THE NAVIGATION OF AN AUTONOMOUS CAR USING DEEP CONVOLUTIONAL NEURAL NETWORK AND LIDAR MEASUREMENTS

A MASTER THESIS SUBMITTED TO THE
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCE
OF
KOCAELI UNIVERSITY

BY

AZEEZ FOLAYEMI ADEBAYO

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
MECHATRONICS ENGINEERING

KOCAELİ 2022

KOCAELI UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCE

MECHATRONICS ENGINEERING

MASTER THESIS

THE NAVIGATION OF AN AUTONOMOUS CAR USING DEEP CONVOLUTIONAL NEURAL NETWORK AND LIDAR MEASUREMENTS

AZEEZ FOLAYEMI ADEBAYO

Prof. Dr. Hüseyin Metin ERTUNÇ **Supervisor, Kocaeli University**	**............................**
Prof. Dr. Hasan OCAK **Jury member, Kocaeli University**	**............................**
Prof. Dr. Raşit KÖKER **Jury member, Sakarya University of Applied Science**	**............................**

Thesis Defense Date: 25.05.2022

ETIK BEYAN VE ARASTIRMA FONU DESTEGI

Kocaeli Üniversitesi Fen Bilimleri Enstitüsü tez yazım kurallarına uygun olarak hazırladığım bu tez/proje çalışmasında,

- Bu tezin/projenin bana ait, özgün bir çalışma olduğunu,
- Çalışmamın hazırlık, veri toplama, analiz ve bilgilerin sunumu olmak üzere tüm aşamalarında bilimsel etik ilke ve kurallara uygun davrandığımı,
- Bu çalışma kapsamında elde edilen tüm veri ve bilgiler için kaynak gösterdiğimi ve bu kaynaklara kaynakçada yer verdiğimi,
- Bu çalışmanın Kocaeli Üniversitesi'nin abone olduğu intihal yazılım programı kullanılarak Fen Bilimleri Enstitüsü'nün belirlemiş olduğu ölçütlere uygun olduğunu,
- Kullanılan verilerde herhangi bir tahrifat yapmadığımı,
- Tezin/Projenin herhangi bir bölümünü bu üniversite veya başka bir üniversitede başka bir tez/proje çalışması olarak sunmadığımı,

beyan ederim.

☑ Bu tez/proje çalışmasının herhangi bir aşaması hiçbir kurum/kuruluş tarafından maddi/alt yapı desteği ile desteklenmemiştir.

☐ Bu tez/proje çalışması kapsamında üretilen veri ve bilgiler .... tarafından .... no'lu proje kapsamında maddi/alt yapı desteği alınarak gerçekleştirilmiştir.

Herhangi bir zamanda, çalışmamla ilgili yaptığım bu beyana aykırı bir durumun saptanması durumunda, ortaya çıkacak tüm ahlaki ve hukuki sonuçları kabul ettiğimi bildiririm.

Azeez Adebayo

YAYIMLAMA VE FIKRI MÜLKIYET HAKLARI

Fen Bilimleri Enstitüsü tarafından onaylanan lisansüstü tezimin/projemin tamamını veya herhangi bir kısmını, basılı ve elektronik formatta arşivleme ve aşağıda belirtilen koşullarla kullanıma açma izninin Kocaeli Üniversitesi'ne verdiğimi beyan ederim. Bu izinle Üniversiteye verilen kullanım hakları dışındaki tüm fikri mülkiyet haklarım bende kalacak, tezimin/projemin tamamının ya da bir bölümünün gelecekteki çalışmalarda (makale, kitap, lisans ve patent vb.) kullanımı bana ait olacaktır.

Tezin/projenin kendi özgün çalışmam olduğunu, başkalarının haklarını ihlal etmediğimi ve tezimin/projenin tek yetkili sahibi olduğumu beyan ve taahhüt ederim. Tezimde yer alan telif hakkı bulunan ve sahiplerinden yazılı izin alınarak kullanılması zorunlu metinlerin yazılı izin alarak kullandığımı ve istenildiğinde suretlerini Üniversiteye teslim etmeyi taahhüt ederim.

Yükseköğretim kurulu tarafından yayınlanan ***"Lisanüstü Tezlerin Elektronik Ortamda Toplanması, Düzenlenmesi ve Erişime Açılmasına İlişkin Yönerge"*** kapsamında tezim aşağıda belirtilen koşullar haricinde YÖK Ulusal Tez Merkezi/ Kocaeli Üniversitesi Kütüphaneleri Açık Erişim Sisteminde erişime açılır.

☐ Enstitü yönetim kurulu kararı ile tezimin/projemin erişime açılması mezuniyet tarihinden itibaren 2 yıl ertelenmiştir.

☐ Enstitü yönetim kurulu gerekçeli kararı ile tezimin/projemin erişime açılması mezuniyet tarihinden itibaren 6 ay ertelenmiştir.

☑ Tezim/projem ile ilgili gizlilik kararı verilmemiştir.

Azeez Adebayo

PREFACE AND ACKNOWLEDGMENT

This thesis presents an algorithm for the completion of an obstacle course for autonomous vehicles. The use of deep convolutional neural network in conjuction with a tree based approach was proposed and tested. The subsequent evaluation helped determine the feasibility of the algorithm for other like projects or studies.

Firstly, I would like to thank almighty Allah for giving me the endurance to successfully complete this work. Also, with great appreciation, I thank my thesis advisor Professor Dr. Huseyin Metin Ertunc for his continuous guidance and support that prompted me to overcome difficulties and encouraged me throughout my research work. Lastly, I would like to express my sincere gratitude to the Presidency for Turks Abroad and Related Communities scholarships (YTB) for providing me with the opportunity to be here in the first place, bringing me much closer to my engineering dream.

However, I am also very inclined to thank myself for persevering through the language barrier and stress of understanding, for striving to be amongst the best and for being able to motivate my peers.

April – 2022 Azeez ADEBAYO

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

A	: Available actions
a_t	: Action selected at time step t
b	: Bias of the convolution kernel
D_{in}	: Input dimension
D_{out}	: Output dimension
e^{z_i}	: Standard exponential function for input vector
e^{z_j}	: Standard exponential function for output vector
FN	: False negative
FP	: False positive
h	: Height
$L(.)$	: The loss function
$L_{cls}(p_i, p_i^*)$	: Classification loss representing the log loss over two first stage prediction classes
$L_{reg}(t_i, t_i^*)$	: Regression loss
l, m	: Row, Column number respectively of kernel/filter
M	: Number of classes
N	: Representing the number of detected or predicted bounding boxes
n	: Number of samples
P	: Transition probability in Markov Decision Processes
p	: Predicted probability observation o of class c
p_i^*	: The ground truth label ($1\ or\ 0$) i.e., object vs no object
p_i	: Output score for the classification branch
$Q(s, a)$	: Q-value function of Q-learning
R	: Reward function in Markov Decision Processes
S	: Set of possible environment states
s_t	: Environment state at time step t
ST	: Stride
σ	: SoftMax
TN	: True negative
TP	: True positive
t_i^*	: Regression target
t_i	: Output prediction of the regression layer
θ	: Parameter of the neural network
u	: ReLU input number
$V(s, a)$	: Value function of Q-learning
W	: Kernel size
w	: Width
ω	: Convolution kernel
$x(i, j)$	: Input vector
(x, y)	: The coordinates of the box center
(x_{min}, y_{min})	: Coordinates of the top left bounding box corner
(x_{max}, y_{max})	: Coordinates of the bottom right bounding box corner
y	: Binary indicator (0 or 1) if class label c is the correct classification for observation o
$\bar{z}$	: SoftMax input vector

Abbreviations

AGV	: Autonomous Guided Vehicle
AI	: Artificial Intelligence
AVX	: Advanced Vector Extensions
BLDC	: Brushless DC electric motor
CNN	: Convolutional Neural Network
CSV	: Comma-separated values
DRL	: Deep Reinforcement Learning
DQN	: Deep Q-network
Faster-RCNN	: Faster Region-based Convolutional Neural Network
FCN	: Fully connected network
FMA	: Fused Multiply Accumulate
GPS	: Global Positioning System
HIL	: Hardware in the Loop
ID3	: Iterative Dichotomiser 3
IMU	: Inertial Measurement Unit
IOU	: Intersection over union
KITTI	: Karlsruhe Institute of Technology and Toyota Technological Institute
LiDAR	: Light Detection and Ranging
LSTM	: Long Short-Term Memory
MAP	: Mean Average Precision
MDP	: Markov Decision Process
ML	: Machine Learning
MSE	: Mean Squared Error
NMS	: Non-Maximum Suppression
NN	: Neural Network
ReLU	: Rectified Linear Unit
RL	: Reinforcement Learning
ROI	: Region of Interest
ROS	: Robot Operating System
RPN	: Region Proposal Network
SGD	: stochastic gradient descent
SVM	: Support Vector Machine
UART	: Universal asynchronous receiver-transmitter
VGG	: Visual Geometry Group

DERİN DÖNÜŞÜMLÜ SİNİR AĞI VE LIDAR ÖLÇÜMLERİYLE OTONOM BİR ARABANIN NAVİGASYONU

ÖZET

Bu tez çalışmasında, elektrikli bir aracın bir engel parkurunu hatasız tamamlamasına yönelik temel kontrolü için kapsamlı bir algoritma geliştirilmiştir. Engel parkurunun boyutları ve ölçüleri Teknofest yarış kurallarına göre belirlenmiş ve Gazebo ortamında ölçekli bir simülasyonu yapılmıştır. Tezde, söz konusu elektrikli aracın hem trafik kurallarına doğru bir şekilde uyarken parkuru başarıyla tamamlaması hem de güvenliği ön planda tutması için gereken optimizasyon çalışmaları yapılmıştır. Bu çalışma için geleneksel otonom sürüşü; perceptron (algı), planlama ve kontrol için ayrı bileşenler olarak kullanılabilecek şekilde seçilmiştir. Otonom araçların yakın bir gelecekte çok fazla yaygın hale geleceği motivasyonuyla çalışma gerçekleştirilmiştir. Evrişimli sinir ağı (CNN), birçok araştırmacının projesinde akan bir görüntü ekranında gerçek zamanlı hızlarda şerit ve araç tespiti gerçekleştirmek için kullanılmış ve umut verici sonuçlar verdiği gösterilmiştir. Diğer taraftan Uçtan-Uca otonom sürüş yaklaşımı genellikle güvenilirdir ve otonom tabanlı proje uygulamalarında sıklıkla tercih edilir. Çünkü söz konusu model, hem pekiştirmeli öğrenmede (reinforcement learning) hem de denetimli öğrenmede (supervised learning) uygulandığı gibi bir giriş görüntüsüne dayalı olarak her araç durumu (direksiyon açıları, gaz kelebeği seviyesi, fren ve hız) için sayısal bir değer öngörür. Ancak, her araç rotası için genelleştirilmiş bir ödül işlevi bulmanın yanı sıra aracın/temsilcinin eğitimsiz bir ortamda veya trafik işareti kombinasyonundaki davranışsal belirsizliği ile ilgili pratik zorluklar, bu yöntemleri bu çalışma için uygun olmayan bir seçim haline getirmektedir. Bu yüzden ilk olarak, düşük hesaplama yükü üzerinde karşılaştırmalı algılama hızı nedeniyle Faster-RCNN modeli seçilmiştir. Hareket tahminlerinin iyi olmasını sağlamak için bir karar ağacı formatında LiDAR ölçümleriyle sürekli bağlantılıyken 12 farklı yol işaretini algılamak üzere eğitilmiştir. Fren, gaz kelebeği veya aracın parkurda güvenli bir şekilde gezinmesi için ne kadar yönlendirileceğine yönelik değerler önceden belirlenir, sonra uygulanır. Önerilen algoritma, beş mantık aşamasına ayrılarak ifade edilebilir: sol mantığı takip, düz mantığı takip, sağ mantığı takip, dur mantığı ve park mantığı. İstenen araç davranışının elde edildiği çeşitli komutları içeren her bir fonksiyon yerine getirilir. Bu komutlar daha sonra yerleşik Arduino mikro denetleyicisi tarafından fren aktüatörleri, direksiyon step motoru ve diğer araç bileşenlerinin kontrolünde yürütülür. Tez çalışması esnasında yaklaşık 4636 görüntü toplanmış ve etiketlenmiştir; bunların %90'ı modelin Tensorflow çerçevesindeki eğitimi için ayrılmıştır. Saniyede 6 kare hızında gerçek zamanlı tek çekim kamera görüntülerinde %95,8 genel doğrulukla 429243 adımda önerilen modelin kayıp değeri, 0,0097'dir. Olasılık belirsizliklerini azaltmak için modelin algılama eşiği de %85 olarak ayarlanmıştır. Gazebo kullanan kapsamlı simülasyonlar algoritmayı doğrularken, sensörler ve kontrolör arasında çok iyi bir bağlantı sağladığı için Robot İşletim Sistemi (ROS) de dahil edilmiştir. Tez kapsamında gerçekleştirilen çalışma başarılı olmasına rağmen, başarı oranı 0 ile 1 arasında ölçeklendirildiğinde 0,8 olarak değerlendirilmiştir. Başarısızlığının büyük bir kısmı araçların doğru park edememesinden kaynaklanmaktadır. Bunun sonucunda, algoritmada daha fazla iyileştirmeye yönelik derinlik sensörlü kameraların kullanılması gibi çeşitli önerilerde bulunulmuştur. Böylece daha kesin kararlar için araçtan gerçek nesne mesafesi elde edebilmek mümkün olmaktadır. LiDAR ölçümleri, otonom aracın bariyerlere çarpmadan sürüşü için

gereklidir. Bununla birlikte, otonom araca şerit algılama teknolojisinin dâhil edilmesi, aracın şerit engel parkurlarında da otonom hareket etmesini sağlayacaktır. Bunlar, gerçek zamanlı algılama doğruluğunu artırmak için eğitim sürecinde daha fazla görüntünün kullanılmasını gerekli kılmaktadır. Son olarak, geliştirilen algoritma otonom araç üzerinde rota takibi için denendiğinde simülasyondaki ile benzer sonuçlar alınmıştır. Test edilen arabanın davranışının, modellenen kopyasına tamamen benzediği görülmüştür. Ham kodlar ve eğitilmiş model, daha fazla erişim veya kullanım için GitHub deposunda kullanıma sunulmuştur. Bu çalışmada, yaklaşımı basit olmakla birlikte, algılama, planlama ve kontrolü içeren otonom sürüş temelleri uygulanmıştır.

Anahtar Kelimeler: Derin öğrenme, Engel Parkuru, Evrişimli Sinir Ağı, Otonom Araç, Robot İşletim Sistemi (ROS).

THE NAVIGATION OF AN AUTONOMOUS CAR USING DEEP CONVOLUTIONAL NEURAL NETWORK AND LIDAR MEASUREMENTS

ABSTRACT

A comprehensive algorithm was developed for the basic control of an electric vehicle towards completing an obstacle course. For this study, the traditional autonomous driving is selected as separate components can be used for perception, planning and control. Faster-RCNN model which was selected because of its comparative detection speed over a low computational burden, was trained to detect 12 different road signs while in constant conjunction with the LiDAR data in a decision tree format towards ensuring that motion predictions are well established before a conversion into action in the form of brake, throttle or how much to steer the car. These commands were executed by the onboard Arduino in controlling the brake actuators, steering stepper motor and other vehicle components. About 4636 images were collected and labelled of which 90% was dedicated for the model's training within the Tensorflow framework. The loss value of the proposed model is 0.0097 at 429243 steps with an overall accuracy of 95.8%. The model's detection threshold was also set to 85% in order to decrease probability ambiguities. Robot Operating System (ROS) functionality is also incorporated as it provided a seamless bridge between sensors and controller. Extensive simulations using Gazebo validated the algorithm, with a success rate of 0.8, before being deployed in real life where the course was completed with similar result. The car's behavior being completely alike to its modelled replica.

Keywords: Deep Learning, Obstacle Course, Convolutional Neural Network, Autonomous Vehicle, ROS.

1. INTRODUCTION

TEKNOFEST, one of the biggest aviation, space and technology festivals in the world, raising awareness to the current trends in technology and science in today's society as a whole, has its autonomous car category, dubbed ROBOTAXI in the form of an obstacle course shown in Figure 1.1, with barriers and road/traffic signs at various intersections (Teknofest Aerospace and Technology Festival, 2020). The ability of the vehicle to complete the course successfully while obeying traffic rules correctly as well as prioritizing safety is desired to be optimized.

Furthermore, it may be expected that the vehicle possesses the ability to perceive the environment, to localize itself, and to navigate robustly from one place to another. That is, a comprehensive integration of several different knowledge bodies or modules, effectively making mobile robotics a multidisciplinary field. In perception, the vehicle's only substitute for the basic senses is its sensor which can be processed and manipulated to give specialized information or interpretable measurement structures to construct reliable perception of its environment. Some commonly used sensors are;

- Camera,
- Light Detection and Ranging (LiDAR) sensor,
- Radar,
- and sometimes Sonar.

On the other hand, localization and navigation require the knowledge of control theory, probability theory and sometimes a combination with artificial intelligence algorithms. They represent internal systems that helps bridge sensing to mobility, with navigation further broken down as path planning and motion control. They are armed with commonly used sensors like;

- Motor/wheel sensors (encoders),
- Inertial Measurement Unit (IMU),
- Global Positioning System (GPS) sensor etc.

From the above, the control scheme of the autonomous mobile robot as summarized in

(Siegwart et al., 2011), have now been put here in the context of the question they answer:

- Perception. What is in my environment?
- Localization. Where am I?
- Path planning. How do I get from point A to B?
- Motion control. How should I get to point B?

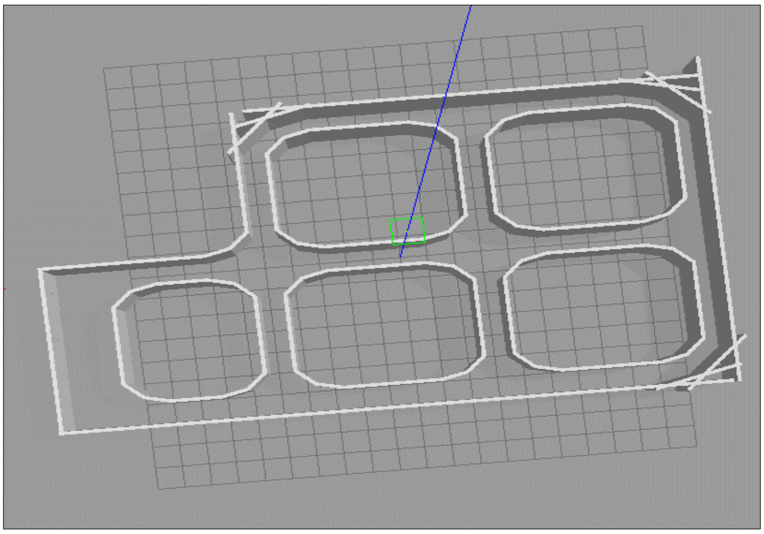
In outdoor cases, the exact pose -position and orientation- can be effectively obtained with some small degree of uncertainty using the GPS sensor technology and as such solving the localization problem as opposed indoor situations where more complex equations need to be generated in monitoring pose, yet with its own uncertainties. Essentially, the pose estimation problem is sensor noise compensation (Milstein et al., 2002). For this study however, it is not necessary that we know where the vehicle is located as even though a GPS can be used, its inclusion is simply not crucial to the completion of the task. This study only proposes the use of perception, and navigation -i.e., planning and control- algorithm in completing the obstacle course.

The obvious drawback of this, as in every vision-based system is its illumination dependency as that constitutes a form of noticeable noise which may blur, lighten the images, picture jitters from motion vibration may also be considered as further noise sources. More specifically, these types of sensor noises reduce the helpful content of the color image. In the case of LiDAR, effects of illumination and/or contact material may not necessarily ensure that the beam is reflected in order for an accurate measure of the obstruction distance or may yield incoherent reflection of the expected signal or beam, resulting in failure. Power related faults are also possible although less common. However, the LiDAR generally succeeds in perceiving the object, and only fails on rare occasions.

Without doubt, the age of self-driving cars is upon us. Convolutional neural network (CNN) has been utilized by various authors and projects in performing lane and vehicle detection at a running image frame (videos) at real-time speeds. The results demonstrated in (Arinaldi et al., 2018; Badioze Zaman et al., 2017; Ekşi & Gökmen, 2020) shows that deep learning architecture can be successfully used for self-driving vehicles. Furthermore, while the End-to-End autonomous driving approach, is reliable and highly favored in similar projects, as the model predicts a numeric value for every vehicle state (steering angles, throttle level, brake, and speed) based on an input image, for supervised learning (Meftah & Braham, 2020; Padhy et al., 2018; Raj & Narendra G, 2021) or the agent estimates the Q-values that signifies the best decision (turn left, right or do nothing) in which reward is maximized, as in reinforcement learning (Chopra & Roy, 2020). However, considering the practical difficulty involved in finding a generalized reward function for every course (Kebria et al., 2020) and that, since a scenic overview is required and learned, a complete change in scenery or a traffic signal/sign placed at a different intersection would make the agent unstable, i.e., not necessarily yielding a desired vehicle/agent behavior. The need for a more tailored approach helped simplified the choice of the traditional autonomous driving (Raj & Narendra G, 2021). Of course, without the reliance on pre-built maps or vehicle localization, as we do not intend to offer way-points nor require a complex dataset structure as in the KITTI dataset, Comma.ai dataset etc. Ultimately, the complexity of design of a controller is reduced.

To this end, the data to be gathered here are in the form of single camera images shot at 6 frames per second and are fed directly into a Faster RCNN model, using its RPN block to propose regions with object presence score of one “1” unto the feature maps, in a bid to achieve real time detection speed (Phon-Amnuaisuk et al., 2018; Ren et al., 2015). Then, subject to a size hierarchy of the bounding boxes on the final output image, a decision tree with LiDAR data as its focus is built.

In order to have seamless communication between components and as such reduce accidents and by extension cost and time, various simulators and their usage limitations were examined in (Meftah & Braham, 2020; Rivera et al., 2019). ROS and Gazebo were selected for this study as it offered code re-usability and design flexibility; ensuring standardized subscriber and publisher programming structure as well as obstacle course drawing and dimensioning. More so, the faster RCNN trained model along with the driving algorithm is written in Python while using Tensorflow as the deep learning framework. It is our desire that the success of this study be replicated in the competitions to come.



(a)

(b)

Fig. 1.1. (a) Scaled simulated replica of the Teknofest Robotaksi arena (b) The real-life parkour

This thesis as such is organized as follows, Chapter I contains the introduction and general overview of the project scope. Chapter II examines the literature and the successful techniques used in like projects, Chapter III describes the methodology; development of the data, model architecture, training, and algorithm design. The results from experimental simulations and real-world demonstration are depicted and discussed in Chapter IV. Finally, in Chapter V, the study’s conclusions and future work suggestions are presented.

2. BACKGROUND KNOWLEDGE

Artificial Intelligence (AI) and Machine Learning (ML) are based on computational models and algorithms for classification, clustering, regression and dimensionality reduction, such as neural networks, genetic algorithms, support vector machines, k-means, kernel regression and discriminant analysis. Machine Learning is the field of study that gives computers the capability to learn without being explicitly programmed and has enabled tasks ranging from driving cars to translating speech. The three main categories of machine learning are supervised learning, unsupervised learning and reinforcement learning. These three are discussed relative to their usage in the autonomous driving context

2.1. Supervised Learning

In supervised learning, giving input variable (x) and an output variable (y), an algorithm is used to learn the mapping function from input to output, with a goal to approximate the mapping function so well that, for any input x an output variable y can be accurately predicted. Algorithms are trained using labeled examples and "Learning" consists of using sophisticated mathematical algorithms to minimize the loss of prediction. To make this measurement, different techniques may be employed depending on the problem but the most common ones are; cross-entropy loss for classification and MSE for regression. Other examples of supervised learning include: Linear Regression, Gaussian Naive Bayes, Decision Trees, Support Vector Machine (SVM), Random Forest, etc. Supervised learning algorithms primarily generate two kinds of results: classification and regression.

Classification: predict discrete responses. In binary class predictions, responses such as 0 for 'No' and 1 for 'Yes' with their respective probabilities are returned. However, responses such as turn left, turn right or remain in same position are returned in multi-class situations. The cross-entropy loss or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability diverges from the actual label. In binary classification, cross-entropy is calculated as:

$$-(y\log(p) + (1-y)\log(1-p)) \tag{2.1}$$

for multiclass classification, a separate loss is calculated for each class label per observation and the results are summed.

$$-\sum_{c=1}^{M} y_{o,c}\log(p_{o,c}) \tag{2.2}$$

M refers to the number of classes, y is binary indicator (0 or 1) if class label c is the correct classification for observation o, and p is the predicted probability observation o of class c.

Regression: predict continuous responses such as price of houses or steering angle having value ranging between 0 and 360. The Mean Squared Error, or $L2$ loss tells us how close a regression line is to a set of points by calculating their respective distances. These distances represent the error and are squared to eliminate negative signs before being averaged. Mathematically,

$$MSE = \frac{1}{n}\sum(actual - forecast)^2 \tag{2.3}$$

n refers to the number of samples.

2.1.1. Convolutional Neural Networks

For image classification or detection, CNNs are good at feature extraction hence are naturally used as image encoders. The convolution network as used in studies often comprises of the application of 2D kernel, w or stacks of kernels as in filter on an input image with pixels location (i,j) as seen in Equation (2.4) and (2.5), to capture certain features, before being processed with activation function f to produce the output, y_t and followed by MaxPooling2D layer shown in Equation (2.6) that acts to down sample the feature maps, all of these are ordered into a stack of the required depth. This ensures that the information being abstracted is greatly reduced in size or dimension and yields an efficient interpretation as internal matrix or vector representation. The final layers of deep CNN consist of one or more fully connected layers with SoftMax regression and is depicted in Figure 2.1.

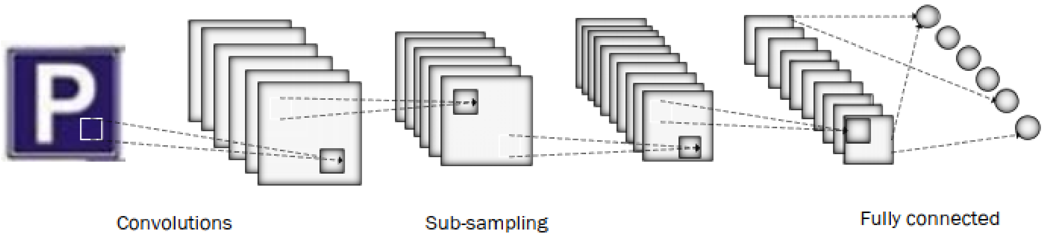


Fig. 2.1. Illustration of convolutional layers and fully-connected layers forming CNN

$$u(i,j) = \sum_l \sum_m x(i+l, j+m)\omega(l,m) + b \tag{2.4}$$

$$y_t = \text{ReLU}(u(i,j)) \tag{2.5}$$

$$D_{out} = \frac{(D_{in} - W)}{ST} + 1 \tag{2.6}$$

where y_t represents the output value after convolution, Rectified Linear Unit (ReLU) defined in Equation (2.7) is a piecewise linear activation function that will output the input, u directly if it is positive, otherwise, it will output zero, $x(i,j)$ is the input vector, ω is the convolution kernel, and b is the bias of the convolution kernel. l and m are row, column number respectively of kernel/filter, stride is ST, W is kernel size and D_{in}, D_{out} are input and output dimension respectively.

$$R(z) = \begin{Bmatrix} u & u > 0 \\ 0 & u \leq 0 \end{Bmatrix} \tag{2.7}$$

The rectified linear activation (ReLU) is the default activation when developing multilayer perceptron and CNNs as it overcomes the vanishing gradient problem for when the layers are many. Other activation functions include:

Linear activation: here, no transform is applied at all to the inputs after they are weighted and summed.

Nonlinear activation: These functions are sometimes preferred as they allow the nodes to learn more complex structures in the data. Traditionally, two widely used nonlinear activation functions are the sigmoid and hyperbolic tangent activation functions. For

Sigmoid, the input to the function is transformed into a value between 0.0 and 1.0. While, for the hyperbolic tangent function, or tanh outputs values between -1.0 and 1.0. These functions are however, only sensitive to changes around their mid-point of their input, such as 0.5 for sigmoid and 0.0 for tanh and, saturate at their limits. Another activation function that is generally used at the last layer of neural networks is the SoftMax - Equation (2.8)- as it applies the standard exponential function to each element z_i of the input vector z and normalizes these values by dividing by the sum of all these exponentials; this normalization ensures that the sum of the components of the output $\sigma(z)$ is 1 i.e. mathematically converts a vector of numbers into a vector of probabilities, where the probabilities of each value are proportional to the relative scale of each value in the vector. The main advantage being that the function is able to handle multiple classes. For this reason, the model chosen for this study uses the ReLU activation within its layers and SoftMax activation as its final layer classifier.

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j-1}^{M} e^{z_j}} \; for \; i = 1, \dots, M \; and \; z = (z_1, \dots, z_M) \in \mathbb{R}^M \tag{2.8}$$

The notation σ= SoftMax, $\bar{z}$ = input vector, e^{z_i}= standard exponential function for input vector, M = number of classes in the multi-class classifier, e^{z_j} = standard exponential function for output vector.

2.1.2. End to End Supervised Learning

The idea of end-to-end learning in autonomous driving is that direct outputs such as steering angle, vehicle velocity or the choice to break or not could be predicted by the model bypassing the need for some control system or theoretical calculation. The model is trained such that it perfectly approximates the functions necessary to generate the desired outputs. As this is a supervised learning, images are labeled in such a way that their output class yields some probability value representing a decision (e.g., left, right or maintain position) of how much to turn the steering, sometimes the CNN is stacked with a Long Short-Term Memory (LSTM) network as in (Liu et al., 2020) such that neurons shared are used to influence the decision of the LSTM in the prediction of velocity. Generally, this technique requires imitation as a human drive the vehicle and records data for training simultaneously.

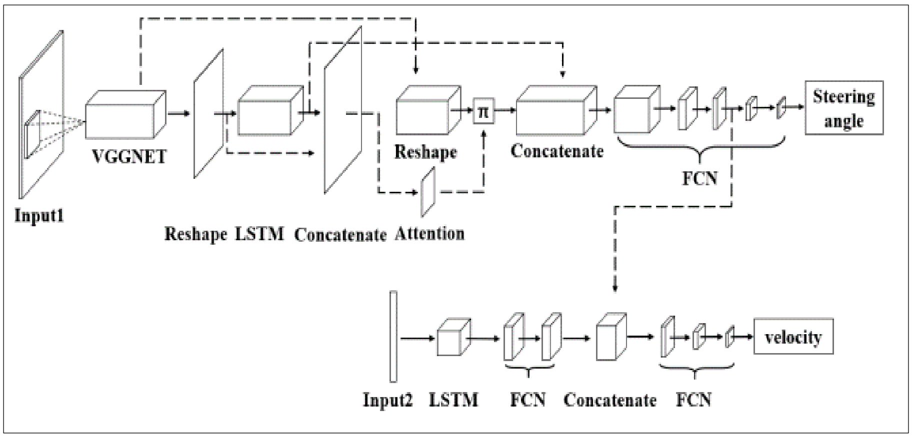


Fig. 2.2. End-to-End control model structure as depicted in (Liu et al., 2020)

2.1.3. Decision Trees

Decision Trees are a type of Supervised Machine Learning wherein decision is narrowed down by splitting based on learned features. In most cases the Gini split is used as the feature with the lowest Gini value helps abstract features the most. Like in every supervised learning, a table or training data of inputs and their corresponding output is provided. The tree can be explained by two entities, namely decision nodes and leaves. Decision nodes are feature abstraction levels with the primary source or root node having branches, also termed edges. While the leaves are the decisions or the final outcomes, in this case, whether the car turns left, maintain position or turn right, respectively. A decision tree is drawn upside down with its root at the top as seen in Figure 2.3.

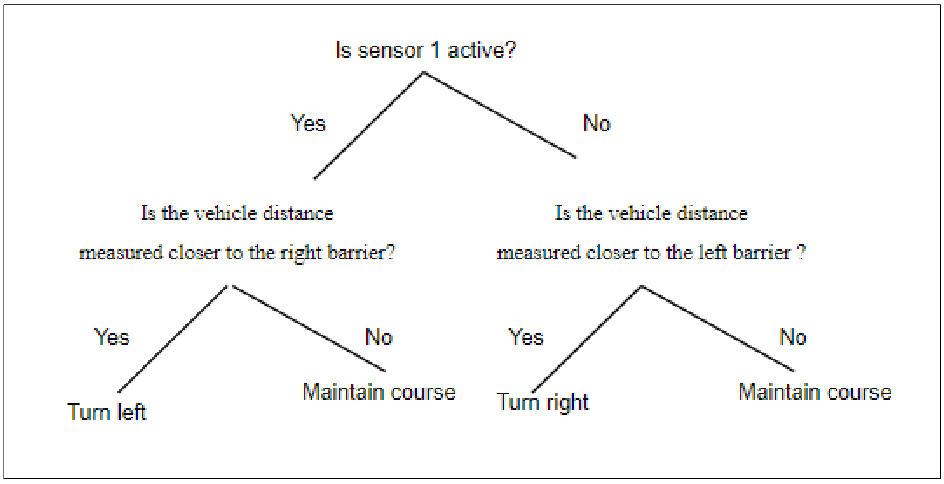


Fig. 2.3. A basic classification decision tree

The above tree is called Classification tree as the target is to classify direction. Regression trees are represented in the same manner however, they predict continuous values like price of a house or steering angle.

The ID3 algorithm is the most popular decision tree technique and it considers the attribute with the most information gain as the root node by using Entropy as loss function. Another way of saying the same thing is the feature that abstracts information the most is used as the root node. Each attribute has its entropy (discussed in section 2.1) calculated in order to obtain how much information gain is possible, before being compared with other attributes. A split is only performed if the Entropy of each the resulting nodes is lower than the Entropy of the parent node. The process is iterated until leaf nodes are achieved in all cases. In our case, based on the LiDAR data values at every instant, we may split the decisions into when to turn left, turn right or maintain a course.

2.1.4. Drawbacks of Supervised Learning in Autonomous Driving

While steering angle, velocity or brake output can be easily predicted to some degree of accuracy, in this form of learning, a change in traffic sign or signal position within the same scene, not previously planned, will not necessarily yield the desired change in behavior as the whole scene is learnt rather than the distinct representation of the signs. Also, if for some reason we are able to label or possible scenarios accordingly, the next problem would be what happens if the design of the road is changed, what behavior is to be expected of the vehicle.

2.2. Unsupervised Learning

In contrast, unsupervised learning tasks algorithms with identifying patterns in data, trying to spot similarities that split that data into categories. Unsupervised learning algorithms aren't designed to single out specific types of data, they simply look for data that can be grouped by similarities, or for anomalies that stand out. Clustering is the most common unsupervised learning technique i.e., finding relationships within data.

There are no training examples used in this process. Instead, the system is given a set data and tasked with finding patterns and correlations therein. In self-driving situations however, semi supervised learning is mostly used wherein a trained model is used in labelling real-time data in order to minimize the cost associated with labelling. Other times, it is used as a back-up model in cases where the supervised model fails to detect certain objects which automatically are clustered or grouped by their similarity to already determined classes. Popular techniques include self-organizing maps, k-means clustering and singular value decomposition. For this study however, unsupervised learning is not necessarily useful.

2.3. Reinforcement Learning

An agent learns by trials and errors. Then becomes able to gain a knowledge through the learning process i.e., from experience wherein, actions with higher reward are reinforced while actions with penalties are avoided. initially, actions in states are usually generated randomly as there is no knowledge of the control policy π.

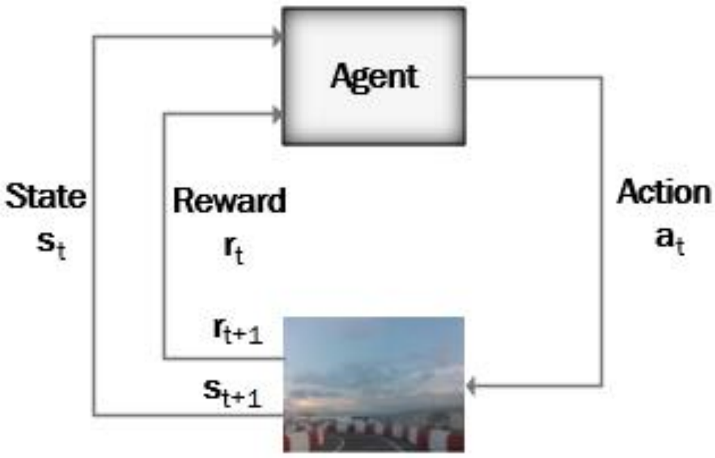


Fig. 2.4. Framework of Reinforcement Learning (RL) system

The policy π can be simplified as the probability of taking an action $a_t \in A$ at time t given a current state $s_t \in S$ as it can be seen as a set of rules that governs how the action is taken. However, iteratively the agent models the dynamics of the controlled environment by interacting with it.

$$policy: \pi(s,a) = P(a = a|s = s) \tag{2.9}$$

The choice of a model-based or model-free RL practice depends on how much information we have on the environment in which the agent is to interact. The Markov Decision Process (MDP) is used in model-based practices wherein the probabilities associated with the states and control are partly random and partly known. MDP property asserts that the next state is only a function of the current state and does not depend on state history or past. Also, expected future rewards are discounted over a gamma factor (0,1) showing our preference for the present over the future. MDP, according to (El Sallab et al., 2017) is defined by (S, A, R, $\mathbb{P}$, γ), in which (S) is a set of possible environment states, (A) is a set of possible actions, (γ) is the discount factor, (R) is the reward function, and ($\mathbb{P}$) is state transition probability. The MDP is modeled as:

$$R(s', s, a) = P(r_{t+1} | s_{t+1} = s', s_t = s, a_t = a) \tag{2.10}$$

The relationship between the policy and the return is introduced in the expected discounted return, as rewards from a policy followed over an episode is summed and its expectation is obtained, since we are dealing with probabilities. While reward is obtained after every step, the term is changed to return when it implies over one episode.

$$R_t(s) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{n-t} r_n = \sum_{t\geq 0} \gamma^t R_{a_t}(s_t, s_{t+1}) \tag{2.11}$$

$$\text{Expected Discounted Return} = E\left|\sum_{t\geq 0} \gamma^t R_{a_t}(s_t, s_{t+1})\right|_{\pi} \tag{2.12}$$

The goal of solving the MDP here is to find or learn through experience what is the optimal policy to maximize future rewards.

In order to find the policy, value-based methods or policy-based methods may be employed. With value-based methods, in moving from state to action we make some intermediate quality values or Q-values that helps quantitatively compare the different possible actions in that state, so like a choice to go full-left in state (t) being quantitatively higher than going half-left or full-right etc. The Bellman equation suggests the evaluation of the value of a state (Sutton, 1988) as shown in equation.

$$V(s) = \max_a (R(s, a) + \gamma V(s')) \tag{2.13}$$

$V(s')$ may be set to a value of how much important the target state is discounted as it is all relative. Again, the discount factor helps tell the agent how far it is from the target state. However, since the choices of the agent at first is to explore, we need a way to accommodate for this randomness in choice. Hence, Equation (2.13) is further modified as

$$V(s) = \max_{a}(R(s,a) + \gamma \sum_{s'} P(s,a,s')\, V(s')) \tag{2.14}$$

$P(s,a,s')$ refers to the probability of moving from state s to s' with action a and sum from $P(s,a,s')V(s')$ is the expectation of the situation that the robot incurs randomness.

Since we do care more about the quality of our actions as opposed just the possible values of the states. In Equation (2.14) we took the max of all possible values which essentially implies that there is more than one and to examine the quality, we transition to q-learning as

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} P(s,a,s') \max_{a} Q\,(s',a') \tag{2.15}$$

$$Q(s,a) = r + \gamma \max_{a} Q\,(s',a') \tag{2.16}$$

Although we are not necessarily sure of what the actual values of the next states are, the Q-value function can be iteratively converged to the optimal Q-value function (Boyan & Moore, 1995) using the temporal difference which is the difference of the estimated utilities/qualities between current state s, and next state s_{t+1} with learning rate α, at time step unit t, as shown below.

$$TD_t(s,a) = R(s,a) + \gamma \sum_{s'}(P(s,a,s') \max_{a'} Q\,(s',a')) - Q(s,a) \tag{2.17}$$

$$Q_t(s,a) = Q_{t-1}(s,a) + \alpha TD_t(s,a) \tag{2.18}$$

which can be re-written as

$$Q_t(s,a) = Q_{t-1}(s,a) + \alpha \left(R(s,a) + \gamma \max_{a'} Q\,(s',a') - Q_{t-1}(s,a)\ \right) \tag{2.19}$$

where alpha is the learning rate that controls how quickly the agent adapts to the random changes in the environment.

For every policy π, there is an optimal value for being in each of the states and its obtained as

$$V^*(s) = \max_a Q^*(s,a) \tag{2.20}$$

The optimal policy can be retrieved from the optimal value function as

$$\pi^*(s) = arg \max_a Q^*(s,a) \tag{2.21}$$

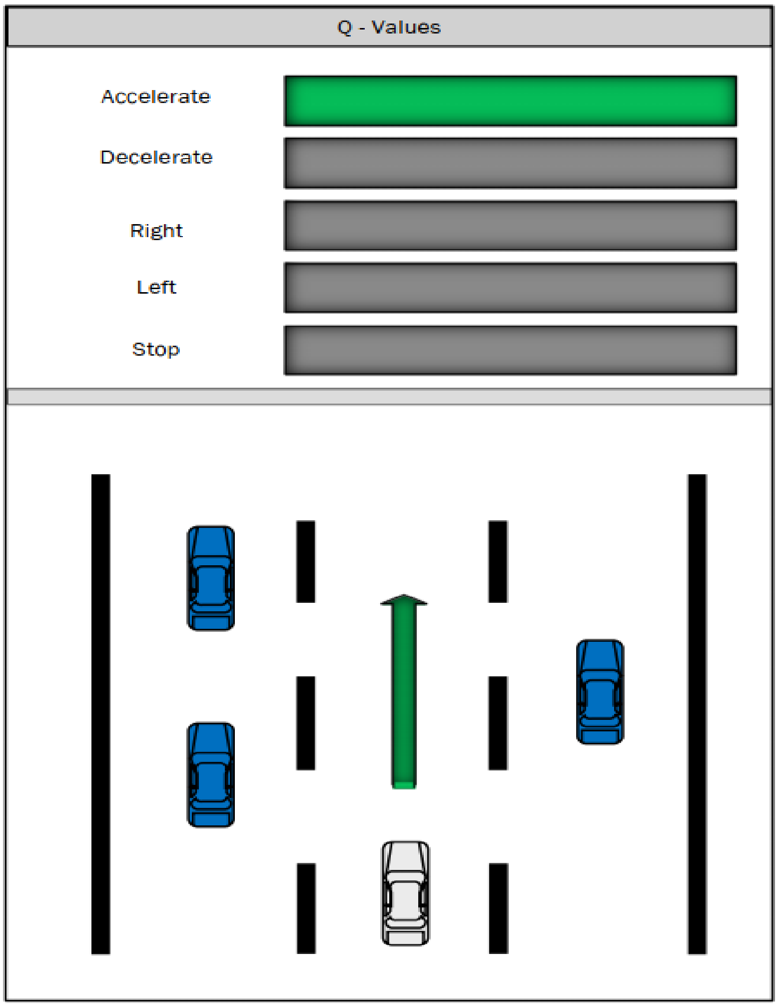


Fig. 2.5. Reinforcement Learning (RL) with Q-values for decision

The classical Q-learning algorithm can only obtain the optimal policy for small state space cases because the look-up table to update the Q-value of each element is limited in the state-action space. When the state space is large, this method is not applicable as extension becomes time-consuming and as such not applicable for real-time continuous decision-making. With policy-based methods, we go directly from state to action without the intermediate Q-values. This offers simplicity in that the table of values do not need to be stored, easily learns through stochasticity and much better for continuous outcomes. However, for a nonlinear dynamic case as in autonomous driving which maintains a more complex environment properties, as we have a multi agent environment wherein states are stochastic and the action space and time are continuous, the Deep Q-Network (DQN) (Mnih et al., 2015) is adopted. The DQN uses neural network (NN) to approximate the nonlinear relationship between states and Q-values of all the actions. The DQN uses the experience replay during training and determines the loss by taking the sum of the squared differences of the Q-values and their targets to update weights through backpropagation:

$$L = E \left| (r_{t+1} + \gamma \max_{a'} Q\,(s', a') - Q(s_t, a_t; \theta))^2 \right| \tag{2.20}$$

The learning goal of DQN is to find the best settings of the parameter θ of $Q(s, a; \theta)$ or the weights w of the neural network function approximator (Mnih et al., 2015). i.e.,

$$Q(s, a|\theta) \ \approx \ Q*(s, a) \tag{2.21}$$

In most self-driving use cases, the output is the steering values i.e., the network is trained end-to-end. In order to make the steering problem as a classification problem, the actions (steer left, or right) are discretized as shown in Figure 2.7.

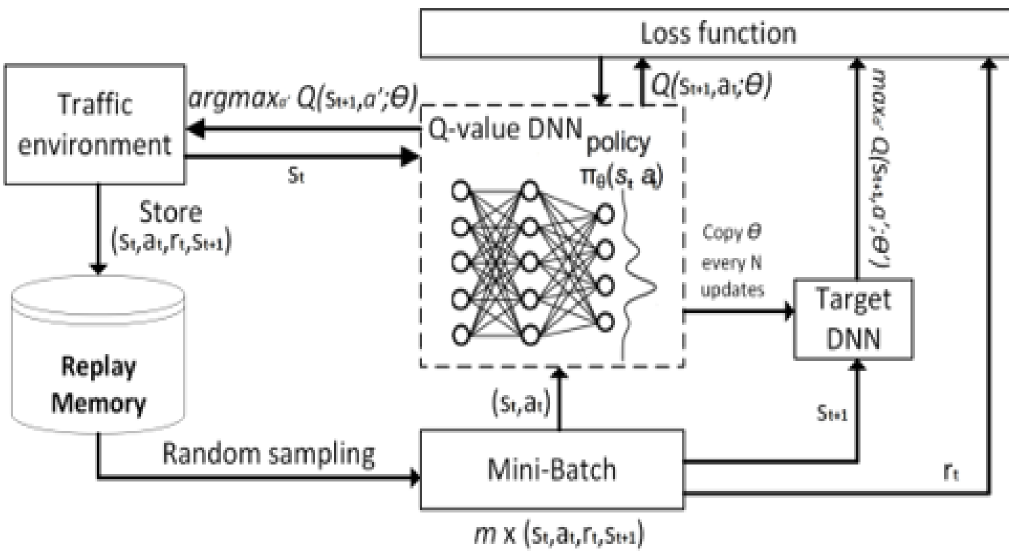


Fig. 2.6. Deeep Q-Network (DQN) system framework

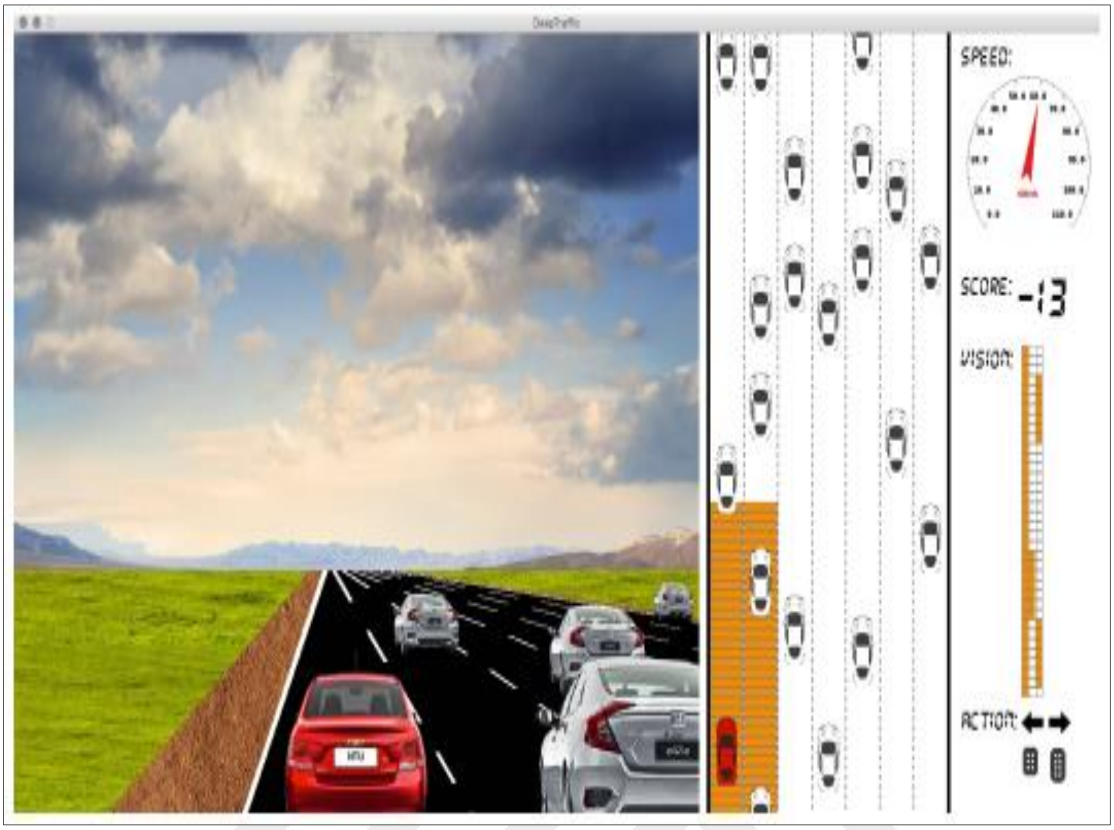


Fig. 2.7. End-to-End RL model simulation as depicted in (Ho, 2018)

2.3.1. Drawbacks of Reinforcement Learning in Autonomous Driving

This constant learning and feedback attribute makes the reinforcement learning (RL)-based autonomous driving reliable. However, in the context of the competition, Teknofest Robotaksi, the traditional Q-learning algorithm in every learning iteration requires a brute force grid search through the entire Q-table which may be computationally infeasible as exponential enlargement of the Q-table in control problems is induced due to the discretization of the action space (Belletti et al., 2018). Also, the DQN does not necessarily help as while a scenic or scenario-based view providing an end-to-end learning of picture to steering angle, if the traffic signal positions are changed within the same lane, the output steering behavior may not necessarily change. RL research in autonomous driving is mainly in intersection scenarios/traffic control, highway scenarios and of course end-to-end learning.

3. MATERIALS AND METHODS

3.1. Hardware Platform

The deep learning model is run on a custom built Tensorflow-1 framework for our Ubuntu 14.04 LTS OS host machine, which has Intel processor, 16GB RAM, and Advanced Vector Extensions (AVX) support which uses Fused Multiply Accumulate (FMA) operations, to accelerate linear algebra computations; dot-product, matrix multiplication, convolution, etc. Making ML operations quicker by up to 300%. Communication between the host machine and Arduino microcontroller is done via wired UART connection using the Robot Operating System (ROS) interface making up the control unit for the autonomous vehicle. The outfit is equipped with a camera, and an RpLiDAR S1 sensor for ground truth measurement. Images are captured at a default resolution of 1280×720×3. These images are passed through the trained Faster RCNN detector, after which a class specific motion command is generated, before being sent to the Arduino to initiate the control.

3.2. Image data collection and descrıption

4636 images -snapped from varying distances from desired object i.e., different camera location-, were collected and resized to 600×600 pixels which are less than 300 Kilobytes with no further edit or processing. These images maintained a somewhat balanced proportion between the 12 classes to be detected. Further partitioning was done to split the images into train and test sets, at 90% and 10%, respectively. The “LabelImg” tool was used to label images. When the Labelling was completed, a single file was created; containing bounding box coordinates as well as object class, image width and height table values, for train and test sets, respectively. The classes as shown in Figure 3.1 are; bus stop, restricted, no right turn, no left turn, green light, red light, park, no parking, stop, mandatory right ahead, mandatory left ahead, and ahead or turn right having corresponding identities: 1 – 12.

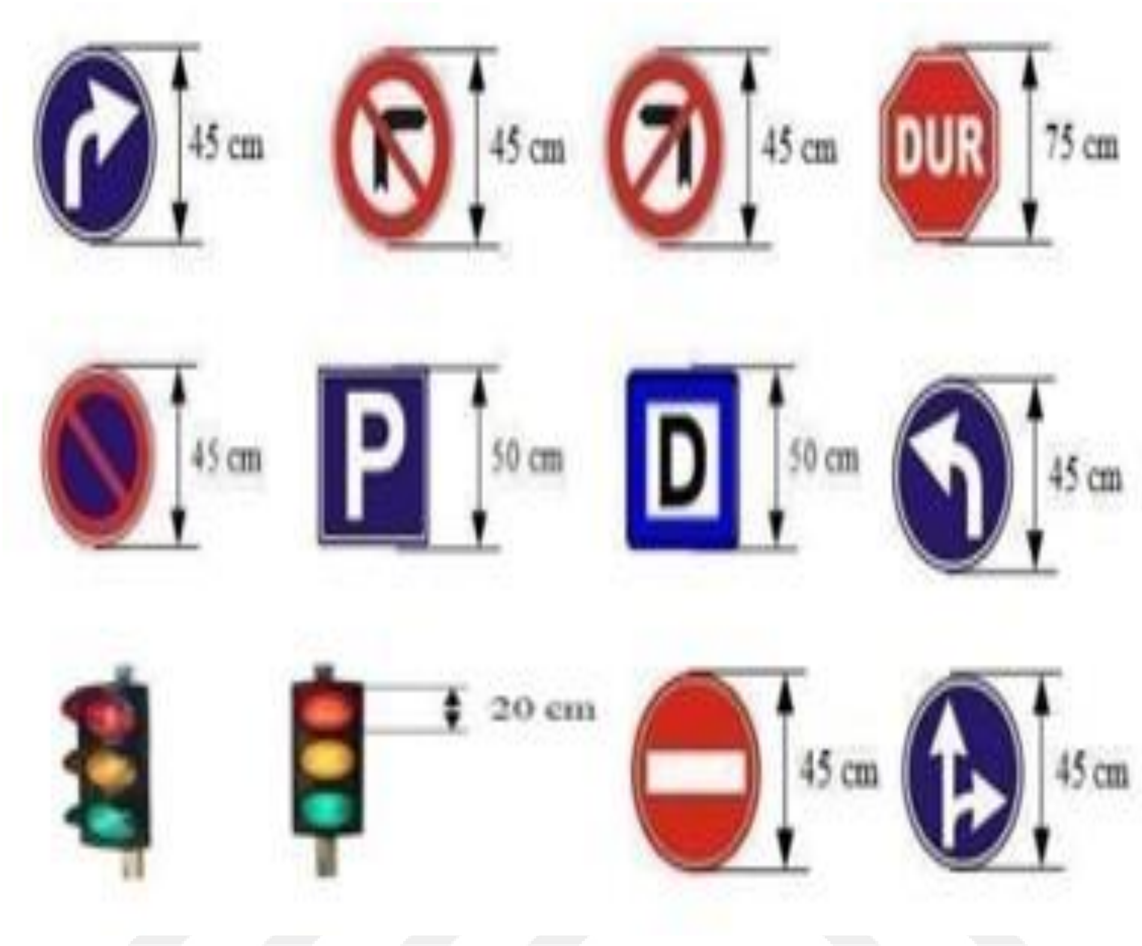


Fig. 3.1. Examples and measurements of the traffic signs

3.3. Training the model with image dataset

Any Convolutional Neural Network (CNN) has convolutional layers as its foundation architecture. With its depth implying the number of layers i.e., the more layers we have, the deeper the CNN. The Convolution refers to the application of a kernel, or stacks of kernels as in filter, of specific size and number, to capture certain features. Other important terms include pooling representing the action of down sampling feature maps, and stride which implies how much left to right, top to bottom, in pixels the filter should slide over an image. Generally, each convolutional layer is followed by a pooling layer with a stipulated stride. The importance of CNN is demonstrated by the mere fact that, 600×600×3 fully connected neurons would naturally be required to learn necessary weights and biases for features to ensure detection, which would no doubt be computationally expensive.

The files from Section 3.2 are Comma-separated values (.csv) files used as attribute in Tensorflow in order to get and obtain images and their information towards training or testing the Faster-RCNN model. The model was trained for 429243 steps with an overall accuracy of 95.8%. The training was not done at once as it was stopped at intervals and the model generated had its performance evaluated in the real-world scenario. The performance here, determined whether or not the training should be continued. Studies in (Chen & Elangovan, 2020) asserts that Faster R-CNN technique is 10 times faster than Fast R-CNN technique and 250 times faster compared to R-CNN.

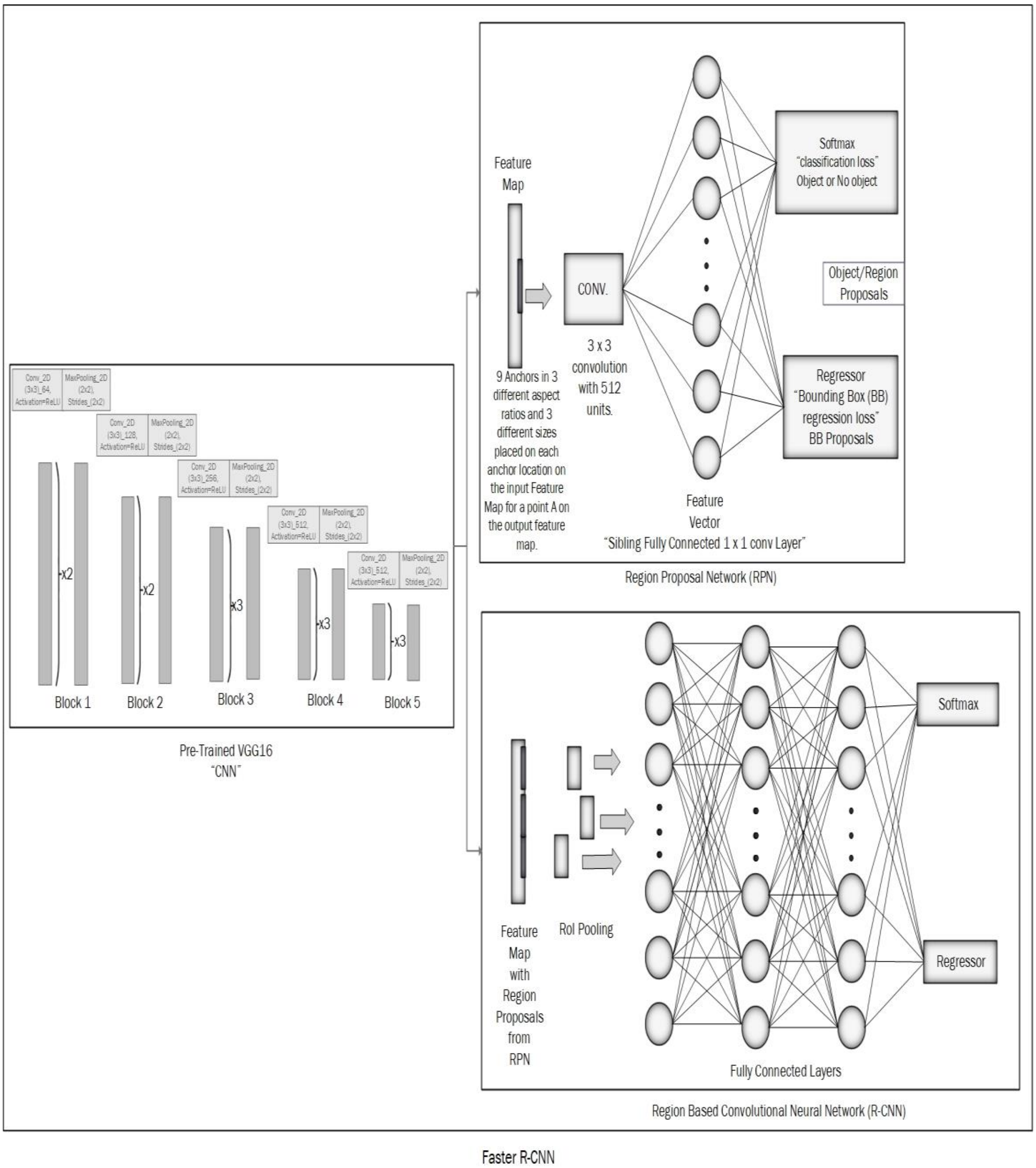


Fig. 3.2. Faster RCNN Architecture

3.3.1. Faster-RCNN model architecture

For this study, our Faster RCNN model, uses the VGG16 pre-trained Conv Net classifier, -without the inclusion of its fully connected layers i.e., just after the final convolution block- as seen in Figure 3.2, to learn and extract features from the input image [1,600,600,3] where 1 is the batch size. Furthermore, the convolutional feature maps can now be used for generating region of interests by constructing a Region Proposal Network (RPN) block, consisting of a sliding 3×3 convolution, 512 units with rectified linear unit (ReLU), followed by two sibling fully connected layers—a box-regression layer with 36 units and a box-classification layer of 18 units (Ren et al., 2015). Backpropagation and stochastic gradient descent (SGD) are used in the training of this block. For this study, the weights and biases are initialized at 0.0 and the learning rate, at 0.0001.

For every point in the output feature map, the network learns whether an object is present in the input image at its corresponding location and estimate its size. As the filter slides over each pixel of the feature map, a set of anchors are placed on the corresponding location in the input image, although for our case a stride of 16×16 is used in placing the anchors, each of which has 9 boxes of 3 varying size and 3 aspect ratios (0.5, 1.0 and 2.0) generated to indicate a possible object at that location.

As the network moves through each pixel in the output feature map containing objects in the corresponding input image, it refines these anchors' coordinates to give bounding boxes as "Object proposals" or regions of interest.

Two different anchor categories are generally considered. One category overlaps a ground-truth object with an Intersection over Union (IoU) greater than 0.69, and are considered "positive". While the other, considered "negative" have less than 0.3 IoU with ground-truth or don't overlap with any ground truth object. The rest are discarded for RPN training. The RPN model is optimized for a multi-task loss function i.e., combining two tasks (classification + localization). The loss function sums up the cost of classification and bounding box prediction via regression and is given below as

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}\,(p_i, p_i^*) + \lambda \frac{1}{N_{reg}}\sum_i p_i^* L_{reg}\,(t_i, t_i^*) \tag{3.1}$$

where $L_{cls}(p_i, p_i^*)$, is the classification loss representing the log loss over two first stage prediction classes. p_i is the output score from the classification branch for a specific anchor i, and p_i^* is the ground truth label ($1\ or\ 0$) i.e., object or no object. If the ground truth classification score is 1, indicating the presence of an object, the regression loss $L_{reg}(t_i, t_i^*)$, is activated. t_i is the output prediction of the regression layer and consists of 4 variables $[t_x, t_y, t_w, t_h]$. The regression target t_i^* is calculated from the ground truth bounding box coordinate x^* and the co-ordinates of the anchor box x_a as —

$$t_x = \frac{x - x_a}{w_a}, t_y = \frac{y - y_a}{h_a}, t_w = \log\left(\frac{w}{w_a}\right), t_h = \log\left(\frac{h}{h_a}\right) \tag{3.2}$$

$$t_x^* = \frac{x^* - x_a}{w_a}, t_y^* = \frac{y^* - y_a}{h_a}, t_w^* = \log\left(\frac{w^*}{w_a}\right), t_h^* = \log\left(\frac{h^*}{h_a}\right) \tag{3.3}$$

while $x, y, w,$ and h correspond to the coordinates of the box centre, width and height, while variables $x, x_a, and\ x^*$ are for the predicted box, anchor box, and ground-truth box respectively.

For maintaining a balance between the negative and positive anchors, random samples of both categories are taken to form a mini batch of size 256, equally. Or padded with the negatives, should the positives be less than 128. Since this does not mean that the anchor boxes aren’t overlapping, the Non-Maximum Suppression (NMS) is used to further filter them by sorting by scores as it iterates, discarding IoU larger than 0.6 pre-set threshold, leaving the highest scoring boxes. After NMS is done, selection is performed by keeping the top *N* proposals sorted by score. where *N* is the number of bounding boxes (Ren et al., 2015) with box regressor predicting better offset values for the bounding boxes. The output from this block serves as an input, added unto the shared feature maps before being pooled -reshaped using a Region of interest (RoI) pooling layer with MaxPool kernel size of 2 and stride 2- to finally, the Detection Network where a fully connected layer and a SoftMax activation and linear regressor which finds out the actual classification label probability from our 12 classes and further adjusts the location of their bounding boxes.

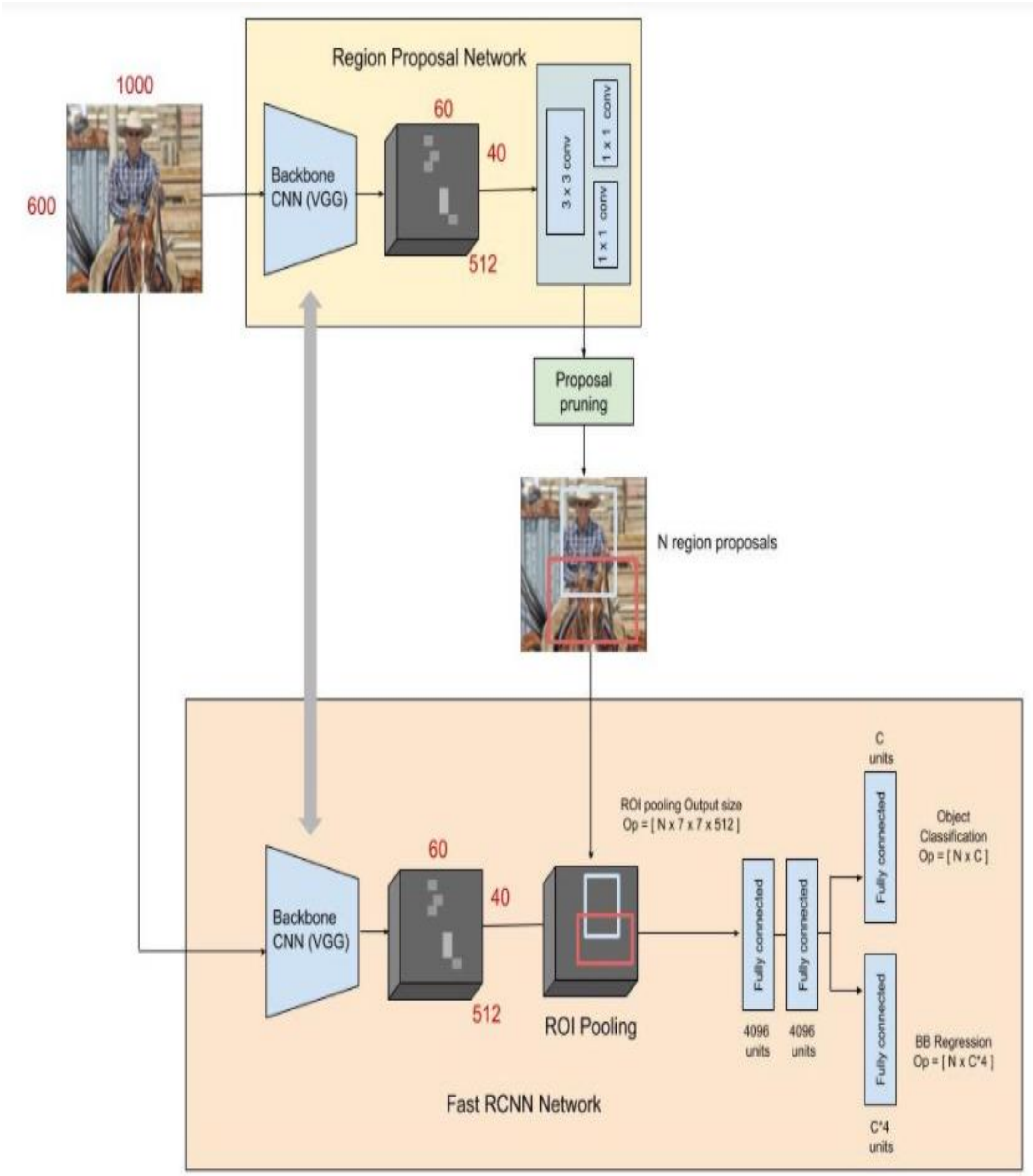


Fig. 3.3. The Faster R-CNN detection pipeline

The detection information for each input image, has the format: detection class, detection score, detection boxes: $[x_{min}, y_{min}, x_{max}, y_{max}]$, where: (x_{min}, y_{min}) - coordinates of the top left bounding box corner and (x_{max}, y_{max}) - coordinates of the bottom right bounding box corner. And detections number, N representing the number of detected or predicted bounding boxes. In this study, the perimeter of the bounding box or size is calculated from the detection box data and it is used in approximating the distance of the vehicle away from the object.

3.4. Measurement data collection and description

After having split the LiDAR readings into 9 sectors representing various areas, per dimension of the vehicle as shown in Figure 3.4. Although these sectors cover some several beams/rays of the LiDAR, only the lowest returned measurement is collected as it represents the part of vehicle with the highest likelihood of crashing. In order to build a decision tree that ensures safe driving, the LiDAR data was collected by having the desired task performed by a human operator i.e., drive the car around the obstacle course a few times, at a constant speed. Although this step is shared with the end-to-end learning, the aim here however, is to obtain the safest measurements for which the car is still permissible to drive freely without crashing with the barriers -measuring 50 cm by 100 cm each-. From (Teknofest Aerospace and Technology Festival, 2020), it is clear that while lane locations or traffic signal locations can change, road width remained 4 m. This therefore provides a basis for comparison with LiDAR data based on position of vehicle. It also helps ensure that the car can be successfully driven with measurements capable of navigating in scenarios where no traffic sign exists e.g., the edges, as the vehicle defaults to a safe driving mode or home mode.

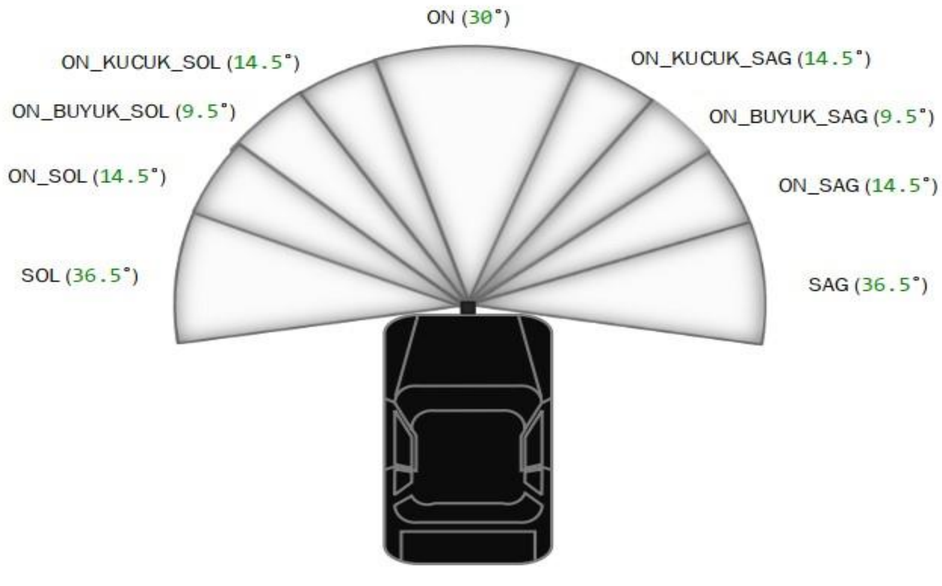


Fig. 3.4. LiDAR scan sections

Together, the camera and LiDAR gather real-time data, then commands are planned for every scenario around this, before control is offered, with the use of Arduino to break, throttle or steer the car. The steering used is a sports car steering with 3 turns to lock, which is about $1080°$ of rotation. That is, from the extreme left position to the extreme right limit, or vice versa, requires 3 full steering turns. A Nema 34 stepper motor $8.5nm$ is connected to a $2:1$ ratio gear as obtained using Equation (3.4), dividing the number of teeth of the driven gear by the number of teeth of the driving gear, implying that for every two rotations of the driving gear, the driven gear makes one rotation. The assembly is connected via a belt -ensuring that they turn in same direction- before being attached to the steering unit as shown in Figure 3.5.

$$Gear\ ratio = \frac{Driven\ gear}{Driving\ gear} \tag{3.4}$$

The stepper motor has a step angle of 1.8° with a possible error of $\pm 5\%$. Hence, $1080°$ implies 600 steps. Transmitting the same number of steps to the driven results in, 1200 steps. Programmatically, the steering rotation was divided into six '6' turns of 200 count each. Each turn representing a $13°$ angle shift in the vehicle's wheel or tyre, with the center 0° position corresponding to the $540°$ of the steering.

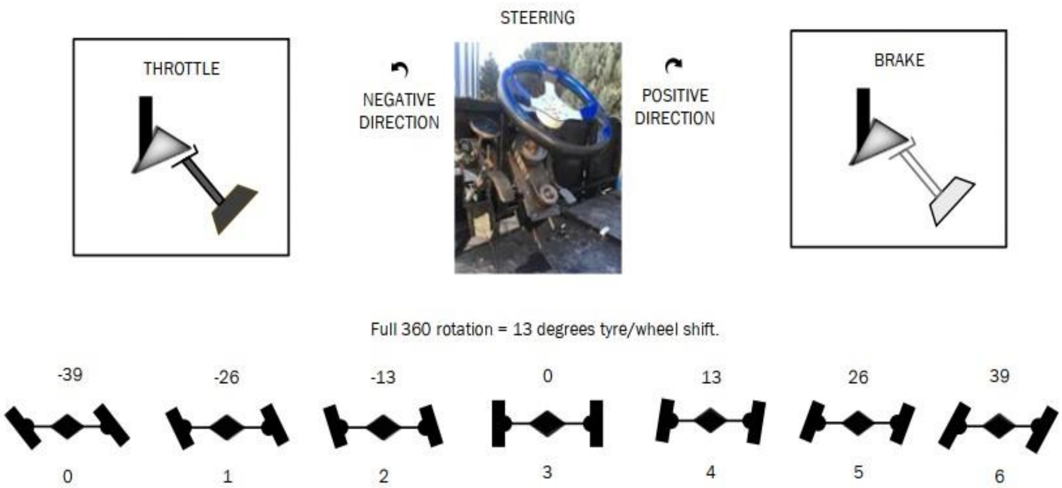


Fig. 3.5. Wheel/Tyre angle representation

Further integration with the brake system and the BLDC motor involves providing a logical one or zero that starts or force stops them respectively. The motor speed is initialized at $35m/s$ and remained fixed. This section completes the planning stage and is instrumental in the conversion of decision to action.

Table 3.1. Algorithm I of the Autonomous Navigation for Robotaksi

Initialize:	**Set** scale = scale value
Repeat	
Input:	**Get** image from the front camera
Compute:	**Determine** the presence of traffic signal, its detected class and score and **Calculate** perimeter of surrounding bounding boxes.
If	(mod(array[last six detected class]) == detected class) **Then**:
Initialize:	**Set** traffic signal = detected class
Update	array[last six detected class]
Else	
Update	array[last six detected class]
Initialize:	**Init** LiDAR scan
Input:	**Obtain** LiDAR scan
If	(perimeter > size threshold) **And** (score > score threshold) **Then**:
	If (LiDAR_scan > crash threshold/scale) **Then**:
	Start vehicle motor or throttle
	Algorithm II
	Else
	Stop vehicle or apply brake
Else	
If	(LiDAR_scan > crash threshold/scale) **Then**:
Start	vehicle motor or throttle
Select	appropriate steering turn for the centering of the vehicle on the road
Else	
Stop	vehicle or apply brake
Until	keyboard pressed or vehicle is stopped

In Table 3.1, “scale value” represents a factor by which the dimensions in the simulated world are to be multiplied in order to obtain their real-world values. The “detected class” is the class of object with highest likelihood or probability score, also used in the algorithm as “score”. It is compared against the last six detections in order to further ascertain the model’s prediction correctness, since images are shot at 6 frames per second. LiDAR “scan” represents the smallest values from each sectors treated and considered separately. The “perimeter” is used as a measurement factor since object appear to be larger, the closer they are to the camera. Thereby, actions can be made to be triggered only when specific “thresholds” are satisfied. “Steering_turn” is a variable that holds a value between 1 and 6, selected based on the considered environment.

3.5. Custom Decision Tree

In our case, the LiDAR data could have been recorded in a table format, with each section of the LiDAR as depicted in Figure 3.4 representing a column. The data may then be passed into a tree-based model as discussed in Section 2.1.3, for which the splits are obtained and the decision to turn left fully, turn left slightly, maintain a course, turn right slightly, or turn right fully are proffered as the 5 end leaves. However, it is also the case that the same data collected in a junction where the road splits in two, and populated in the table for which the signal indicated a turn right will also be indicative of its behavior for when the sign or traffic signal says “no right turns”. Since the values measured are not so easy to split and same data measurement occur in multiple places with different decisions as the output choices are not binary. Although since the main decider remains the output of the image detector, one might think it wise to make a root node off of each detection and thereby make the split easier. The problem with this logic would then be, what happens when the detection is false. That is, the effect of an error being more detrimental than the error itself as the car may turn right even though a variable had been changed.

This study uses a check and balance or look before leaping approach to ensure that decisions are well established before actions are carried out. A custom decision tree was built and tested in multiple scenarios for which the core modes are left turn, right turn and move straight. Each of which has the 5 end leaves as the possible choice within them.

In Table 3.2 -algorithm II-, which is a function within algorithm I, a left turn decision tree is shown. The tree was designed after observing the real time data collected from the LiDAR by driving the car to complete the course a few times while taking into consideration known dimensions of the parkour. Algorithm II shows the logic executed when the left turn is initiated or called. The right turn follows similar suit but with opposite rules.

Since the maximum turn capable at the vehicles wheel or tyre is 39° and the steering turns thrice (3 times) to lock. İn order to move from 0° at the wheel to 39°, each steering turn represents Each turn representing a 13° angle shift in the vehicle’s wheel or tyre. Hence, the simulation variable “degrees_per_sec” is set to 13°, in order for the behaviour to closely emulate the real world.

Also, “scale value” is indicative of the measurements used in creating the simulation world as to what degree values in the code have to be divided in order for the dimensions to be same as the real-world measurements. A scale value of one ‘1’ implies the real world. The “degree_per_sec” variable is also converted to “angular speed” in radian as that is demanded in ROS command velocity twist message.

Table 3.2. Algorithm II of the Autonomous Navigation for Robotaksi

```
Initialize:  Set scale = scale value
             Degrees_per_sec = 13
     Input:  Get lidar data corresponding to all sectors.
   Compute:  Calculate angular_speed = degrees_per_sec*2*PI/360
        If   (LiDAR top > 1.7/scale ):
             Start vehicle motor or throttle
             Simulation_drive = vehicle speed
             if (LiDAR_top <  (5.6/sim_scale)) and LiDAR_top_left > (6.6/sim_scale):
                 Turn steering full left
                 Simulation turn = 3 * angular_speed
             elif ((LiDAR_left or LiDAR_top_left) <  (1.5/sim_scale)) and LiDAR_right <
             (4/sim_scale):
                 Turn steering full right
                 Simulation turn = -3 * angular_speed
             elif ((LiDAR_left or LiDAR_top_left) <  (2.1225/sim_scale)) and LiDAR_right <
             (4/sim_scale):
                 Turn steering slightly right
                 Simulation turn = -1 * angular_speed
             elif ((LiDAR_left) >  (2.4975/sim_scale)):
                 Turn steering full left
                 Simulation turn = 3 * angular_speed
             elif ((LiDAR_left) >  (2.31/sim_scale)):
                 Turn steering slightly left
                 Simulation turn = 2 * angular_speed
             elif ((LiDAR_left) >  (2.1225/sim_scale)):
                 Turn steering even more slightly left
                 Simulation turn = 1 * angular_speed
             elif (LiDAR_top <  (5.6/sim_scale)) and LiDAR_top_right > (6.6/sim_scale):
                 Turn steering full right
                 Simulation turn = -3 * angular_speed
             else:
                 Maintain course
                 Simulation turn = 0.0
      Else:
             Stop vehicle or apply brake
             Simulation_drive = 0.0
             Simulation turn = 0.0
```

As it can be seen in Algorithm II, in turning left the left sector of the LiDAR data is considered more and favored in the decision-making process.

3.6. Decision Flow Chart

The flow chart of the overall algorithm is depicted in stages representing perception, planning and control below. The arrow represents the data connection of each stage with the next.

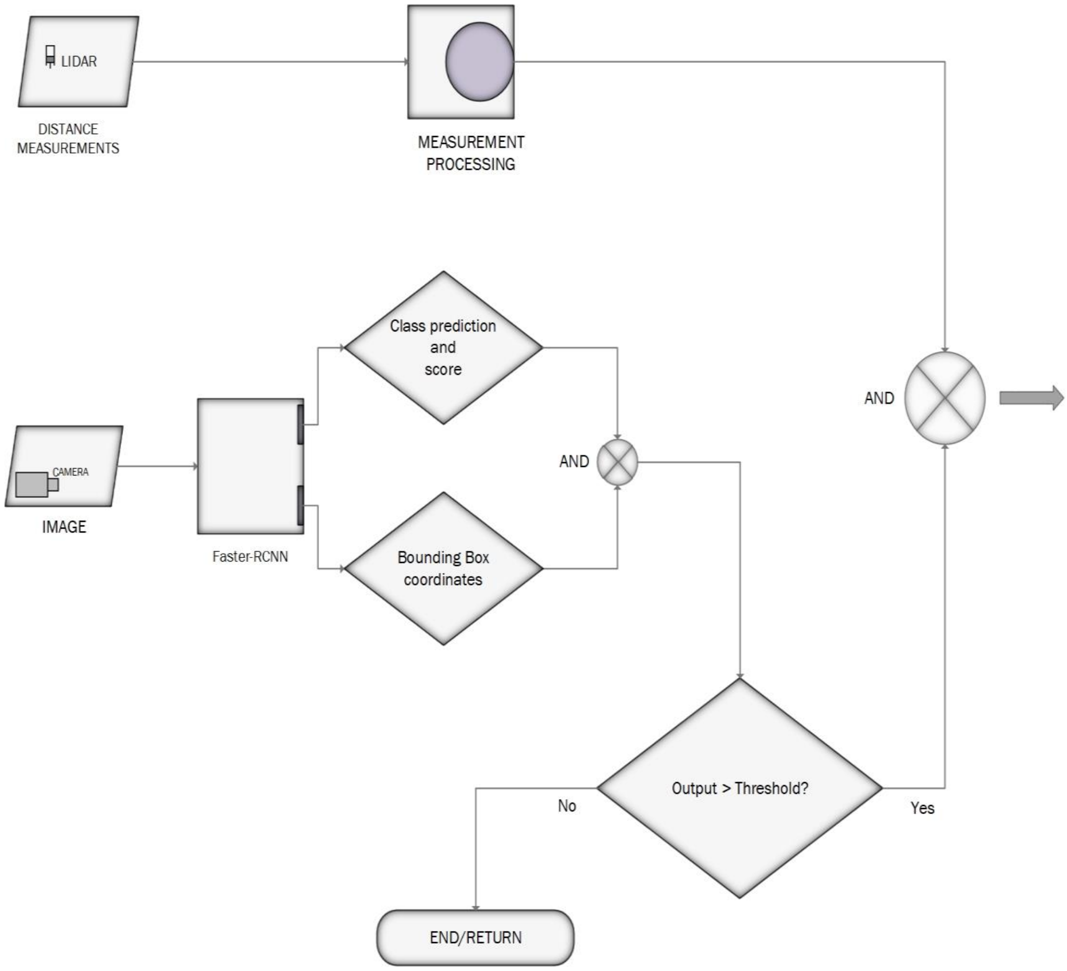


Fig. 3.6. Process flow chart (Perception)

Image from the camera is passed to the Faster-RCNN model for detection and perimeter of the bounding boxes is calculated for an approximate measure of the object's closeness. Simultaneously, the data from the lidar is also combined to make a definite decision on the path to go and how to avoid barriers while at it. These decisions are further broken down into five '5' logic functions and explained as they make up the planning stage.

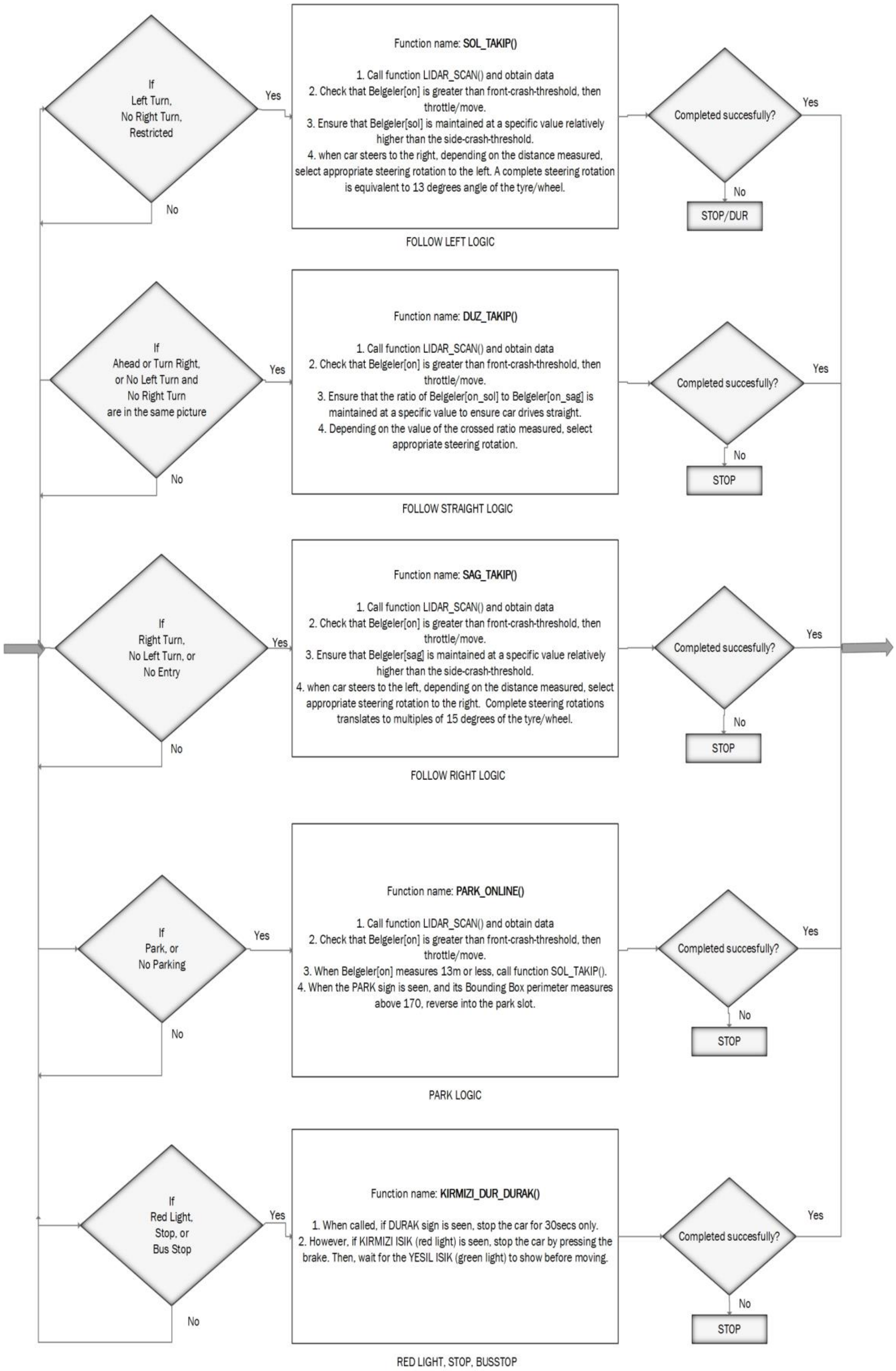


Fig. 3.7. Process flow chart (Planning)

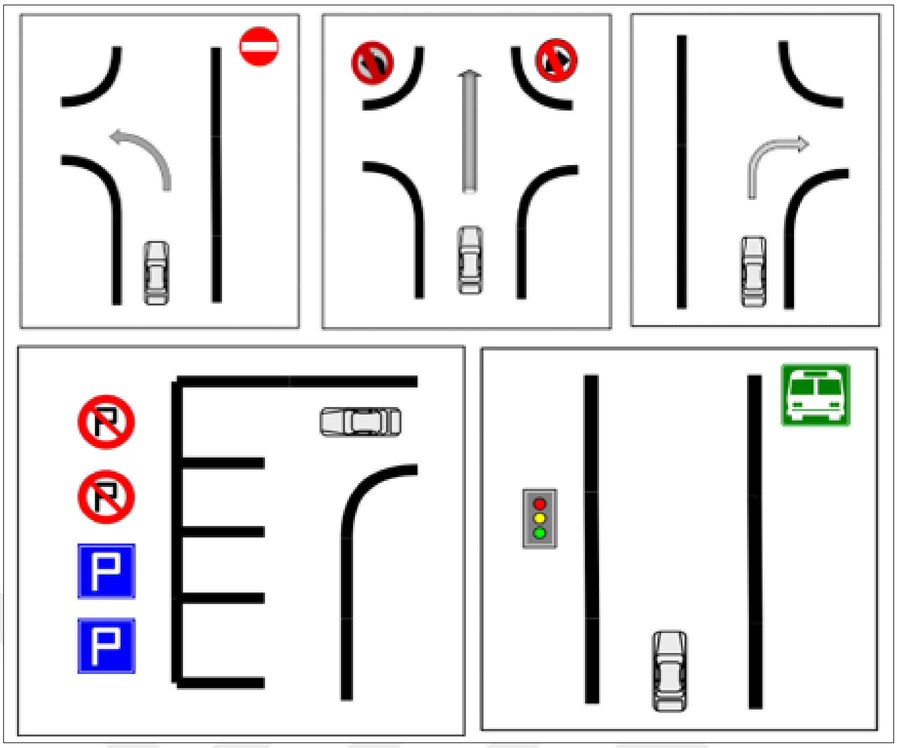


Fig. 3.8. Scenario representation of the respective algorithm functions

The output of the planning stage is the control commands via the Arduino aimed at throttling, braking or steering the vehicle.

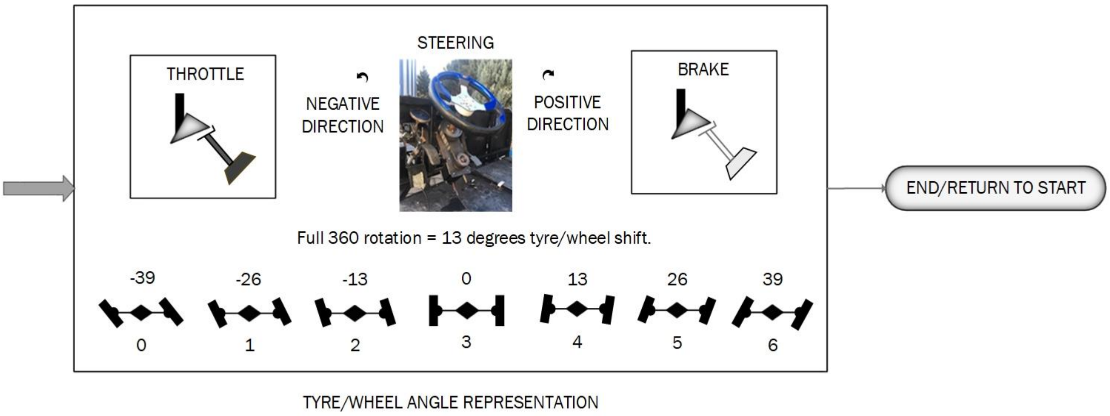


Fig. 3.9. Process flow chart (Control)

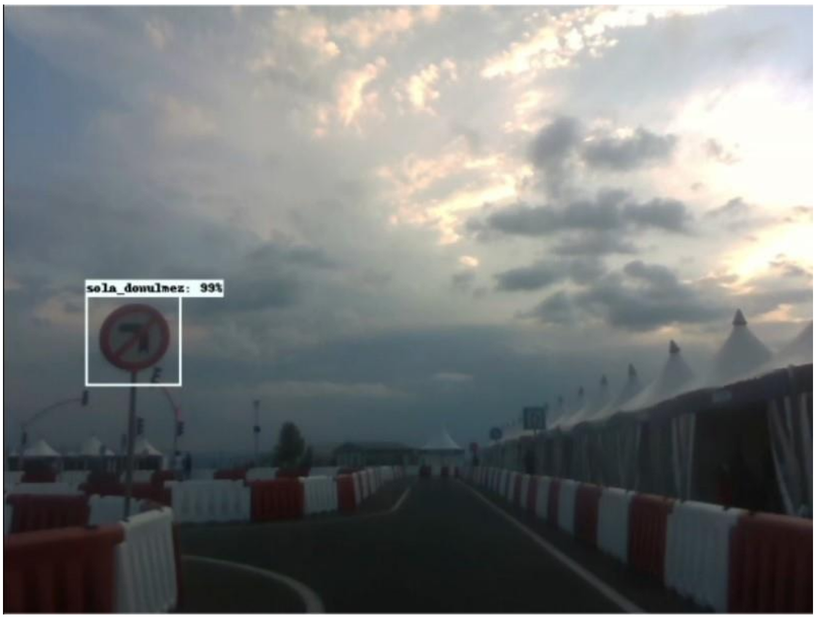
The “end/return to start” block represents a loop as the whole process begins all over once an action is executed while the “stop/dur” block represents the point where the code terminates as the vehicle has detected a barrier that it will crash into if it proceeds. These code modules were integrated by the KOU Mekatronom team into the vehicle system as the algorithm was deployed.

Fig. 4.0. The KOU Mekatronom team (2021)

In the next chapter, the algorithm is tested and the results are collected and analyzed in order to evaluate its feasibility.

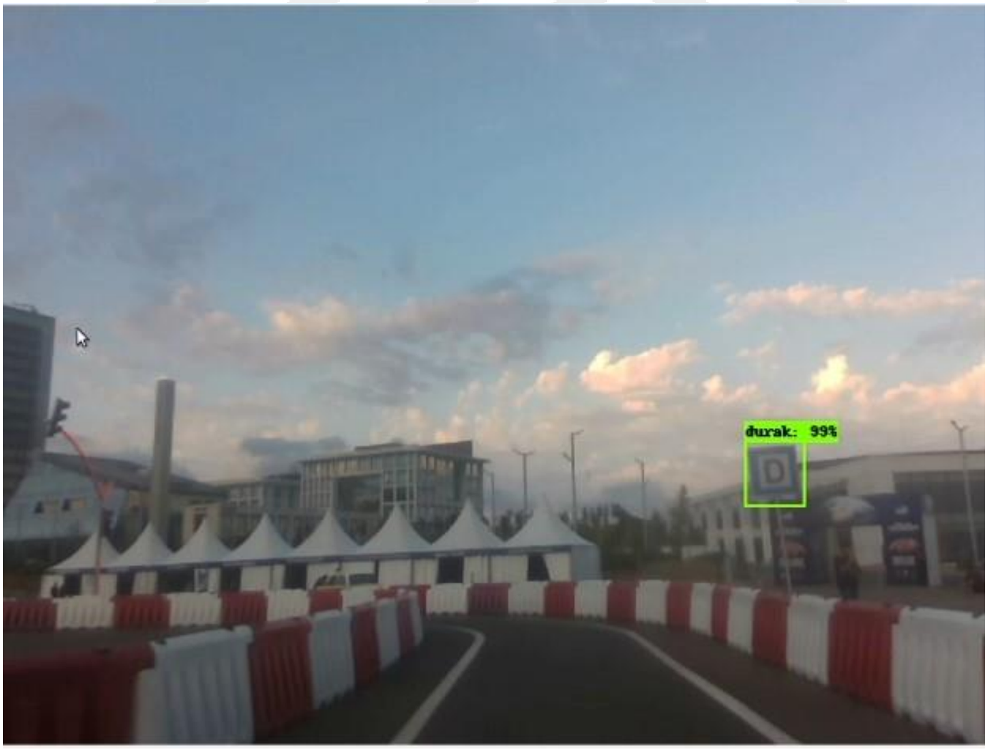
4. EXPERIMENTAL RESULTS

Simulations were carried out using ROS and Gazebo. The "mybot" ROS package, equipped with a LiDAR and a camera, was used for the simulation. The simulated robot required two entries: the linear velocity of the robot and its angle or orientation, as it can be seen in Table 3.2 -algorithm II-. Positive and negative angles implied clockwise and anti-clockwise robot direction, respectively. The wheel/tyre angles discussed in Section 3.4 were fed directly to the simulator, depending on the LiDAR data returned. The main algorithm was written in Python and run-on Ubuntu while Arduino maintains its C++ language. ROS publishes serial data via wired connection to Arduino in the form of characters signaling specific actions to be carried out. The camera was plugged directly to the host machine, after which traffic signs were showed to it and detected in real-time, a form of Hardware in the Loop (HIL) experience. The subscribers were the camera and the LiDAR as ROS constantly listened for changes in state for which certain functions are then called. The ROS spin function is used to keep Python from exiting until these nodes are stopped manually or forced within the code and since they are not, effectively it puts their activities in a loop. Gazebo provided an effective world drawing with scaled dimensions of object to the real-world counterpart. The various scenarios tested for are depicted below, the real-world test is shown in Figure 4.1 and 4.2 while the Gazebo simulated world is shown in Figure 4.3.



(a)

(b)



(c)

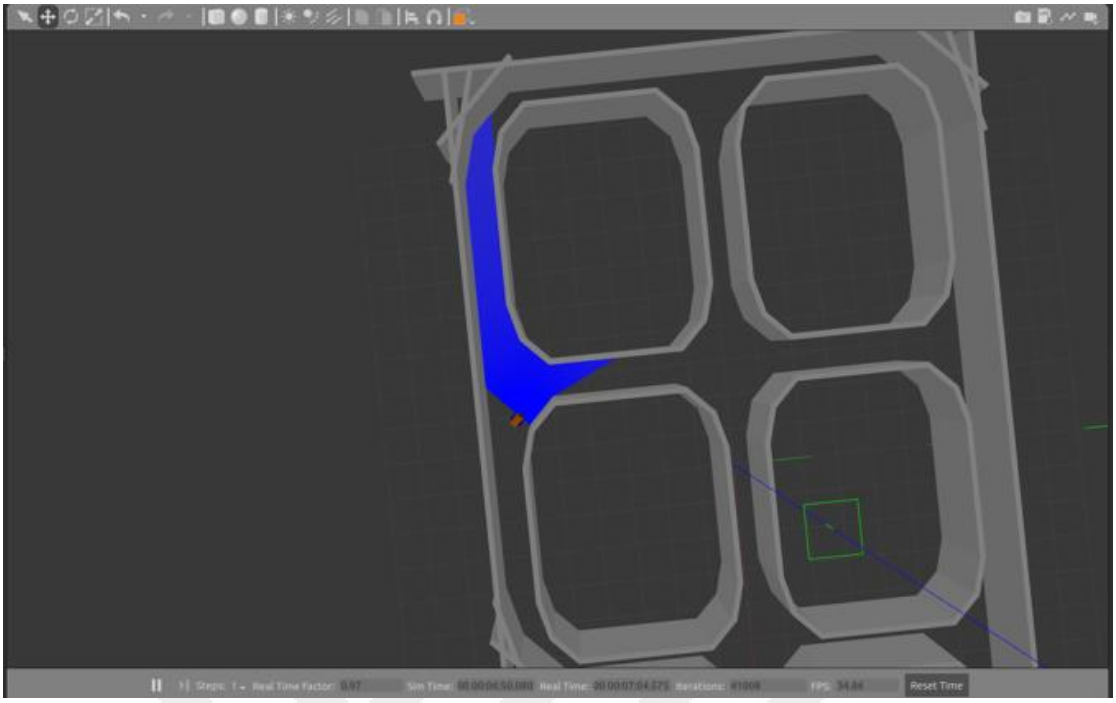
Fig. 4.1. (a) Real-time detection of no left turn at Teknofest (b) Park, and no parking (c) Bus stop

(a)

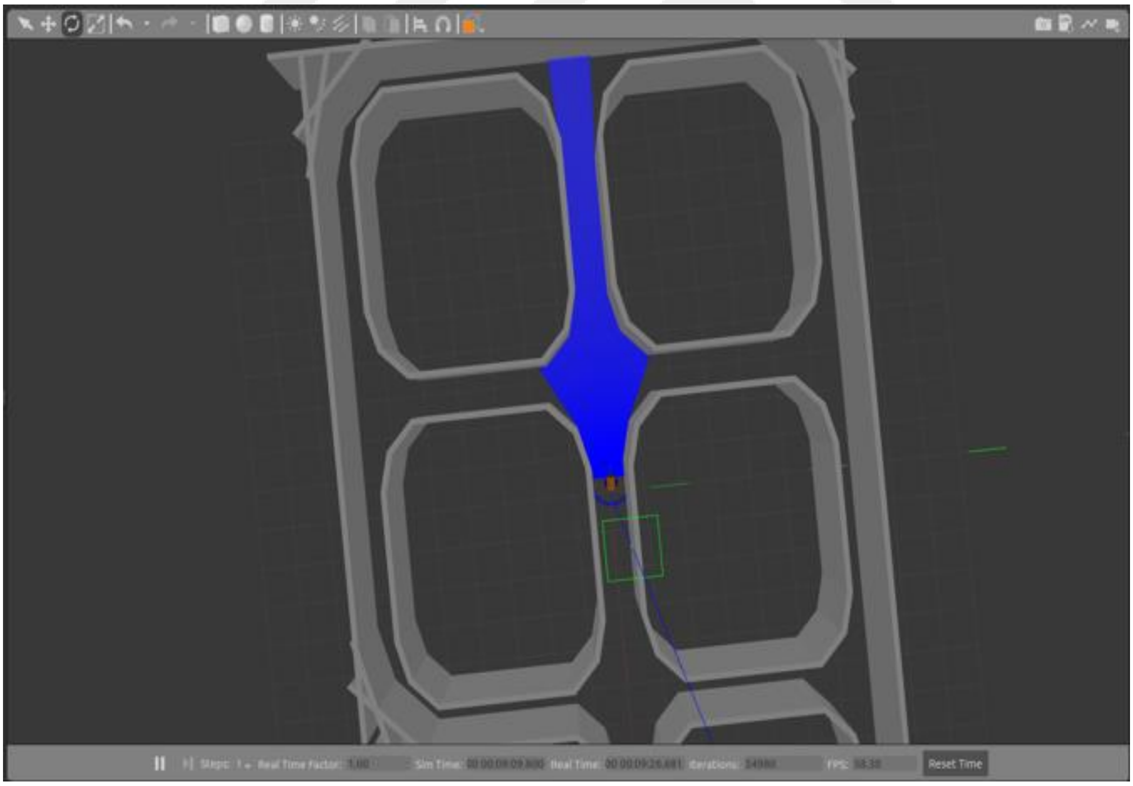
(b)

(c)

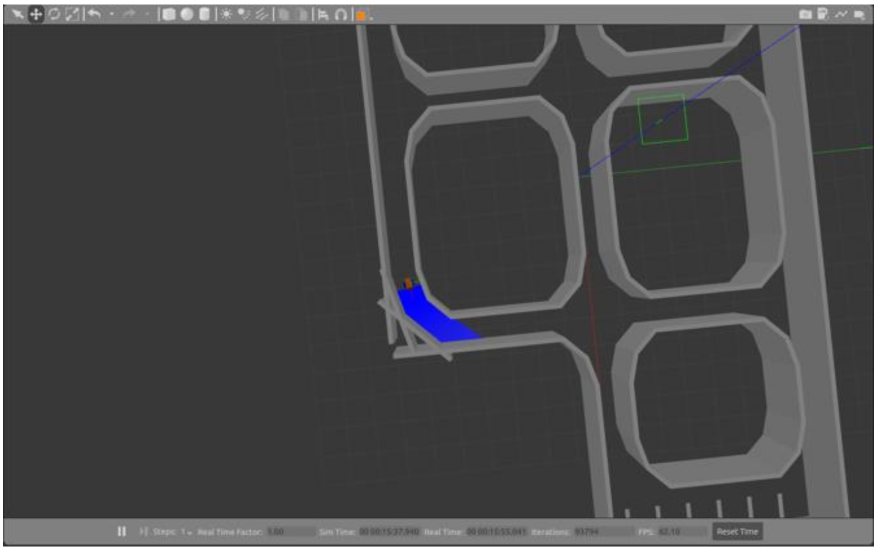
Fig. 4.2. (a) Right turn command in real-time control safety level verification (b) Moving straight (c) Left turn



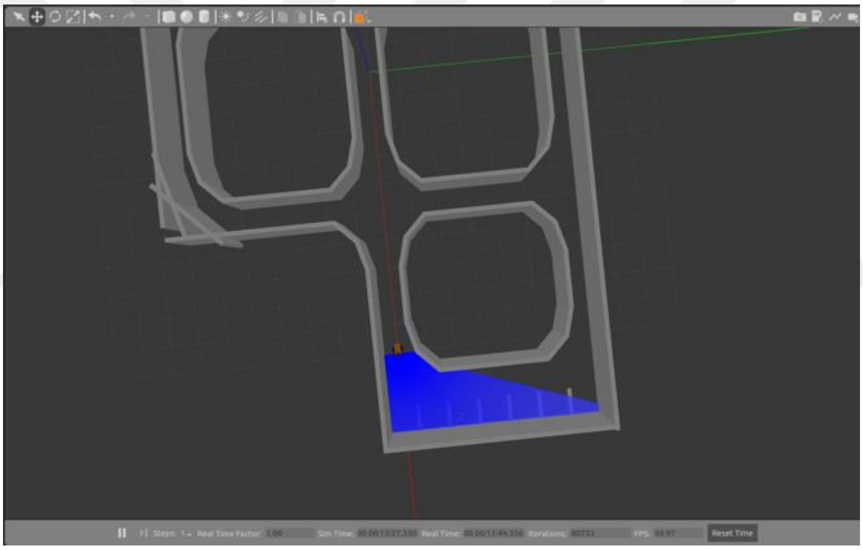
(a)



(b)



(c)



(d)

Fig. 4.3. (a) Right turn in Gazebo/ROS simulation (b) Moving straight at the intersection (c) Edge turn (d) Left turn preparation

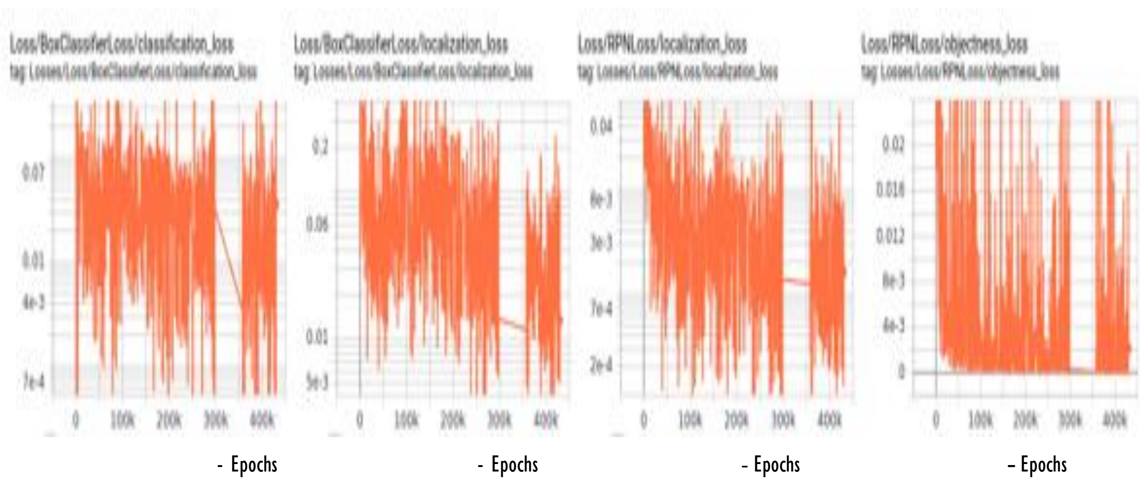
4.1. Performance Evaluation and Discussion

The vehicle's performance was checked over the obstacle course with various changes to the placement of the traffic signs over each test. The methodology outlined in Section 2 was followed and algorithm I is run, with its performance evaluated under two categories: The ability of the vehicle to obey the traffic rules, as well as its ability complete the course successfully while prioritizing safety. With the former, solely dependent on the accuracy of the model and the later on the algorithm. If in a trial, the vehicle is not able to achieve either, it is said to have failed that trial. Accordingly, the statistical success-ratio or success rate (SR) giving in Equation (4.1), defined here as the number of successful trails over the total number of trials is selected for this study.

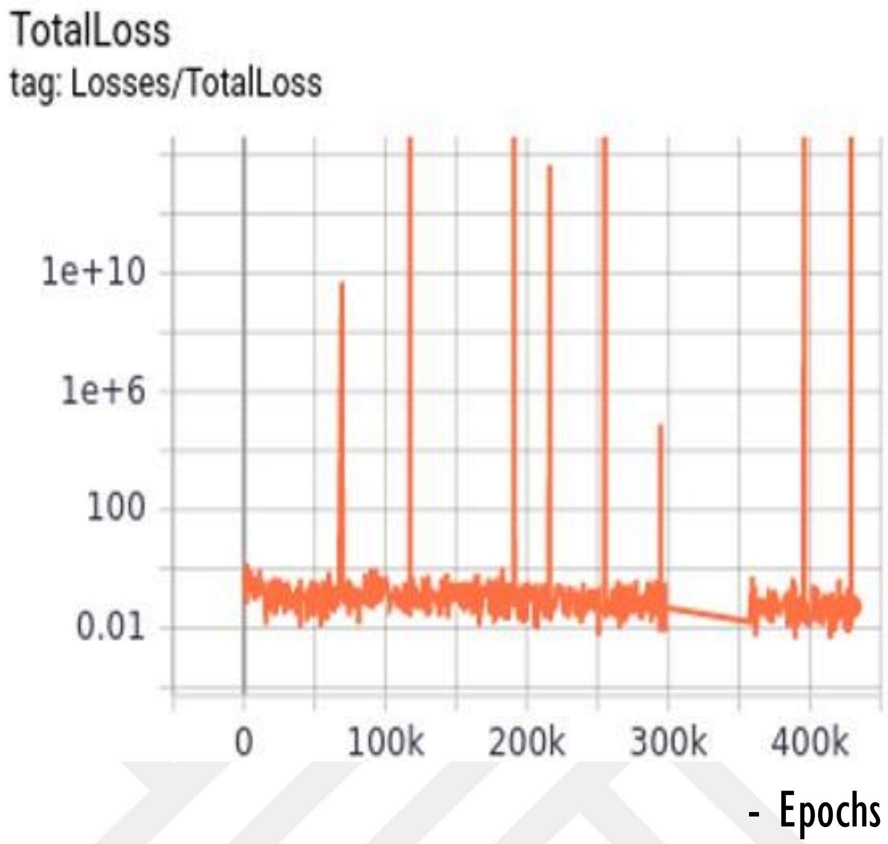
$$success\ rate = \frac{number\ of\ success}{(total\ number\ of\ trials\,)} \tag{4.1}$$

The SR across 20 simulated trials is evaluated as 0.8. With the bulk of the failed trials resulting from the algorithm's inability to successfully park the car. This is understandably so, as the concept of depth is not being registered and as such object sizes were used in approximating their distances. Overall, improvements can be achieved by including a lane detection technology using deep learning or edge detection with OpenCV so that the car is better centered on the road or parking slot. Cameras with depth sensors can be used to obtain the perceptive distance of object to camera, translated, to ultimately improve the park planning process.

Furthermore, with regards to the configuration properties of the CNN model, it was observed that deeper model with many convolutional layers imply a high number of training parameters and as such require high training time. However, with minimum loss value. This was noted as 429243 steps over 3-days yielded a corresponding loss value of 0.0097, shown in Figure 4.4.



(a)



(b)

Fig. 4.4. (a) Training localization and classification losses (y-axis) against training epoch (x-axis) (b) Total loss (y-axis) against training epoch (x-axis)

In addition, more images could be added to the train set in order to improve the learned weights and by extension, accuracy of detection. Images, both for training and testing, could also be cleaned by applying a Gaussian filter to remove noise before being subsequently labelled or detected. This will no doubt raise the accuracy of the detection. The detection model is often evaluated using mean average precision (mAP) at a specified IoU value. This is analogous to finding the area under the precision-recall curve for each class of object and then averaging it. At IoU of 0.5, Equation (4.2) and (4.3) is applied to the respective precision and recall values of each class, as seen in Table 4.1. With these values having been obtained from the confusion matrix, shown in Table 4.3. The confusion matrix, although used for evaluating the performance of classification models relative to some ground truth data, can be tailored for detection models by adding an extra class signifying "nothingness" for when images contain more than one object class or generally contains other detections with a lesser IoU value.

Table 4.1. Precision and Recall values per class

	Category	Precision@0.5IOU	Recall@0.5IOU
0	bus_stop	1	1
1	restricted	0.969879518	0.981707317
2	no_right_turn	0.994949495	1
3	no_left_turn	0.928654971	0.998742138
4	green_light	0.972972973	0.993865031
5	red_light	0.922666667	0.997118156
6	park	0.998830409	0.998830409
7	no_parking	0.982725528	0.998050682
8	stop	0	0
9	mandatory_right_ahead	0.984848485	0.994897959
10	mandatory_left_ahead	0.99103139	0.995495495
11	ahead_or_turn_right	0.974264706	1

$$Precision = \frac{TP}{TP+FP} \tag{4.2}$$

$$Recall = \frac{TP}{TP+FN} \tag{4.3}$$

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{4.4}$$

where, the TP, TN, FP, FN; indicating true positive, true negative, false positive, and false negative, respectively. TP implies the correct prediction of a class relative to ground truth or IoU greater than threshold, while TN represents a correct prediction of the absence of a class. FP denotes a false prediction of a class relative to ground truth or IoU value less than set threshold and FN points out that we predicted no class for a present class. For detection, the "nothingness" class is used to account for both FP (last row) and FN (last column), Table 4.3. Therefore, predictions that have a correct predicted label but not enough IoU go into the last row, whereas the predictions that have an incorrect predicted label go into the corresponding (ground truth, predicted) index of the matrix. An example is shown in Table 4.2 for a per class basis below.

Table 4.2. Confusion Matrix format for a sample class

Actual	Class1	True Positive	False Negative
	Nothingness	False Positive	True Positive
		Class1	Nothingness
		Prediction	

Using Eq.4 over the detected classes, the proposed model achieves an accuracy of 95.8%, with a total of 2521 images. Although most images contained more than one class of object, no images contained the “stop” sign.

Table 4.3. Confusion Matrix for the trained detection model

Category	Confusion Matrix:												
bus_stop	187.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
restricted	0.	322.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	6.
no_right_turn	0.	0.	394.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
no_left_turn	0.	0.	0.	794.	0.	0.	0.	0.	0.	0.	0.	0.	1.
green_light	0.	0.	0.	0.	648.	0.	0.	0.	0.	0.	0.	0.	4.
red_light	0.	0.	0.	0.	0.	346.	0.	0.	0.	0.	0.	0.	1.
park	0.	0.	0.	0.	0.	0.	854.	0.	0.	0.	0.	0.	1.
no_parking	0.	0.	0.	0.	0.	0.	0.	512.	0.	0.	0.	0.	1.
stop	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
mandatory_right_ahead	0.	0.	0.	0.	0.	0.	0.	0.	0.	390.	0.	0.	2.
mandatory_left_ahead	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	221.	0.	1.
ahead_or_turn_right	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	265.	0.
nothingness	0.	10.	2.	61.	18.	29.	1.	9.	0.	6.	2.	7.	53.

The simulated behavior and the real-world test were found to be alike. With dimensions in the simulation only being 4.285 times smaller than real-world objects counterpart. This scale value could be re-calculated if lane or vehicle dimensions changes in the future as it serves as a variable within the code which could be initialized. This ensures code re-usability as well as better synergy between simulation and real-world performance.

5. CONCLUSION

A deep learning-based solution was proposed to the completion of a barriered obstacle course. The obstacle course's dimensions and measurements have been given by Teknofest and a scaled replica was constructed within Gazebo.

Various known algorithms like the use of reinforcement learning, end-to-end learning with convolutional neural networks (CNN) as well as tree-based networks were evaluated and their performances contrasted before a final algorithm was proposed.

The proposed algorithm uses the Faster RCNN model to detect the images obtained from the front-facing camera. 12 classes were learned, with an overall loss value of 0.0097. Based on detected class probabilities of the traffic signs, the result with the highest probability is used to initiate the necessary control command via a decision tree-based logic generated with the LiDAR to safely navigate the vehicle through the obstacle course. The control commands are issued via the Arduino with well-defined steering rotation angles for which the desired wheel/tyre behavior is obtained.

With the use of the algorithm's scale variable in adjusting from simulated to real life measurements, the study was successful in achieving its objectives and tasked to help win in the competition or at least finish the course. The success rate of the algorithm was evaluated as 0.8 on a scale from 0 to 1, over 20 tests.

Although deemed successful, further improvements to the algorithm are proposed. Should an updated system be desired in the near future, a line following or lane detection algorithm using Hough Transform may be included to replace the decision tree-based logic as the LiDAR measurement is heavily dependent on barriered obstacle course. A line following logic that alternates between left and right lane views when path have been decided by the CNN.

In addition, to improve the accuracy of detection, more images could be added to the train set. Also, both train and test images, could be cleaned by applying a Gaussian filter to remove noise before being subsequently labelled or detected.

Also, currently the size of the detection bounding box is used in approximating the object's distance from the vehicle. Although, accurate in the driving process, it yields varying performance when it comes to parking the vehicle. This solely contributed to the 0.8 score in its success rate due to its inability to park properly. Cameras equipped with distance or depth sensors can be used in obtaining actual object or traffic signal distance and as such yield a better parking experience.

The image test-train set used as well as raw codes and trained model have been made available on my GitHub repository for further contributory access or usage. In this study, the basics of autonomy which includes perception, planning and control were implemented, however simplistic in its approach.

REFERENCES

Teknofest Aerospace and Technology Festival. (2020). *Robotaxi-Full Scale Autonomous Vehicle Competition Specification*. Teknofest. https://cdn.teknofest.org/media/upload/userFormUpload/ROBOTAXI-AUTONOMOUS_VEHICLE_COMPETITION_SPECIFICATION_v5_DNKOY_gnj0P.pdf.

Arinaldi, A., Pradana, J. A., & Gurusinga, A. A. (2018). Detection and classification of vehicles for traffic video analytics. *Procedia Computer Science*, *144*, 259–268. https://doi.org/10.1016/J.PROCS.2018.10.527.

Badioze Zaman, H., Robinson, P., Smeaton, A. F., Shih, T. K., Velastin, S., Terutoshi, T., Jaafar, A., & Mohamad Ali, N. (2017). Advances in visual informatics: 5th International Visual Informatics Conference, IVIC 2017, Bangi, Malaysia, November 28-30, 2017, Proceedings. *Springer*. http://link.springer.com/10.1007/978-3-319-70010-6.

Belletti, F., Haziza, D., Gomes, G., & Bayen, A. M. (2018). Expert Level Control of Ramp Metering Based on Multi-Task Deep Reinforcement Learning. *IEEE Transactions on Intelligent Transportation Systems*, *19*(4), 1198–1207. https://doi.org/10.1109/TITS.2017.2725912.

Boyan, J. A., & Moore, A. W. (1995). Generalization in Reinforcement Learning: Safely Approximating the Value Function. *Advances in Neural Information Processing Systems 7 (NIPS 1994)*, *7*.

Chen, P., & Elangovan, V. (2020). Object sorting using faster R-CNN. *International Journal of Artificial Intelligence & Applications*, *11*(6), 27–36. https://doi.org/10.48550/arxiv.2012.14840.

Chopra, R., & Roy, S. S. (2020). End-to-End Reinforcement Learning for Self-driving Car. *Advances in Intelligent Systems and Computing*, *1082*, 53–61. https://doi.org/10.1007/978-981-15-1081-6_5.

Eksi, O. T., & Gokmen, G. (2020). Comparing of Some Convolutional Neural Network (CNN) Architectures for Lane Detection. *Balkan Journal of Electrical and Computer Engineering*, *8*(4), 314–319. https://doi.org/10.17694/BAJECE.752177.

El Sallab, A., Abdou, M., Perot, E., & Yogamani, S. (2017). Deep Reinforcement Learning framework for Autonomous Driving. *IS and T International Symposium on Electronic Imaging Science and Technology*, 70–76. https://doi.org/10.2352/issn.2470-1173.2017.19.avm-023.

Ho, S. Y. (2018). *Reinforcement learning for self-driving cars | NTU Singapore*. Nanyang Technological University, SCSE Student Reports (FYP/IA/PA/PI). https://dr.ntu.edu.sg/handle/10356/74098.

Kebria, P. M., Khosravi, A., Salaken, S. M., Nahavandi, S., Kebria, P. M., Khosravi, A., Salaken, S. M., & Nahavandi, S. (2020). Deep Imitation Learning for Autonomous Vehicles Based on Convolutional Neural Networks. *IEEE/CAA Journal of Automatica Sinica, 2020, Vol. 7, Issue 1, Pages: 82-95*, *7*(1), 82–95. https://doi.org/10.1109/JAS.2019.1911825.

Liu, Z., Wang, K., Yu, J., & He, J. (2020). End-to-end control of autonomous vehicles based on deep learning with visual attention. *2020 4th CAA International Conference on Vehicular Control and Intelligence, CVCI 2020*, 584–589. https://doi.org/10.1109/CVCI51460.2020.9338558.

Meftah, L. H., & Braham, R. (2020). A virtual simulation environment using deep learning for autonomous vehicles obstacle avoidance. *Proceedings - 2020 IEEE International Conference on Intelligence and Security Informatics, ISI 2020*. https://doi.org/10.1109/ISI49825.2020.9280513.

Milstein, A., Nicolás Sánchez, J., & Williamson, E. T. (2002). Robust Global Localization Using Clustered Particle Filtering. *American Association for Artificial Intelligence (AAAI/IAA)*. www.aaai.org.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature 2015 518:7540*, *518*(7540), 529–533. https://doi.org/10.1038/nature14236.

Padhy, R. P., Verma, S., Ahmad, S., Choudhury, S. K., & Sa, P. K. (2018). Deep Neural Network for Autonomous UAV Navigation in Indoor Corridor Environments. *Procedia Computer Science*, *133*, 643–650. https://doi.org/10.1016/J.PROCS.2018.07.099.

Phon-Amnuaisuk, S., Murata, K. T., Pavarangkoon, P., Yamamoto, K., & Mizuhara, T. (2018). Exploring the Applications of Faster R-CNN and Single-Shot Multi-box Detection in a Smart Nursery Domain. *CoRR*, *abs/1808.08675*. https://doi.org/10.48550/arxiv.1808.08675.

Raj, M., & Narendra, G. (2021). Deep Neural Network approach for navigation of Autonomous Vehicles. *2021 6th International Conference for Convergence in Technology, I2CT 2021*. https://doi.org/10.1109/I2CT51068.2021.9418189.

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(6), 1137–1149. https://doi.org/10.48550/arxiv.1506.01497.

Rivera, Z. B., de Simone, M. C., & Guida, D. (2019). Unmanned Ground Vehicle Modelling in Gazebo/ROS-Based Environments. *Machines 2019, Vol. 7, Page 42*, *7*(2), 42. https://doi.org/10.3390/MACHINES7020042.

Siegwart, R., Nourbakhsh, I. R., & Scaramuzza, D. (2011). Introduction to autonomous mobile robots. In *Introduction to Autonomous Mobile Robots*. https://0-ieeexplore-ieee-org.oasis.unisa.ac.za/xpl/ebooks/bookPdfWithBanner.jsp?fileName=6277379.pdf&bkn=6267528&pdfType=chapter.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning 1988 3:1*, *3*(1), 9–44. https://doi.org/10.1007/BF00115009.

APPENDICES

APPENDIX-A

TABLE A.1. Complete algorithm for the Autonomous Vehicle

```
import rospy
from rospy.names import scoped_name
import time
from sensor_msgs.msg import LaserScan
from geometry_msgs.msg import Twist
from std_msgs.msg import String
import serial
Arduinodata = serial.Serial('/dev/ttyUSB1',115200)

def tabela(tabela_veri):

    global boyut
    global score
    global tabela_id
    global eski_tabela_id
    global tabela_data_length

    global sim_scale

    global durak                   # id: 1
    global girilmez                # id: 2
    global saga_donulmez           # id: 3
    global sola_donulmez           # id: 4
    global yesil_isik              # id: 5
    global kirmizi_isik            # id: 6
    global park                    # id: 7
    global park_edilmez            # id: 8
    global dur                     # id: 9
    global sag_don                 # id: 10
    global sol_don                 # id: 11
    global ileri_sag               # id: 12
    # id 14 = saga_donulmez True and sola_donulmez True

   tabela_data_length = str(tabela_veri)

    numbers = [int(word) for word in tabela_veri_uzunluk.split() if word.isdigit()]

    tabela_gelen_id = int(numbers[0])

    boyut = int(numbers[1])

    tabela_gelen_score = int(numbers[2])

    res =  max(set(eski_tabela_id), key=eski_tabela_id.count)
    print('element with heighest frequency: \n', res) #res = [ 1,2,2,4,1,1 ]

    if tabela_gelen_id == 1 and boyut>200 and tabela_gelen_score>9000:   #id: 1 'durak'
        if res == tabela_gelen_id:
            durak, girilmez, saga_donulmez, sola_donulmez = 'True','False','False','False'
            kirmizi_isik, park, park_edilmez,yesil_isik = 'False','False','False','False'
            sag_don, sol_don, ileri_sag, dur  = 'False','False','False','False'
            tabela_id = tabela_gelen_id
            score = tabela_gelen_score

    elif tabela_gelen_id == 2 and boyut>110 and tabela_gelen_score>9000:   #id: 2 'girilmez'
        if res == tabela_gelen_id:
            girilmez, durak, saga_donulmez, sola_donulmez = 'True','False','False','False'
            kirmizi_isik, park, park_edilmez,yesil_isik = 'False','False','False','False'
            sag_don, sol_don, ileri_sag, dur  = 'False','False','Fa          tabela_id = tabela_gelen_id
            score = tabela_gelen_score

    elif tabela_gelen_id == 3 and boyut>=150 and tabela_gelen_score>9000:   #id: 3 'saga_donulmez'
        if res == tabela_gelen_id:
            saga_donulmez, girilmez, durak, sola_donulmez = 'True','False','False','False'
```

```
        kirmizi_isik, park, park_edilmez,yesil_isik = 'False','False','False','False'
        sag_don, sol_don, ileri_sag, dur  = 'False','False','False','False'
        tabela_id = tabela_gelen_id
        score = tabela_gelen_score

elif tabela_gelen_id == 4 and boyut>=150 and tabela_gelen_score>9000:   #id: 4 'sola_donulmez'
    if res == tabela_gelen_id:
        sola_donulmez, girilmez, durak, saga_donulmez = 'True','False','False','False'
        kirmizi_isik, park, park_edilmez,yesil_isik = 'False','False','False','False'
        sag_don, sol_don, ileri_sag, dur  = 'False','False','False','False'
        tabela_id = tabela_gelen_id
        score = tabela_gelen_score

elif tabela_gelen_id == 5 and boyut>=170 and tabela_gelen_score>9000:   #id: 5 'yesil_isik'
    if res == tabela_gelen_id:
        sola_donulmez, girilmez, durak, saga_donulmez = 'False','False','False','False'
        kirmizi_isik, park, park_edilmez,yesil_isik = 'False','False','False','True'
        sag_don, sol_don, ileri_sag, dur  = 'False','False','False','False'
        tabela_id = tabela_gelen_id
        score = tabela_gelen_score

elif tabela_gelen_id == 6 and boyut>=170 and tabela_gelen_score>9000:   #id: 6 'kirmizi_isik'
    if res == tabela_gelen_id:
        sola_donulmez, girilmez, durak, saga_donulmez = 'False','False','False','False'
        kirmizi_isik, park, park_edilmez,yesil_isik = 'True','False','False','False'
        sag_don, sol_don, ileri_sag, dur  = 'False','False','False','False'
        tabela_id = tabela_gelen_id
        score = tabela_gelen_score

elif tabela_gelen_id == 7 and boyut>=110 and tabela_gelen_score>9000:   #id: 7 'park'
    if res == tabela_gelen_id:
        sola_donulmez, girilmez, durak, saga_donulmez = 'False','False','False','False'
        kirmizi_isik, park, park_edilmez,yesil_isik = 'False','True','False','False'
        sag_don, sol_don, ileri_sag, dur  = 'False','False','False','False'
        tabela_id = tabela_gelen_id
        score = tabela_gelen_score

elif tabela_gelen_id == 8 and boyut>=110 and tabela_gelen_score>9000:   #id: 8 'park_edilmez'
    if res == tabela_gelen_id:
        sola_donulmez, girilmez, durak, saga_donulmez = 'False','False','False','False'
        kirmizi_isik, park, park_edilmez,yesil_isik = 'False','False','True','False'
        sag_don, sol_don, ileri_sag, dur  = 'False','False','False','False'
        tabela_id = tabela_gelen_id
        score = tabela_gelen_score

elif tabela_gelen_id == 9 and boyut>110 and tabela_gelen_score>9000:   #id: 9 'dur'
    if res == tabela_gelen_id:
        sola_donulmez, girilmez, durak, saga_donulmez = 'False','False','False','False'
        kirmizi_isik, park, park_edilmez,yesil_isik = 'False','False','False','False'
        sag_don, sol_don, ileri_sag, dur  = 'False','False','False','True'
        tabela_id = tabela_gelen_id
        score = tabela_gelen_score

elif tabela_gelen_id == 10 and boyut>110 and tabela_gelen_score>9000:   #id: 10 'sag_don'
    if res == tabela_gelen_id:
        sola_donulmez, girilmez, durak, saga_donulmez = 'False','False','False','False'
        kirmizi_isik, park, park_edilmez,yesil_isik = 'False','False','False','False'
        sag_don, sol_don, ileri_sag, dur  = 'True','False','False','False'
        tabela_id = tabela_gelen_id
        score = tabela_gelen_score

elif tabela_gelen_id == 11 and boyut>110 and tabela_gelen_score>9000:   #id: 11 'sol_don'
    if res == tabela_gelen_id:
        sola_donulmez, girilmez, durak, saga_donulmez = 'False','False','False','False'
        kirmizi_isik, park, park_edilmez,yesil_isik = 'False','False','False','False'
        sag_don, sol_don, ileri_sag, dur  = 'False','True','False','False'
        tabela_id = tabela_gelen_id
        score = tabela_gelen_score

elif tabela_gelen_id == 12 and boyut>140 and tabela_gelen_score>9000:   #id: 12 'ileri_sag'
```

```
        if res == tabela_gelen_id:
            sola_donulmez, girilmez, durak, saga_donulmez = 'False','False','False','False'
            kirmizi_isik, park, park_edilmez,yesil_isik = 'False','False','False','False'
            sag_don, sol_don, ileri_sag, dur  = 'False','False','True','False'
            tabela_id = tabela_gelen_id
            score = tabela_gelen_score

    elif tabela_gelen_id == 14 and boyut>=110 and tabela_gelen_score>9000:   #id: 14 'sola_donulmez and saga_donulmez'
        if tabela_gelen_id in eski_tabela_id: #res == tabela_gelen_id:
            eski_tabela_id = [14, 14, 14] # skew res in favour of 14
            sola_donulmez, girilmez, durak, saga_donulmez = 'True','False','False','True'
            kirmizi_isik, park, park_edilmez,yesil_isik = 'False','False','False','False'
            sag_don, sol_don, ileri_sag, dur  = 'False','False','False','False'
            tabela_id = tabela_gelen_id
            score = tabela_gelen_score

    if boyut >= 110 and tabela_gelen_score>= 9000:
        eski_tabela_id.append(tabela_gelen_id)
        if len(eski_tabela_id) >= 6: #if equal to 6 pop the first one. so it will always be 5 everytime.
            eski_tabela_id.pop(0)



def func(scan):

    bolgeler = {

#       SIMULATION LIDAR
#        'on2':  min(min(scan.ranges[301:360]), 30),                #90 - 75 = 15 degrees
#        'on_kucuk_sag': min(min(scan.ranges[243:300]), 30),  #75 - 60.5 = 14.5 degrees
#        'on_buyuk_sag': min(min(scan.ranges[205:242]), 30),  #60.5 - 51 = 9.5 degrees
#        'on_sag': min(min(scan.ranges[147:204]), 30),           #51 - 36.5 = 14.5 degrees
#        'sag': min(min(scan.ranges[0:146]), 30),                    #36.5 degrees

#        'on1': min(min(scan.ranges[361:420]), 30),                 #105 - 90 = 15 degrees
#        'on_kucuk_sol': min(min(scan.ranges[421:478]), 30),    #119.5 = 105 + 14.5 degrees
#        'on_buyuk_sol': min(min(scan.ranges[479:516]), 30),    #129 = 119.5 + 9.5 degrees
#        'on_sol': min(min(scan.ranges[517:574]), 30),              #143.5 = 129 + 14.5 degrees
#        'sol': min(min(scan.ranges[575:720]), 30)

#        RPLIDAR S1
#        RPLIDAR S1 _cable_side
        'sag':  min(min(scan.ranges[511:613]), 40),                     #   306.5 degrees
        'on_sag': min(min(scan.ranges[614:675]), 40),                  #   14.5
        'on_buyuk_sag': min(min(scan.ranges[676:709]), 40),        #   9.5  #  600:620
        'on_kucuk_sag': min(min(scan.ranges[710:749]), 40),         #   14.5
        'on2':  min(min(scan.ranges[750:799]), 40),                     #    15 not so much but...

#       RPLIDAR S1 _right_of_cable_side
        'on1':  min(min(scan.ranges[770:848]), 40),                     # 'on1':
        'on_kucuk_sol': min(min(scan.ranges[849:881]), 40),         #   14.5
        'on_buyuk_sol': min(min(scan.ranges[882:912]), 40),         #  9.5
        'on_sol': min(min(scan.ranges[913:974]), 40),                  #   14.5
        'sol':  min(min(scan.ranges[975:1124]), 40)                     #    15 not so much but...

    }

    global boyut
    global score
    global tabela_id
    global eski_tabela_id
    global tabela_data_length

    global sim_scale
    global park_online_final
    global final_step
    global car_desired_dist_1
    global car_desired_dist_2
```

```
    global durak                       # id: 1
    global girilmez                    # id: 2
    global saga_donulmez               # id: 3
    global sola_donulmez               # id: 4
    global yesil_isik                  # id: 5
    global kirmizi_isik                # id: 6
    global park                        # id: 7
    global park_edilmez                # id: 8
    global dur                         # id: 9
    global sag_don                     # id: 10
    global sol_don                     # id: 11
    global ileri_sag                   # id: 12
    # id 14 = saga_donulmez True and sola_donulmez True

    global saga_takip

    print(' ')
    print(bolgeler)

    if durak == 'True' or dur == 'True': # should be fixed
        print('DURAK')
        saga_takip = 'False'
        kirmizi_dur_durak('D')
        sol_takip(bolgeler['on1'],bolgeler['on2'],bolgeler['sol'],bolgeler['on_sol'],bolgeler['sag'], bolgeler['on_sag'],'d')

    elif girilmez == 'True':
        print('Girilmez')
        saga_takip = 'False'
        if ((bolgeler['on_buyuk_sol']-bolgeler['on_buyuk_sag']>(3/sim_scale)) and (bolgeler['on_sol']>bolgeler['on_sag'])):
            print('Girilmez - Sol Takip')
            sol_takip(bolgeler['on1'],bolgeler['on2'],bolgeler['sol'],bolgeler['on_sol'],bolgeler['sag'], bolgeler['on_sag'],'g')

        elif ((bolgeler['on_buyuk_sag']-bolgeler['on_buyuk_sol']>(3/sim_scale)) and (bolgeler['on_sag']> bolgeler['on_sol'])):
            print('Girilmez - Sag Takip')
            sag_takip(bolgeler['on1'],bolgeler['on2'],bolgeler['sol'],bolgeler['on_sag'],bolgeler['sag'], bolgeler['on_sol'],'g')

        elif ((bolgeler['on1']< (15/sim_scale) or bolgeler['on2']<(15/sim_scale)) and (bolgeler['on_sag']>(10/sim_scale) and
bolgeler['on_sol']>(10/sim_scale))):
            print('Girilmez - Sag Takip')  # or it has left and right available and only forward is girilmez
            sag_takip(bolgeler['on1'],bolgeler['on2'],bolgeler['sol'],bolgeler['on_sag'],bolgeler['sag'], bolgeler['on_sol'],'g')

    elif (saga_donulmez == 'True' and sola_donulmez == 'True'):
        print('sola_saga_donulmez - DUZ TAKIP')
        duz_takip(bolgeler['on1'],bolgeler['on2'],bolgeler['sol'],bolgeler['on_sol'],bolgeler['sag'], bolgeler['on_sag'],'i')

    elif saga_donulmez == 'True' or sol_don == 'True':
        print('Saga Donulmez or SOL TAKIP')
        saga_takip = 'False'
        sol_takip(bolgeler['on1'],bolgeler['on2'],bolgeler['sol'],bolgeler['on_sol'],bolgeler['sag'], bolgeler['on_sag'],'s')

    elif sola_donulmez == 'True' or sag_don == 'True' or ileri_sag == 'True':
        print('Sola Donulmez or SAG TAKIP or ileri_sag')
        saga_takip = 'False'
        sag_takip(bolgeler['on1'],bolgeler['on2'],bolgeler['sol'],bolgeler['on_sag'],bolgeler['sag'], bolgeler['on_sol'],'s')

    elif yesil_isik == 'True':
        print('Yesil Isik - MOTOR ON - SAG TAKIP')
        saga_takip = 'False'
        Arduinodata.write('Y'.encode())
        sag_takip(bolgeler['on1'],bolgeler['on2'],bolgeler['sol'],bolgeler['on_sag'],bolgeler['sag'], bolgeler['on_sol'],'y')

    elif kirmizi_isik == 'True':
        print('Kirmizi Isik - MOTOR STOPPED')
        saga_takip = 'False'
        kirmizi_dur_durak('K')

    elif (park == 'True' or park_edilmez == 'True'): # size condition is already met.
        print('Park or Park edilmez')
        saga_takip = 'False'
        if park_online_final == 'False':
```

```
            if (bolgeler['sol'] < (4/sim_scale) ) and park == 'True' and boyut >= 155:
                if final_step == 'False':
                    car_desired_dist_1 = bolgeler['on1'] - 0.5 # real_car_length = 3.3, sim_car_length = 0.4
                    #car_desired_dist_2 = bolgeler['on2'] - 0.5
                    final_step = 'True'
                if (bolgeler['on1'] <= car_desired_dist_1 ):
                    kirmizi_dur_durak('K')     # stop. # something to stop simulation car too.
                    park_online_final = 'True' # then, initialize reverse.
                else:
                    sol_takip(bolgeler['on1'],bolgeler['on2'],bolgeler['sol'],bolgeler['on_sol'],bolgeler['sag'], bolgeler['on_sag'],'s')
            else:
                sol_takip(bolgeler['on1'],bolgeler['on2'],bolgeler['sol'],bolgeler['on_sol'],bolgeler['sag'], bolgeler['on_sag'],'s')
        else:
            Arduinodata.write('Y'.encode()) # start the vehicle again...
            park_online(bolgeler['on1'],bolgeler['on2'],bolgeler['on_kucuk_sol'],bolgeler['on_kucuk_sag']) # reverse and turn steering to the right

    else:
        #sag_takip(bolgeler['on1'],bolgeler['on2'],bolgeler['sol'],bolgeler['on_sag'],bolgeler['sag'], bolgeler['on_sol'],'s')
        #duz_takip(bolgeler['on1'],bolgeler['on2'],bolgeler['sol'],bolgeler['on_sol'],bolgeler['sag'], bolgeler['on_sag'],'i', bolgeler['on_buyuk_sag'],
bolgeler['on_buyuk_sol'])
        #sol_takip(bolgeler['on1'],bolgeler['on2'],bolgeler['sol'],bolgeler['on_sol'],bolgeler['sag'], bolgeler['on_sag'],'s')
        #park_online(bolgeler['on1'],bolgeler['on2'],bolgeler['on_kucuk_sol'],bolgeler['on_kucuk_sag'])
        print("waiting for camera")

##########################################   kirmizi_dur_durak   ###############################

def kirmizi_dur_durak(symbol):

    global milliseconds
    global count

    if symbol == 'D':
        Arduinodata.write('C'.encode())
        if (int(round(time.time() * 1000)) - milliseconds) > 100000:
            count = 0
        while count <= 30:
            count = count + 1
            milliseconds = int(round(time.time() * 1000)) #print(milliseconds)

    if symbol == 'K':
        Arduinodata.write('K'.encode())
        vel_msg.linear.x = 0.0
        vel_msg.angular.z = 0.0
        pub.publish(vel_msg)

######################################   park_online   #####################################

def park_online(bolgeler_on1, bolgeler_on2, bolgeler_on_kucuk_sol, bolgeler_on_kucuk_sag):

    global sim_scale

    # say 45 degrees maximum. hence, 360 degrees yields 45/3 but 360 is represented with 300 counters hence; 300 counts = 15 degrees
    PI = 3.1415926535897
    degrees_per_sec = 13 #(speed = degrees/sec) i.e. how many degrees does the steering turn per second
    angular_speed = degrees_per_sec*2*PI/360 #Setting angular.z = 1.0 gives a rotation of ~57 degrees per second. 1 radian ~= 57 degrees (180/PI).

    if (bolgeler_on1>(6.5/sim_scale)) or ( bolgeler_on_kucuk_sag>bolgeler_on_kucuk_sol):# and (bolgeler_on_kucuk_sol>(7.2/sim_scale) or
bolgeler_on_kucuk_sag>(7.2/sim_scale)) : # reverse and turn steering to the right
        print('REVERSE') # opposite direction
        Arduinodata.write('P'.encode()) # a new number is needed to turn A and reverse, i.e. P
        simulation_drive = -0.2
        simulation_turn = 3 * angular_speed

    else:
        print('DUR')
        Arduinodata.write('2'.encode())
        simulation_drive = 0.0
        simulation_turn = 0.0

    vel_msg.linear.x = simulation_drive
```

```
    vel_msg.angular.z = simulation_turn
    pub.publish(vel_msg)


########################################   duz_takip   ###################################

def duz_takip(bolgeler_on1, bolgeler_on2, bolgeler_sol, bolgeler_on_sol, bolgeler_sag, bolgeler_on_sag, symbol, bolgeler_on_buyuk_sag,
bolgeler_on_buyuk_sol):

    global sim_scale
    global saga_takip

    crossed_ratio = bolgeler_on_sag/bolgeler_on_sol
    print("crossed_ratio: ",crossed_ratio)
    # say 45 degrees maximum. hence, 360 degrees yields 45/3 but 360 is represented with 300 counters hence; 300 counts = 15 degrees
    PI = 3.1415926535897
    degrees_per_sec = 13 #(speed = degrees/sec) i.e. how many degrees does the steering turn per second
    angular_speed = degrees_per_sec*2*PI/360 #Setting angular.z = 1.0 gives a rotation of ~57 degrees per second. 1 radian ~= 57 degrees (180/PI).

    if saga_takip == 'True':
        sag_takip(bolgeler_on1, 0, bolgeler_sol, bolgeler_on_sag, bolgeler_sag, bolgeler_on_sol, symbol)

    else:
        if bolgeler_on1>(1.2/sim_scale):
            print('GAZ')
            Arduinodata.write('5'.encode())
            simulation_drive = 0.2 # linear vel in x_direction

            if (crossed_ratio >= (0.76) and crossed_ratio <= (1.3)) and (bolgeler_on1 > (7.6/sim_scale) ): # 5.6
                print('DUZ crossed')
                Arduinodata.write('4'.encode())
                simulation_turn = 0.0 # make is straight
                #saga_takip = 'False'

            elif (crossed_ratio < (0.76)) and (bolgeler_on1 > (5.6/sim_scale)):
                print('SOL : 1 crossed')
                Arduinodata.write('D'.encode())
                simulation_turn = 1 * angular_speed

            elif (crossed_ratio > (1.3)) and (bolgeler_on1 > (5.6/sim_scale)):
                print('SAG : 1 crossed')
                Arduinodata.write('F'.encode())
                simulation_turn = -1 * angular_speed

            elif (bolgeler_on1 < (7.6/sim_scale)) and (((bolgeler_on_sol > (6.6/sim_scale)) and (bolgeler_on_sag < (4.6/sim_scale))) or ((bolgeler_on_sag >
(6.6/sim_scale)) and (bolgeler_on_sol < (4.6/sim_scale)))) :
                print('Duz --> Sag takip 2')
                Arduinodata.write('2'.encode())
                simulation_turn = 0.0
                saga_takip = 'True'

            elif (bolgeler_on1 < (7.6/sim_scale)) and ((bolgeler_on_sol > (8/sim_scale)) and (bolgeler_on_sag > (8/sim_scale))) :
                print('Duz --> Sag takip 3')
                Arduinodata.write('2'.encode())
                simulation_turn = 0.0
                saga_takip = 'True'

            else:
                print('DUZ')
                Arduinodata.write('4'.encode())
                simulation_turn = 0.0

        else:
            print('DUR')
            Arduinodata.write('2'.encode())
            simulation_drive = 0.0
            simulation_turn = 0.0

        vel_msg.linear.x = simulation_drive
        vel_msg.angular.z = simulation_turn
```

```
    pub.publish(vel_msg)


########################################   sol_takip   ###################################

def sol_takip(bolgeler_on1, bolgeler_on2, bolgeler_sol, bolgeler_on_sol, bolgeler_sag, bolgeler_on_sag, symbol):

  global sim_scale

  # say 45 degrees maximum. hence, 360 degrees yields 45/3 but 360 is represented with 300 counters hence; 300 counts = 15 degrees
  PI = 3.1415926535897
  degrees_per_sec = 13 #(speed = degrees/sec) i.e. how many degrees does the steering turn per second
  angular_speed = degrees_per_sec*2*PI/360 #Setting angular.z = 1.0 gives a rotation of ~57 degrees per second. 1 radian ~= 57 degrees (180/PI).

  if bolgeler_on1>(1.7/sim_scale):

    print('GAZ')
    Arduinodata.write('5'.encode())
    simulation_drive = 0.2

    if (bolgeler_on1 < (5.6/sim_scale) ) and bolgeler_on_sol > (6.6/sim_scale) and symbol != 'g': # and bolgeler_on_sag < 3.0 and bolgeler_sag > 0.7:
      print('SOL : 3 __ q1')
      Arduinodata.write('A'.encode())
      simulation_turn = 3 * angular_speed

    elif (bolgeler_sol<(1.5/sim_scale) or bolgeler_on_sol<(1.5/sim_scale)) and bolgeler_sag < (4/sim_scale): # 2.1225
      print('SAG : 3 __ q3')
      Arduinodata.write('H'.encode())
      simulation_turn = -3 * angular_speed

    elif (bolgeler_sol<(2.1225/sim_scale) or bolgeler_on_sol<(2.1225/sim_scale)) and bolgeler_sag < (4/sim_scale) :
      print('SAG : 1 __ q2')
      Arduinodata.write('F'.encode())
      simulation_turn = -1 * angular_speed

    elif bolgeler_sol >(2.4975/sim_scale):
      print('SOL : 3 __ q4')
      Arduinodata.write('A'.encode())
      simulation_turn = 3 * angular_speed

    elif bolgeler_sol >(2.31/sim_scale):
      print('SOL : 2 __ q5')
      Arduinodata.write('S'.encode())
      simulation_turn = 2 * angular_speed

    elif bolgeler_sol>(2.1225/sim_scale):
      print('SOL : 1 __ q6')
      Arduinodata.write('D'.encode())
      simulation_turn = 1 * angular_speed

    elif (bolgeler_on1 < (5.6/sim_scale)) and bolgeler_on_sag > (6.6/sim_scale) and symbol != 'g': # and bolgeler_on_sol < 3.0 and bolgeler_sol > 0.7:
      print('SAG : 3 __ q7')
      Arduinodata.write('H'.encode())
      simulation_turn = -3 * angular_speed

    else:
      print('DUZ __ q8')
      Arduinodata.write('4'.encode())
      simulation_turn = 0.0

  else:
    print('DUR __ q9')
    Arduinodata.write('2'.encode())
    simulation_turn = 0.0
    simulation_drive = 0.0

  vel_msg.linear.x = simulation_drive
  vel_msg.angular.z = simulation_turn
  pub.publish(vel_msg)
```

```
##########################################   sag_takip   ######################################

def sag_takip(bolgeler_on1, bolgeler_on2, bolgeler_sol, bolgeler_on_sag, bolgeler_sag, bolgeler_on_sol, symbol):

    global sim_scale

    # say 45 degrees maximum. hence, 360 degrees yields 45/3 but 360 is represented with 300 counters hence; 300 counts = 15 degrees
    PI = 3.1415926535897
    degrees_per_sec = 14.1 #(speed = degrees/sec) i.e. how many degrees does the steering turn per second
    angular_speed = degrees_per_sec*2*PI/360 #Setting angular.z = 1.0 gives a rotation of ~57 degrees per second. 1 radian ~= 57 degrees (180/PI).

    if bolgeler_on1>(1.3/sim_scale):
        print('GAZ')
        Arduinodata.write('5'.encode())
        simulation_drive = 0.2

        if (bolgeler_on1 < (5.6/sim_scale) ) and bolgeler_on_sag > (6.6/sim_scale) and symbol != 'g': # and bolgeler_on_sag < 3.0 and bolgeler_sag > 0.7:
            print('SAG : 3 __ q1')
            Arduinodata.write('H'.encode())
            simulation_turn = -3 * angular_speed

        elif (bolgeler_sag<(1.5/sim_scale) or bolgeler_on_sag<(1.5/sim_scale)) and bolgeler_sol < (4/sim_scale): # 2.1225
            print('SOL : 3 __ q3')
            Arduinodata.write('A'.encode())
            simulation_turn = 3 * angular_speed

        elif (bolgeler_sag<(2.1225/sim_scale) or bolgeler_on_sag<(2.1225/sim_scale)) and bolgeler_sol < (4/sim_scale) :
            print('SOL : 1 __ q2')
            Arduinodata.write('D'.encode())
            simulation_turn = 1 * angular_speed

        elif bolgeler_sag >(2.4975/sim_scale):
            print('SAG : 3 __ q4')
            Arduinodata.write('H'.encode())
            simulation_turn = -3 * angular_speed

        elif bolgeler_sag >(2.31/sim_scale):
            print('SAG : 2 __ q5')
            Arduinodata.write('G'.encode())
            simulation_turn = -2 * angular_speed

        elif bolgeler_sag>(2.1225/sim_scale):
            print('SAG : 1 __ q6')
            Arduinodata.write('F'.encode())
            simulation_turn = -1 * angular_speed

        elif (bolgeler_on1 < (5.6/sim_scale)) and bolgeler_on_sol > (6.6/sim_scale) and symbol != 'g': # and bolgeler_on_sol < 3.0 and bolgeler_sol > 0.7:
            print('SOL : 3 __ q7')
            Arduinodata.write('A'.encode())
            simulation_turn = 3 * angular_speed

        else:
            print('DUZ __ q8')
            Arduinodata.write('4'.encode())
            simulation_turn = 0.0



    else:
        print('DUR')
        Arduinodata.write('2'.encode())
        simulation_turn = 0.0
        simulation_drive = 0.0

    vel_msg.linear.x = simulation_drive
    vel_msg.angular.z = simulation_turn
    pub.publish(vel_msg)
```

```
################################      stop simulation      #####################################

def force_stop_robot():
    rospy.loginfo("robot stopped!")
    Arduinodata.write('2'.encode())
    pub.publish(Twist())


####################################      main      ###########################################

if __name__ == "__main__":
    boyut = 0
    tabela_id = 0
    eski_tabela_id = [0]
    score = 0
    tabela_data_length = 0

    sim_scale = 1 # 4.285
    saga_takip, park_online_final, final_step = 'False','False','False'
    car_desired_dist_1, car_desired_dist_2, milliseconds, count = 0.0, 0.0, 0, 0

    durak, girilmez, saga_donulmez, sola_donulmez = 'False','False','False','False'
    kirmizi_isik, park, park_edilmez,yesil_isik = 'False','False','False','False'
    sag_don, sol_don, ileri_sag, dur  = 'False','False','False','False'

    rospy.init_node('OTONOM',anonymous=True)

    rospy.loginfo("To stop the robot use: CTRL + C")
    rospy.on_shutdown(force_stop_robot)

    rospy.Subscriber('tabela_topic', String, tabela)

    rospy.Subscriber('/scan',LaserScan,func)                              #rplidar

    #rospy.Subscriber('/mybot/laser/scan', LaserScan, func)          #sim
    pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)                 #sim
    vel_msg = Twist()                                                     #sim

    rospy.spin()
```

GLOSSARY

Turkish words used in the code are translated here, with sentences used also being a combination of some of these words.

Bolgeler	: Regions or sectors for which LiDAR’s data is divided.
Boyut	: Dimension or size of the image.
Buyuk	: Large or big.
Don	: Turn.
Donulmez	: No Turn.
Dur	: Stop.
Durak	: Bus stop.
Duz	: Straight.
Eski	: Old.
Girilmez	: No entry.
Ileri	: Forward.
Isik	: Light.
Kirmizi	: Red.
Kucuk	: Small.
On	: Front.
Otonom	: Autonomous.
Park_Edilmez	: No parking.
Sag	: Right.
Sol	: Left.
Tabela	: Sign.
Takip	: Follow.
Veri	: Data.
Yesil	: Green.

CONFERENCES AND PUBLICATION

Adebayo A. & Ertunç, H. M. (2022). The utilization of machine learning algorithms (deep convolutional neural network detector) in training a vehicle to navigate autonomously. *2nd International Symposium of Scientific Research and Innovative Studies (ISSRIS'22)*, pp.365-375. ISBN: 978-605-73639-1-6.

Adebayo, A. & Ertunç, H. M. (2022). Natural Navigation System Design for Indoor Mobile Robots. *Kocaeli Journal of Science and Engineering*, 5(1), 73-83. DOI: 10.34088/kojose.1012914.

RESUME

Azeez received his bachelors in mechanical engineering from the University of Ibadan, Nigeria in 2018. In the year 2019, he was awarded the Turkish government scholarship (YTB) for his master's degree in Mechatronics Engineering at Kocaeli University, where he currently studies. His research interests include machine learning, image and signal processing, as well as navigation and automatic control for mobile robots.