AND EXPERIMENTS/METHODS:

1) Wireless connection of the remote controller

This is required for the seamless control of the car on the paths during the data collection section. The added nRF24L01 module allows for the wireless transmission of data between the remote controller and the toy car, while the Arduino joystick shield provides the input for the movements of the car.

To get this setup to work, a lot of effort had to be expanded since our nRF24L01s broke down from the start. Different types of codes that tries to send basic numerals through the communication channel all the way to different joystick communication codes for car control were tested. In total more than 10 types of codes were tested and the result was not changing. During this process, all types of different combinations for the electronic equipment was tried. For example, the Arduino Leonardo's were swapped with Arduino Uno, nRF24s were interchanged between each other. The cables were tested to understand if there was a physical contact issue. Capacitors were added from a breadboard to regulate the power flow. ESP32 modules were brought as a substitute.

In the end, none of the codes alongside with changed electronic trials resulted in a data flow. This cemented in our minds that the fault was on the nRF24s themselves since all the codes worked on other users. It was only possible to test this hypothesis when the new modules arrived at school though, and since we had to solve this issue before progressing to other topics, a lot of weeks were wasted in the first semester.

After the new nRF24s arrived, the problem was solved, and the remote controller Arduino codes were tweaked for it to work with the car effectively. In the beginning, the Joystick values were continuous from 0 to 680, however, when the car was ridden for testing, it was seen that our remote controller and motors were not sensitive enough to register the differences in the joystick inputs. This caused a change for the input scheme from the joystick to specific numbers, and change the regression problem to classification.

2) Connecting Intel Realsense camera for the RGB and depth

Making the Intel Realsense camera work with our model and car is an important part of our project since the depth sensory information is necessary for the originality of our project. The camera works alongside the Jetson Nano, so we connected the camera using its connection cable to the Jetson Nano and started to work on it. It was not just possible to install using commands like pip install since Jetson Nano is based on an ARM64 processor unlike typical computers that generally uses x86. This was also the reason a different version of Arduino had to be installed. Firstly, Archiconda was to be downloaded since regular anaconda does not work

either. It was used to create a new environment. The steps in the following website by Lieu Zheng Hong [16] were followed that combined and explained in more detail the main Github pages for downloading and implementing the intelrealsense2 for python. However, after following it to the end, at the testing part, some errors arose that were caused by the conflict between the virtual environments. The process was restarted after going to base environment, deleting the archiconda environments, and correcting the CMake versions. However, because of the partitions there was a limited space, and when it filled all the way without the complete installation, the device was formatted and its disk space partition increased. On this try, the method worked, and we had in our hands a camera that records RGB as well as depth information that is useable through python scripts.

However, when the project was close to the end, and the time for inference came, it was seen that the camera working with python2 was not compatible with the other parts working in python3. It was seen that most people had a hard time applying the intelrealsense2 to python3, and it was not innately compatible. After exhausting trials and research though with the correct commands and folders in the right places, the camera started to work with python3.

3) Mounting and connecting the modules

Assembling the car and making it useful in terms of deep learning purposes was a scope component. The base for the car was already built when the project started from previous projects, so it had motors and wheels and a plastic skeleton. On top of this, the Arduino Uno and Uno shield, Intel Realsense camera, and Jetson Nano was mounted. For the protection and mounting process of the Intel Realsense camera and the Jetson Nano, specially designed plastic encasings were 3D printed. For communication between the remote controller and the car, the nRF24 was used as mentioned. For the Arduino and Jetson, a direct connection was used. Camera and Jetson were also connected using the camera's connection cable.

Two 7.4 V Lipo battery with 2 cores, 2200 mAh, and 35C was used for powering the Jetson Nano and motors individually. Jetson Nano requires the voltage to not exceed 5 V, so a DC convertor was used to limit the voltage given. The camera and Arduino Uno takes the power necessary from the connection to Jetson Nano. When nRF24 was present, it took power from this battery by its connection to the Arduino Uno. Other battery is directly connected to the motor driver to power up the motors for the car to move.

4) Dataset Generation

4.1 Gathering data:

After the car was built and desired functionalities obtained, project team worked on data gathering. First step was building the modular roads. Puzzle-like playing mats were used to provide the modularity. By covering one surface of the mats with black cardboard and taping white lanes on top of them, 16 unit of playing mats were shaped in a way that they can form various paths when combined. Some of the constructed paths for data gathering are given below. These paths are only used to gather "forward", "right" and "left" labelled RGB frames. No obstacle were used on these paths.

Forward label corresponds to the left and right motor speed values of 155, 155; respectively.
Right label corresponds to the left and right motor speed values of -100, 160; respectively.
Left label corresponds to the left and right motor speed values of 160, -100; respectively.

Figure-1: Training path-1

Figure-2: Training path-2

Figure-3: Training path-3

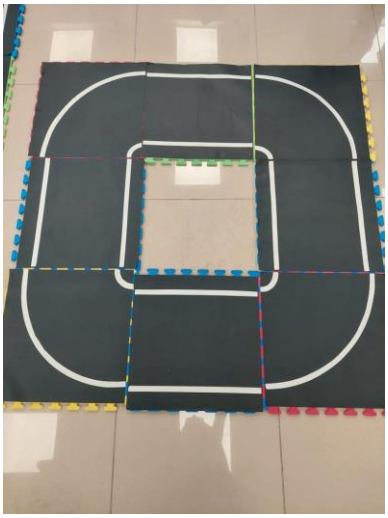
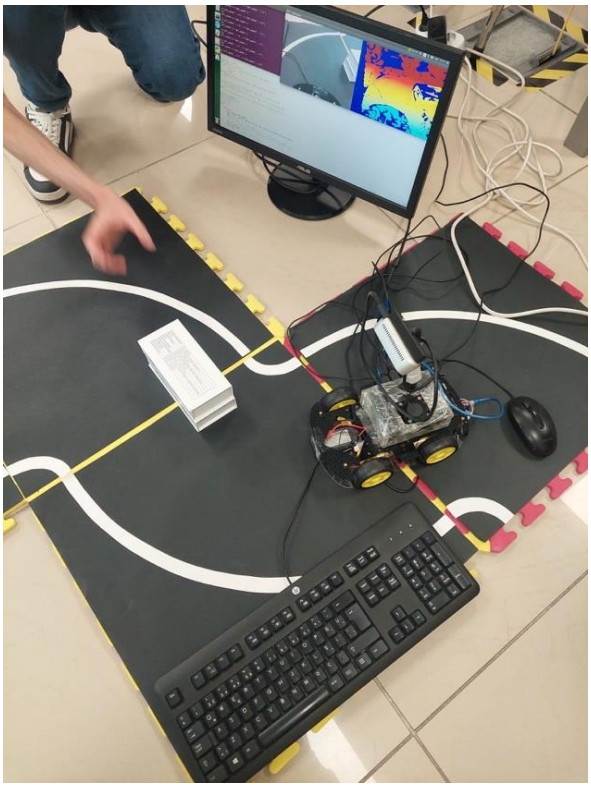


Figure-4: Training path-4

To gather the "stop" labelled depth data, various obstacles were put in front of the car while the speed values of the motors were both 0. An image representing the way of gathering this data is given below.

Figure-5: Depth information gathering



An instance of each label is given below:

Figure-6: Forward labeled - RGB

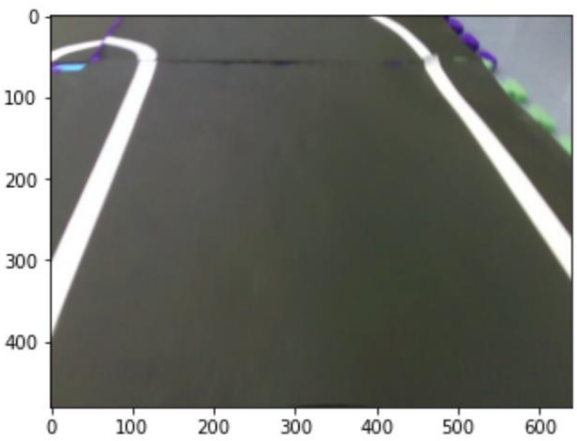


Figure-7: Left labeled - RGB

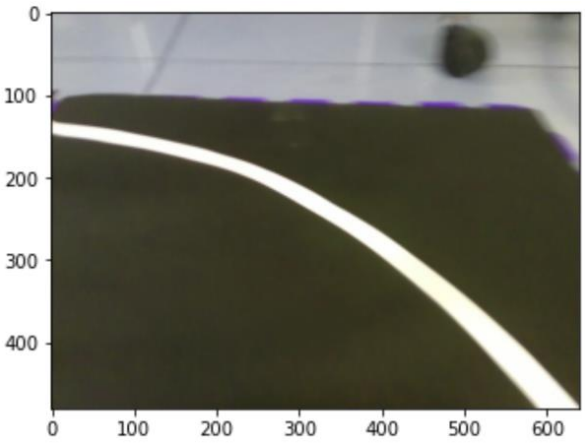


Figure-8: Right labeled - RGB

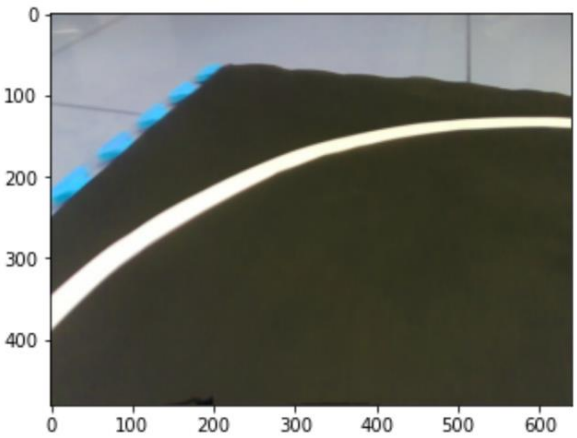
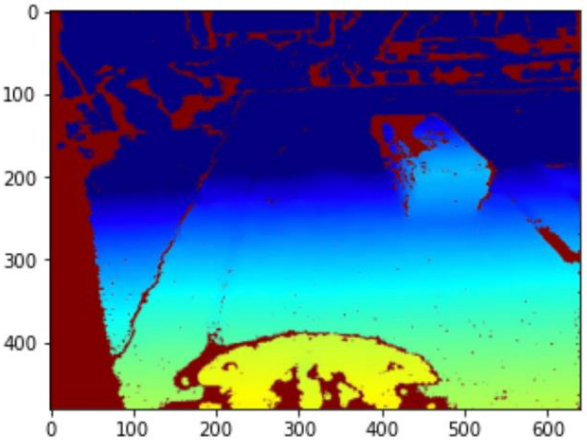


Figure-9: Stop labeled - Depth



All the frames were saved into a couple fundamental folders while driving the car. The image names and speeds were recorded into csv files for each folder. After deleting the faulty frames, all of the frames and csv files were merged individually.

4.2 Data augmentation:

After the raw data was gathered, data augmentation was done by flipping "right" and "left" labeled frames to increase the dataset even more and have a much more balanced one. Augmentation is represented below.

Figure-10: Left labeled (before flipping)

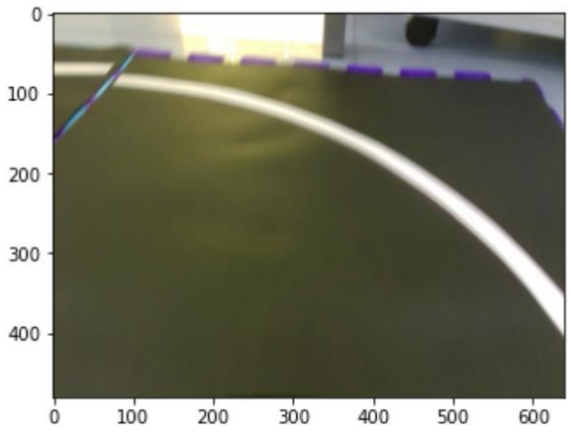
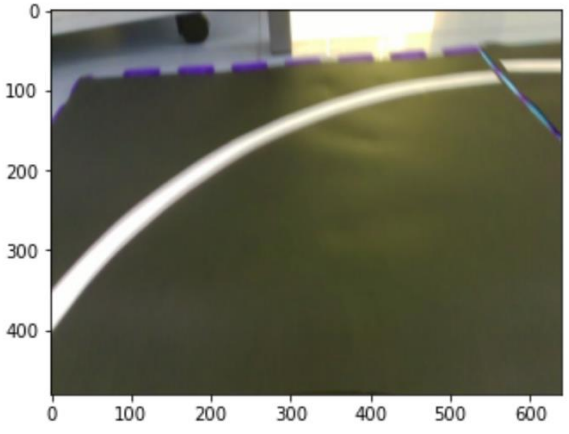


Figure-11: Right labeled (after flipping)



4.3 Preprocessing:

After the raw data was gathered, preprocessing was done. First of all, each image was resized to 64x64 from their original size 640x480. After rescaling, all images were converted to gray scale including the depth images since its coloring is artificially done and initially it is

in grayscale. RGB frames were also converted since our roads are black and white, and colour information does not add anything fruitful. Finally, the upper 14 pixel of images were cropped to prevent misleading information make the car deciding easier. Each step of preprocessing is given below.

Figure-12: Raw image

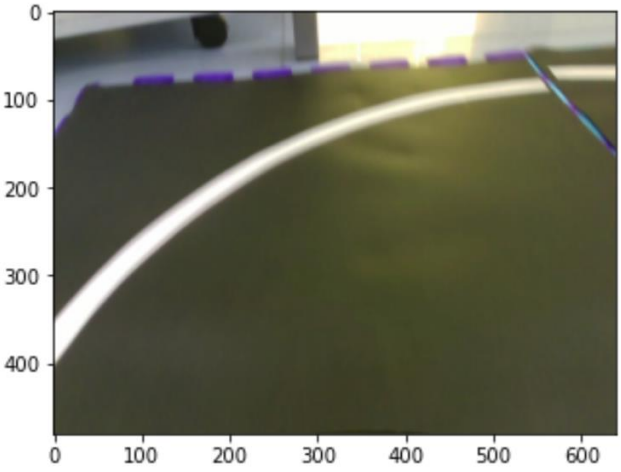


Figure-13: Rescaled image

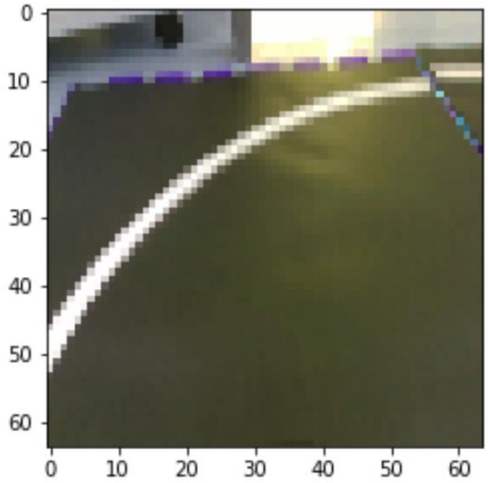


Figure-14: Gray scaled image

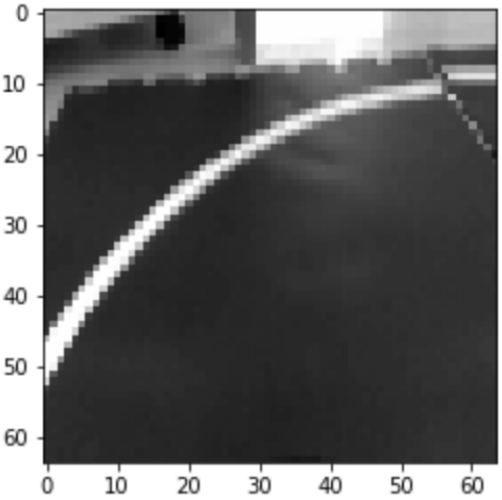
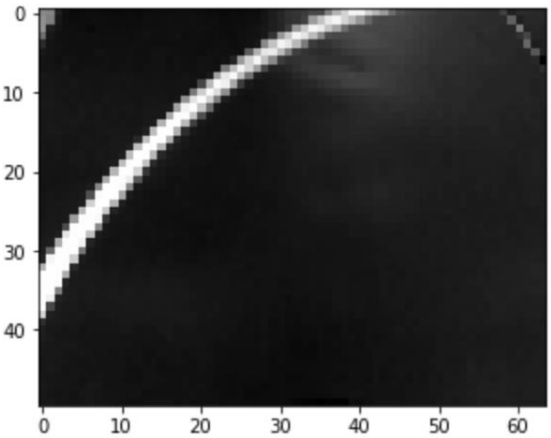


Figure-15: Cropped image



To create the labels, image names were used to match the images and wheel speeds from the csv file. Afterwards, each type of wheel speed was changed each one into a classification number, for example (0,0) speed was changed into the label 0. While a single model was used, the structure of the npy file was as follows:

Figure-16: Dataset Representation

0	RGB FRAMES	DEPTH	LABEL
1	[RGB image array]	[Depth Array]	Number from 0 to 4
…	…	…	…

However, when two model structure was getting worked on, raw dataset was split into two different npy files one for RGB and one for depth:

Figure-17: RGB Representation

0	RGB FRAMES	LABEL
1	[RGB image array]	Number from 0 to 3
…	…	…

Figure-18: Depth Representation

0	DEPTH FRAMES	LABEL
1	[Depth image array]	0 or 1
…	…	…

While creating these files, for RGB the (0,0) speed frames were excluded and the wheel speeds were changed into 0-1-2 for forward, right and left. For the depth, if the speed was 0, it was labeled as 0 i.e. stop, and otherwise, whether it was forward, right, or left, it was still labeled as 1, so go.

For the testing dataset a similar approach was followed. Different paths were created so that there is not a cross pollination between the train and test sets, and the results are not misleading. A couple examples of test paths is given below.

Figure-19: Testing path -1

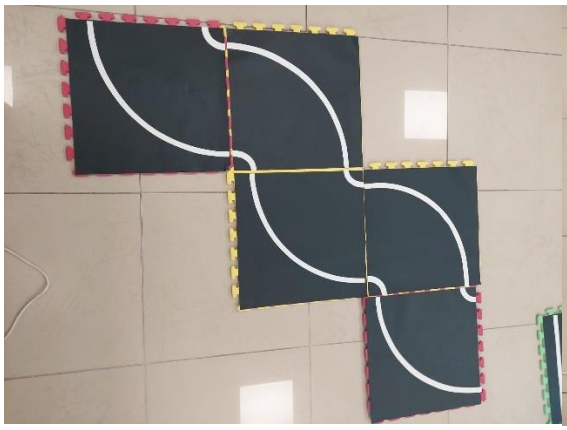


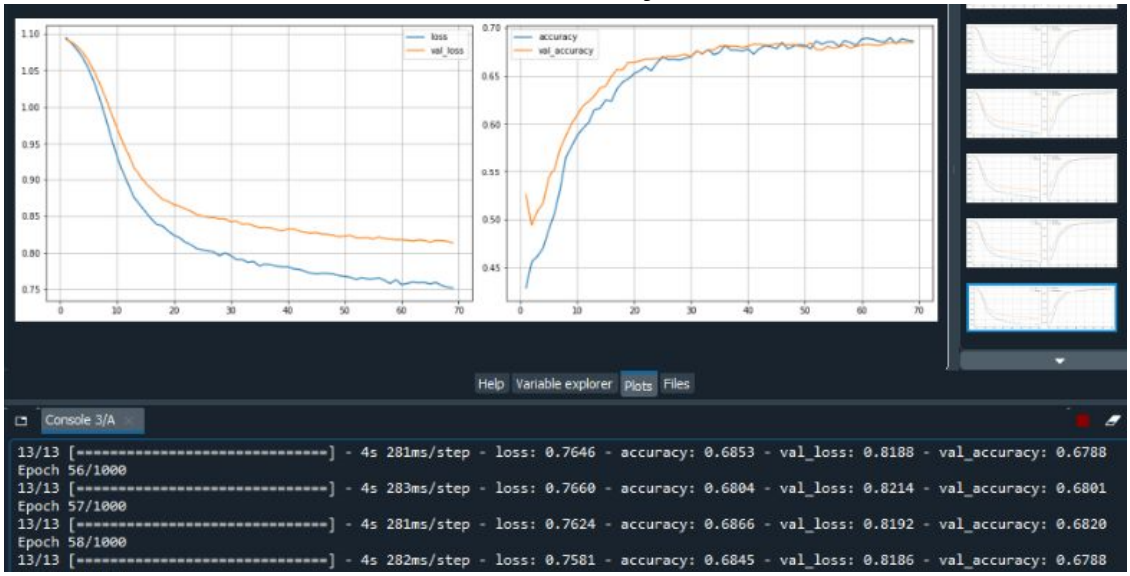
Figure-20: Testing path -2

After following the same process as the training preprocessing, in the end, two different npy files were obtained with the same structure as the train dataset for the two models and one npy file for the single models.

5) Building Neural Network

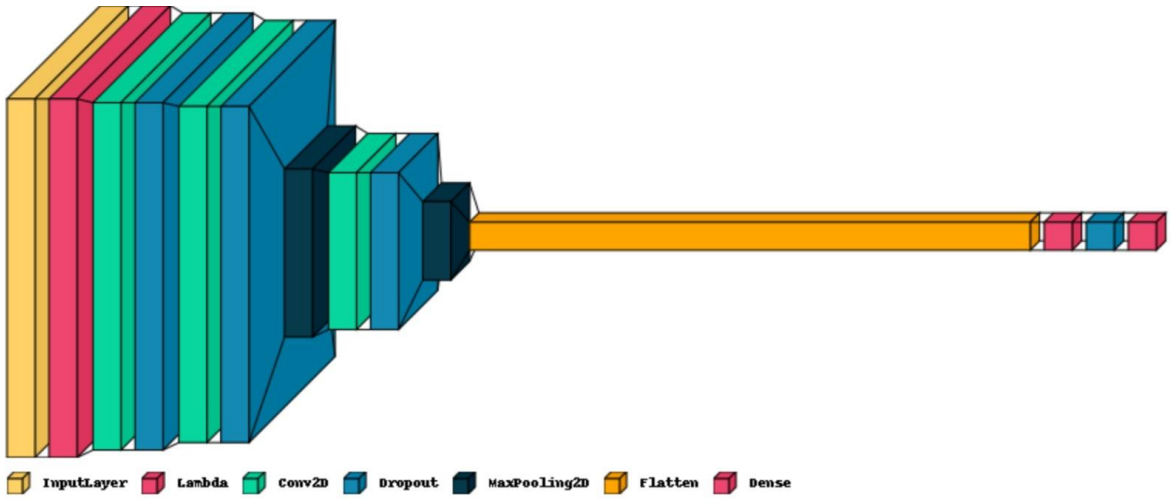
When the dataset with more than 50.000 frames was generated, next step of the project was to build the best performing neural network(s). The priority of the project team was to use their own unique model which was achieved at the end. Best performance was obtained by using 2 convolutional neural networks; one for classifying depth frames to decide on when to stop and when to go, one for classifying RGB frames to decide on where to go. All the models were built by using Keras library of Python programming language. The progression of the training was observed from either the CMD or the python console and plots sections. During an exemplary training as below, the epochs and the graphs would be simultaneously watched to see how the training was going.

Figure-21: method of training



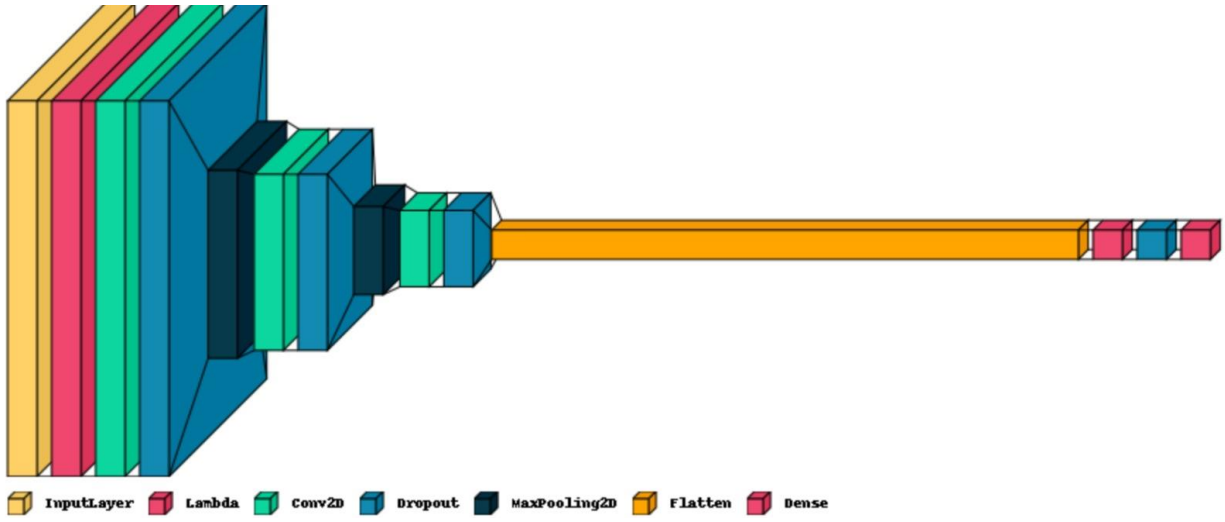
Depth model was trained by using only depth frames. Depth frames which were captured when there was no obstacle in front of the car belonged to class-0 which stands for "go", depth frames captured when there was an obstacle in front of the car belonged to class-1 which stands for "stop". This network consists of a normalization layer followed by 3 convolutional layers and 1 fully connected layer with 100 neurons. After each of these 4 main layers, a dropout layer with the dropout rate of 0.3 was added and after second and third convolutional layers 2D max pooling layers were added with pool size (2,2). Convolutional layers all have the kernel size of (3,3) and their filter numbers are 24, 32 and 64 respectively. Representation of the model is given below.

Figure-22: depth model photo



RGB model was trained by using only RGB frames which were captured when the car was being driven. There are 3 classes that this convolutional network tries to predict correctly: Class-0 which stands for "forward" with corresponding right and left motor speed values of (155,155), Class-1 which stands for "right" with corresponding right and left motor speed values of (-100,160), Class-2 which stands for "left" with corresponding right and left motor speed values of (160,-100). This network consists of a normalization layer followed by 3 convolutional layers and 1 fully connected layer with 100 neurons. After each of these 4 main layers, a dropout layer with the dropout rate of 0.3 was added and after first and second convolutional layers 2D max pooling layers were added with pool size (2,2). Convolutional layers all have the kernel size of (3,3) and their filter numbers are 32, 64 and 128 respectively. Representation of the model is given below.

Figure-23: RGB model photo



Both of these models were then converted to TensorFlow Lite (TFLite) models to have the light-weighted version of each model and to increase the input processing speed. TFLite is a framework that converts a TensorFlow model to an optimized version of itself in terms of speed and storage.

Logic behind this structure with 2 neural networks is as follows: When an RGB frame and its depth information was captured by the camera. First the depth information was processed by the first model explained above. If the depth information is classified as class-1 which means "stop", this means there is an obstacle encountered and both right and left motor speed values are set to 0 and the car stops. If depth information is classified as class-0 which means "go", corresponding RGB frame of this depth information become processed by the

second model explained above. If the RGB frame is classified as class-0, class-1 or class-2, the car moves forward, turns right or turns left, respectively, by setting the corresponding motor speed values of each class which are also mentioned above.

Before concluding on this 2 modeled structure. various approaches have been tried to solve the classification task with optimal results. First approach was trying to make the classification by using a single model. Project team worked on two variations of this approach. One was feeding the model with multiple input, RGB and depth frame, and obtaining a single output for prediction of all 4 classes. The other one was feeding the model with multiple input and obtaining multiple output, similar logic to the one with 2 modeled structures, but a merged variation of it. All these approaches performed poorly either in testing or inference. After these models MobileNet model have been tried by adjusting the frozen layer number over and over but desired outcomes have not been achieved even with this pre-trained model. As a result, project teamed decided on using the 2 modeled structured and obtained the results meeting success criteria. All performance rates of each tried model are given in part-10.

Another approach that was tried to improve the performance was using keras-tuner. Keras tuner is a library of Keras that is responsible for creating different models by testing different parameters randomly, and possibly leading to better model parameters without trying each parameter by hand. For this section, both different keras tuner modes and parametrizing different parts of the model were tested.

Three different keras tuner modes were tried: RandomSearch, Hyperband, and BayesianOptimization.

The RandomSearch tuner randomly selects hyperparameter combinations from a user defined search space. This is more useful than grid search that systematically chooses hyperparameter combinations since somewhere in between the grids a better value can be hidden. Therefore, grid search was eliminated, and the search started with randomsearch. [17]

The Hyperband tuner halves the models at each run step to efficiently allocate resources during the search. It trains multiple models simultaneously with different hyperparameter configurations, getting rid of poor performance models, and allocating more resources to better models. This type is better if there is limited time and resources. [17]

The Bayesian Optimization tuner uses Bayesian algorithms to intelligently explore the hyperparameter search space. It creates a probabilistic model of the objective function and uses it to find the most promising hyperparameter configurations for evaluation. This method is efficient since it builds up on previous information gathered. [17]

While testing with keras tuner, at different times different parameters were chosen to be searched upon. In the beginning, a couple parameters like the number of filters in the convolution layer and number of units in dense layers were chosen as the parameters. As the time went on, more parameters were added to explore upon. These tuners could even decide on adding how many convolution layers and max pooling layers, effectively building different types of models at each pass. They could even choose the kernel size, regularization parameters etc.

Neural neural networks have been tested in two ways using our test roads:

Quantitative Evaluation: In this method, theoretical performance has been evaluated. This is to check if the neural network is giving the correct output value when the RGB frames and depth information is given. This evaluation has been done by using matplotlib and scikit libraries for the confusion matrix and the model evaluation metrics respectively.

Qualitative Evaluation: In this method, practical performance has been evaluated. Number lane violations and obstacle crashes recorded by hand and percentage of car's ability to drive safe has been be calculated.

5. PROJECT TARGETS AND SUCCESS CRITERIA:

Work package 1 – Literature review

Success can be measured by reviewing 15 different articles. We successfully completed this section by investigating 15 articles.

Work package 2 – Hardware

For the successful completion of the hardware section, firstly, all the hardware modules have to be chosen to be compatible and placed on the skeleton of the car. The modules should be able to communicate in real time and work harmoniously for both the data collection section and the inference section. In addition, we should be able to drive the car manually for at least 5 consecutive laps. All of this was accomplished in the end by writing the correct codes and doing enough testing.

Work package 3 – Dataset Preparation

For this part, the collection of at least 20 thousand frames of RGB and depth images individually by driving the car manually is required. The frames should be preprocessed to be effective for training. We have done this by driving the car around the modular roads, recording the images, and preprocessing and data augmenting by our codes. In the end, we have around 50 thousand frames, more than the double of the success criteria preprocessed and fed to the model.

Work package 4 – Deep learning

Firstly, the classification accuracy of the deep learning model should be higher than 95% for the model to work properly on our modular roads. Accuracy is described as the ratio between the correctly labeled photos, the true positives and the true negatives (TP and TN) to the whole of the dataset which also includes false negatives and false positives. In short, accuracy is

$$\frac{TP + TN}{TP + TN + FP + FN}$$

A higher accuracy is needed in our case so that the correctly labeled photos are more abundant than the other ones which results in a better working model. In our case, the average accuracy was up to the standard due to correct training.

Confusion matrix is a type of table that has the actual labels in the left and the predicted ones on the top. This way the correctly labeled photos are in the diagonal, and it is possible to see how the model gets confused between different labels. When the majority of the results is in the diagonal, it is possible to say the model is efficient at correctly classifying images.

The line keeping rate measures how well the car is able to stay within the created road lines while driving. A pretty high line keeping rate is important for the car to not go out of bounds all the time which would create a non-safe environment and non-properly working output product. That's why the cut-off value for the average line keeping rate was agreed as 96%. This will be calculated by conducting 50 runs on different paths, recording any line violations that occur, and averaging the value.

Our next target is about the efficiency of the car. Car has to work in real time while operating deep learning algorithms on a Jetson Nano. The Jetson Nano's lower processing power and lower number of GPU cores compared to a regular computer GPU will handicap the complexity of the model. This will limit the model types that can be used in the car whether it

is a prewritten model or our own. It will be important to carefully consider the model's number of parameters, number of layers etc. and simplify and make the process more efficient if the processing gets alarmingly slow.

The avoidance rate of cars and pedestrians should be at the very least 98%. Since this rate is a measure of how well the model is able to avoid obstacles i.e. other cars and pedestrians while driving, a high avoidance rate is of importance to the safety of the vehicle, its passengers, and the pedestrians.

Work package 5 – Optimization

The car has to work on 25 FPS as mentioned and if the deep learning part is not enough to meet this criteria optimization part will come into play. For the optimization to be successful, TFLite will be added to the end of the model pipeline which makes the model more lightweight and the inference faster.

6. RISKS AND B PLANS:

Table 1. risks and b plans

Work Package #	Risk	B-Plan
WP 2	Electronic equipment broke down.	We stocked extra equipment and bought new ones when needed
WP 3	Insufficient dataset was encountered	We diversified the roads and added it to our dataset. Also, data augmentation was used.
WP 4	Model not achieving the expected performance	Our models achieved the expected performance.

7. WORK TIME PLAN OF THE PROJECT:

Literature Review work-package was done at the beginnings of each semester and through the end of the second semester, when it was time to work on deep learning models. Hardware work-package took much more time than expected due to failures of electronic equipment. This was the first expected risk of this project as it is mentioned in part-6. The biggest problem was regarding the nRF24L01 modules which were used to provide to

communication between remote controller and the car. nRF24L01 modules kept burning and causing inconsistencies and project team needed to stock new ones. Another problem occurred was regarding the powering of Jetson Nano computer. External batteries were not enough to power Jetson Nano to run a neural network while all the peripherals like Arduino Uno and Intel RealSense Camera were on. So, the project team decided to switch to Jetson Xavier platform, a more powerful platform and suitable to the powering way that was tried on Jetson Nano. So, all the setups were done on Jetson Xavier one more time. However, after a while a failure occurred on the ports of Jetson Xavier and it needed to be sent to warranty. As a result, project teams had to switch back to Jetson Nano platform and solved the powering problem by powering Arduino platform and Jetson Nano separately. This platform switching costed project team 3-4 weeks. Because of these problems, hardware work-package which was started at the 4th week of semester-1 was completed at the 10th week of semester-2. Dataset preparation work-package was done in 2 steps. First step took first 2 weeks of the semester-2 where project team construct the modular roads which is one of the primary originalities of the project. Second step which is data gathering by manually driving the car and preprocessing of the data was done when the Hardware work-package was completed and it also took 2 weeks. Deep learning work-package started at the 10th week of semester-2 and continued until the end of the project. This work-package was mostly done in parallel with the optimization work-package which was done in last 3 weeks of the project. This work time plan is shown in Table 1 and Table 2 at Part 16.

All team members shared the workload of each work package equally. Completion rate and measure of completion of each work-package is given below.

WP-1 Literature Review:
Completion Rate: 100%
Measure of Completion: Reviewing 15 articles and having enough knowledge of the works and approaches done before.

WP-2 Hardware:
Completion Rate: 100%
Measure of Completion: Being able to drive the car manually by obtaining RGB frames and depth information by the camera for at least 5 consecutive laps.

WP-3 Dataset Preparation:
Completion Rate: 100%
Measure of Completion: Obtaining a dataset consisting of at least 20 thousand frames and preprocessing it to make it ready to fit neural networks.

WP-4: Deep Learning:
Completion Rate: 100%
Measure of Completion: Making the car going on random roads and avoiding obstacles autonomously using Convolutional Neural Networks by achieving the proposed accuracy, lane keeping rate, avoidance rate and obtaining a confusion matrix with majority of the results in the diagonal.

WP-5 Optimization:
Completion Rate: 90%
Measure of Completion: Trying pre-trained models, increasing the speed of the car and being able to process 25 frames per second.

WP-5 is not 100% completed because the neural network is able to process 15 frames per second while the success criteria is 25. Reasons of this unmet success criteria is discussed at part-11.

8. DEMO PLAN:

The demo of our project will be using our physical car and self-generated roads. For the demo, the teachers will be given the chance to create a path using our modular roads in whichever way they desire. Next, the car will be tested on this road that it may or may not have encountered beforehand. This increases the chances to demonstrate that the car is not trained for a specific couple of roads and instead can work on any kind of path, proving its versatility and adaptability. It is possible to test the car on two different roads in the given demo time to increase the chances of unforeseen paths.

In addition to the testing of the car on the physical roads, there is also the possibility of demonstrating the test results on the computer test dataset. It will show that the car works sufficiently on all the paths that were created specifically for the test dataset. On top of that, it will give a better chance to see values of the accuracy, confusion matrix, average line keeping rate, and FPS.

9. FINANCIAL EVALUATION:

Predicted total cost of the project was 14.555 TRY. However, due to the need of new NRF modules to replace the burnt ones and extra material to design the road, required total cost increased to 15.070 TRY. Machine-Instrument and Material cost took 14.070 and 1000 TRY of the total, respectively. No cost was needed for People, Service and Travel categories. Inner distribution of Machine-Instrument and Material spending is given below.

Machine-Instrument (14.070 TRY)

Jetson Nano Development Kit: 5100 TRY
Intel RealSense Camera : 7500 TRY
Arduino Boards : 450 TRY
Voltage Regulator & Battery : 610 TRY
Remote Controller : 110 TRY
Wireless Connection Modules: 300 TRY

Materials to Design Road (1000 TRY)

Playing mats : 800 TRY
Cardboard & Tape: 200 TRY

Distribution of predicted and actual spendings, classified into 4 main categories, are also represented in Table 6 and 7, respectively.

10. RESULTS:

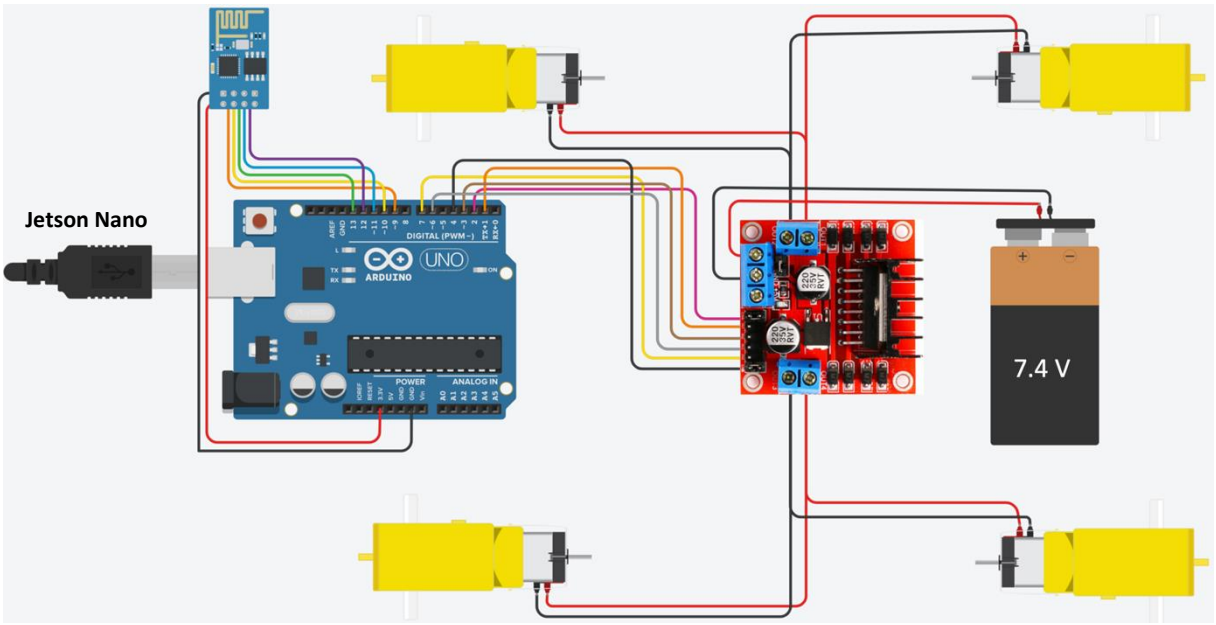
1. Literature review:

The results of it can be seen in the second section of the report. It was helpful in our journey while deciding on the path to take for our project.

2. Hardware:

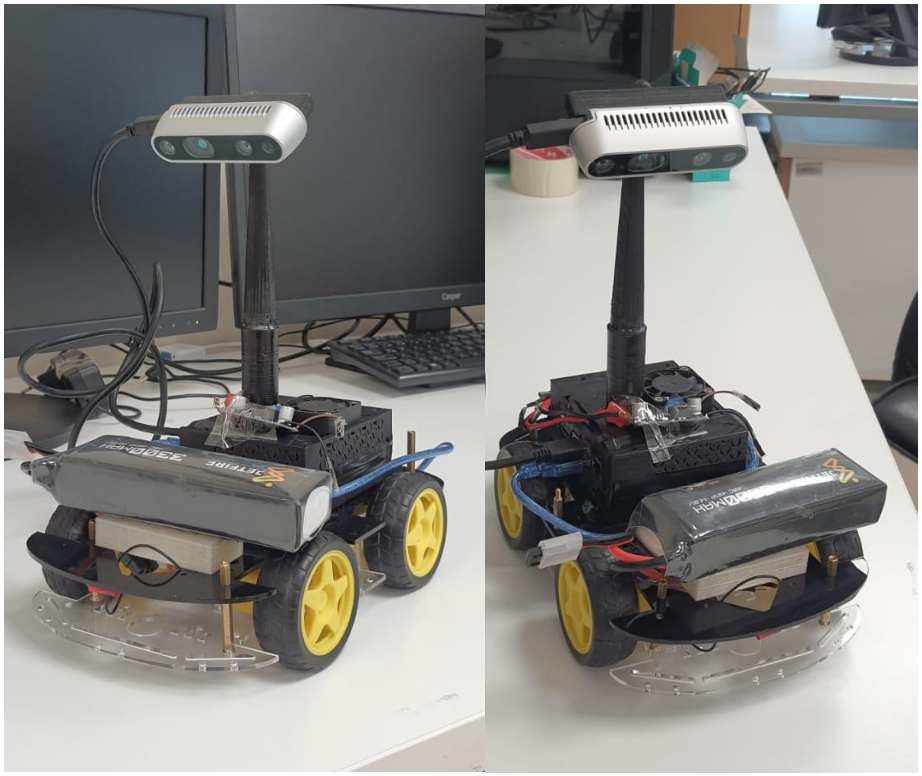
The connection scheme of our car can be seen in the figure below that lets the car work without any problems. When the dataset collection section is over, the remote connection module was removed.

Figure-24: Connection scheme



The final look of our car can be seen in the photos below.

Figure-25: final car photos



3. Dataset Preparation:

3.1 Train dataset

When the data gathering was completed the project team obtained a dataset as follows:

Table 2. number of frames from each class

Frame Label	Count (RGB - Depth Pair)
Forward (155, 155)	15.775
Right (-100, 160)	5.691
Left (160, -100)	8.626
Stop (0, 0)	9.302
TOTAL:	39.394

When the data augmentation was done by flipping the right and left images, the final dataset was obtained given in the following table:

Table 3. number of frames from each class after data augmentation

Frame Label	Count (RGB - Depth Pair)
Forward (155, 155)	15.775
Right (-100, 160)	14.317
Left (160, -100)	14.317
Stop (0, 0)	9.302
TOTAL:	53.711

3.2 Test dataset

For the test set a similar process was followed. Initial frame numbers are given below:

Table 4. test set raw data for RGB and depth

Frame Label	Count (RGB)
Forward (155, 155)	3781
Right (-100, 160)	1192
Left (160, -100)	1462
TOTAL:	6435

Frame Label	Count (Depth)
Others	6435
Stop (0, 0)	3202
TOTAL:	9637

After data augmentation, the number of frames were equalized across the board for easier to read testing results. Final dataset numbers can be seen below.

Table 5. test set finals

Frame Label	Count (RGB)
Forward (155, 155)	2400
Right (-100, 160)	2400
Left (160, -100)	2400
TOTAL:	7200

Frame Label	Count (Depth)
Left (160, -100)	3600
Stop (0, 0)	3600
TOTAL:	7200

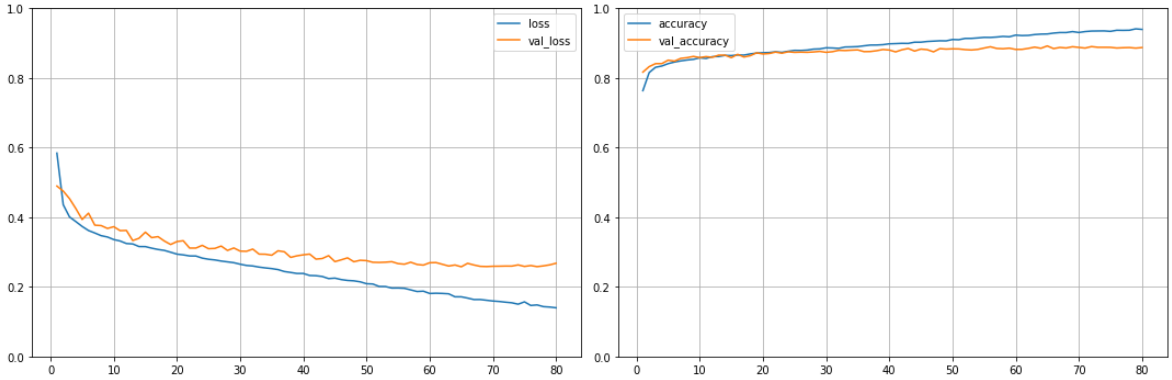
4. Deep Learning

Firstly, results for the best RGB and depth models will be discussed based on the training-validation losses and training-validation accuracies, evaluation results, and confusion matrices.

4.1 RGB:

For the trainings, generally, we had the early stopping mechanism that stopped the training whenever the val_loss did not decrease for a number of epochs. This prevented overfitting in the model as well as not wasting our resources. The training curve for the final RGB can be seen below. When the validation loss stopped decreasing and instead is having an upward curve with a value of 0.3 when the training is stopped as can be seen. Training loss is decreasing steadily and not plateauing meaning a good learning rate is chosen. Accuracy is increasing steadily albeit slowly until higher than 0.95 while validation accuracy is increasing even slower and stopping a little after 0.9, not reaching 0.95. Validation loss and accuracy is a little bumpy probably stemming from inconsistent dataset since when the car was nearing the road limiters, it had to be centered, hence right or left classifications on a straight road, and vice versa when turning.

Figure-26: the training- validation losses and training-validation accuracies



From the scikit summary below it can be seen that the testing accuracy is 0.92. The accuracy did not change much from the training and test meaning the model did not overfit to the train dataset and performs well with unseen data. Precision and recall are high for the right and left classes, i.e. 1 and 2 respectively, but as it has struggles with the forward class, its values are lower.

Figure-27: evaluation metrics

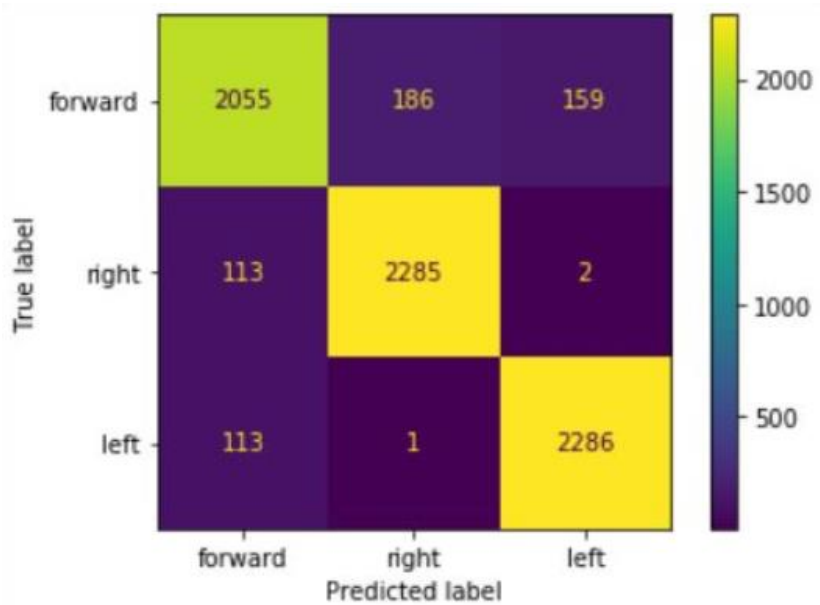
```
113/113 [==============================] - 10s 6ms/step
              precision    recall  f1-score   support

           0       0.90      0.86      0.88      2400
           1       0.92      0.95      0.94      2400
           2       0.93      0.95      0.94      2400

    accuracy                           0.92      7200
   macro avg       0.92      0.92      0.92      7200
weighted avg       0.92      0.92      0.92      7200
```

This forward struggle can also be seen on the confusion matrix. It demonstrates that while the majority of the elements are in the diagonal, more of the forward is mistaken for other classes while left and right are relatively way better classified.

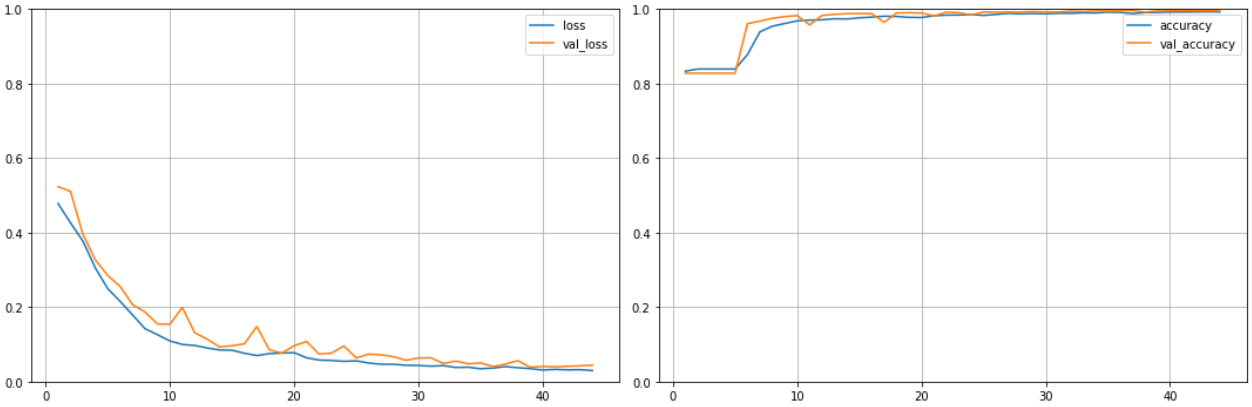
Figure-28: confusion matrix RGB



4.2 Depth

From the training, it can be seen that the general curve for the loss is good, therefore, a fitting learning rate was chosen. There are a couple validation bumps but overall it is decreasing meaning it did not overfit. Accuracy and validation acuracy are similar and start high and increase to be even higher.

Figure-29: the training- validation losses and training-validation accuracies for depth



As can be seen from the results, the accuracy, precision, recall of the model is all 1.00's meaning a nearly perfect model if not a perfect one. The confusion matrix shows that the model only confuses 8 stop frames out of 3600 for go, and only 2 go frames as stop. An overwhelming majority of the items are in the diagonal so depth part is achieved with flying colours.

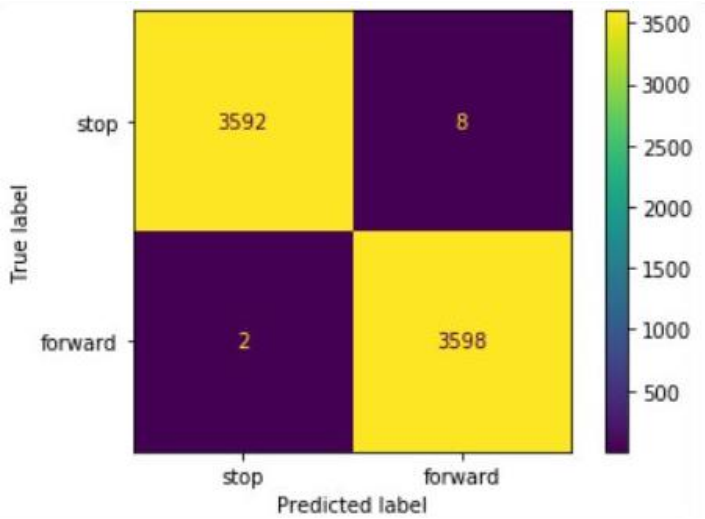
Figure-30: depth evaluation metrics

```
113/113 [==============================] - 1s 8ms/step
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      3600
           1       1.00      1.00      1.00      3600

    accuracy                           1.00      7200
   macro avg       1.00      1.00      1.00      7200
weighted avg       1.00      1.00      1.00      7200
```

Figure-31: depth confusion matrix



Overall, the average accuracy of both models is 0.96 owing it to

$$\frac{0.92 + 1{,}00}{2} = 0.96$$

4.3 Other Trials

Table 6. Some other trials

Model type	Data size	Conv number & filters	Learning rate	Batch size	Crop	Dropout	Regularization	Accuracy
MIOO*	128	2/32,64	0.0002	32				0.75
MIOO	64	3/32,64,64	0.0002	128		✓		0.78
MIOO	32	4/24,32,64,128	0.0002	128		✓		0.77
MIOO	64	3/32,64,128	0.0003	64		✓	✓	0.83
MIMO*	128	2/32,64	0.0003	32				0.77
MIMO	32	3/24,64,256	0.0001	64		✓		0.78
MIMO	64	3/32,64,128	0.0004	64	✓	✓		0.85
RGB	64	4/24,32,64,256	0.0002	128	✓	✓	✓	0.82
RGB	64	3/24,32,512	0.0002	128	✓	✓	✓	0.82
RGB	32	3/24,32,64	0.0002	1024		✓	✓	0.89
RGB	64	4/24,32,64,128	0.0002	512	✓	✓	✓	0.90
DEPTH	32	3/32,64,128	0.0001	1024		✓	✓	0.97
DEPTH	32	3/32,64,128	0.0001	1024	✓	✓	✓	0.92
DEPTH	64	3/32,64,128	0.0004	512		✓	✓	0.95
DEPTH	64	3/32,64,128	0.0004	512	✓	✓	✓	0.98

MIOO = multiple input one output
MIMO = multiple input multiple output

For one output, a medium size with a higher than others gave a better result. Higher size performed badly since it was learning more and higher level features that would cause it to learn slower and perform worse in a basic classification. It might also be the lower batch size. For multiple output, better results were obtained with crop and more basic network. From then on with some exceptions, similar things were tested. Results in RGB also confirmed that less complex models performed better with crops, dropout, and regularization added. With depth, the crop helped with the outcome with 64, but negatively affected the 32 since it was already pretty small.

It was seen as the experiments progressed that the more separate the two parts for the depth and the RGB was, the higher the accuracy tended to be. This is why it was fully separated in the end. This could be caused by the depth and RGB features not being compatible.

4.4 Keras tuner trials

Figure-32: randomsearch

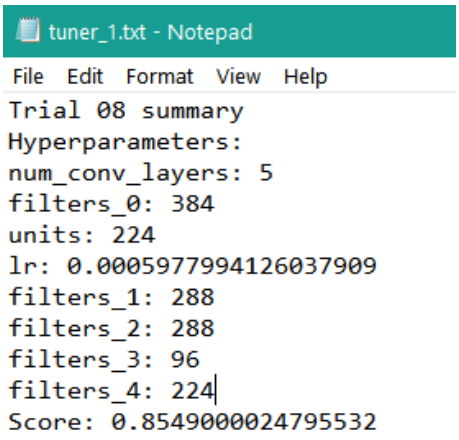


Figure-33. Best result from hyperband:

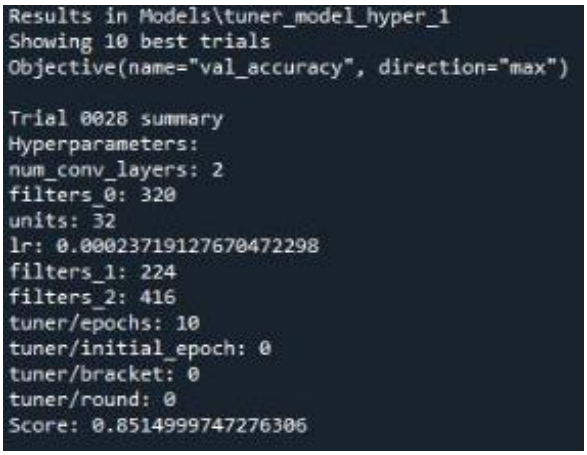
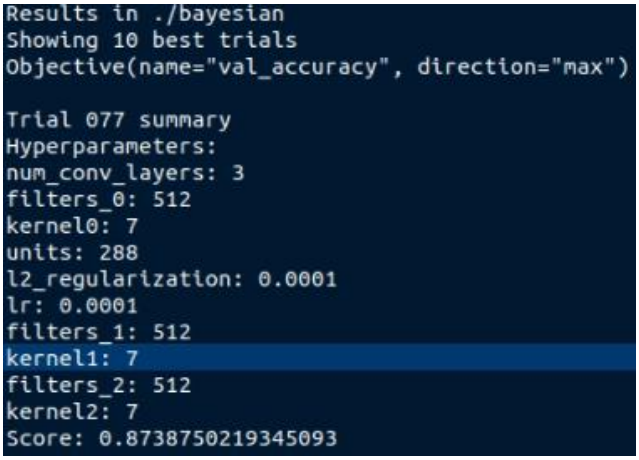


Figure-34: best result from bayesianoptimization

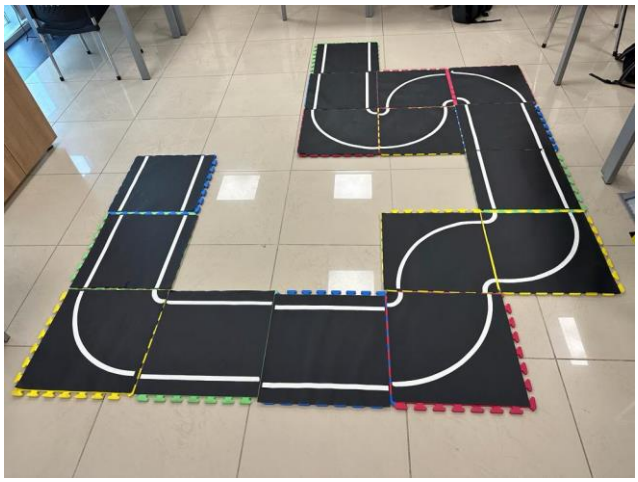


As can be seen from the results, the best performing tuner was Bayesian with 0.88 validation accuracy. This can be attributed to it searching based on previous results. RandomSearch was able to accomplish 0.854 while hyperband with more trials only obtained 0.851. Overall, while these numbers gave us some ideas in the beginning, maybe because they were suited to shorter trainings -since the tuners only trained for a small number of epochs to test faster- when we tried them for full length trainings, they did not fare better than something we wrote.

4.5 Inference

For the inference, some roads the car has never rode upon were built. One example is seen below. The final test for our models was the car driving on this road without going out of bounds, and stopping when obstacles were put randomly on different sections.

Figure-35: Inference path 1



On these paths, which included some loops also, the car autonomously drove around. For the paths that did not include loops, when the car reached one end, its direction was reversed for it to parse the road in the opposite direction. Changing the direction was also done for the loop paths half of the time for a fair comparison of going into both directions. For the line violation rate, it was observed for a total of 50 loops, and it was seen that it stayed inside the bounds of the white lines 100 percent of the time. Afterwards, a couple road blocks were put on its course 50 times inside the bounds, and its rate to stop was investigated. In the end, it was found as 98% since it missed one of it that was probably placed too close to the outer bounds when the car was going in the opposite side of the line.

5. Optimization

5.1 Mobilenet

Pretrained model MobileNet was used in an attempt to obtain a better model. The results of our trials can be seen in the table below:

Unfrozen layers	Data size	Dense layer & unit	Learning rate	Batch size	Crop	Regularization	Accuracy
4	128	3/100,50,10	0.0001	32			0.74
4	64	3/100,50,10	0.0001	32			0.76
2	64	3/100,50,10	0.0002	64			0.79
0	64	2/100,10	0.0004	128			0.78
0	32	1/200	0.0002	128			0.79
0	32	1/100	0.0003	256		✓	0.80
0	64	1/100	0.0003	128		✓	0.83
0	64	1/100	0.0002	256	✓	✓	0.85

It was seen that the model performed better with smaller data size from the initial tests. Then it was observed that less unfreezing resulted in better results. Which caused the later trials to be done with all layers frozen, and only working on the output feature map of the pretrained model. Then it was seen that less dense layers resulted in better outcomes. Afterwards, it was seen that with everything else the same, 64 size performed better than 32. Finally, a higher batch size and a lower learning rate concluded our trials with way lower accuracy at 0.85 compared to our models.

It was decided that wasting anymore time on Mobilenet or other pretrained models was not productive, but since keras tuner did not take individual effort, it was also tested on it to see if it was a case of choosing wrong hyperparameters by us. Since the best results were obtained without any unfrozen layers, the tuner was tested with differing number of dense units, and learning rates. Two best models out of all the tests can be seen below.

Figure-36: best result from mobilenet tuner

```
Results summary
Results in Models\tuner_mobile
Showing 10 best trials
Objective(name="val_accuracy", direction="max")

Trial 2 summary
Hyperparameters:
units: 64
lr: 0.0034599236521799413
Score: 0.5671499967575073

Trial 1 summary
Hyperparameters:
units: 512
lr: 0.0011500145713744404
Score: 0.5658999979496002
```

It was not comparable to the values achieved by our models. With the best score being 0.57, so research on pretrained models was promptly dropped. In our opinion, the pretrained model did not result in the same level as our model due to it being more complex than what our deep learning project needed.

5.2 TFLite

TFLite made our model work faster with FPS increasing from 9-10 band to 14-15 which corresponds to around a 50% increase.

11. DISCUSSION:

This project was based on the idea of developing a final product carried out by implementing latest software technologies to hardware. In this sense, an autonomous car which is trained by self-captured videos was developed. Overall progression of the project to satisfy the success criteria can be discussed in 2 parts: Hardware and Software

Hardware applications of the project started with the efforts to mount the modules to build the car, connecting Intel RealSense camera and to provide the connection between remote controller and the car itself. The goal was to be able to drive the car for at least 5 consecutive laps and make it ready to collect enough data which are two of the success criteria of the project. These criteria were met by successfully accomplishing methods 1-2-3 and 4.1 stated in Part-4. Mounting the modules and connecting electronic equipment needed less effort than the other methodologies as some of them were already ready. Project team designed and printed some 3d models to protect the equipment, provide durability and place each equipment to correct spots. Connections of Jetson Nano and Arduino Uno has been done. Then, the Intel RealSense camera has been connected and been ready to gather data. To achieve the connection between the car and the remote controller, nRF24L01s modules were preferred. As stated in the former parts, these modules caused so many problems due to their nonpersistent behavior. Problem has been solved only by stocking extra equipment but unfortunately it costed weeks to the project team. After the car was ready to be driven, modular roads which is one of the major originality of the project were constructed. The car was driven on randomly constructed paths and labelled RGB and depth frames were collected. Stated two success criteria were met by these methods.

Software applications include data pre-processing, building the neural network and optimization. The aims of these applications were reaching a classification accuracy of 95%, obtaining the desired confusion matrix, building a neural network that can process 25 frames per second, having the car going with 95% lane keeping rate and 98% obstacle avoidance rate

which are the other success criteria of the project. These criteria were met by the methods 4.2-4.3 and 5 except 25 FPS processing rate. Gathered data was augmented and pre-processed by using various techniques. Flipping was used to increase the number of right and left labeled frames. After that; rescaling, gray scaling and cropping was applied to have much more meaningful data. When the data was ready to fit to the model, project team constructed various neural networks and decided on using two separate models, one for processing RGB frames and one for processing depth frames. These model were the converted to TFLite models to optimize their speed and make them light-weighted. Obtained accuracy from these models were 0.92 and 1.00, respectively. These values gave an average accuracy of 0.96 and both models' confusion matrix were pretty successful. By these results, theoretical success criteria of software applications, at least 0.95 accuracy and confusion matrix with majority of instances in the diagonal, were met. These models were also enough to meet the criteria of at least 95% lane keeping rate and 98% obstacle avoidance rate. These were tested by practical evaluations which are letting the car performing its skills and counting the desired an undesired behaviors. However, one success criteria which is 25 FPS input processing was not met. Reason of this unmet criteria was regarding the powering of Jetson Nano. Jetson Nano computer has two powering modes: MAX(10W) and 5W. The desired FPS rate was only reachable by MAX powering mode. However, external batteries were not enough to boot up Jetson Nano on MAX powering mode with all the peripherals on. So, project team needed to switch to 5W powering mode to be able to use the computer and process the data in real-time. But the best reachable FPS rate with 5W powering was 15.

As a result, it can be seen that followed methodologies were pretty successful as the project team achieved almost every success criteria. Although so many problems were encountered, project team dealt with all of them and develop the desired final product. The project is successfully completed with great effort of the team members.

12. CONCLUSION:

Our project aimed to develop an autonomous car by using advanced artificial intelligence technologies and training it with self-collected data. The car uses the Intel Realsense camera for depth and color information as input to deep learning models to make autonomous decisions on the road. Modular roads were used for training and testing, with custom models created for this purpose.

The project consisted of five work packages, consisting of literature review, designing and assembling the car, creating the dataset, developing deep learning models, and optimizing their speed and accuracy. The first four packages were completed successfully. Reviewing 15 articles were enough for the first package. For the hardware section, we created a car that can

work with a remote controller and without it for the inference that can loop around the paths consecutively despite some hiccups along the way. Data collection part was finished swiftly by collecting around 50 thousand frames after creating the puzzle like structure of the road and preparing the proper hardware and software for it. Our dataset is more diverse than most other research mentioned here owing it to our modular roads. Deep learning part package was achieved by obtaining the expected accuracy, by having the most of the items in the diagonal of the confusion matrix, by working in real-time, by having an avoidance rate of more than 98%, and line keeping rate of 100%. Optimization part was not accomplished completely since the FPS is not 25 as intended even with using TFLite, but the car can drive around without any problems.

Despite encountering hardware issues along the way, such as the remote connection module and Jetson Nano/Xavier indecision, the project was successfully finished on time due to the team's perseverance and better knowledge on the software sections. Thanks to our efforts, we now have a car in our hands that possesses the ability to autonomously make decisions regarding its movement, including the choice of whether to move or not and the timing of its movements.

13. FUTURE STUDIES:

It is planned to continue this project to make the car able to bypass the small obstacles which are not covering the road completely. By this functionality the car will not stop for every obstacle it sees and it makes the project much more relatable to real-world scenarios. Also, object detection can be added to the project to make the car behave differently to various obstacles. Making the car more cautious to pedestrian-like objects will be an important development to advance the project.

It is also possible to try to improve the project or change some parts to make it more applicable to more diverse situations by trying the below:

- Trying single-model multiple output model type
- Trying TensorRT for even faster inference
- Data collection on other types of roads with different characteristics

14. ASSESSMENT OF ENGINEERING COURSES:

Courses that helped or will help us during project is as follows:

Introduction to CoE&EEE: Building the car required Arduino and electronics knowledge. Thanks to our past experience in this course we knew the methods to build a car and use Arduino Boards. In the course project we experienced these concepts which helped us to finish this graduation project in an effective way.

Intro to Machine Learning and Intro to Deep Learning: Since we went through advanced concepts of machine learning and deep learning algorithms in these two courses, it was easy to make progress in the artificial intelligence part of the project. We had high-level knowledge on artificial intelligence thanks to these courses.

Intro to Computer Vision: In this project we dealt with visual data and its processing steps which are the concepts that are included in this course's syllabus. This course helped us to predict the possible challenges and the ways to handle them.

Advanced Programming: In the project we used Python programming language for building neural networks and configuring the camera. This course which helped us to improve our Python skills eased our jobs critically.

15. REFERENCES:

[1] Magnusson, F. (2018). Evaluating Deep Learning Algorithms for Steering an Autonomous Vehicle.

[2] Do, T. D., Duong, M. T., Dang, Q. V., & Le, M. H. 2018. "Real-time self-driving car navigation using deep neural network". In 2018 4th International Conference on Green Technology and Sustainable Development (GTSD) (pp. 7-12). IEEE.

[3] Viswanath, P., Nagori, S., Mody, M., Mathew, M., & Swami, P. 2018. "End to end learning based self-driving using JacintoNet". In 2018 IEEE 8th International Conference on Consumer Electronics-Berlin (ICCE-Berlin) (pp. 1-4). IEEE.

[4] Pannu, G. S., Ansari, M. D., & Gupta, P. 2015. "Design and implementation of autonomous car using Raspberry Pi". International Journal of Computer Applications, *113*(9)

[5] Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., & Zieba, K. 2016. "End to end learning for self-driving cars". Retrieved January 5, 2023, from *https://arxiv.org/pdf/1604.07316.pdf*

[6] Vijitkunsawat, W., & Chantngarm, P. 2020. "Comparison of machine learning algorithm's on self-driving car navigation using Nvidia Jetson Nano". In 2020 17th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON) (pp. 201-204). IEEE.

[7] Dangskul, W., Phattaravatin, K., Rattanaporn, K., & Kidjaidure, Y. 2021. "Real-Time Control Using Convolution Neural Network for Self-Driving Cars". In 2021 7th International Conference on Engineering, Applied Sciences and Technology (ICEAST) (pp. 125-128). IEEE.

[8] Agnihotri, A., Saraf, P., & Bapnad, K. R. 2019. "A convolutional neural network approach towards self-driving cars". In 2019 IEEE 16th India Council International Conference (INDICON) (pp. 1-4). IEEE.

[9] Al-Qizwini, M., Barjasteh, I., Al-Qassab, H. 2017. "Deep learning algorithm for autonomous driving using GoogLeNet". Retrieved January 4, 2023, from https://ieeexplore.ieee.org/abstract/document/7995703

[10] Darapaneni, N., R, P. R., Paduri, A. R. 2021. "Autonomous Car Driving Using Deep Learning". Retrieved January 5, 2023, from https://ieeexplore.ieee.org/abstract/document/9478090

[11] Sonata, I., Heryadi, Y., Lukas, L., Wibowo, A. 2020. "Autonomous car using CNN deep learning algorithm". Retrieved January 5, 2023, from https://iopscience.iop.org/article/10.1088/1742-6596/1869/1/012071/pdf

[12] Bechtel, M. G., Mcellhiney, E., Kim, M., Yun, H. 2019. "DeepPicar: A Low-Cost Deep Neural Network-Based Autonomous Car". Retrieved January 5, 2023, from https://ieeexplore.ieee.org/abstract/document/8607229

[13] Xu, H., Gao, Y., Yu, F., Darrell, T. 2017. "End-to-end Learning of Driving Models from Large-scale Video Datasets". Retrieved January 6, 2023, from https://arxiv.org/pdf/1612.01079v2.pdf

[14] Dong, D., Li, X., Sun, X. 2019. "Deep Learning Techniques for Obstacle Detection and Avoidance in Driverless Cars". Retrieved January 6, 2023, from https://ieeexplore.ieee.org/abstract/document/9073155

[15] Hu, Y., Shum, H. P. H., Ho, E. S. L. 2021. "Multi-task deep learning with optical flow features for self-driving cars". Retrieved January 7, 2023, from https://ietresearch.onlinelibrary.wiley.com/doi/full/10.1049/iet-its.2020.0439

[16] Hong, L. H. 2020. "How to install librealsense and pyrealsense2 on the Jetson NX". Retrieved January 3, 2023, from https://lieuzhenghong.com/how_to_install_librealsense_on_the_jetson_nx/

[17] "Getting started with KerasTuner". Retrieved January 3, 2023, from https://keras.io/guides/keras_tuner/getting_started/

16. PROJECT ACTIVITIES AND WORK PLAN

For this part, we want to see how you distribute the work items into weeks and who in your team will perform these work items. A better plan should distribute work uniformly over the weeks and project team members. Also, make sure you finish some work items during the planning so that you assess if the project is going well or not. The tables after the one below targets this requirement.

Table 1 *The Work-Activity Plan for Project 1*

Work and Activity Project 1	**Responsible Group Member**	**Timeline**													
		1. week	2. week	3. week	4. week	5. week	6. week	7. week	8. week	9. week	10. week	11. week	12. week	13. week	14. week
1. Literature Review	Taha - Büşra														
2. Hardware	Taha - Büşra														

Table 2 *The Work-Activity Plan for Project 2*

Work and Activity Project 2	**Responsible Group Member**	**Timeline**													
		1. week	2. week	3. week	4. week	5. week	6. week	7. week	8. week	9. week	10. week	11. week	12. week	13. week	14. week
1. Literature Review	Taha - Büşra														
2. Hardware	Taha - Büşra														
3. Dataset Preparation	Taha - Büşra														
4. Deep Learning	Taha - Büşra														
5. Optimization	Taha - Büşra														

15.1 LIST OF WORK PACKAGES

Table 3 *Detailed Definition of Work and Activity*

WP No	Detailed Definition of Work and Activity
1	Literature survey for having desired knowledge by seeing similar projects and publications
2	Building the car completely. (Module connection, remote controller, camera connection, 3d printing for car design)
3	Creating the roads, driving the car for gathering data, data preprocessing.
4	Creating unique model, training and testing, inference.
5	Trying existing models, fine tuning, increasing the car speed, trying TFLite, Keras-tuner.

Table 4 *Work package targets, their assessment, and the contribution of each work package to the overall project success.*

Work package	Target	Measurable outcome	Contribution to overall success(%)
1	Examining various examples of previous works and having knowledge of various methods and processes.	Reviewing 15 articles.	10
2	Building the car by having all desired functionalities available.	Being able to drive the car manually by obtaining RGB frames and depth information by the camera.	20
3	Designing the roads to drive the car on. Gathering the data and preprocessing it to fit it into neural network	Obtaining the dataset consisting of RGB frames and their depth information which are labelled by remote controller commands.	20
4	Building a neural network and completing training and testing.	Having the driving itself without lane violation and obstacle crashes.	35
5	Optimization	Faster inference	15
			Total:100

Table 5 *The work package distribution to project team members*

WORK PACKAGE DISTRIBUTION					
Project Member	WP1	WP2	WP3	WP4	WP5
Fatma Büşra Yaman	50	50	50	50	50
Taha Yunus Hamamcıoğlu	50	50	50	50	50
Total	100%	100%	100%	100%	100%

17. BUDGET

Table 6 *Proposed Budget in TL*

	ITEMS				
	PEOPLE	MACHINE-INSTRUMENT	MATERIALS	SERVICE	TRAVEL
IMU FUND	0	13.800	800	0	0
SPONSOR COMPANY FUND	-	-	-	-	-
TOTAL	0	13.800	800	0	0

Table 7 *Actual Budget in TL (what you spent indeed)*

	ITEMS				
	PEOPLE	MACHINE-INSTRUMENT*	MATERIALS*	SERVICE	TRAVEL
IMU FUND	0	14.070	1000	0	0
SPONSOR COMPANY FUND	-	-	-	-	-
TOTAL	0	14.070	1000	0	0

*Provide proforma invoice for machines and materials to be purchased.
*Provide technical specifications for machines and services to be purchased.
*Make a contract for services if necessary

18. CURRICULUM VITAE

TAHA YUNUS HAMAMCIOĞLU

Istanbul-Turkey • +90 530 340 12 40
hamamcioglutaha@gmail.com · linkedin.com/in/tahayunus/

PROFILE

I am a senior Computer Engineering student at Istanbul Medipol University with a CGPA of 3,42 and an excellent command of English. I enrolled at my university by earning 100% scholarship and I am currently in the 4th place in my department. I am highly skilled in Python, Java, C/C++ and JavaScript programming languages. I have a great passion for **Artificial Intelligence** with comprehensive knowledge of the field concepts and related technologies. I am experienced in building complex neural networks.

EXPERIENCE

JULY – AUG 2022
ARTIFICIAL INTELLIGENCE INTERN, TÜBİTAK
Kocaeli, Turkey

Hands-on experience on image processing and deep learning models; particularly on Convolutional Neural Networks. Specialized in TensorFlow, Keras, scikit-learn and OpenCV libraries of Python.

SEPT – OCT 2021
DATA SCIENCE INTERN, TÜBİTAK
Kocaeli, Turkey

Project data analysis and visualization on the front-end of the web page built by using Angular Framework. Specialized in NumPy, Pandas, Plotly, Matplotlib and seaborn libraries of Python.

EDUCATION

2019 - 2023 (June)
COMPUTER ENGINEERING, ISTANBUL MEDİPOL UNIVERSITY
CGPA: 3.42
100% Scholarship
4th place in department

2014 - 2018
SCIENCE HIGH SCHOOL, IZMIR ÇAĞDAŞ EĞİTİM COLLEGE
100% Scholarship

SOFTWARE SKILLS

- **PYTHON** (+3 years)
- **JAVA** (+2 years)
- **C/C++** (+1 year)
- **JAVASCRIPT** (+1 year)
- **Deep Learning:** Specialized in Convolutional Neural Networks
- **Machine Learning:** Supervised and unsupervised Learning
- **Artificial Neural Networks:** Multilayer perceptrons
- **Data Science:** Data analysis, visualization and interpretation

Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

Fatma Büşra Yaman

fatma.yaman@std.medipol.edu.tr · +90 538 769 4738 · 28.10.2000

LinkedIn: https://www.linkedin.com/in/fatma-busra-yaman-9182a01a9/

İstanbul,Turkey

EDUCATION

- **Istanbul Medipol University**
  Computer Engineering Department, 4th year
  Current GPA: 3.79, Ranked first in department every year
  2019-ongoing
- **IBDP (International Baccalaurate Diploma Programme)**
  Graduation grade: 36/45
  2017-2019
- **Cambridge IGCSE**
  Average Grade: A^+
  2015-2017
- **Kartal Anatolian Imam Hatip High School**
  Graduation grade: 96.94
  2014-2019

INTERNSHIPS

- **2021-TÜBİTAK Information Technologies Lab**
  Worked on an anti face spoofing model to prevent print and replay attacks to enter restricted areas
  Deep learning
- **2022-HAVELSAN Video Analysis Department**
  Worked on state of the art object tracking models
  Deep learning

SKILLS

- **Deep Learning**
  PyTorch, Tensorflow OD API, Keras, Tensorflow, YOLOv4 and YOLOv5,; Trainings over GPU server, Google Colab, and Docker
- **Machine Learning**
  Scikit-learn
- **Computer Vision**
  OpenCV in C++ and Python
- **Programming Languages**
  C/C++, Python, Java, C#
- **Embedded Software**
  Verilog, STM32F407vg - Discovery