Autonomous robot using computer vision with CNN and OpenCV: an experimental comparison

Duong Dinh Tu*
Department of Automatic Control
Vinh University
Nghean, Vietnam
duongdinhtu@vinhuni.edu.vn
*Corresponding author

Ho Sy Phuong
Department of Automatic Control
Vinh University
Nghean, Vietnam
hophuong@vinhuni.edu.vn

Dang Thai Son
Institute of ET
Vinh University
Nghean, Vietnam
sondt@vinhuni.edu.vn

Mai The Anh
Department of Automatic Control
Vinh University
Nghean, Vietnam
anhmt@vinhuni.edu.vn

Nguyen Xuan Hung
Department of Automatic Control
Vinh University
Nghean, Vietnam
xuanhung19@outlook.com.vn

Tran Huy Hoang
Department of Automatic Control
Vinh University
Nghean, Vietnam
tranhuyhoang02022002@gmail.com

***Abstract* — Various machine learning and artificial intelligence applications have gained prominence, propelled by recent technological advancements. Among these applications, autonomous robots stand out as particularly significant. Their integration is anticipated to yield substantial and transformative impacts on society. Current computer vision and deep learning research has been instrumental in developing automated driving systems. Lane-keeping, a critical aspect of autonomous vehicles, has been extensively studied using traditional computer vision techniques and novel deep learning and artificial intelligence models. This study details the design and fabrication of a mobile robot equipped with differential wheels and the development of a PID controller for the robot. It also presents the implementation of OpenCV and CNN models to enable the robot to follow a lane. Experiments were conducted in environments with varying lighting conditions. The experimental results demonstrate the superior effectiveness of the proposed CNN model.**

Keywords— Autonomous robot, Convolutional Neural Network, OpenCV, computer vision

I. INTRODUCTION

Autonomous robots have experienced significant advancements in recent years due to the integration of computer vision and artificial intelligence techniques. Numerous methods and models have been researched and developed for lane tracking, with Convolutional Neural Networks (CNNs) particularly prominent. Numerous studies [1-7] have explored the application of the CNN architectures in self-driving cars. These investigations utilize Raspberry Pi microprocessors in conjunction with Pi cameras to collect road data, with CNNs providing predictions for steering angles to facilitate lane control. Most of these studies highlight the superiority of CNNs in robot control based on computer vision. Overall, CNN models demonstrate powerful, flexible, and automatic computational capabilities. However, they are limited by the need for high-quality datasets and substantial computational resources, which can hinder real-time performance.

In contrast, traditional techniques, such as those employing the OpenCV library, still offer significant advantages. These techniques generally require fewer computing resources, deliver high real-time performance, and are suitable for embedded systems. However, they are constrained by predefined rules. Studies [8-12] that rely on traditional image processing techniques use OpenCV libraries for lane detection and navigation, underscoring the enduring utility of these methods. For instance, one article [8] emphasizes the importance of traditional computer vision techniques in autonomous robotic systems. Additionally, a report [13] integrates image processing with OpenCV and CNN algorithms to detect road damage.

In this study, we design and fabricate a differential wheeled robot and develop a PID controller. We then build and experiment with both OpenCV and CNN models to enable the robot to follow lanes under different conditions. We aim to elucidate the advantages of CNN models and discuss the challenges associated with using OpenCV and CNN for lane tracking. The block diagram depicting the architecture of the differential drive-wheeled mobile robot is illustrated in Fig. 1.

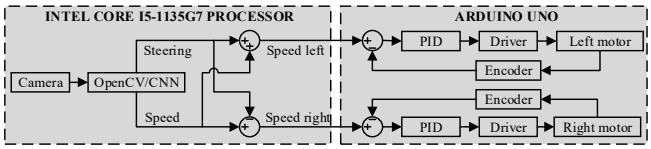


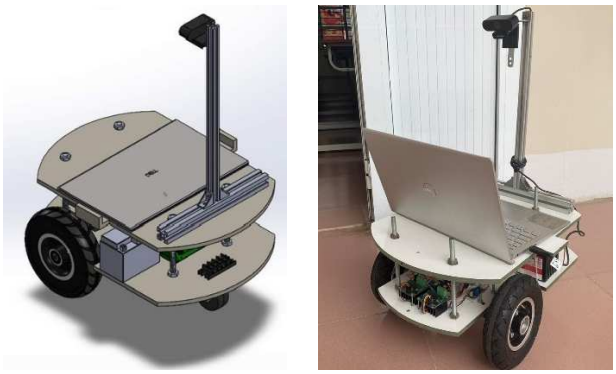
Fig. 1. Block diagram of differential drive wheeled mobile robot.

II. HARDWARE DESIGN AND FABRICATION

Initially, the differential drive-wheeled mobile robot (DDWMR) design was executed using SolidWorks software. Fig. 2a depicts the 3D rendering of the robot. Subsequently, equipment selection and hardware fabrication ensued. The robot frame comprises two 1cm thick wooden panels, assembled with 1cm diameter screws, resulting in a maximum frame size of 50cm x 41cm x 13cm. The wheel system encompasses two differential drive wheels, each boasting a diameter of 19 cm, alongside a guide wheel with a diameter of 7.5 cm. To ensure proper alignment, cushioning materials were employed during wheel installation, maintaining the robot frame parallel to the ground. Providing movement for the robot are two Planet motors equipped with encoders, each weighing 450g and operating at a 12V voltage source, 320 rpm decelerated speed, and 30W capacity, featuring encoders with 12 pulses and two channels (A, B). Providing the requisite power supply are two Globe batteries weighing 1.45kg each, with a voltage of 12V. The robot's control system integrates an Arduino UNO microcontroller and two BTS7960 motor control modules. For implementing computer vision algorithms, an Intel Core i5-1135G7 processor with a speed of 2.4 GHz and 8GB of internal memory is used. A Hikvision DS-U02 camera, featuring a 2MP CMOS sensor with a resolution of 1920 x 1080, is mounted on a shaped aluminum bar measuring 2cm x 2cm, positioned at 77cm

979-8-3503-7979-2/24/$31.00 ©2024 IEEE

above the ground. The overall weight of the robot totals 11.78kg, with the completed assembly depicted in Fig. 2b.



(a) (b)

Fig. 2. (a) 3D robot drawing on Solidworks 2021 software; (b) image of the completed robot.

III. Design a PID Controller

We use a PID controller to control the robot's speed. This is the most used controller in robotics and industry thanks to its simple structure, ease of implementation in simulation, and ease of hardware implementation. The mathematical model of the PID controller is expressed as follows:

$$u = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt} \quad (1)$$

where the variable K_p is the proportional constant, the variable K_i is the integral constant, and the variable K_d is the derivative constant. The primary challenge in implementing a PID controller lies in appropriately adjusting the parameters K_p, K_i, and K_d. Various methods have been proposed for tuning these parameters, particularly for DC motors [14-19]. However, constructing an accurate transfer function proves challenging when dealing with Planet motors utilized in robotic applications due to insufficient motor parameters. Moreover, the transition from simulation to real-world control environments presents significant disparities. Consequently, we conducted experiments to calibrate the PID controller using empirical robot operation data. In the first approach, we employed system identification techniques. The input dataset comprises 3945 voltage samples provided to the motor, ranging from 0V to 10V, with a sampling time of 0.025 seconds. The output dataset consists of 3945 speed samples measured in revolutions per minute (rpm), ranging from 0 rpm to 251.12 rpm. Leveraging MATLAB's system identification tool, we derived the transfer function, yielding a stability enforced of 74.17%:

$$G(s) = \frac{35.19s + 2.859}{s^2 + 1.382s + 0.1046}$$

Utilize MATLAB's PID tuner tool to determine PID controller parameters. It is essential to configure the Saturation block following the PID controller block to constrain the input voltage within the range of 0 V to 10 V. The obtained results are as follows: K_{p1} = 0.058, K_{i1} = 0.149, K_{d1} = -0.0002.

In the second approach, a trial and error methodology is employed. Initially, the PID controller parameters are assigned small values, specifically $K_p = K_i = K_d = 0.1$. Subsequent adjustments are made through experimental iterations conducted on the robot. These adjustments involve varying the K_p value while holding K_i and K_d constant and observing the resultant output response for refinement. Once an appropriate K_p value is determined, it is updated in the controller, with K_d maintained unchanged while adjusting K_i. Eventually, the K_i value is updated, and K_d is adjusted accordingly. The PID controller parameters obtained through this trial and error method are as follows: K_{p2} = 0.2, K_{i2} = 0.0008 and K_{d2} = 0.2.

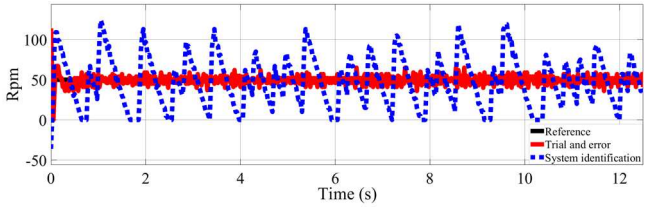


Fig. 3. Comparison of output response of PID controllers calibrated by trial and error method and system identification method.

Two calibrated PID controllers were tested. The desired motor speed was set to 50 rpm, and the actual motor speed was collected using the two controllers with a sampling time of 0.025s. The PID controller parameters were tuned using the trial and error method for better efficiency (Fig. 3). Therefore, the parameter set was selected according to this method.

IV. Experimental Track Design

(a)

(b)

(c)

Fig. 4. The outdoor robot test track: (a) robot experimental track during the day; (b) robot experimental track during the day; (c) robot experimental road at night.

An outdoor robot test track was meticulously designed for experimentation purposes. The track consisted of a two-lane elliptical road, each lane measuring 0.8 m in width. The total length of the road spanned 110 m, and distinct white road markings, 10 cm in width, were painted along its surface (Fig. 4a). Two additional areas with pedestrian markings were arranged to create more realistic situations (Fig. 4b). During night testing conditions, a 100W street light would illuminate the road. Under these conditions, the area marked with the number "**1**" is the most dimly lit area of the test track (Fig. 4c).

V. Lane Detection with OpenCV

Several lane detection algorithms utilizing OpenCV have been introduced. In this study, we synthesize these algorithms [8-12] and propose a lane detection algorithm that uses OpenCV with parameters optimized based on the device status and testing environment. Initially, images captured by the camera are converted to grayscale format to expedite processing compared to color images with three channels (red, green, and blue). To mitigate noise, a Gaussian filter is applied to the image:

$$G(x,y) = e^{-\left(\frac{x^2+y^2}{2\sigma^2}\right)} \quad (2)$$

The subsequent step involves threshold filtering of the image. The threshold is established in daytime conditions with ample illumination at T = 200. Conversely, under low-light conditions at night, the threshold is adjusted to T = 100:

$$t(x,y) = \begin{cases} 1, if\ f(x,y) > T \\ 0, if\ f(x,y) \le T \end{cases} \quad (3)$$

where $f(x,y)$ presents the image before filtering, $t(x,y)$ represents the image after filtering.

Before delineating the edges of the road markings, the noise surrounding the object within the image is eliminated, thereby smoothing the edges through the erosion operation. Subsequently, the dilate operation is employed to accentuate the edges of the road markings. A contour-finding algorithm was utilized to localize road markings. To control the robot's movement within the correct lane, the method of comparing the frame's center with the lane's center was employed. The robot was required to maneuver such that the frame's center coincided with the lane's center. Let x_1 and x_2 denote the centers of the two-lane markings, respectively. The center of the lane could be determined as follows:

$$x_c = \frac{(x_1 + x_2)}{2} \quad (4)$$

To determine whether the robot was deviating towards the left or right side of the lane, the deviation ratio d_c, representing the robot's displacement relative to the lane's center, was calculated as follows:

$$d_c = \frac{x_c}{640} \quad (5)$$

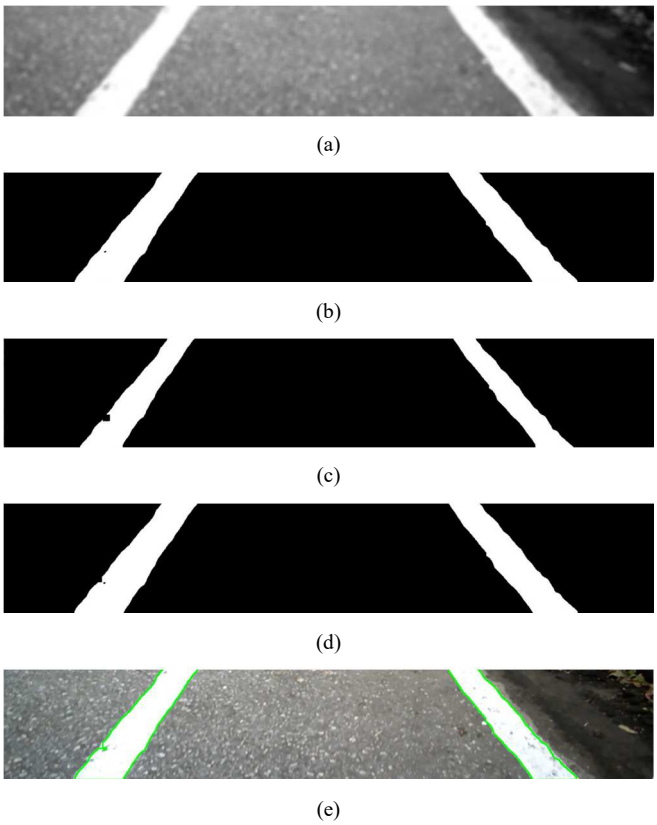
where 640 is the X-axis coordinate value of the center of the frame. The speed deviation to be adjusted d_v is determined:

$$d_v = (d_c - 1) \times 15 \quad (6)$$

where 15 is the conversion value from the deviation ratio d_c to speed deviation d_v. If the initial set value of the robot's speed is v_f, then the right wheel speed v_r and the left wheel speed v_l will be recalculated as follows:

$$\begin{aligned} v_r &= v_f - d_v \\ v_l &= v_f + d_v \end{aligned} \quad (7)$$

The basic steps of the lane detection algorithm using OpenCV are shown in Fig. 5.



(a) (b) (c) (d) (e)

Fig. 5. The basic steps of the lane detection algorithm using OpenCV: (a) Gaussian filter; (b) threshold filtering; (c) erode operator; (d) dilate operator; (e) find the contours of the road markings.

VI. Lane Detection with CNN

Recent studies have demonstrated significant advancements in applying CNNs for autonomous vehicles. Numerous CNN models have been developed successfully for lane detection in mobile robots. However, the majority of results have been obtained through simulation. Although some experimental results have been achieved using robots equipped with Raspberry Pi microcontrollers, these experiments have been limited to steering angle prediction without incorporating motor speed control. Additionally, the structural details of the proposed CNN models are often inadequately reported [1-7]. One study [20] proposes a CNN architecture based on Nvidia's model, consisting of nine layers: five convolutional layers, three densely connected layers, and one output layer. This model employs the Exponential Linear Unit (ELU) activation function and the Adam optimizer. When tested on a robot using a Raspberry Pi 3 Model B+, the model demonstrated acceptable performance in steering angle prediction. Another study [21] introduces an architecture based on the LeNet CNN, comprising one input layer, six interleaved layers (including Convolutional, Subsampling, and Fully Connected layers), and one output layer, utilizing the Sigmoid activation function. This architecture is intended for implementation on a robot using a Raspberry Pi 3B+ microcontroller and a PID controller. However, experimental results have not yet been reported. A further study [22] presents a CNN architecture with eighteen layers, including one input layer, six convolutional layers, four fully connected layers, several auxiliary layers, and one output layer, employing the Rectified Linear Unit (ReLU) activation

function. This model is also proposed for use on a robot with a Raspberry Pi 3B+ microcontroller and a PID controller, though experimental validation is still pending.

A. Design a CNN Model

The proposed CNN architecture comprises one input layer, five convolutional layers, one flatten layer, three fully connected layers, and one output layer. The model begins with a series of convolutional layers, specifically a Conv2D layer with 24 filters, a kernel size of 5 x 5, a stride of 2 x 2, and an Exponential Linear Unit (ELU) activation function applied to the input shape of 66 x 200 x 3; a Conv2D layer with 36 filters, a kernel size of 5 x 5, a stride of 2 x 2, and an ELU activation function; a Conv2D layer with 48 filters, a kernel size of 5 x 5, a stride of 2 x 2, and an ELU activation function; two successive Conv2D layers, each with 64 filters, a kernel size of 3x3, and an ELU activation function. Following the convolutional layers, the network includes a flatten layer to convert the 2D matrix data to a 1D vector. This is followed by three densely connected (fully connected) layers with 100, 50, and 15 units, each utilizing the ELU activation function. The final layer is dense with a single unit and designed for regression output. The model is compiled using the Adam optimizer with a learning rate of 0.0001 and employs the Mean Squared Error (MSE) loss function. This architecture aims to provide robust feature extraction and regression capabilities suitable for autonomous vehicles' steering angle and speed prediction tasks. The architecture will be implemented on an Intel Core i5-1135G7 processor, running two parallel models to predict the steering angle and the robot's speed. The architecture of the proposed CNN model is shown in Table I.

TABLE I. THE ARCHITECTURE OF THE PROPOSED CNN MODEL

Layer (Type)	Output Shape	Parameters
Input	(66, 200, 3)	0
Conv1 (f=5,s=2)	(31, 98, 24)	1,824
Conv2 (f=5,s=2)	(14, 47, 36)	21,636
Conv3 (f=5,s=2)	(5, 22, 48)	43,248
Conv4 (f=3,s=1)	(3, 20, 64)	27,712
Conv5 (f=3,s=1)	(1, 18, 64)	36,928
Flatten	1152	0
Fully connected	100	115,300
Fully connected	50	5,050
Fully connected	15	765
Output	1	0
Total		252,463

B. Data Collection and Preparation

For road data collection, a gamepad is utilized to maneuver the robot within the right lane while concurrently capturing corresponding images, speed, and steering angle data using the Hikvision DS-U02 camera. The dataset comprises 20,000 distinct images, each with dimensions of 1280 x 720 pixels. In conjunction with applying deep learning models, data augmentation techniques enhance the training dataset by randomly transforming existing samples. This augmentation strategy serves to expand the training set size and mitigate overfitting. Common transformations include zoom, horizontal flip, brightness adjustment, random shadow addition, and height and width shifts. Additionally, Pixel-wise Affine Normalization (PAN), a preprocessing technique, is implemented to uniformly rebalance pixel values across the entire image, mitigating unwanted variations and enhancing the model's generalization capabilities. To facilitate the application of the proposed CNN model, additional preprocessing steps, such as resizing each image to 66 x 200 pixels, are performed (Fig. 6). All these techniques are executed using the OpenCV library and other Python libraries.

Fig. 6. Data collection.

C. Model Training

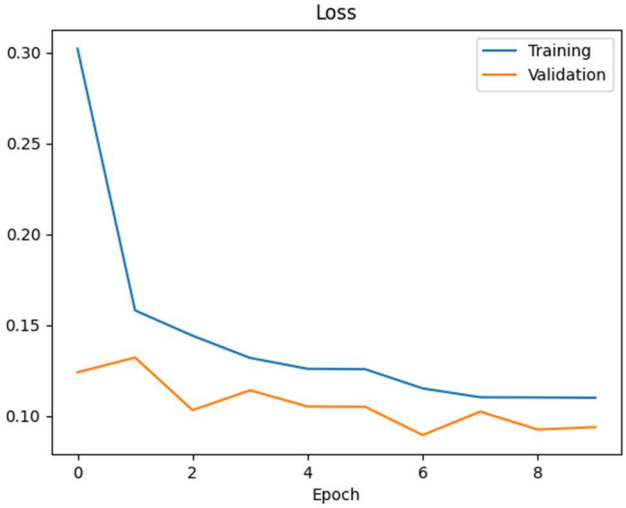


Fig. 7. The training error rate of the steering angle prediction.

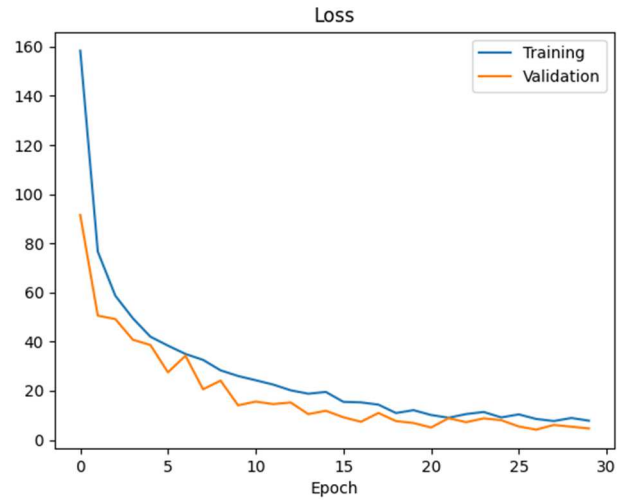


Fig. 8. The training error rate of the speed prediction.

The proposed CNN model requires training with the collected data. The input data, which consists of the collected and preprocessed images, is partitioned into an 80:20 ratio, with 80% allocated for model training and the remaining 20% for validation purposes. The CNN model is trained for two

tasks: steering angle prediction and speed prediction. This dual-task approach enhances the robot's flexibility and performance during experiments. For the steering angle prediction model, training requires approximately 15 minutes for ten iterations. In contrast, training the speed prediction model takes approximately 45 minutes for 30 iterations. The steering angle values range from -1 to 1, with the loss function converging to a minimum value of around 0.1 (Fig. 7). Additionally, at a straight-line speed of 60 rpm and 30 rpm on curves, the loss function achieves its minimum value of approximately 10 (Fig. 8).

VII. Experimental Results

Experiments were conducted on the proposed robot along a designated experimental route. The OpenCV library was used to control the robot for lane-following, while the CNN model was employed to predict steering angles and speed. These computations were performed using an Intel Core i5-1135G7 processor. A PID controller for robot speed control was also implemented on the Arduino UNO microcontroller.

A. In Daylight Conditions

One limitation of OpenCV is that the robot could only move at a preset speed. This limitation precluded the robot from decelerating at road corners, deviating from its lane. The experimental process demonstrated that the robot could only move stably at a maximum speed of 30 rpm under daytime conditions. In contrast, employing a CNN model enabled the robot to automatically adjust its speed on different road segments, increasing speed on straight roads and decreasing speed on corners. Fig. 9 illustrates the change in the steering angle and speed of the robot when utilizing the CNN model. It can be observed that the minimum speed of the robot was approximately 30 rpm, while the maximum speed was around 60 rpm. This adaptive adjustment allowed the robot to maneuver more flexibly and accurately than when using OpenCV.

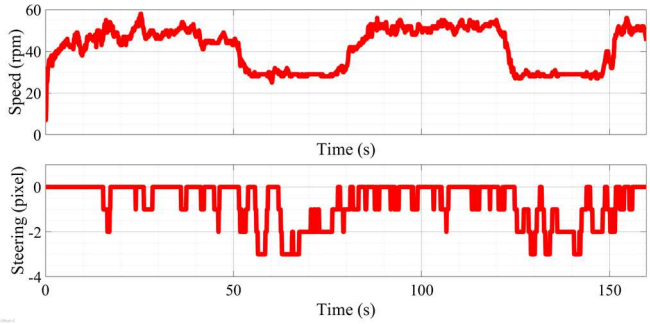


Fig. 9. Predicted steering angle and predicted speed by CNN.

Under well-lit conditions, experimental findings demonstrate that utilizing the CNN results in more stable movement of the robot within the correct lane than when employing OpenCV. Moreover, pedestrian crossings, obscured road markings, and uneven road surfaces may cause the robot to deviate from the lane when using OpenCV. Fig. 10 compares the robot's steering angle between OpenCV and the CNN model, indicating superior stability achieved when utilizing the CNN model. Furthermore, the robot tends to drift towards the right side of the lane using the CNN model, whereas it maintains movement around the lane center with OpenCV. Fig. 11 illustrates a case where a robot utilizing OpenCV experienced difficulty recognizing the lane when passing through an area with pedestrian crossings. It can be observed that, at the eighty-fifth second, the robot failed to detect the lane correctly and deviated from it. In contrast, when employing CNN, the robot accurately predicted the lane in the area with pedestrian crossings and continued to move within the correct lane.

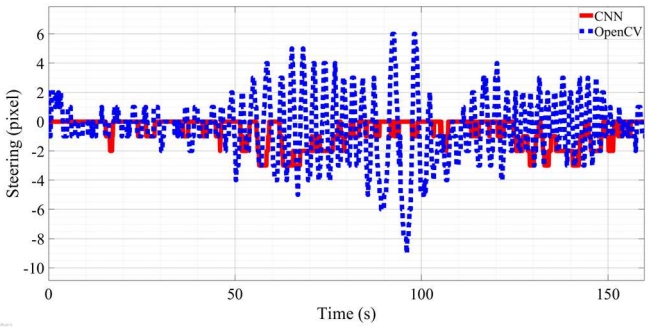


Fig. 10. Comparison of steering angle when using CNN model and OpenCV in daylight conditions.

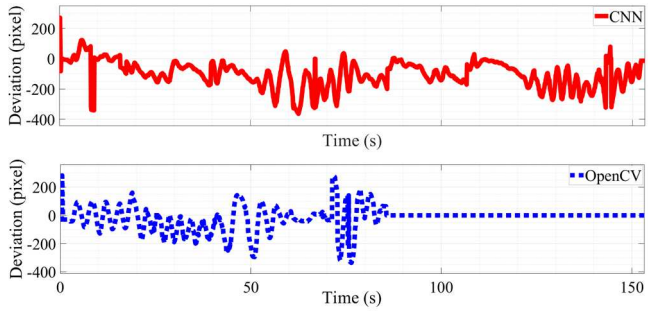


Fig. 11. Comparison of deviation in pixels when using CNN model and OpenCV in daylight conditions.

B. In Nightlight Conditions

The robot was tested under night conditions, illuminated by 100W street lights. When employing OpenCV, adjusting the filtering threshold from T = 200 to T = 100 became necessary. In general, both the CNN model and OpenCV encountered difficulties under these environmental conditions. In most cases, neither the CNN model nor OpenCV could control the robot to move within the correct lane throughout one lap of the 110m test track. The robot lost control in the dimly lit area of the road (the area marked with number 1 in Figure 4c). However, the CNN model still demonstrated an advantage over OpenCV.

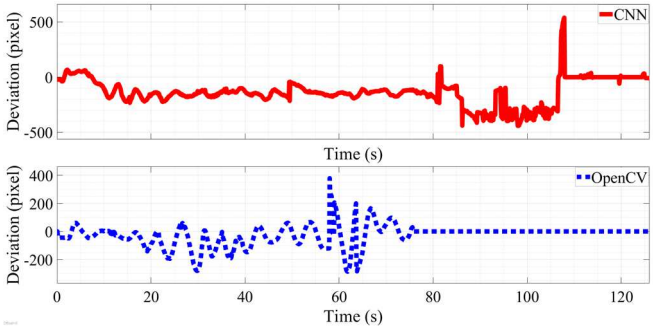


Fig. 12. Comparison of eccentricity in pixels when using CNN model and OpenCV in nightlight conditions.

Fig. 12 presents the deviation observed when using the CNN model and OpenCV. The robot utilizing CNN could move within the correct lane for a considerable distance, only losing control at the one hundred and tenth second when the robot

entered the minor illuminated area of the road. Meanwhile, the robot employing OpenCV would move into the wrong lane at the seventy-fifth second.

VIII. CONCLUSIONS

We fabricated a wheeled mobile robot with a differential drive system and a PID control scheme optimized through a trial-and-error approach. We have synthesized and proposed control methodologies utilizing OpenCV and the proposed CNN model to enable lane-tracking capabilities. Experiments were conducted on a pre-designed experimental track under varied outdoor lighting conditions. The results confirm previous studies that The CNN model excels in lane recognition for autonomous robots, particularly in challenging conditions involving lighting interference or image noise due to pedestrian or unclear road markings.

It needs to be further discussed that although the CNN model shows superiority, using OpenCV still has significant advantages. CNN models are beneficial in complex conditions such as lighting noise, shadows, and curves. The CNN model can also generalize and recognize better in various situations after training on a large and diverse dataset. However, the CNN model has notable disadvantages, including the requirement for powerful hardware and substantial computational resources for training and deployment. Building a high-quality and extensive dataset is also expensive and time-consuming. Compared to the CNN model, OpenCV results in lower accuracy, especially under complex noise conditions. OpenCV detects features manually, limiting its generalization ability and linking it to specific environmental conditions. Nevertheless, OpenCV has significant advantages, such as not requiring powerful hardware or large computing capacity, making it more suitable for real-time applications. Furthermore, OpenCV does not require training data, simplifying deployment. These advantages make OpenCV ideal for certain types of mobile robots, such as Automatic Guided Vehicles (AGVs) operating in fixed environments or robots used for educational purposes.

REFERENCES

[1] Gupta, S., Upadhyay, D., Dubey, A.K. (2019). Self-Driving Car Using Artificial Intelligence. In: Kumar, M., Pandey, R., Kumar, V. (eds) Advances in Interdisciplinary Engineering. Lecture Notes in Mechanical Engineering. Springer, Singapore. https://doi.org/10.1007/978-981-13-6577-5_49.

[2] J. Kim, G. Lim, Y. Kim, B. Kim and C. Bae, "Deep Learning Algorithm using Virtual Environment Data for Self-driving Car," 2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC), Okinawa, Japan, 2019, pp. 444-448, doi: 10.1109/ICAIIC.2019.8669037.

[3] Del Egio, J., Bergasa, L.M., Romera, E., Gómez Huélamo, C., Araluce, J., Barea, R. (2019). Self-driving a Car in Simulation Through a CNN. In: Fuentetaja Pizán, R., García Olaya, Á., Sesmero Lorente, M., Iglesias Martínez, J., Ledezma Espino, A. (eds) Advances in Physical Agents. WAF 2018. Advances in Intelligent Systems and Computing, vol 855. Springer, Cham. https://doi.org/10.1007/978-3-319-99885-5_3.

[4] I. Ahmad and K. Pothuganti, "Design & implementation of the real-time autonomous car by using image processing & IoT," 2020 Third International Conference on Smart Systems and Inventive Technology (ICSSIT), Tirunelveli, India, 2020, pp. 107-113, doi: 10.1109/ICSSIT48917.2020.9214125.

[5] P. G. Chaitra, V. Deepthi, S. Gautami, H. M. Suraj and N. Kumar, "Convolutional Neural Network based Working Model of Self Driving Car - a Study," 2020 International Conference on Electronics and Sustainable Communication Systems (ICESC), Coimbatore, India, 2020, pp. 645-650, doi: 10.1109/ICESC48915.2020.9155826.

[6] W. Dangskul, K. Phattaravatin, K. Rattanapom, Y. Kidjaidure, "Real-time control using Convolution Neural Network for self-driving cars," 2021 7th International Conference on Engineering, Applied Sciences and Technology (ICEAST), 2021, pp. 125–128, doi: 10.1109/ICEAST52143.2021.9426255.

[7] R. Farkh, S. Alhuwaimel, S. Alzahrani, K. Al Jaloud and M. Tabrez Quasim, "Deep learning control for autonomous robot," Computers, Materials & Continua, vol. 72, no.2, pp. 2811–2824, 2022.

[8] J. B., S. V., V. Purohit, D. Oswald Tauro and V. J., "Design and Development of Automated Intelligent Robot Using OpenCV," 2018 International Conference on Design Innovations for 3Cs Compute Communicate Control (ICDI3C), Bangalore, India, 2018, pp. 92-96, doi: 10.1109/ICDI3C.2018.00028.

[9] R. Murad, A. Jones and J. Straub, "Use of Computer Vision for White Line Detection for Robotic Applications," 2019 IEEE International Conference on Electro Information Technology (EIT), Brookings, SD, USA, 2019, pp. 509-514, doi: 10.1109/EIT.2019.8834015.

[10] A. Ma'arif, A. A. Nuryono and Iswanto, "Vision-Based Line Following Robot in Webots," 2020 FORTEI-International Conference on Electrical Engineering (FORTEI-ICEE), Bandung, Indonesia, 2020, pp. 24-28, doi: 10.1109/FORTEI-ICEE50915.2020.9249943.

[11] Wael Farag, "Real-Time Detection of Road Lane-Lines for Autonomous Driving", Recent Patents on Computer Science, 2019, Vol. 12, No. 1, pp. 1–10. DOI: 10.2174/2213275912666190126095547.

[12] Karkera, T., Singh, C. Autonomous Bot Using Machine Learning and Computer Vision. SN COMPUT. SCI. 2, 251 (2021). https://doi.org/10.1007/s42979-021-00640-6.

[13] Sang-Hyun Lee, "A study on road damage detection for safe driving of autonomous vehicles based on OpenCV and CNN", international Journal of Internet, Broadcasting and Communication Vol.14, No.2, (2022), pp. 47–54. http://dx.doi.org/10.7236/IJIBC.2022.14.2.47.

[14] A. Ma'arif, Iswanto, N. M. Raharja, P. Aditya Rosyady, A. R. Cahya Baswara and A. Anggari Nuryono, "Control of DC Motor Using Proportional Integral Derivative (PID): Arduino Hardware Implementation," 2020 2nd International Conference on Industrial Electrical and Electronics (ICIEE), Lombok, Indonesia, 2020, pp. 74-78, doi: 10.1109/ICIEE49813.2020.9277258.

[15] Borase, R.P., Maghade, D.K., Sondkar, S.Y. et al. A review of PID control, tuning methods and applications. Int. J. Dynam. Control 9, 818–827 (2021). https://doi.org/10.1007/s40435-020-00665-4.

[16] P. Shah and R. Sekhar, "Closed Loop System Identification of a DC Motor using Fractional Order Model," 2019 International Conference on Mechatronics, Robotics and Systems Engineering (MoRSE), Bali, Indonesia, 2019, pp. 69-74, doi: 10.1109/MoRSE48060.2019.8998744.

[17] Y. Naung, A. Schagin, H. L. Oo, K. Z. Ye and Z. M. Khaing, "Implementation of data driven control system of DC motor by using system identification process," 2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), Moscow and St. Petersburg, Russia, 2018, pp. 1801-1804, doi: 10.1109/EIConRus.2018.8317455.

[18] N. Donjaroennon, S. Nuchkum and U. Leeton, "Mathematical model construction of DC Motor by closed-loop system Identification technique Using Matlab/Simulink," 2021 9th International Electrical Engineering Congress (iEECON), Pattaya, Thailand, 2021, pp. 289-292, doi: 10.1109/iEECON51072.2021.9440305.

[19] K. Sekarsari, T. Tata, Performance analysis of PID control in DC Brushless motor using trial and error method, 2021 IOP Conf. Ser.: Mater. Sci. Eng., 2021, DOI 10.1088/1757-899X/1098/4/042027.

[20] Chy MKA, Masum AKM, Sayeed KAM, Uddin MZ. Delicar: A Smart Deep Learning Based Self Driving Product Delivery Car in Perspective of Bangladesh. Sensors. 2022; 22(1):126. https://doi.org/10.3390/s22010126.

[21] Farkh, R., Quasim, M.T., jaloud, K.A., Alhuwaimel, S., Siddiqui, S.T. (2021). Computer vision-control-based CNN-PID for mobile robot. Computers, Materials & Continua, 68(1), 1065-1079. https://doi.org/10.32604/cmc.2021.016600.

[22] Farkh, R., Alhuwaimel, S., Alzahrani, S., Jaloud, K.A., Quasim, M.T. (2022). Deep learning control for autonomous robot. Computers, Materials & Continua, 72(2), 2811-2824. https://doi.org/10.32604/cmc.2022.020259.