Journal of Integrated and Advanced Engineering (JIAE)
Vol. 4, No. 1, March 2024: 9-20
http://asasijournal.id/index.php/jiae
http://doi.org/10.51662/jiae.v4i1.118

Collision avoidance of mobile robot using AlexNet and NVIDIA Jetson Nano B01

Ferryawan Harris Kristanto, Zendi Iklima[*]
Department of Electrical Engineering, Faculty of Engineering, Universitas Mercu Buana, Indonesia

Abstract
In this research, an intelligence collision avoidance system on a mobile robot was designed using the AlexNet image classifier method. AlexNet is a convolutional neural network architecture that managed to win the ImageNet Large Scale Visual Recognition Challenge in 2012. The dataset consists of three categorical labels: blocked right, blocked left, and free. Images of 224 x 224 pixels were trained into two CNN architectures: AlexNet and ResNet-18. The performance of both architectures was examined in a testing environment. The system was built without real-time obstacles, instead using the side boundaries of the test lane. Analogously, if the mobile robot moves either through the side lane or off track, then these conditions are defined as a crash. From the entire research that was done, it was determined that intelligence collision avoidance models based on AlexNet were the most reliable models, with an average accuracy deviation rate of 6,00%. The true pre-trained AlexNet adopted from PyTorch Transfer Learning had 92.22% overall accuracy, while the non-trained AlexNet achieved 90.81% accuracy. It is also supported by the evidence that Intelligence Collision Avoidance Model-1 and Model-3 based on AlexNet didn't lead the mobile robot to spin out and were stable in the test lane.

Keywords:
Collision avoidance;
Deep Learning;
Mobile robot;
NVIDIA Jetson Nano B01;

Article History:
Received: August 28, 2023
Revised: November 26, 2023
Accepted: December 6, 2023
Published: February 14, 2024

Corresponding Author:
Zendi Iklima
Electrical Engineering Department, Universitas Mercu Buana, Indonesia
Email:
zendi.iklima@mercubuana.ac.id

INTRODUCTION

Artificial Intelligence (AI) is a field in computer science that is aimed at making software and hardware capable of thinking, such as humans. AI is widely used to solve problems in areas such as business, robotics, natural language, mathematics, games, perception, medical diagnosis, engineering, financial analysis, scientific analysis, and reasoning [1][2]. Deep learning (DL) is the deepest structure of AI based on artificial neural networks (ANN), which has been widely used in recent years. DL is divided into three main tasks: supervised learning, unsupervised learning, and reinforcement learning. DL is very suitable for solving classic problems in computer vision, namely image classification. One method in DL that is often used in image processing is convolutional neural network (CNN), which is the enhancement of multi-layer perceptron by providing deep neural network layers.

The control of mobile robot motion has been incorporated into deep learning systems for robotics. Research on perceptual capabilities in robotic systems has been greatly influenced by the DL model [3]. Different techniques can be used to create collision avoidance systems for mobile robots. The Improved Artificial Potential Field (IAPF) and PID Adaptative Tracking Control Algorithm [4] are two collision avoidance techniques that have been developed, as well as a mid-vehicle collision avoidance system based on fuzzy inference [5]. A collision avoidance system may also be designed using a straightforward perceptron network [6].

A mobile robot will be used to test the implementation of the proposed research's intelligence collision avoidance system, which will be constructed using DL principles. The model's performance will be assessed in the testing environment once the collision avoidance method, which is based on prediction probability, is utilized. The testing area consists of two different test lanes with no actual barriers. Instead, side lane boundaries are referred to as the barriers or obstructions that will be used as test cases to determine how an intelligent system based on the AlexNet image classifier method would be able to resolve the current problem. The intelligent collision system is supposed to be capable of identifying and correctly anticipating the side lane, ensuring that the robot does not collide or deviate from the test track. The ResNet-18 architecture is also incorporated to develop the intelligence collision avoidance model as a performance comparison indication between AlexNet and ResNet-18.

METHOD

The research methodology determines the research objectives: first, to know how to design an intelligent collision avoidance system using AlexNet, and second, to evaluate the performance of the model. A literature review is carried out in order to find the references for the proposed project, then the research kit is designed and built. Later, the research kit will be examined in a testing environment. The model is analyzed according to the mobile robot's performance in the arena and taken as the final conclusion. The methodology shown in Figure 1.

Based on a literature review, neural networks are the best method to design a collision avoidance system. CNN based on AlexNet could produce image classification performance with a low overfitting rate [7, 8, 9, 10]. The NVIDIA Jetson Nano B01 would be the preferred processing unit over the Raspberry Pi since it has a dedicated Maxwell GPU. As for the DL Frameworks, PyTorch is suitable for low-end devices such as the Jetson Nano B01 since it can reduce the bottleneck effect between CPU and GPU [11, 12, 13, 14, 15].

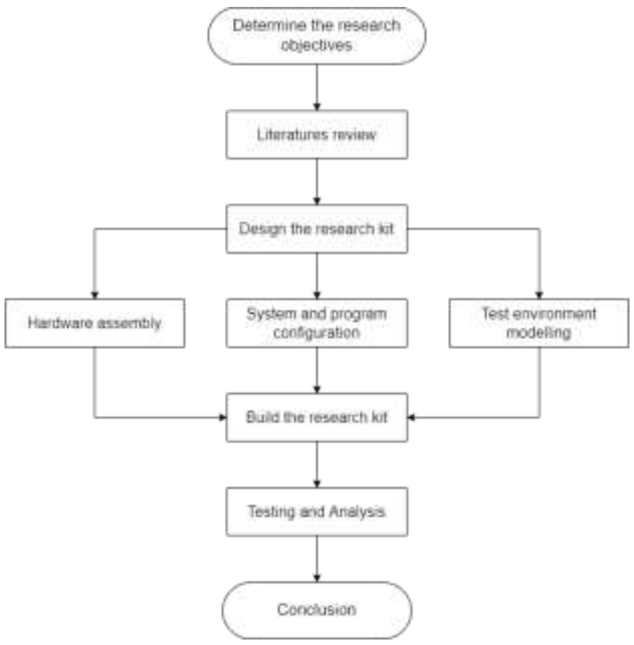


Figure 1. Research Methodology

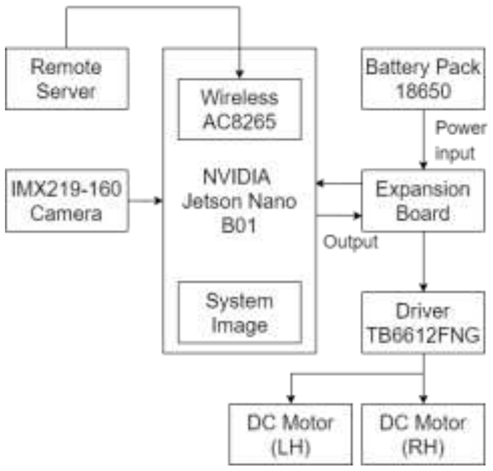


Figure 2. Hardware Assembly

Hardware Assembly

Figure 2 show the hardware assembly that construct the mobile robot components. The Waveshare Robot Kit was used to construct the mobile robot's components. Because it decreased design time, the produced robot kit was chosen as the primary option. The IMX219-160 webcam serves as an input device. The microprocessor was the NVIDIA Jetson Nano B01, while the output block included the TB6612FNG Motor Driver and two DC motors. The NVIDIA Jetson Nano B01 was chosen as the robotic system microcontroller since it has DL requirements. The Jetson Nano is equipped with a Cortex-ARM A57 CPU and a dedicated Maxwell GPU [16][17]. Image processing can be done correctly with Compute Unit Device Architecture support. Figure 3 depicts the CAD design of the mobile robot kit. The package's operating system (OS) is packed as a system image that is available on the official NVIDIA developer forum through NVIDIA JetPack v4.3. Figure 3 shows the mobile robot design that construct using Waveshare Robot Kit.

Environment Design

Figure 4 show the environment design which contains the lane boundaries that represents in Figure 5. There were two types of tracks established: circuit and sprint. The test tracks were built at the scale depicted in Figure 5. They were given a colorful ribbon at the track's border that acts as the arena's barrier.

In general, the collision avoidance technique that was developed is unique. In this study, no physical barriers were placed in the arena; instead, we used the track's border as impediments. The concept of a crash is stated to be present when the robot exceeds the track limit, as seen in Figure 6.

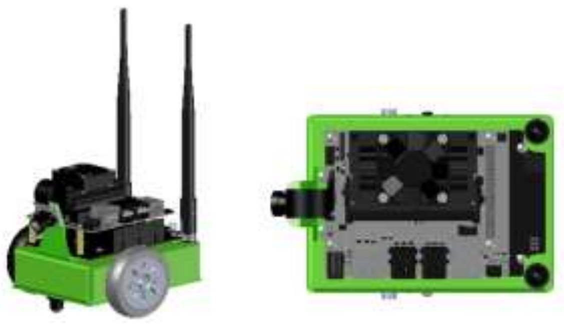


Figure 3. Mobile Robot Design

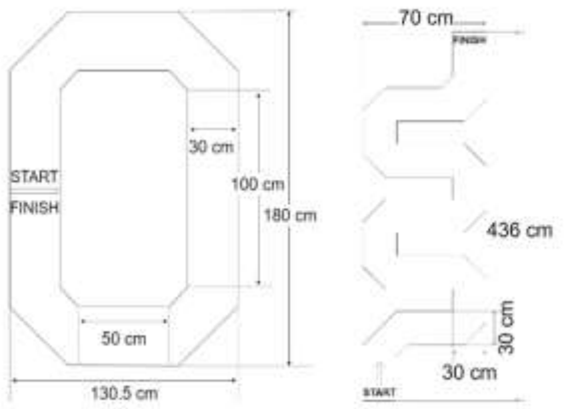


Figure 4. Environment Design

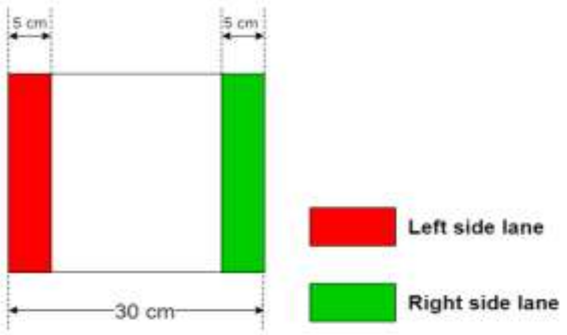
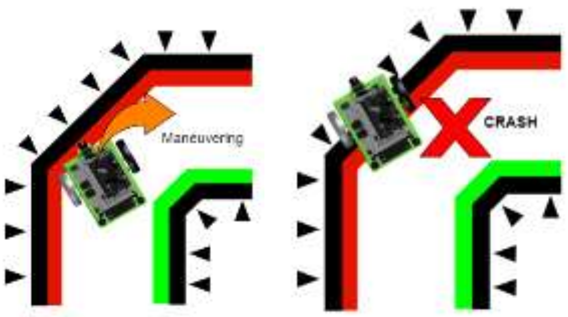


Figure 5. Side Line Boundaries



(a) (b)
Figure 6. Scheme of (a) Maneuvering, (b) Crash State

The identical approach was applied to both test tracks by creating exterior limits in the form of black bars. As depicted in Figure 6(a), the intelligent collision avoidance system is supposed to steer the robot into the arena without striking the black barrier. If the robot reaches the track's black boundary or travels outside of the track's outer barrier shown in Figure 6(b), the algorithm has failed to produce an accurate forecast.

Intelligence collision avoidance is programmed based on the user-specified threshold value for the anticipated label value provided by CNN's softmax function. There are three sorts of labels: blocked_left, blocked_right, and free. For example, if the robot detects the route edge and the obstacle is classed as blocked_left with a probability value greater than the threshold, the robot should shift to the right, avoiding contact with the black edge and going off-track. This rule applies to the whole labels.

The dataset was divided into three groups, each with 75 photos measuring 224 by 224 pixels. The complete data set of 225 photos was divided about 80% for training and 20% for testing. There are no definitive criteria for determining the best or optimal ratio for a specific dataset [18]. The 80:20 ratio is just an explanation for the notion known as Pareto's notion [19]. The division ratio difference is noticeable in small datasets, while division ratios of 70% and 80% definitely work better on large datasets [20][21].

Methods

To aid user understanding in system planning, a flowchart is presented. In summary, the system process begins with data collection, model training, and model evaluation, as shown in Figure 7.

During the initial step, the user connects in to the JupyterLab Integrated Development Environment (IDE) through a remote server on a wireless network (Wifi). JupyterLab is controlled by the user using the Python programming language [22][23]. The dataset folder is created at the root and the IMX219-160 webcam feed is launched which capable to capture 3280 x 2464 resolution and real time monitoring. The data collection step is when you collect the datasets that will be utilized to train the model. The dataset was taken using the camera's capture function and saved in a folder based on the label category. Following the completion of the dataset collection, the next phase is model training. The torchvision.transform package is used to preprocess the picture data. Torchvision is a PyTorch package that includes CNN and transfer learning models [24][25]. After the model has been trained, the model's performance is evaluated in a test environment. Its goal is to evaluate the model's performance. At this point, it is unclear if the model can successfully avoid collisions. Figure 8 show the mobile robot observation flow when perfom using IMX219-160.

System initialization begins by ensuring that the complete system can start, run smoothly, and is ready for use. Connect the server device (Jetson Nano) to the Wi-Fi network. The next step is to start streaming the webcam from the IMX219-160 camera. This indicates that the server device is ready for use in taking and storing photos for the dataset. Intelligence Collision Avoidance is built by using a pre-trained CNN model, loading the model, and starting ROS. After successfully loading ROS into the system, the mobile robot may begin moving down the track.

When obstacles are detected and predicted, a probability distribution mechanism for the indicated obstacle types on the track is initiated. For each predicted category, a probability threshold of 0.5 is used. When the greatest likelihood value matches to a certain category, the robot's orientation is maneuvered or rotated. This method is repeated throughout the robot's journey. When the user reaches the end of the track, he or she manually issues an order to stop the robot's motor movement.

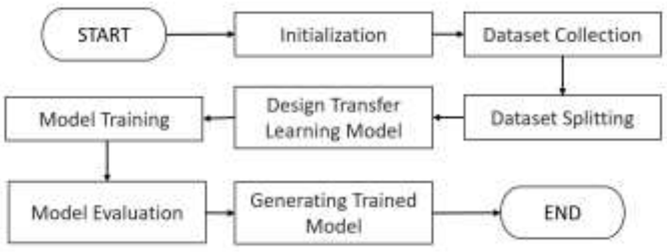


Figure 7. System Flow

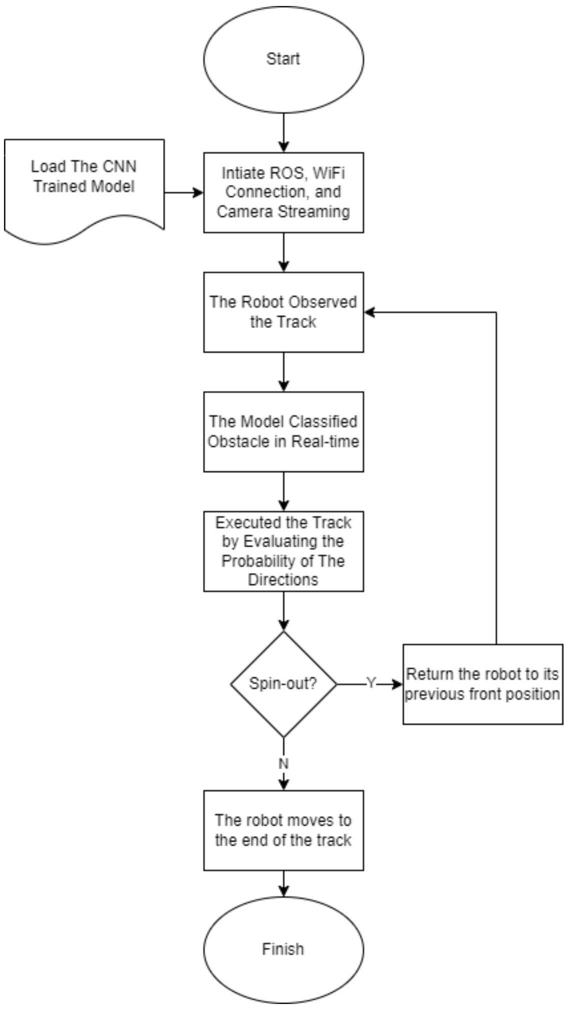

Start
Load The CNN Trained Model
Intiate ROS, WiFi Connection, and Camera Streaming
The Robot Observed the Track
The Model Classified Obstacle in Real-time
Executed the Track by Evaluating the Probability of The Directions
Spin-out?
Y
Return the robot to its previous front position
N
The robot moves to the end of the track
Finish

Figure 8. Mobile Robot System Flow

This experiment proposes six models of intelligent collision avoidance. The six models are separated into three AlexNet-based models and three ResNet-18-based models. All models will be put through their paces on a test track. Table 1 lists the model's spesifications.

TRUE Pre-Trained is a transfer learning model with inherited weights and biases. In contrast, with the FALSE PRE-TRAINED, the model must retrain the neurons' weights and biases. The initial model employs 20 epochs, which are expanded to 40 in succeeding models.

The model overview of AlexNet is provided in Table 2. This summary is part of the PyTorch torchvision package and is accessible via the model.summary() function. The model overview is intended to help people understand how the model is developed. AlexNet includes 21 sequential layers in total, 5 convolutional layers and 3 fully linked layers. It contains around 61 million trainable parameters.

Table 1. Proposed Models

Model	***Architecture***	***Pre-Trained***	***Epochs***
1	AlexNet	*TRUE*	20
2	AlexNet	*FALSE*	20
3	AlexNet	*FALSE*	40
4	ResNet-18	*TRUE*	20
5	ResNet-18	*FALSE*	20
6	ResNet-18	*FALSE*	40

Table 2. AlexNet Model Summary

AlexNet			
#	***Layer (type)***	***Output Shape***	**Param#**
1	Conv2d	[-1, 64, 55, 55]	23,296
2	ReLU	[-1, 64, 55, 55]	0
3	MaxPool2d	[-1, 64, 27, 27]	0
4	Conv2d	[-1, 192, 27, 27]	307,392
5	ReLU	[-1, 192, 27, 27]	0
6	MaxPool2d	[-1, 192, 13, 13]	0
7	Conv2d	[-1, 384, 13, 13]	663,936
8	ReLU	[-1, 384, 13, 13]	0
9	Conv2d	[-1, 256, 13, 13]	884,992
10	ReLU	[-1, 256, 13, 13]	0
11	Conv2d	[-1, 256, 13, 13]	590,08
12	ReLU	[-1, 256, 13, 13]	0
13	MaxPool2d	[-1, 256, 6, 6]	0
14	AvgPool2d	[-1, 256, 6, 6]	0
15	Dropout	[-1, 9216]	0
16	Linear	[-1, 4096]	37,752,832
17	ReLU	[-1, 4096]	0
18	Dropout	[-1, 4096]	0
19	Linear	[-1, 4096]	16,781,312
20	ReLU	[-1, 4096]	0
21	Linear	[-1, 1000]	4,097,000

Nonetheless, an overview of ResNet-18 is provided in Table 3. This transfer learning model includes 68 total sequential layers, including 18 depth levels. ResNet-18 is slightly more sophisticated than AlexNet in terms of the number of successive layers. Because of its complexity, building a ResNet-18 model may necessitate a lengthier computing procedure, particularly during the network training phase [18].

Table 3. ResNet Model Summary

ResNet-18			
#	***Layer (type)***	***Output Shape***	**Param#**
1	Conv2d	[-1, 64, 112, 112]	9,408
2	BatchNorm2d	[-1, 64, 112, 112]	128
3	ReLU	[-1, 64, 112, 112]	0
4	MaxPool2d	[-1, 64, 56, 56]	0
5	Conv2d	[-1, 64, 56, 56]	36,864
6	BatchNorm2d	[-1, 64, 56, 56]	128
7	ReLU	[-1, 64, 56, 56]	0
…	…	…	…
…	…	…	…
…	…	…	…
60	Conv2d	[-1, 512, 7, 7]	2,359,296
61	BatchNorm2d	[-1, 512, 7, 7]	1,024
62	ReLU	[-1, 512, 7, 7]	0
63	Conv2d	[-1, 512, 7, 7]	2,359,296
64	BatchNorm2d	[-1, 512, 7, 7]	1,024
65	ReLU	[-1, 512, 7, 7]	0
66	BasicBlock	[-1, 512, 7, 7]	0
67	AvgPool2d	[-1, 512, 1, 1]	0
68	Linear	[-1, 1000]	513,000

RESULTS AND DISCUSSION

This chapter discusses the analysis of the training and testing phase's outcomes. The first round of testing was supplying the ROS parameter value that was associated to the robot's motor speed and the projected threshold value. Overall model performance was determined by the amount of time each model took to train, the actual prediction accuracy, and the frequency of spin-outs along the observation.

Because the mobile robot's kinematic planning and route planning were not included in this study, those values were decided based on user subjectivity. ROS settings are 0.125 and 0.1 for forward and left-right motor speeds, respectively as listed in Table 4. The selected value produced a positive response in conjunction with the wheel traction. The prob_threshold setting of 0.5 might improve the robot's mobility by removing excessive spin. The time necessary to complete the training phase was recorded using an external timer. Table 5 displays the findings of the observation.

Model-1 and Model-4, both with pre-trained status and the same number of epochs, may be compared linearly. The sole distinction is the architectural foundation. This is likewise true for the Model-2 and Model-5, as well as the Model-3 and Model-6. Overall, AlexNet has a shorter training time than ResNet-18 [18].

The system's real performance in a test environment was used to determine the correctness of each model. The training accuracy rate was compared to the real-time accuracy. As a crucial indication, we used the accuracy numbers from the network training stage's test set.

Table 4. ROS Parameter

ROS Parameter	
speed.forward	0.125
speed.left	0.1
speed.right	0.1
prob_threshold	0.5

*ROS Parameter values given based on subjectivity since kinematic planning & analysis were not included in the research

Table 5. Training Duration

Model	**Architecture**	**Duration**
1	AlexNet	09:41:882
2	AlexNet	12:02:656
3	AlexNet	17:41:154
4	ResNet-18	13:40:752
5	ResNet-18	19:55:357
6	ResNet-18	32:43:104

Table 6. Prediction Accuracy Comparison

Model	***Test Accuracy***	***Real Accuracy***		
		Circuit	**Sprint**	***Overall***
1	1.00	0.9252	0.9191	0.9222
2	0.85	0.8453	0.7341	0.7897
3	0.95	0.9118	0.9043	0.9081
4	1.00	0.6290	0.5038	0.5664
5	1.00	0.8120	0.8404	0.8262
6	1.00	0.7570	0.6223	0.6897

Table 7. Deviation Rate

Model	***Dev***	***Dev (%)***	***AVG DEV***
1	0.0779	7.79%	
2	0.0603	6.03%	6.00%
3	0.0419	4.19%	
4	0.4336	43.36%	
5	0.1738	17.38%	30.59%
6	0.3104	31.04%	

According to the data in Table 6, the accuracy of some models from the training stage can exceed 1.00. These figures, however, are not trustworthy since we discovered evidence that those models were unable to achieve the accuracy generated from the training step. When the system's forecast was tested in the test environment, a deviation occurred. Table 7 shows the overall deviation values.

AlexNet has an overall accuracy deviation of 6.00%. ResNet-18, on the other hand, has a concerningly low average deviation of 30.59%. This scenario implies that ResNet-18 models have been overfitted. This overfitting problem would not have been discovered if the actual performance test had not been performed.

Spin Out Rate

In this study, "spin out" is a self-described condition. It is a situation in which the label prediction mistake happens repeatedly, causing the robot to spin in the wrong direction or to go off track. A "crash" state includes a "spin out" situation. The data shown below was derived from observations of mobile robot movement behavior. Spin out happens when the system constantly makes incorrect predictions, as seen in Figure 9. The robot recognizes the left boundary of the track and, according to the collision avoidance software, should do a right-hand maneuver. When a false prediction occurs repeatedly, the robot continues to spin, either in the wrong direction or off the track.

This metric can tell us how well the intelligence collision avoidance model performs. The spin out condition is relevant to the model's real accuracy. The poorer the model prediction accuracy, the more frequently the robot would spin off during testing.

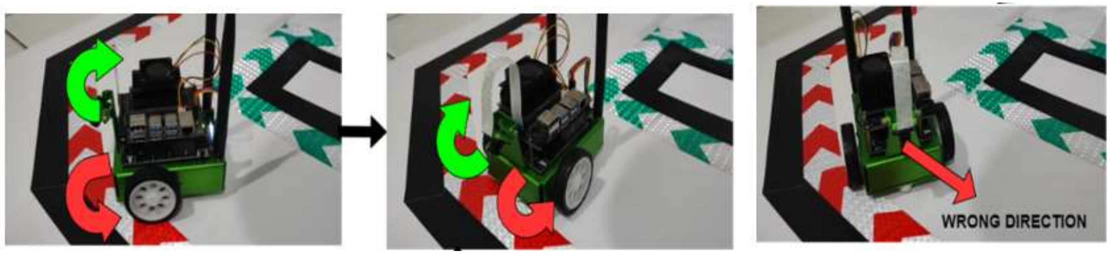
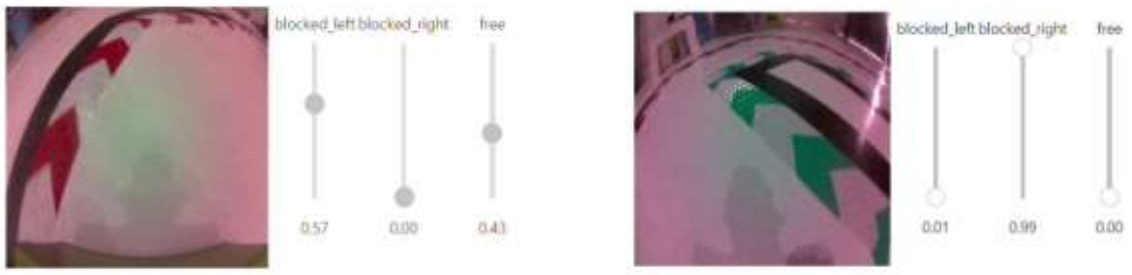


Figure 9. Spin Out Scheme

Table 8. Spin Out Rate

Model	**Circuit**		**Sprint**	
	Observation Time	**Spin Out**	**Observation Time**	**Spin Out**
1	01:47:40	N/A	01:39:60	N/A
2	03:14:31	4	02:53:13	9
3	01:42:70	N/A	01:55:62	N/A
4	02:04:61	5	04:24:07	16
5	02:13:12	4	02:43:53	3
6	01:57:14	6	03:08:91	9



(a) (b)

Figure 10. Captured Frame of (a) Right Maneuver and (b) Left Maneuver.

The ResNet-18 architecture is untrustworthy for building intelligent collision avoidance on mobile robots based on the facts supplied above. This is due to the ResNet-18-based models' proclivity to spin out mobile robots during observation. The Spin out rate can be observed by calculate the observation time over both the circuit or the sprint itself, shown in Table 8. Using Model 1, the robot completed the track in 1 minute and 47 seconds, with no spin-outs. Model 2 performs in 03 minutes 14 seconds with 4 times spin-outs in circuit and in 02 minutes and 53 seconds with 9 times spin outs. Table 8 evaluates the model performance of spin-outs scenario using the whole model. Figure 10 shows the captured frame while the robot observed the track.

CONCLUSION

This section includes an overview of the findings as well as a commentary. This section is split into two sections. The first is learning how to develop an intelligence collision avoidance on a mobile robot utilizing the AlexNet image classifier approach, which is relevant to the study purpose. Second, determine the performance of the chosen technique.

Intelligence Collision Avoidance on Mobile Robot was constructed in steps that included building hardware, configuring systems, writing collision avoidance programs, modeling test trajectories, and finally testing the real model. The Waveshare Robot Kit was used to build the robot's hardware, which sped up the assembling process. The NVIDIA Jetson Nano B01 was chosen as the microcomputer because it is capable of image processing. The Intelligent Collision Avoidance System was created with PyTorch as the DL framework and a CNN-based technique. The models were built using the transfer learning model. The test pathways are used as a testing environment to evaluate the model's real predicted performance. A competent model can anticipate the track's edge based on the label category, ensuring that the robot does not touch or exit the track.

Based on the results of the experiments, the accuracy value produced from the neural network training stage cannot be confirmed as a model performance indicator. To evaluate the model's real performance, it should be tested in a test environment. With a very short training period and a 6.00% average accuracy deviation, the Intelligence Collision Avoidance model based on the AlexNet image classifier approach was demonstrated to be the more dependable method when compared to ResNet-18. The best models were the Model-1 and Model-3, as indicated by no spin outs during the test.

ACKNOWLEDGMENT

This research was supported by Department of Electrical Engineering, Faculty of Engineering, Universitas Mercu Buana. We thank our colleagues from Department of Electrical Engineering, Faculty of Engineering, Universitas Mercu Buana who provided insights and expertise that greatly assisted the research, although they may not agree with all of the interpretations/conclusions of this paper.

REFERENCES

[1] Z. Iklima, A. Adriansyah, and S. Hitimana, “Self-Collision Avoidance of Arm Robot Using Generative Adversarial Network and Particles Swarm Optimization (GAN-PSO),” *SINERGI*, vol. 25, no. 2, p. 141, 2021, doi: 10.22441/sinergi.2021.2.005.

[2] Z. Iklima, B. N. Rohman, R. Muwardi, and Z. Arifiansyah, “Defect classification of radius shaping in the tire curing process using Fine-tuned Deep Neural Network,” *SINERGI*, vol. 26, no. 3, pp. 335-342, 2022, doi: 10.22441/sinergi.2022.3.009

[3] M. Andronie *et al.*, “Big Data Management Algorithms, Deep Learning-Based Object Detection Technologies, and Geospatial Simulation and Sensor Fusion Tools in the Internet of Robotic Things,” *ISPRS Int. J. Geo-Information 2023*, vol. 12, no. 2, p. 35, Jan. 2023, doi: 10.3390/IJGI12020035.

[4] Z. Pan, D. Li, K. Yang, and H. Deng, “Multi-Robot Obstacle Avoidance Based on the Improved Artificial Potential Field and PID Adaptive Tracking Control Algorithm,” *Robotica*, vol. 37, no. 11, pp. 1883–1903, Nov. 2019, doi: 10.1017/S026357471900033X.

[5] P. Narayanan *et al.*, “Analysis and design of fuzzy-based manoeuvring model for mid-vehicle collision avoidance system,” *J. Ambient Intell. Humaniz. Comput.*, vol. 12, no. 10, pp. 9909–9922, Oct. 2021, doi: 10.1007/S12652-020-02737-X/METRICS.

[6] E. Galan-Uribe, J. P. Amezquita-Sanchez, and L. Morales-Velazquez, “Supervised Machine-Learning Methodology for Industrial Robot Positional Health Using Artificial Neural Networks, Discrete Wavelet Transform, and Nonlinear Indicators,” *Sensors 2023*, vol. 23, no. 6, pp. 3213, Mar. 2023, doi: 10.3390/S23063213.

[7] S. Li, L. Wang, J. Li, and Y. Yao, “Image Classification Algorithm Based on Improved AlexNet,” *J. Phys. Conf. Ser.*, vol. 1813, p. 12051, 2021, doi: 10.1088/1742-6596/1813/1/012051.

[8] Y. Lu, “Image Classification Algorithm Based on Improved AlexNet in Cloud Computing Environment,” *2020 IEEE Int. Conf. Ind. Appl. Artif. Intell. IAAI 2020*, pp. 250–253, Dec. 2020, doi: 10.1109/IAAI51705.2020.9332891.

[9] J. Naranjo-Torres, M. Mora, R. Hernández-García, R. J. Barrientos, C. Fredes, and A. Valenzuela, “A Review of Convolutional Neural Network Applied to Fruit Image Processing,” *Appl. Sci. 2020*, vol. 10, no. 10, p. 3443, May 2020, doi: 10.3390/APP10103443.

[10] X. Luo, W. Wen, J. Wang, S. Xu, Y. Gao, and J. Huang, “Health classification of Meibomian gland images using keratography 5M based on AlexNet model,” *Comput. Methods Programs Biomed.*, vol. 219, p. 106742, Jun. 2022, doi: 10.1016/J.CMPB.2022.106742.

[11] F. Florencio, T. Valença, E. D. Moreno, and M. Colaço Junior, “Performance Analysis of Deep Learning Libraries: TensorFlow and PyTorch,” *J. Comput. Sci.*, vol. 15, no. 6, pp. 785–799, Apr. 2019, doi: 10.3844/JCSSP.2019.785.799.

[12] G. Dewantoro, J. Mansuri, and F. D. Setiaji, “Comparative Study of Computer Vision Based Line Followers Using Raspberry Pi and Jetson Nano,” *J. Rekayasa Elektr.*, vol. 17, no. 4, Dec. 2021, doi: 10.17529/JRE.V17I4.21324.

[13] A. Paszke *et al.*, “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” *Adv. Neural Inf. Process. Syst.*, vol. 32, Dec. 2019, Accessed: Feb. 14, 2024. [Online]. Available: https://arxiv.org/abs/1912.01703v1

[14] H. Dai, X. Peng, X. Shi, L. He, Q. Xiong, and H. Jin, “Reveal training performance mystery between TensorFlow and PyTorch in the single GPU environment,” *Sci. China Inf. Sci.*, vol. 65, no. 1, pp. 1–17, Jan. 2022, doi: 10.1007/S11432-020-3182-1/metrics.

[15] N. Mahmoud, Y. Essam, R. Elshawi, and S. Sakr, “DLBench: An Experimental Evaluation of Deep Learning Frameworks,” *Proc. - 2019 IEEE Int. Congr. Big Data, BigData Congr. 2019 - Part 2019 IEEE World Congr. Serv.*, pp. 149–156, Jul. 2019, doi: 10.1109/bigdatacongress.2019.00034.

[16] A. Kurniawan, “IoT Projects with NVIDIA Jetson Nano: AI-Enabled Internet of Things Projects for Beginners,” *IoT Proj. with NVIDIA Jetson Nano AI-Enabled Internet Things Proj. Beginners*, pp. 1–123, Dec. 2020, doi: 10.1007/978-1-4842-6452-2.

[17] S. Cass, “Nvidia makes it easy to embed AI: The Jetson nano packs a lot of machine-learning power into DIY projects - [Hands on],” *IEEE Spectr.*, vol. 57, no. 7, pp. 14–16, Jul. 2020, doi: 10.1109/MSPEC.2020.9126102.

[18] Q. H. Nguyen *et al.*, “Influence of data splitting on performance of machine learning models in prediction of shear strength of soil,” *Math. Probl. Eng.*, vol. 2021, 2021, doi: 10.1155/2021/4832864.

[19] V. Roshan, J. H. M. Stewart, R. Joseph, and H. M. Stewart, "Optimal ratio for data splitting," *Stat. Anal. Data Min. ASA Data Sci. J.*, vol. 15, no. 4, pp. 531–538, Aug. 2022, doi: 10.1002/SAM.11583.

[20] A. Rácz, D. Bajusz, and K. Héberger, "Effect of Dataset Size and Train/Test Split Ratios in QSAR/QSPR Multiclass Classification," *Molecules*, vol. 26, no. 4, Feb. 2021, doi: 10.3390/molecules26041111.

[21] V. F. Ochkov, A. Stevens, and A. I. Tikhonov, "Jupyter Notebook, JupyterLab-Integrated Environment for STEM Education," *2022 6th Int. Conf. Inf. Technol. Eng. Educ. Inforino 2022 - Proc.*, 2022, doi: 10.1109/inforino53888.2022.9782924.

[22] F. Albardi, H. M. Di. Kabir, M. M. I. Bhuiyan, P. M. Kebria, A. Khosravi, and S. Nahavandi, "A Comprehensive Study on Torchvision Pre-trained Models for Fine-grained Inter-species Classification," *Conf. Proc. - IEEE Int. Conf. Syst. Man Cybern.*, pp. 2767–2774, Oct. 2021, doi: 10.1109/SMC52423.2021.9659161.

[23] E. Bisong, "Building Machine Learning and Deep Learning Models on Google Cloud Platform," *Build. Mach. Learn. Deep Learn. Model. Google Cloud Platf.*, 2019, doi: 10.1007/978-1-4842-4470-8.

[24] K. K. Bressem, L. Adams, C. Erxleben, B. Hamm, S. Niehues, and J. Vahldiek, "Comparing Different Deep Learning Architectures for Classification of Chest Radiographs," *Sci. Rep.*, vol. 10, no. 1, Feb. 2020, doi: 10.1038/s41598-020-70479-z.

[25] S. I. (Sheldon) Winston and A. (Annisa) Jamali, "Axis Manipulation to Solve Inverse Kinematics of Hyper-Redundant Robot in 3D Space," *Journal of Integrated and Advanced Engineering (JIAE)*, vol. 2, no. 2, pp. 113–122, Sep. 2022, doi: 10.51662/jiae.V2I2.81.