Autonomous Driving Support System with Image Processing Techniques

Onur Alaca, Fahriye Gemci*
Department of Computer Engineering, Kahramanmaras Sutcu Imam University, Kahramanmaras, Turkiye
*onuralcaa@gmail.com; *fahriyegemci@ksu.edu.tr*

***Abstract*—In this study, an autonomous driving support system is developed using image processing and machine learning techniques. The aim of the study is to create a driving support system that can perform basic autonomous driving functions on low-cost hardware. Within the scope of the project, important driving functions, such as lane following, red light detection, sign recognition, and obstacle recognition, are addressed. While the lane following algorithm guides the vehicle by keeping it in the lane within the camera angle, the red light detection algorithm, sign recognition and obstacle recognition algorithms aim to identify the situations that need to be performed and perform the necessary activities autonomously.**

Within the scope of the study, the images taken through a camera were processed using the C++ module of the OpenCV library and the specified tasks were performed using the necessary decision-making algorithms. The machine learning algorithms developed were also tested on a Raspberry Pi computer in real time and their performance was evaluated. In this way, the feasibility of autonomous driving technologies on a low-cost platform was examined, and successful results were obtained.

***Index Terms*—Autonomous vehicle; Haar cascade classifier; Lane keeping; Image processing; Machine learning.**

I. INTRODUCTION

There are always risks involved in driving. However, this effect can be minimized if dangers are known in advance and appropriate measures are taken. Since it is known that there cannot be a single cause of traffic accidents, it would be appropriate to focus on many factors. If these are analyzed well and healthy solutions can be produced, the damages caused by accidents for both the national economy and the people will be reduced [1].

According to research, 94 % (±2.2 %) of traffic accidents are caused by driver errors, 2 % (±0.7 %) - by vehicle problems, and 2 % (±1.3 %) - by environmental conditions, such as slippery roads and bad weather. On the other hand, 41 % (±2.2 %) of accidents caused by drivers were due to internal and external distractions, 33 % (± 3.7 %) were due to excessive speed, 11 % (±2.7 %) were due to speeding in corners, and 7 % (±1.0 %) were due to lack of sleep [2]. In addition, sleep deprived driving is seen to significantly increase the accident rate in vehicles [3]. With autonomous vehicle technology, deaths, injuries, and material losses resulting from serious vehicle accidents can be prevented by zeroing the number of accidents caused by driver error, which is as high as 94 %, by eliminating the human element in driving. Due to this technology, the transportation of elderly people will become easier, fuel consumption will be more optimal, and toxic exhaust gas emissions will be reduced [4]. Many companies continue to work on these issues [5].

Extensive research and development on autonomous vehicles [6] include topics such as location determination, mapping, object detection, lane detection, lane following, traffic sign detection, route optimization, vehicle control, accident and object avoidance, adaptive cruise control, and communication with other vehicles [7].

In this study, driving assistance systems, which are a sub-field of fully autonomous driving systems, are considered, and an autonomous driving assistance system that works with image processing techniques is developed. This system is designed to perform basic driving functions, such as lane following, red light detection, and vehicle distance tracking. The aim is to develop a driving support system using Raspberry Pi, a low-cost and accessible hardware, by integrating real-time image processing techniques with C++ [8] and OpenCV libraries, and to investigate the potential of this system to prevent traffic accidents.

The developed system aims to run complex driving support algorithms in the most efficient way with low-level hardware.

The lane tracking algorithm integrated into the system will ensure that the vehicle continues to move stably in the specified lane, while the traffic light detection function will determine the points where it should stop in traffic. The vehicle distance tracking algorithm will help maintain a safe distance from the vehicle in front, and the sign reading feature will detect traffic signs.

These features support the decision-making process and work towards minimizing human errors.

In the study, an autonomous vehicle system was developed employing image processing techniques, Haar cascade-based object detection, and PID control methods. In recent years, the implementation of such integrated systems in autonomous vehicles has attracted significant research interest due to their practical applicability and technological relevance.

In conclusion, this study is an important example of a driving assistance system developed on Raspberry Pi, a low-cost and accessible platform, to demonstrate the basic principles of autonomous driving technologies in a practical way. For future research, this study is expected to provide

Manuscript received 13 June, 2025; accepted 9 October, 2025.

academic and practical contributions to the development of autonomous driving systems.

II. RELATED WORK

– Hasani, Putra, and Setyawan (2022) [9]

Lane Detection and Steering Control: In this study [9], the transformation of the BGR to the RGB color space, the transformation of the perspective, the greying/thresholding, the detection of the Canny edge, and the analysis based on the histogram were used as image processing steps. In this way, the lanes of the road were detected from the camera image and the position (in pixels) of the lane relative to the vehicle camera was determined. The amount of deviation from the detected lane centre is given as input to a proportional-integral-derivative (PID) control system to steer the vehicle. The authors report that with appropriate PID parameters, vehicle steering control performance reaches steady state in about five seconds and has a position overshoot of only 1.5 cm. This result shows that precise lane following performance can be achieved with PID control even in a small-scale vehicle. In our project, although there is a similar lane following mechanism, the PID parameters are not directly mentioned. Probably a simpler proportional control or rule-based steering correction was applied. However, since in our project an additional method such as virtual lane creation is used in case of lane loss, this unique approach, which is not mentioned in the Hasani, Putra, and Setyawan study [9], may make our system more robust in lane following. On the other hand, in our project, a PID tuning and a numerical performance analysis as reported by Hasani, Putra, and Setyawan [9] were not performed (or not mentioned).

Traffic Sign Detection: Hasani, Putra, and Setyawan [9] used Haar cascade-based object recognition for traffic sign detection. It is stated that a statistical classifier was trained using 1000 positive and 1400 negative sample images for model training. Although the type of traffic signs detected is not specified (probably critical signs such as stop signs), they evaluated their detection performance in terms of distance and angle. As a result of static tests, a relationship was found between the distance between the camera and the sign and the pixel size in the image, 1 cm = 10 pixels. This indicates that the system can measure the position with a resolution of approximately 0.1 cm/pixel. Traffic signs could be detected at distances between 5 cm and 120 cm, with the optimum accuracy obtained in the range of 5 cm to 90 cm. In terms of viewing angle, sign detection was possible at angles between 10 ° and 140 ° relative to the camera axis, with the best results obtained at angles between 20 ° and 130 °. Dynamic tests (with the vehicle in motion) show that detection performance is highly dependent on these distance and angle factors, as well as the ambient lighting and conditions in the training data. Hasani, Putra, and Setyawan [9] reported an average overall traffic sign detection accuracy of 92.3 %.

Comparison: There is a great deal of similarity between the sign recognition part of the project under review and Hasani Putra, and Setyawan [9] approach, both of which used object recognition with Haar cascade. In our project, four different models (Stop, No Entry, red light, yellow light) were trained separately, while Hasani, Putra, and Setyawan [9] probably only recognised specific signs (e.g., “stop” sign) with one or two models. The 92 % accuracy achieved by the authors in [9] is quite high and is probably valid for a single type of sign under controlled conditions. Since different types of objects (both signs and traffic lights) were detected in our project, it may be more difficult to achieve similar accuracy for each of them; however, the project description states that all targets were successfully detected on the test track and the required response was given. Although traffic light detection is not mentioned in [9], our system also has the color recognition feature of traffic lights. In this respect, in addition to this work in the literature, our project extends the scope by providing light detection and appropriate stop/start decision mechanism.

Hardware and Platform: Although hardware details are limited in [9], some of the studies the authors cite point to the use of Raspberry Pi. Most likely, they also used a Raspberry Pi-based model vehicle or worked with a similar embedded system. In a study by Pannu, Ansari, and Gupta [10], which is included in the reference list of work in [9], it is reported that a prototype of a mini autonomous vehicle was developed using Raspberry Pi (Model B) and components, such as camera, motor drives, and ultrasonic sensor. In this system, monocular image processing was performed with a single camera, and lane following and obstacle detection were performed. Raspberry Pi was used to run multiple visual algorithms in real time with its low power and size advantage. Therefore, the hardware choices of our project are compatible with other prototypes in the literature. All of them place a Raspberry Pi on a DC motorised toy car chassis powered by a similar 5-V power supply, take images via USB or Pi-camera, and steer the vehicle with OpenCV-based operations. In [9], simple drivers (such as L293D) and extra sensors (such as ultrasonic) are typically used for motor control. In this context, in terms of hardware cost and components, our project is equivalent or simpler than its counterparts in the literature (for example, there is no ultrasonic obstacle sensor in our project, only camera data are sufficient).

– Pannu, Ansari, and Gupta (2015) [10]

“Design and Implementation of Autonomous Car using Raspberry Pi”: In this early work [10], a Raspberry Pi-based autonomous vehicle prototype is presented. The system aims to deliver the vehicle to the destination by performing lane following and obstacle detection with a single camera and ultrasonic sensor data. Basic autonomous driving features are provided by combining image processing with lane detection and ultrasonic with distance measurement. Comparison: The Pannu, Ansari, and Gupta [10] system does not include traffic sign recognition and is more focused on lane following and collision avoidance. The fact that our project also responds to traffic signs indicates that it targets a more comprehensive urban driving scenario compared to this prototype from 2015 [10]. In terms of hardware, both projects use a similar combination of a four-wheel mini chassis, Raspberry Pi, and camera.

– Tiwari and Singh (2017) [11]

“Vehicle Control Using Raspberry Pi and Image Processing”: In this study [11], a system implemented with Python-OpenCV on Raspberry Pi 3 is integrated into autonomous driving by detecting the STOP sign and the red traffic light. The Haar cascade technique is used to recognise the STOP sign, while color-based processes such as image

masking and contour detection are used for red light detection. In addition, the distance to the vehicle in front was measured with an ultrasonic sensor, and adaptive speed control was also performed. Comparison: The Tiwari and Singh [11] system is quite close to our project in terms of task: Stopping at a red light and stopping at a STOP sign in the same way.

– Vinothini and Jayanthy (2019) [12]

"Road Sign Recognition System for Autonomous Vehicle using Raspberry Pi": This study [12] focuses on the recognition of five different traffic signs for a Raspberry Pi-based autonomous vehicle. The authors have developed a system that successfully detects these five road signs in real time on a Raspberry Pi. The methodology used is referred to as the Haar cascade in the literature, but it is likely that customised machine learning techniques were used for specific signs. Comparison: The work of Vinothini and Jayanthy [12] is valuable in that it addresses the challenge of multiple traffic sign recognition in a Raspberry Pi environment. The STOP and No Entry signs and red/yellow light detection in our project are actually a subset of these five signs (presumably there were common signs like STOP in their work). However, while the authors in [12] focused only on sign recognition, our project realises both sign/light recognition and vehicle motion control (stopping, turning) according to this information. In other words, our project is more comprehensive in terms of integrating sensing output into vehicle control.

– Mohit, Kumar, Vats, and Sathyalakshmi (2020) [13]

"Analysing the Features of Self-Driving Cars Using Haar Classification Methodology": In this Indian study [13], the implementation of various autonomous driving features is demonstrated in a model car using the Haar cascade method. The aim is to present these features as a module that can be added to existing vehicles. In the model presented, it is stated that basic features such as obeying traffic signs (e.g., stop at the STOP sign) and lane following are realised with Haar-based sensors. The challenges of the prototype in Indian traffic scenarios (e.g., complex road conditions) are also discussed. The authors in [13] emphasised that the Haar cascade algorithm works successfully in basic tasks such as STOP signalling and lane detection. Comparison: This work [13] is methodologically parallel to our project; both have experimented with autonomous features in small-scale vehicles using classical image processing and Haar classifiers. Mohit, Kumar, Vats, and Sathyalakshmi [13] specifically addressed the challenges of localised conditions (light changes, different views of signs, etc.). Since our project is also tested on a real-world test track, it has dealt with similar practical challenges. Performance degradation is expected when, e.g., the illumination conditions or the scale of the cues do not match the data on which the Haar cascade model is trained - the authors in [13] address these issues, and our system probably has similar limitations. In [13], the challenges in localised conditions (light changes, different views of signs, etc.) are specifically addressed. Since our project is also tested on a real-world test track, it has dealt with similar practical challenges. However, whereas the work in [13] provides a more conceptual framework, our project provides a more practical solution with concrete rule responses (stop, turn) and unique additions such as virtual lanes.

III. CONTRIBUTION OF PAPER

– Hardware Cost and Field Applicability

While many studies [14] in the literature have been carried out with high-end systems (e.g., Nvidia Jetson, Intel NUC), in this project, only Raspberry Pi 4B and Arduino UNO were used for real-time implementation.

This is a strong contribution to the literature in terms of demonstrating that autonomous driving is possible with low-cost systems.

– Real-Time Simulation Test Track

While other studies [15] usually use simulation environments or ready-made datasets, here a completely real mini-test track was prepared and tapes, lights, and signs were installed.

The use of real field-tested hardware is an approach not often seen in academic projects.

– Separate Model Training

For STOP and NO ENTRY signs, the GTSRB dataset was used, while for red and yellow traffic lights, the original dataset was collected from scratch and Haar Cascade models were trained.

This is a more comprehensive image processing approach than studies in the literature, which generally focus only on sign recognition [16].

– Virtual Lane and Sharp Turn Algorithm

While there are mostly fixed turn decisions in the literature [17], in this project a PID controlled steering algorithm that optimises vehicle turns has been developed.

Especially in case of lane loss or at bends, a virtual lane estimation was used. This provides a more responsive and safer driving than conventional algorithms.

IV. MATERIAL AND METHODS

A. The Hardware Components

– Raspberry Pi 4B

Raspberry Pi is a product developed by the Raspberry Pi Foundation, a UK-based company that offers users an affordable and small-sized computer. It is basically a single board computer and has minimal hardware. Widely used in education, prototyping, and hobby projects, this device can run Linux-based operating systems and supports various programming languages. Thanks to its hardware flexibility and wide range of connectivity options, it can be easily integrated with sensors, cameras, and other electronic components in projects.

– Raspberry Pi Camera Module

A compact, high-resolution imaging unit designed specifically for Raspberry Pi devices. Usually with 8 MP or 12 MP resolution, it is suitable for still images and video capture up to 30 FPS. The camera module is used in projects for applications such as face recognition, image processing, and video streams. It easily connects to the Raspberry Pi via the CSI connector and performs various image processing tasks, working in harmony with libraries such as OpenCV.

– Arduino Uno R3 Microcontroller

Arduino Uno is an easily programmable microcontroller board that is widely used for electronic projects and prototype development. It is preferred in educational and hobby projects

and works compatible with various sensors and components.

B. Image Preprocessing Steps

– Image Preprocessing

The image from the camera is converted to greyscale using the cvtColor function of the OpenCV library.

OpenCV supports many color spaces and can convert between them. There is a cvtColor() method in the Imgproc class for conversion between these color spaces with OpenCV. The cvtColor method takes two matte objects and the color space to be converted as parameters.

– Image Thresholding

To avoid errors due to variations in ambient light, fixed and dynamic thresholding values are applied. Here, fixed thresholds are used first. The main disadvantage of working with fixed thresholds is that at different times of the day and in different environments, i.e., when the light levels are different, the system needs to reconfigure these fixed thresholds according to the ambient light level. For this reason, dynamic thresholding algorithms are used to automate this reconfiguration in software.

In image processing, thresholding is a technique used to create binary images from greyscale images. The process involves setting a threshold value and converting all pixels in the greyscale image to black or white according to whether their intensity values are above or below the threshold value. This technique is widely used in various applications such as image segmentation, object detection, and feature extraction.

– Edge Detection

Edge detection is an image processing technique used to identify the boundaries (edges) of objects or regions in an image. Edges are among the most important features associated with images. The Canny function of the OpenCV library was used to identify the edges in the image.

– Perspective Transformation

The region where the stripes will be detected is transformed into a "bird-eye perspective" with the warpPerspective function of the OpenCV library. The purpose of the warpPerspective operation is to move an image taken from a different angle to the camera plane. This operation allowed the image to be flattened in a certain perspective.

– Block Processing

The sub-region is divided into vertical blocks of 1 pixel resolution, and the mean intensity of white pixels is calculated with the Mean Filter function of the OpenCV library. The pixel intensities (e.g., 0 black and 255 white) are analyzed to determine the stripe positions. Horizontal and vertical positions of the stripes are determined using intensity values. The start and end points of the detected stripes are denoted by a and b.

– Deflection Angle Calculation and Processing

After calculating the center of the stripes with the strip center calculation function, the horizontal window center point of the camera angle was calculated. This is a fixed value of half of the horizontal pixels of the final window. The strip centers are represented as a moving line across the entire window with a green vertical line. The window center point is also represented as a blue vertical line and is a fixed line.

– PID Control Algorithm

Proportional-integral-derivative (PID) controllers are essential components of today's automation and control systems. PIDs use a closed-loop feedback control mechanism that continuously adjusts the outputs according to the difference between a desired setpoint and a measured value. PID controllers represent a sophisticated feedback mechanism that is vital for controlling dynamic systems. In essence, they operate using three basic terms: Proportional (P), Integral (I), and Derivative (D). Each term uniquely modulates the output signal depending on the difference between the desired set point and the actual measured value, commonly known as the error.

– Time and Performance Optimization

To increase processing speed, only the lower part of the image was analyzed. This method allows eliminating unnecessary processing in the upper parts. The lane detection algorithm is optimized to process only the target region to ensure real-time operation.

C. Machine Learning

Machine learning is a subset of artificial intelligence (AI). It focuses on teaching computers to learn from data and developing them through experience, rather than being explicitly programmed to do so. In machine learning, algorithms are trained to find patterns and correlations in large data sets and make the best decisions and predictions based on this analysis. Machine learning applications improve with use and become more accurate as more data become available [18].

– The Haar Cascade model

The Haar cascade model, which is a simple and fast model, is preferred for sign recognition. Haar cascade is a machine learning-based classifier system for object recognition tasks. The model learns to recognize a specific object (e.g., a face, a traffic sign, a red light) by learning from positive and negative samples. The Haar cascade classifier is a machine learning algorithm developed by Viola and Jones [19] that is frequently used in object detection applications. It is used to quickly and efficiently detect the presence of a particular object (e.g., a traffic sign, a human face, or a traffic light). It is especially preferred in real-time applications because it provides high performance with low computational cost.

V. EXPRIMENTAL RESULTS

A. System Design

– Mini Test Vehicle

Before assembling the hardware parts, the vehicle body had to be formed and the component layout had to be determined in advance. For this purpose, photoblock cardboard material was cut and shaped according to the needs. Then, a steering system was designed for the assembly of the high torque motors that will enable the vehicle to move and for the vehicle to make right-left steering movements.

The vehicle is designed in layers to minimize the dimensions of the vehicle and to make maximum use of the body dimensions. The bottom layer contains the motor and steering system, while the second layer contains the battery, voltage reducer, servo motor, ONN/OFF button, charging circuit, and motor driver elements. In addition, to increase the robustness of the body made of photoblock material, some points of the vehicle were supported with wooden parts.

Finally, the third layer is the central processing layer,

which includes Arduino Uno, Raspberry Pi 4B, camera, and cooling elements.

Computers such as Raspberry Pis are devices that show performance loss as the operating temperature increases while performing operations that require intensive processing power. Image processing algorithms are also among these performance-intensive operations. For this reason, active and passive cooling methods were used to ensure that the operating temperature of the device remains at an optimum level.

Camera position is one of the most important factors in keeping the vehicle in the lane (Fig. 1.) Therefore, the camera is positioned high above the vehicle. In this way, the viewing angle is maximized.

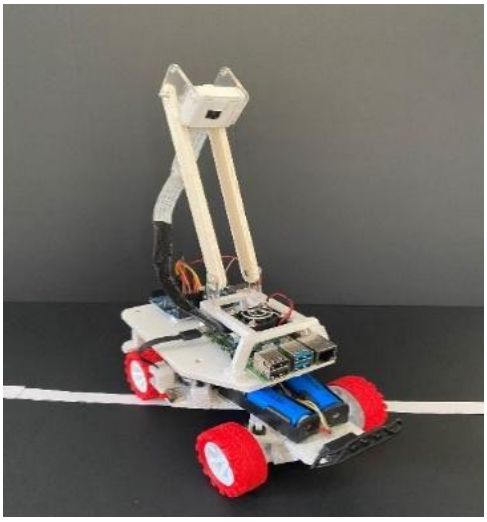

Fig. 1. Camera position in vehicle.

Two power systems have been designed for the power required by the vehicle. One of them is designed with lithium ion batteries, while the other is designed using lithium polymer batteries. Lithium ion batteries are located directly in the second layer, while the lithium polymer battery, which is placed in the system later on as needed, is placed in the rear part of the second layer of the vehicle (Fig. 1). The vehicle can be used in two modes to be more controlled in testing and development processes. One of them is Manual mode; in this mode, the vehicle is enabled to perform basic control movements such as mode switching and forward-backward, right-left, etc. via infrared (IR) control. IR module control is used for this mode. Thanks to the IR receiver connected to the Arduino, commands from the remote control are received and processed.

– Preparation of the Test Track

During preparation of the test track, a design was realized in which the autonomous driving assistant could show its capabilities at the maximum level (Fig. 2). Considering the dimensions of the vehicle and the portability of the track, the track was designed with the photoblock material used in the vehicle body. On this track, a 5 mm thick black photoblock material measuring 50 cm × 70 cm was used as the floor material. The reason why black is preferred here is that the vehicle uses white stripes on a black background in the lane detection algorithm. For the stripes, pieces of white photoblock material were cut in various lengths and angles.

Fig. 2. Test track real view.

B. Data Collection and Sampling

In this study, positive-negative samples for the detection of traffic lights and certain traffic signs (e.g., “STOP”, “NO ENTRY”) were prepared and the model was trained with OpenCV’s opencv_traincascade tool. After the training was completed, the resulting .xml file was put into use to run in real time on the Raspberry Pi.

For negative sampling to be used in the sign recognition process, a code was developed that runs on the Raspberry Pi and captures images using the camera on the vehicle. With this code, negative samples were automatically collected from scenes without signage. Negative sampling (NS) is a technique widely used in machine learning to improve the accuracy of a model. The main goal is to select meaningful samples from a large pool of object-free (i.e., non-positive) images to include in the learning process [19], [20].

In this context, 800 different images were recorded on the test track without signage in the vehicle’s field of view. These images were used during model training as negative samples representing scenarios without signage. Thus, the objective was to provide the model with the ability to distinguish between scenes with and without signage.

This negative sample set is used in other models to be created. Different negative data sets can be used according to the test paths designed for the vehicle. For the positive sampling process, the signboard photographs were taken from possible angles within the vehicle field of view (Fig. 3). Then, using the cropper tool of the Cascade Trainer GUI application, the labeling of the signboard that we want to be identified from the positive sampling images was performed (Fig. 4).

The specific region of interest for each positive sample is defined through a manual bounding process. As illustrated in Fig. 5, the “DUR” sign is selected and labeled using the cropper tool to ensure the classifier learns the correct features of the object. This labeling step is vital for generating the accurate coordinate data required by the opencv_traincascade utility.

Following the labeling of individual objects, the entire experimental setup, consisting of the mini traffic light module, various road signs, and the track layout, is organized as shown in Fig. 6. This comprehensive environment enables the vehicle to capture integrated data, ensuring that the detection models function effectively within a realistic track scenario.

Fig. 3. Negative sampling.

Fig. 4. Positive sampling.

Fig. 5. Labeling.

In this study, the widely used open source German Traffic Sign Recognition Benchmark (GTSRB) [21] dataset is used to train models for the recognition of STOP and NO ENTRY signs. GTSRB is a collection of labeled images containing different classes of traffic signs under various illumination, distortion, and background conditions [22]. The images of these classes were resized (24×24 pixels) to fit the OpenCV Haar cascade training format and positive sample lists (info.lst) were created. Negative samples were obtained from images taken directly at the test site on a Raspberry Pi without signage.

The positive and negative samples were used with the opencv_createsamples and opencv_traincascade tools to create two separate model files:

– For the STOP sign: STOP_cascade.xml;

– For the NO ENTRY sign: NO_ENTRY_cascade.xml.

These models were transferred to a Raspberry Pi for use in a real-time detection system and tested under different viewing angles, distances, and lighting conditions. Due to the rich content of the GTSRB dataset, the false positive rate of the models was reduced and the detection accuracy was improved.

To accurately detect traffic lights (red and yellow), a small-scale prototype of real-world traffic lights was designed. The mini traffic light module is LED-based and placed on a fixed tripod, and the vehicle is positioned in the test environment in the field of view of the Raspberry Pi camera.

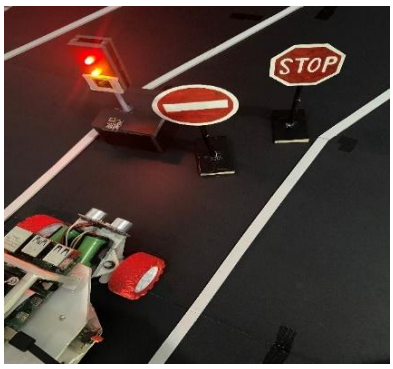

Fig. 6. Data collection.

With this special module, red and yellow light images were easily obtained from different angles and lighting conditions. The camera captured multiple frames per second while the module was active and converted these images into a data set. Approximately 400 to 500 positive samples were collected for each color. In addition, images taken when the light module was off or when other colors were active were treated as negative samples.

The collected data were labeled with the OpenCV labeling tool (Cascade Trainer GUI), converted to .vec files with the opencv_createsamples command, and separate model files were trained for each color with the opencv_traincascade tool:

– Model for red light: RED_cascade.xml;

– Model for yellow light: YELLOW_cascade.xml.

These models were run in real time on the Raspberry Pi, and appropriate actions (e.g., stop on red, warning on yellow) were initiated when the light colors specified in the camera image were detected. The mini-module makes it possible to obtain training data in a controlled and repeatable manner.

After the labeling process was completed, the Haar cascade model was trained using positive and negative samples. The training process was run on a Jupyter Notebook and executed by calling OpenCV's opencv_traincascade tool from the command line.

C. Model Training

The Haar cascade model is trained with positive and negative image samples. Positive samples consist of images containing the object to be detected, while negative samples are images that do not contain that object. The model is trained on these samples to create cascade classifiers that can discriminate the presence of objects [23]–[25].

The model is based on the calculation of Haar-like features on the integral image. This allows areas on the image to be scanned very quickly. Each feature provides a weak clue to the presence of the object. Weak classifiers are combined with the AdaBoost algorithm to create a strong classifier. These classifiers are optimized to focus only on positive regions by going through multiple cascades. This structure allows to reduce false positives and shorten the processing time [26].

The training took approximately 45 minutes to two hours. The processing time varies depending on the number of samples used and the hardware power. The cascade.xml file and the training parameter file obtained at the end of the training consist of files showing the training data in each layer (Fig. 7). It is ready to be used for plate detection in real-time images on a Raspberry Pi.

At the end of the training process, the model was compiled into a file named NO_ENTRY_cascade.xml. In addition, the same training procedure was applied for all traffic signs and lights except the "No Entry" sign. For each object, positive and negative sampling was performed separately; the images were labeled and converted to .vec format with opencv_createsamples and trained with the opencv_traincascade tool.

In this context, separate Haar cascade classifier models were created for the following traffic objects:

– Stop Sign (STOP): The mini STOP sign placed in front of the vehicle was labeled by viewing it from different angles and the model was saved as STOP_cascade.xml.

– NO ENTRY sign: This is the model that was sampled and parameterized in detail in this study.

– Red Light: Detailed sampling was performed using a mini traffic light model.

– Yellow Light: The yellow light model was trained for detection to generate a pre-crossing warning condition.

All models were optimized to run on a Raspberry Pi and exported for use in .xml format. Since each model is trained with object-specific images, it only reacts to the target object, thus avoiding false positives.

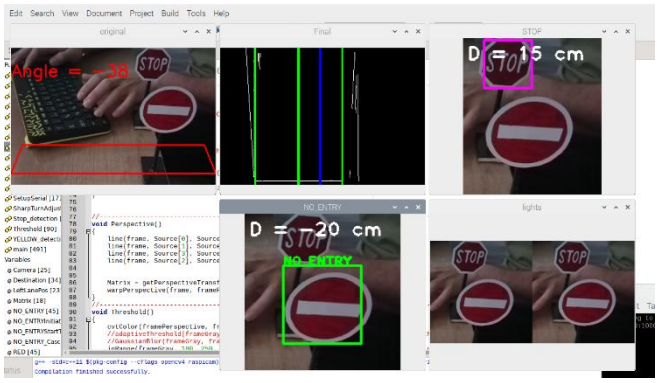

Fig. 7. Working image from the tool interface.

D. Model Performance Evaluation

– Red Light Model

Accurate red light detection is critical for the autonomous driving assistance system to issue stop commands. For this purpose, a mini traffic light module representing a red light was designed, and a training dataset was created by taking images of this module from different angles, light levels, and distances.

After the model training was completed, two separate test sets were created to evaluate the performance of the model:

– Positive test set: 741 images containing red light;

– Negative test set: 800 images without red light.

The Haar cascade classifier was run separately on these images, and it was recorded whether the model predicted correctly or incorrectly for each image. As a result of this process:

– Correct detection in 640 of the red light images (True Positive);

– 101 were missed (False Negative);

– Not detected correctly in 799 of non-red light images (True Negative);

– False detection occurred in one image (False Positive).

Precision, Recall, Accuracy, F1-Score values of the red light model are 0.90, 0.83, 0.92, and 0.91, respectively, in Fig. 8.

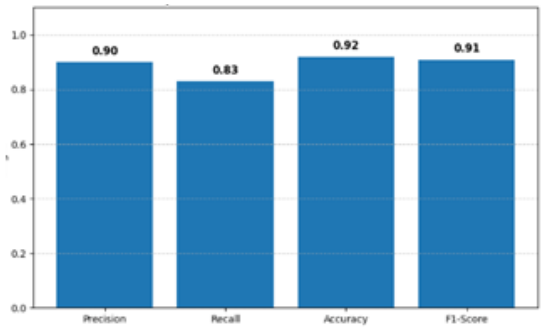

Fig. 8. Red light performance.

– Yellow Light Model

The yellow light serves as a warning to drivers to pass with caution. In this study, a yellow LED traffic light prototype was prepared, similar to the red light model, for the autonomous vehicle to detect this warning. The Raspberry Pi camera was used to record images from different angles and lighting conditions with and without the yellow light on, and positive and negative data sets were created.

After training the model, the following tests were performed:

– Positive test set: 760 images with yellow light;

– Negative test set: 800 images without yellow light.

Precision, Recall, Accuracy, F1-Score values of the yellow light model are 1, 0.71, 0.86, and 0.83 respectively in Fig. 9.

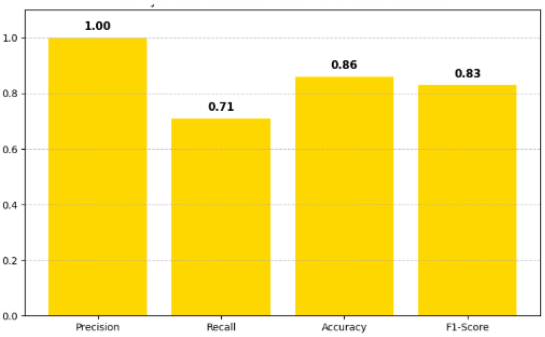

Fig. 9. Yellow light performance.

– STOP Sign Model

The "STOP" sign, which indicates where vehicles should stop, is a critical sign for safe driving. In this study, in order to detect the STOP sign with high accuracy, model training was performed using images labeled with class 14 (Stop sign) in the GTSRB dataset. Positive samples were selected from images that contained this class of sign, while negative samples were created from scenes taken with a Raspberry Pi that did not contain a STOP sign.

Two separate test sets were created to test the performance of the model:

– Positive test set: 1253 images containing a STOP sign;

– Negative test set: 800 images without STOP plate.

After testing the model, the following results were obtained:

– Plate successfully detected in 1151 images (True Positive);

– 102 images with missed plate (False Negative);

– 799 negative samples were correctly error-free (True Negative);

– False positive occurred in one image (False Positive).

Precision, Recall, Accuracy, F1-Score values of the STOP sign model are 1, 0.92, 0.95, and 0.96, respectively, in Fig. 10.

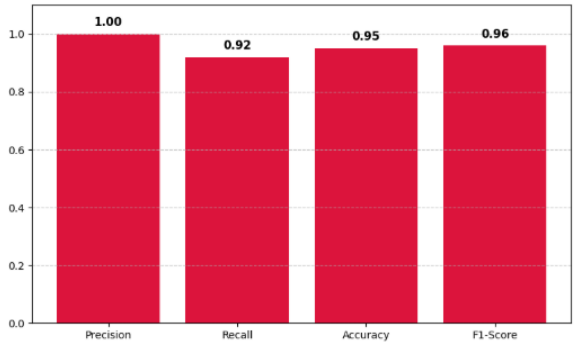

Fig. 10. STOP sign performance.

– NO ENTRY Sign Model

The No Entry sign is a critical warning sign that prevents traffic from entering in the opposite direction. Therefore, it is very important for the autonomous vehicle system to detect this sign correctly for safe driving. In this study, for the training of the NO ENTRY sign model, images of the relevant class were filtered from the GTSRB dataset and positive samples were prepared in this way. Negative samples were created from images taken with a Raspberry Pi in the field, which did not contain a sign.

In the tests performed after model training:

– Positive test set: 1110 images evaluated;

– Negative test set: 800 images evaluated.

According to the results obtained:

– 624 images were correctly identified (True Positive);

– In 486 images the model missed the plate (False Negative);

– 796 images were not correctly errored (True Negative);

– Four images produced false positive results (False Positive).

Precision, Recall, Accuracy, F1-Score values of the yellow light model are 0.99, 0.56, 0.74, and 0.72, respectively, in Fig. 11.

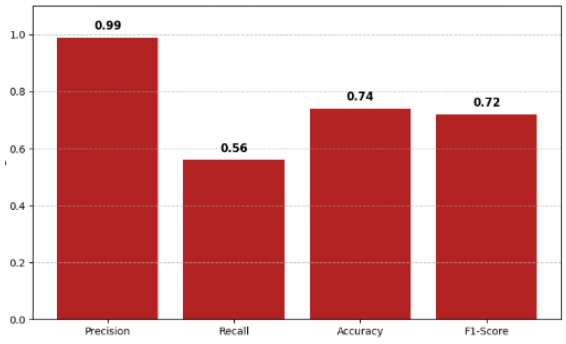

Fig. 11. NO ENTRY sign performance.

VI. CONCLUSIONS

This project achieves tasks such as lane following and traffic sign/light recognition with only camera data, showing that maximum functionality can be achieved with minimal hardware. This is valuable for research on autonomous driving in resource-constrained environments.

The project used specially collected data to train two Haar cascade models. This method demonstrates that classical model training is possible and successful with low data quantities.

This project was tested on a real test track and exhibited behaviours in accordance with traffic rules. The application of specific scenarios such as setting the stopping time at a STOP sign, waiting at a red light, and turning at a no-entry sign is a level of application that is rare in the literature. In this way, it has been shown that the developed system works in practice beyond its success on paper.

In this project, a "virtual lane" generation algorithm is uniquely integrated to help keep the vehicle on the road even when the lane image is lost. This approach allows us to maintain stable vehicle control during temporary road line breaks or faint lines. Whereas in other studies, the vehicle may often react incorrectly when lane detection fails, here a predictive solution is provided.

Another contribution of the project is the inclusion of traffic light conditions in the decision-making mechanism by detecting traffic lights, as well as traffic signs. Although some studies in the literature focus only on signage or only on traffic lights, this project was addressed both simultaneously and in a coordinated manner. In particular, the fiction that the yellow light is detected and the vehicle stops (stop-and-go logic) is an important detail in terms of reflecting a realistic traffic scenario.

In this study, supervised learning, one of the machine learning techniques, is used to perform autonomous activities such as sign recognition, traffic light detection, and obstacle recognition. Supervised learning is a machine learning method that aims to create a function that covers these data and their results by using previously observed data with known results (labeled). In this study, the sign recognition processes will be determined by classification application.

CONFLICTS OF INTEREST

The authors declare that they have no conflicts of interest.

REFERENCES

[1] E. Selimoğlu, "Trafik kazalarının nedenleri, sonuçları ve kazaların önlenmesine ilişkin öneriler", *Ziraat Mühendisliği*, vol. 361, pp. 51–54, 2014.

[2] S. Singh, "Critical reasons for crashes investigated in the national motor vehicle crash causation survey", Technical Report No. DOT HS 812 115, NHTSA, 2015.

[3] B. C. Tefft, "Acute sleep deprivation and risk of motor vehicle crash involvement", Technical Report, Washington, D. C., Foundation for Traffic Safety, 2016.

[4] T. J. Crayton and B. M. Meier, "Autonomous vehicles: Developing a public health research agenda to frame the future of transportation policy", *Journal of Transport & Health*, vol. 6, pp. 245–252, 2017. DOI: 10.1016/j.jth.2017.04.004.

[5] "On the road to fully self-driving", Waymo Safety Report, pp. 1–43, 2017.

[6] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies", *IEEE Access*, vol. 8, pp. 58443–58469, 2020. DOI: 10.1109/ACCESS.2020.2983149.

[7] E. Duman and K. Fidanboylu, "Otonom araçlarda nesne tespiti, şerit tespiti, haritalama ve konumlandırmaya yönelik sensör füzyon tekniklerinin uygulanması", *Veri Bilimi*, vol. 7, no. 2, pp. 11–21, 2024.

[8] A. Kalla and E. Tenekeci, "C++ programlama dili için yerel nesne-ilişkisel haritalama yaklaşımı", *Harran University Journal of Engineering*, vol. 8, no. 2, pp. 151–158, 2023. DOI: 10.46578/humder.1219421.

[9] M. C. Hasani, R. J. Putra, and N. Setyawan, "Autonomous car steering control and sign detection utilizing Haar Cascade and PID", *AIP Conference Proceedings*, vol. 2453, no. 1, art. 020063, 2022. DOI: 10.1063/5.0094256.

[10] G. S. Pannu, M. D. Ansari, and P. Gupta, "Design and implementation of autonomous car using Raspberry Pi", *International Journal of Computer Applications*, vol. 113, no. 9, pp. 22–29, 2015. DOI: 10.5120/19854-1789.

[11] R. Tiwari and D. K. Singh, "Vehicle control using raspberrypi and image processing", *Innovative Systems Design and Engineering*, vol. 8, no. 2, pp. 45–49, 2017.

[12] K. Vinothini and S. Jayanthy, "Road sign recognition system for autonomous vehicle using Raspberry Pi", in *Proc. of 2019 5th International Conference on Advanced Computing & Communication Systems (ICACCS)*, 2019, pp. 78–83. DOI: 10.1109/ICACCS.2019.8728463.

[13] R. S. Mohit, A. S. Kumar, M. Vats, and S. Sathyalakshmi, "Analyzing the features of self-driving cars using Haar classification methodology", in *Proc. of 5th International Conference on Communication and Electronics Systems (ICCES)*, 2020, pp. 1344–1350. DOI: 10.1109/ICCES48766.2020.9137991

[14] K. Sarvajcz, L. Ari, and J. Menyhart, "AI on the road: NVIDIA Jetson nano-powered computer vision-based system for real-time pedestrian and priority sign detection", *Applied Sciences*, vol. 14, no. 4, p. 1440, 2024. DOI: 10.3390/app14041440.

[15] F. N. Ortataş and E. Çetin, "Solution of real-time traffic signs detection problem for autonomous vehicles by using YOLOV4 and Haar Cascade algorithms", *International Journal of Automotive Science and Technology*, vol. 7, no. 2, pp. 125–140, 2023. DOI: 10.30939/ijastech..1231646.

[16] C. Dewi, R.-C. Chen, X. Jiang, and H. Yu, "Deep convolutional neural network for enhancing traffic sign recognition developed on Yolo V4", *Multimedia Tools and Applications*, vol. 81, pp. 37821–37845, 2022. DOI: 10.1007/s11042-022-12962-5.

[17] Z. Ding, C. Sun, M. Zhou, Z. Liu, and C. Wu, "Intersection vehicle turning control for fully autonomous driving scenarios", *Sensors*, vol. 21, no. 12, p. 3995, 2021. DOI: 10.3390/s21123995.

[18] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality", in *Proc. of Advances in Neural Information Processing Systems 26 (NIPS 2013)*, 2013, pp. 1–9.

[19] P. Viola and M. J. Jones, "Rapid object detection using a boosted cascade of simple features", in *Proc. of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR 2001, 2001, pp. I–I. DOI: 10.1109/CVPR.2001.990517.

[20] L. Bourdev and J. Brandt, "Robust object detection via soft cascade", in *Proc. of 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 2005, pp. 236–243, vol. 2. DOI: 10.1109/CVPR.2005.310.

[21] GTSRB - German Traffic Sign Recognition Benchmark. [Online]. Available: https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign

[22] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition", *Neural Net*works, vol. 32, pp. 323–332, 2012. DOI: 10.1016/j.neunet.2012.02.016.

[23] D.-V. Bratu, S.-A. Moraru, and L. G. Guşeilă, "A performance comparison between deep learning network and Haar Cascade on an IoT device", in *Proc. of 2019 International Conference on Sensing and Instrumentation in IoT Era (ISSI)*, 2019, pp. 1–6. DOI: 10.1109/ISSI47111.2019.9043714.

[24] A. Al-Rahayfeh and M. Faezipour, "Eye tracking and head movement detection: A state-of-art survey", *IEEE Journal of Translational Engineering in Health and Medicine*, vol. 1, art no. 2100212, pp. 2100212–2100212, 2013. DOI: 10.1109/JTEHM.2013.2289879.

[25] R. Yustiawati *et al.*, "Analyzing of different features using Haar cascade classifier", in *Proc. of 2018 International Conference on Electrical Engineering and Computer Science (ICECOS)*, 2018, pp. 129–134. DOI: 10.1109/ICECOS.2018.8605266.

[26] G. Bradski, "The OpenCV Library", *Dr. Dobb's Journal of Software Tools*, vol. 120, pp. 122–125, 2000.