

The implementation of the K-nearest neighbor algorithm to detect the KRSRI robot obstacles

Tigor Hamonangan Nasution¹, Arza Muhammad Prihandoyo¹, Seniman²

¹Department of Electrical Engineering, Faculty of Engineering, Universitas Sumatera Utara, Medan, Indonesia

²Department of Technology Information, Faculty of Computer Science and Information Technology, Universitas Sumatera Utara, Medan, Indonesia

Article Info

Article history:

Received Jan 23, 2024

Revised Oct 22, 2024

Accepted Nov 19, 2024

Keywords:

Gray-level co-occurrence matrix

K-nearest neighbor

Machine learning

Obstacle detection

Robot contests

Robots

ABSTRACT

The Indonesian SAR robot contest (KRSRI) is a development of the fire extinguisher robot contest (KRPAL); initially, the robot at KRPAL only put out fires. Still, at KRSRI, the robot was asked to prioritize the SAR function. The robot had to overcome obstacles in this contest to complete it. Based on this, an obstacle detection system for the robot was designed using machine learning with the K-nearest neighbor algorithm and gray level co-occurrence matrix feature extraction. Later, the robot is expected to be able to carry out accurate obstacle detection to prioritize efficiency so that no more time is consumed due to the robot incorrectly detecting an obstacle. The results of the tests that have been carried out show that the detection accuracy based on the test dataset is 80% for rising barriers, 100% for debris obstacles, and 90% for step obstacles, and an error value of 20% for increasing obstacles is obtained, 0% for debris obstacles, and 10% for stair obstacles.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



Corresponding Author:

Tigor Hamonangan Nasution

Department of Electrical Engineering, Faculty of Engineering, Universitas Sumatera Utara

Medan, Sumatera Utara, Indonesia

Email: tigor.nasution@usu.ac.id

1. INTRODUCTION

Nowadays, the development of research on robots continues to increase [1]-[4] including in Indonesia. The development of robotics in Indonesia is said to have been rapid, as indicated by the large number of participants participating in the Indonesian robot competition (KRI). The National Achievement Center and the Ministry of Education, Culture, Research, and Technology of the Republic of Indonesia organized the KRI. This contest is a forum for student creativity in robotics design and engineering. One of the Indonesian robot contest branches is the Indonesian SAR robot contest (KRSRI). This robot contest is a development of the fire extinguisher robot contest (KRPAL). The difference between the KRSRI and KRPAL competitions only lies in the task, but the robot's shape is still the same: a legged robot [5].

This Indonesian robot contest will continue the international robot competition, namely the RoboCup robot league RoboCup rescue division, if there is a winner in the KRI competition. The robot's task in the competition is to search for simulated victims showing signs of life in a maze with various rugged terrain. Apart from that, the robot must also be able to mark the victim's location and landmarks detected automatically on a map created by the robot online. These signs of life can include the victim's visual appearance and movements, simulated body heat, CO² levels, and audio signals [6].

The KRSRI robot was built by utilizing several distance sensors placed on the robot's body as a guide. However, the weakness of this system lies in the robot's limited ability to avoid obstacles, such as walls or corners. In the latest development of the KRSRI robot, the focus is more on the search and rescue

(SAR) function, so we decided to add obstacle elements such as debris, hollow up-and-down terrain, and pyramid-shaped steps. So, it is necessary to detect obstacles to prioritize the efficiency of the robot's ability to overcome these obstacles. This research uses machine learning with the K-nearest neighbor (KNN) classification algorithm, which is assisted by an image processing algorithm, namely, the gray-level co-occurrence matrix (GLCM) feature extraction, to recognize obstacle images. In this research, we use Python programming to process machine learning using the Scikit-learn Package library [7], [8]. Python programming is also used to program Arduino [9].

Several studies used the KNN algorithm to recognize images, such as prior research conducted by Islama *et al.* [10] entitled "HOG feature extraction and KNN classification for detecting vehicles on the highway," where this research used KNN and extraction image processing the histogram of oriented gradients (HOG) feature, this research aims to recognize car using the KNN algorithm, this research produces a detection accuracy of 84% in detecting cars. In the following previous study conducted by Mulyono *et al.* [11] entitled "Parijoto fruits classification using K-nearest neighbor based on gray level co-occurrence matrix texture extraction," in this research the KNN classification was used on parity fruit, this research used GLCM in obtaining the extraction features, the results of this research got an accuracy of 80% in detecting the parity fruit. In Akila and Pavithra [12] study entitled "Optimized scale invariant HOG descriptors for object and human detection," research using KNN and HOG feature extraction as human and object detectors in this research resulted in a positive detection rate of 86%. These studies show that using the KNN and GLCM algorithms is very good in designing systems for recognizing particular objects [13].

2. METHOD

2.1. System overview

In this research, the system for recognizing types of coffee bean roasting levels uses a computer to run the KNN classification algorithm for obstacle detection. This algorithm performs classification based on GLCM feature extraction of an image [14]. When the photo is taken, the system will immediately extract GLCM features based on degrees 0, 45, 90, and 135 to obtain energy, homogeneity, entropy, and contrast from each degree [15]-[17]. Figure 1 shows the design of the system.

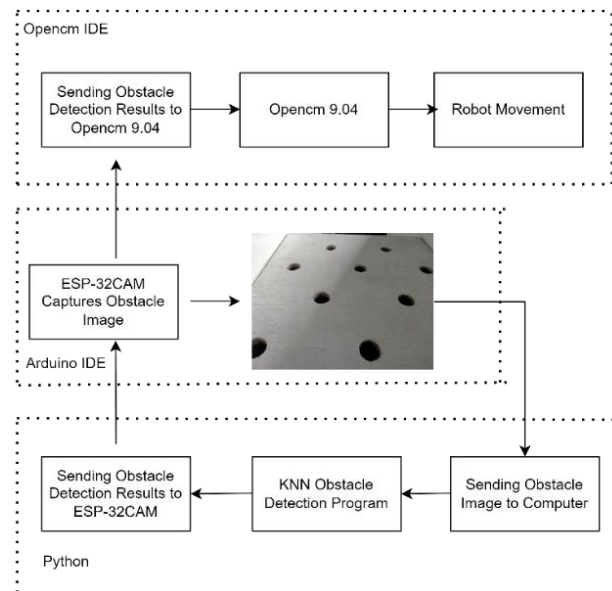


Figure 1. System block diagram

The system works in the block diagram through the integration and communication process carried out in this research, which begins with carrying out an obstacle recognition/detection program on the researcher's computer using the Python programming language. After that, communication is carried out between Python and ESP32-CAM using Wi-Fi; this is done to capture images in real time and receive characters for detected obstacles. After that, the characters will be sent to the OpenCM 9.04 microcontroller, and then the symbols received by OpenCM will influence the robot's movement. The workflow for the second stage of the system to be designed can be shown in Figure 2.

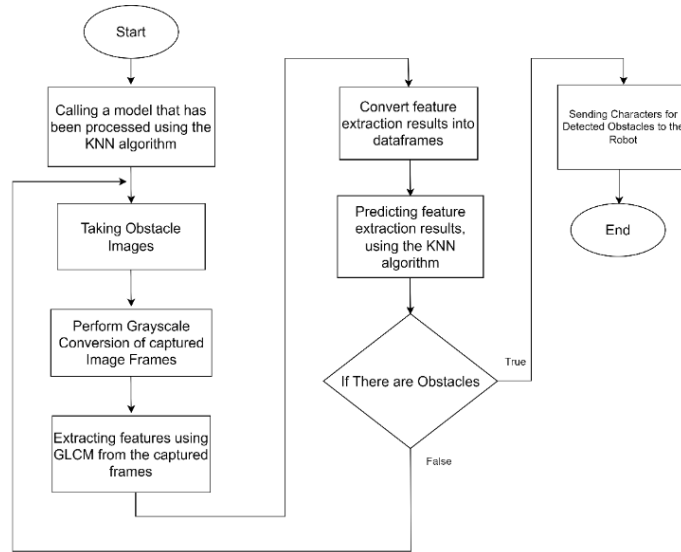


Figure 2. Obstacle detection system flowchart diagram

2.2. System design

Overall, as explained in the general description, this research has two workflow stages, where the first stage is machine learning modeling, and the second stage is the obstacle detection stage. Based on this, a system design was designed, which started by taking the obstacle dataset; for the dataset used, there were 150 datasets; after taking the dataset from the obstacles, it was continued by building a machine learning model using the KNN algorithm, after being trained it was continued with taking images of the challenges which would be detected, after that the final stage is sending the character to the KRSRI robot. The flow is arranged in a flowchart, as seen in Figure 3.

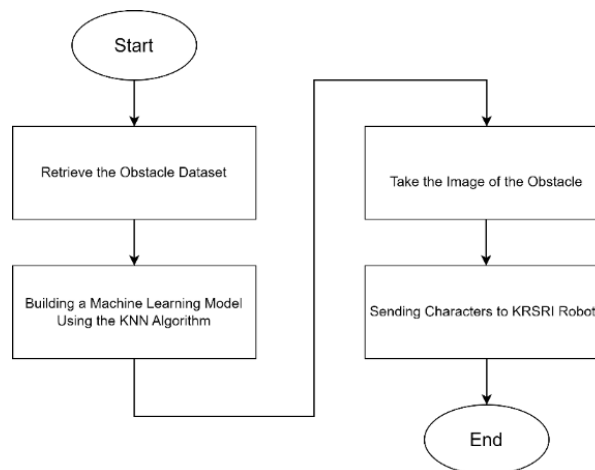


Figure 3. System design flowchart

Based on the flow diagram in Figure 3, the first stage begins with taking the obstacle dataset, which is taken so that the system can classify obstacle detection. This is because machine learning requires data to build models regarding machine learning later. The dataset is in the form of images of obstacles, inclines, and debris steps. Fifty photos were taken for each block, so the dataset had 150 images.

Based on the previous background, the researchers decided to build a machine learning model according to the design, and this research would use the KNN algorithm to classify obstacles. KNN cannot detect obstacles alone, so feature extraction is required. The feature extraction used is GLCM for the KNN algorithm, and this feature extraction is carried out on the researcher's computer. This feature extraction

involves a library from Python, namely open computer vision (OpenCV), where this library functions to simplify digital image processing [18]-[20].

The following is the GLCM feature extraction programming, which will later be used in the KNN algorithm. GLCM feature extraction is performed to help provide accuracy in the subsequent process in the learning machine. First, convert the initial image to grayscale, as shown in the following program code snippet.

```
gray = cv.cvtColor(im, cv.COLOR_BGR2GRAY) gray = cv.resize(gray, (128,128))
```

The following program is to extract GLCM features from angles 0, 45, 90, and 135, along with a snippet of the program code. In this programming, there is a table name variable where this variable holds the label from the file where the brand contains the location of the dataset, then the zero degree extraction label which has the extracted value of energy_0, homogeneity_0, entrophy_0, and contrast_0, as well as the extraction result label of degrees 45, 90, and 135.

```
namatabel=['file','energy_0','homogeneity_0',  
'entrophy_0','contrast_0','energy_45','homogeneity_45',  
'entrophy_45','contrast_45','energy_90','homogeneity_90',  
'entrophy_90','contrast_90','energy_135','homogeneity_135',  
'entrophy_135','contrast_135']  
df = pd.DataFrame(hasilnya, columns=namatabel)
```

After getting the extraction results from several features, the results are converted into a comma separated values (CSV) extension file, as shown in the following program code snippet. The CSV file conversion uses the Pandas library; this library is a Python library that has functions for carrying out data processing and statistical calculations aimed at data preprocessing [13], [21], [22].

```
df.to_csv(r'dataku.csv',index=False)
```

The next stage is to build a machine learning model using the KNN algorithm; after saving the extraction results, the next step is to create a machine learning model, which starts by making calls to the feature extraction results in CSV form using the Pandas library, then assigning a label to each feature extraction result where from data 0-50 has a rising brand, data 51-100 has a debris label, and the rest have a step label, then these labels are converted into an array which the model will later recognize, here is a snippet of the program code.

```
label = rintangan['label'].to_numpy()
```

The program code uses the numpy library, which is a library that Python has for carrying out scientific and mathematical needs [23], [9]. The next stage of building the model is dividing the training data and testing data using the Scikit-learn library, which is used for making machine learning models, from preprocessing to creating the model [7], [8].

```
from sklearn.model_selection import train_test_split xtrain, xtest, ytrain,  
ytest = train_test_split(data,label, test_size = 0.20, random_state=42)
```

The program code snippet above shows that the data is divided with a weight of 20% for test data and 80% for training data. After splitting the training and test data, the next step is to build a KNN model using the Scikit-learn library in Python, as in the following program code snippet.

```
model=KNeighborsClassifier(n_neighbors=3) model.fit(xtrain,ytrain)
```

In the program code snippet above, you can see a variable that contains a syntax for making classifications using KNN; in this syntax, there is a parameter, namely n_neighbors or the number of n neighbors; this is the KNN way of determining an object based on its nearest neighbors, where the n_neighbors parameter is used is 3. The next step is the process of taking pictures of obstacles. Obstacle image-taking is carried out by the ESP32-CAM, which acts as a camera module and a microcontroller. The role of the ESP32-CAM is as a container for capturing images in real time. In taking real-time photos using

The implementation of the K-nearest neighbor algorithm to detect ... (Tigor Hamonangan Nasution)

the esp32cam. The library is available on the Arduino IDE, which is used to access the ESP32CAM device via the Arduino IDE, making it possible to take videos or pictures [24], [25]. The following is a snippet of program code for image capture by ESP32-CAM.

```
void serverJpg() {
  auto frame = esp32cam::capture();
  if (frame == nullptr) {
    Serial.println("CAPTURE FAIL"); server.send(503, "", ""); return;}
  Serial.printf("CAPTURE OK %dx%d %db\n", frame->getWidth(), frame->getHeight(), static_cast<int>(frame->size()));
  server.setContentLength(frame->size()); server.send(200, "image/jpeg");
  WiFiClient client = server.client(); frame->writeTo(client);}

```

The program code snippet above is the program code for taking an image, where if the capture is successful, the function will print information about the photo taken, such as the image resolution and image size in bytes. This image information will be used in the following process. The captured image will be processed by machine learning so that the system can decide on detected obstacles by sending characters to the ESP32-CAM. The communication used between the ESP32-CAM and the KNN algorithm is Wi-Fi communication. We are sending feelings to robots. This character sending is carried out between the ESP32-CAM and the OpenCM 9.04 microcontroller with serial communication using the serial library in Python, which allows parallel communication between Python and Arduino for sending characters [9]. The following is a snippet of program code from the process.

```
void loop() {
  server.handleClient();
  WiFiClient kirim = ser.available();
  if (kirim){
    Serial.print("koneksi done"); while(kirim.connected())}
  if(kirim.available()){
    char terima = kirim.read(); Serial.print("terima ini");}
}

```

The program code snippet above shows the process for making client requests to the web server using the WebServer.h library. Which has the function of running the web server on the ESP32-CAM and checking the Wi-Fi connection that reads data from the link, if any, after taking images on the ESP32CAM then proceed with classification for obstacle detection. The program will display a message when the connection is successful, and the message is received.

3. RESULTS AND DISCUSSION

3.1. Preparation phase

The first stage is preparing the obstacles that the robot will detect. The obstacles that the robot will detect are debris obstacles, obstacles rising and falling with holes, and pyramid steps obstacles. Images of these obstacles are shown in Figure 4. Figure 4 shows the different levels of difficulty for each obstacle. Figure 4(a) shows a debris obstacle where debris is scattered on the floor. The next obstacle is a challenge to pass through an up-and-down field, as seen in Figure 4(b). Finally, in Figure 4(c), there is a pyramid obstacle where the obstacle is shaped like a pyramid the robot will pass.

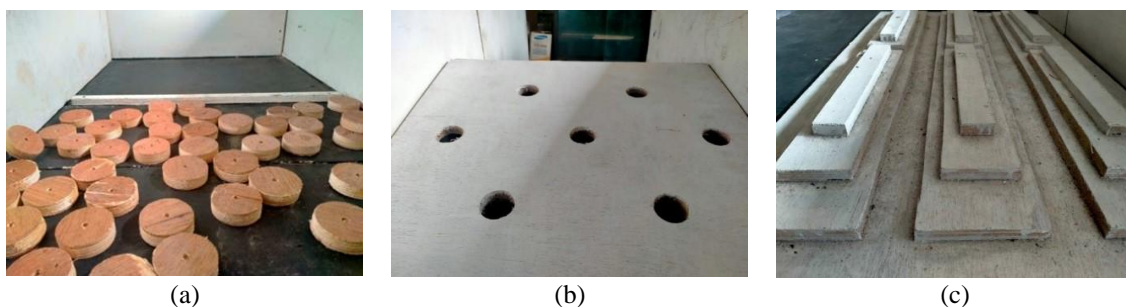


Figure 4. Obstacles that will be detected are; (a) debris obstacles (*puing*), (b) up and down obstacles (*naik*), and (c) pyramid step obstacles (*undakan*)

The second stage is determining the light intensity in the room where the test will occur. The power given must be sufficient so that when image processing is carried out, obstacles can be appropriately detected so the robot can pass through them. The test was carried out indoors because the light intensity from outside was not constant. Inconstant light can affect GLCM feature extraction values. The next step is preparing testing requirements. First, upload the ESP32CAM live streaming program, set up Wi-Fi for ESP32 CAM communication with Python, and upload the movement code for Opencm9.04. Figure 5 shows the configuration process to determine the light intensity. Figure 5(a) shows the position of Esp32 CAM on OpenCM 9.05. Figure 5(b) shows the laptop used to configure and monitor the configuration results.



Figure 5. Test preparation configuration; (a) position of Esp32 CAM on OpenCM 9.05 robot and microcontroller and (b) position of laptop with Python programming

3.2. Testing of the K-nearest neighbor algorithm

The test carried out on the KNN algorithm is to find out how much accuracy is obtained when the model makes predictions on test data. Before this test, the data received from the GLCM feature extraction results are shown in Table 1.

Table 1. Results of GLCM feature extraction on the obstacle dataset

Feature extraction	Up and down obstacles	Debris obstacles	Pyramid step obstacles
energy_0	14.136	5.373	6.718
homogeneity_0	16.825	635.932	4.100
entrophy_0	72.872	8.480	814.311
energy_0	16.725	6.349	4.090
energy_45	12.235	5.192	6.418
homogeneity_45	34.737	9.448	3.945
entrophy_45	7.452	8.578	819.00
energy_45	3.463	94.389	39.354
energy_90	13.508	6.677	10.204
homogeneity_90	2.836	6.445	13.645
entrophy_90	7.326	8.371	7.698
energy_90	2.826	6.435	13.545
energy_135	12.538	4.957	6.245
homogeneity_135	3.565	10.283	48.244
entrophy_135	73.935	8.596	819.420
energy_135	3.555	10.273	4.814

The dataset is 150, divided into 80% for training data and 20% for testing data. The features used are contrast, entropy, homogeneity, and energy obtained from the results of GLCM matrix extraction in degrees 0, 45, 90, and 135. Then, the model was tested on the test data and the results were obtained, as shown in Table 2.

$$Accuracy = \frac{\text{Number of Obstacles Detected}}{\text{Number of Obstacles in Test Data}} \times 100\% \quad (1)$$

Based on (1), the accuracy of the rising barrier is 80%, the debris barrier is 100%, and the step barrier is 90%. Then, after looking for the accuracy value, look for the error value of the detection results for

each obstacle. The error results for the obstacle rise are 20%, the debris obstacle is 0%, and the step obstacle is 10%.

Table 2. Model classification test results on test data

Obstacles	Number of obstacles in test data	Number of obstacles detected
Up and down obstacles	10	8
Debris obstacles	9	9
Pyramid step obstacles	11	10

Apart from analyzing the accuracy and error values in Table 2, obstacles that were not detected according to the test data were also examined. For example, with rising barriers, the number of blocks in the test data is 10. Still, when the detection results are obtained, the number is eight growing obstacles, so two growing obstacles are incorrectly detected. Likewise, with step obstacles, there are 11-step obstacles in the test data, but only 10-step obstacles are detected, so there is a 1-step obstacle that is incorrectly detected. Because there are detection errors in rising barriers and steps in the test data, an analysis is carried out to see what type of obstacle the system predicts for the wrong block. The following are the analysis steps.

The first step in this analysis is to compare the results of the test data output and the detection data from the obstacle objects shown in Figures 6 and 7. Figures 6 and 7 show the output of the test data and the prediction results for obstacle detection. The two works are in an array whose data length is 0 to 29. The test data output shown in Figure 6 has the 0th data, namely in the form of debris obstacles, the 1st data, namely rising barriers, and the 29th data, namely rising data. Likewise, the prediction output shown in Figure 7 has the 0th data, namely debris obstacles, the 1st data, climbing obstacles, and finally, the 29th data, namely steps data. In the two images, there are differences in the output; in the test data, the 20th data array shows a rising obstacle, but in the detection data, the 20th data from the variety shows a step obstacle. This can be seen from the test data output in the 24th array data, where the data shows a step obstacle, but the prediction data shows a debris obstacle. Finally, the 29th test data shows rising impediments, but the detection results show a step obstacle. Based on the detection error. Researchers try to analyze how the system makes errors in detecting these obstacles.

```
▶ ytest
array(['puing', 'naik', 'undakan', 'puing', 'puing', 'naik', 'puing',
      'undakan', 'puing', 'puing', 'undakan', 'naik', 'naik', 'naik',
      'naik', 'puing', 'undakan', 'puing', 'puing', 'undakan', 'naik',
      'undakan', 'naik', 'undakan', 'undakan', 'undakan', 'undakan',
      'undakan', 'naik', 'naik'], dtype=object)
```

Figure 6. Test data output results

```
▶ model.predict(xtest)
[]: array(['puing', 'naik', 'undakan', 'puing', 'puing', 'naik', 'puing',
        'undakan', 'puing', 'puing', 'undakan', 'naik', 'naik', 'naik',
        'naik', 'puing', 'undakan', 'puing', 'puing', 'undakan', 'undakan',
        'undakan', 'naik', 'undakan', 'puing', 'undakan', 'undakan',
        'undakan', 'naik', 'undakan'], dtype=object)
```

Figure 7. Test data detection prediction results

The second step is to carry out feature extraction. The feature extraction results will be stored in a variable in the form of an array. The results of feature extraction from the 20th array data output are shown in Figure 8.

From the data feature extraction results, the system will then make predictions to find the type of obstacle from the feature extraction. After getting the feature extraction results, the next step is to carry out calculations using the KNN algorithm, where in this algorithm, there is a distance measurement metric, namely Euclidean distance. Later, this measurement will carry out a decision obtained from voting the closest distance based on the nearest neighbors, as shown in Table 3.

```

xtest[20]
array([1.46703479e-03, 3.00714136e+02, 7.22822332e+00, 2.99714136e+02,
1.42596441e-03, 3.43756960e+02, 7.24803239e+00, 3.42756960e+02,
2.04848463e-03, 1.62433871e+02, 6.96448180e+00, 1.61433871e+02,
1.42022337e-03, 3.57580445e+02, 7.25685392e+00, 3.56580445e+02])
    
```

Figure 8. 20th test data output results

Table 3. Euclidean distance calculation preparation data

Feature extraction	Pyramid step obstacles	The 20th test data
Energy_0	0.0005.712	0.00146703479
homogeneity_0	338.570.374	300.714.136
Entropy_0	822.300.312	722.822.332
Contras_0	337.570.374	299.714.136
energy_45	0.0005.3532	0.00142596441
homogeneity_45	401.247.504	343.756.960
entropy_45	831.356.067	724.803.239
Contras_45	400.247.504	342756960
energy_90	0.0007.67416147	0.00204848463
homogeneity_90	214.285.433	162.433.871
entropy_90	794.648.228	696.448.180
Contras_90	213.285.433	161.433.871
energy_135	0.0005264	0.00142022337
homogeneity_135	459.182.528	357.580.445
entropy_135	830.466.252	725.685.392
Contras_135	458.182.528	356.580.445

In Table 3, the last date for the training data is a step obstacle, while the 20th test data obstacle is made as a question mark. This is done to see what obstacles the system will detect later. The following is a calculation of the last training data with the 20th test data. In this calculation, the last training data, namely the 119th training data, is represented by P, and the 20th test data is represented by Q:

$$\begin{aligned}
 d_{(P,Q)} &= \sqrt{(energy0_P - energy0_Q)^2 \dots + (contras135_P - contras135_Q)^2} \\
 d_{(P,Q)} &= \sqrt{(-0.000895)^2 + (37.8562377)^2 + \dots + (101.1602083)^2} \\
 d_{(P,Q)} &= \sqrt{0.0000008024 + 1433.09473 + \dots + 1032.29833} \tag{2}
 \end{aligned}$$

where P is final training data, Q is 20th test data. So, we get the Euclidean distance from the 120th training data and the 20th test data.

$$\begin{aligned}
 d_{(P,Q)} &= \sqrt{33503.837848968906} \\
 d_{(P,Q)} &= 118.424 \tag{3}
 \end{aligned}$$

After getting the Euclidean distance from a piece of data, sorting is done from the smallest distance to the most significant distance, then poll or vote based on the smallest space; the poll is done using the neighbor value approach; the neighbor value in this system is three so the poll selects the three closest neighbors and looks at what types of obstacles appear most often in the three neighbors? The following is an Euclidean distance table sorted from the smallest distance, shown in Table 4.

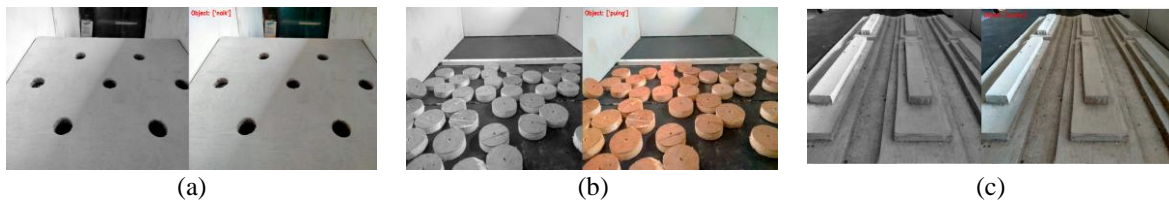
Table 4 represents the Euclidean distance calculation table for the entire training data against the 20th test data, where the table has been sorted based on the smallest Euclidean distance. After sorting, polling starts based on the three closest Euclidean distances. This is based on using the nearest neighbor in this algorithm, namely 3. A step obstacle is detected for the first most relative distance, and a step obstacle is detected for the second closest distance. At the third closest distance, a rising obstacle is detected. Then, a poll is carried out at that distance on the number of blocks that appear. In this case, the obstacle that occurs most frequently is the step obstacle. Therefore, the 20th test data is detected as a step by the system.

Table 4. Data result of Euclidean distance calculation

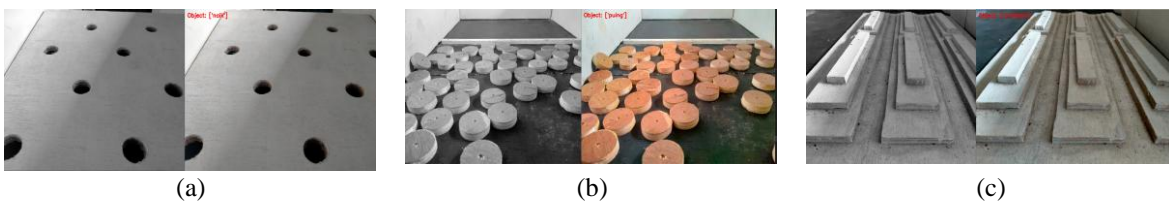
Feature extraction	Pyramid step obstacles	Pyramid step obstacles	Up and down obstacles	Pyramid step obstacles	Pyramid step obstacles
Energ_0	0.000574	0.000362	0.001942	0.000770	0.000687
homogeneity_0	314.448.019	293.388.226	274.136.257	308.078.371	324.365.157
Entropy_0	8.209.360	8.545.028	7.053.243	7.991.112	8.079.789
Contras_0	313.448.019	292.388.226	273.136.257	307.078.371	323.365.157
energy_45	0.000554	0.000344	0.001885	0.000719	0.000645
homogeneity_45	348.416.703	381.997.458	334.023.064	317.058.218	333.723.541
entropy_45	8.258.206	8.639.878	7.086.753	8.058.085	8.142.195
Contras_45	347.416.703	380.997.458	333.023.064	316.058.218	332.723.541
energy_90	0.000883	0.000533	0.002733	0.001045	0.000927
homogeneity_90	118.747.109	133.485.482	149.680.795	120.708.477	132.745.509
entropy_90	7.772.838	8.143.454	6.777.181	7.653.675	7.768.198
Contras_90	117.747.109	132.485.482	148.680.795	119.708.477	131.745.509
energy_135	0.000540	0.000322	0.001931	0.000704	0.000641
homogeneity_135	350.236.034	371.556.265	317.621.923	375.553.103	396.446.463
entropy_135	8.269.659	8.658.497	7.069.122	8.063.625	8.136.208
Contras_135	349.236.034	370.556.265	316.621.923	374.553.103	395.446.463
Euclidean distance	65.949.032	71.454.186	71.561.446	75.263.028	78.147.020

3.3. Obstacle detection testing of robot position

In the experiment, the robot will detect debris obstacles, up-and-down obstacles with holes, and pyramid steps obstacles. The robot's detection of obstacles can be influenced by the distance at which the obstacles are taken. The space for taking obstacles is divided into three, namely at a distance of 5 cm, 10 cm, and 15 cm. The results of obstacle detection tests at distances of 5 cm, 10 cm, and 15 cm are shown in Figures 9 to 11. Figures 9(a), 10(a), and 11(a) are test result images for up-down obstacles. Then, Figures 9(b), 10(b), and 11(b) are test result images for debris obstacles. Finally, Figures 9(c), 10(c), and 11(c) show test result images for pyramid obstacles. The results of the obstacle detection test based on robot distance can be shown in Table 5.



Figures 9. Obstacle detection experiment at a distance of 5 cm; (a) up and down obstacles, (b) debris obstacles, and (c) pyramid step obstacles

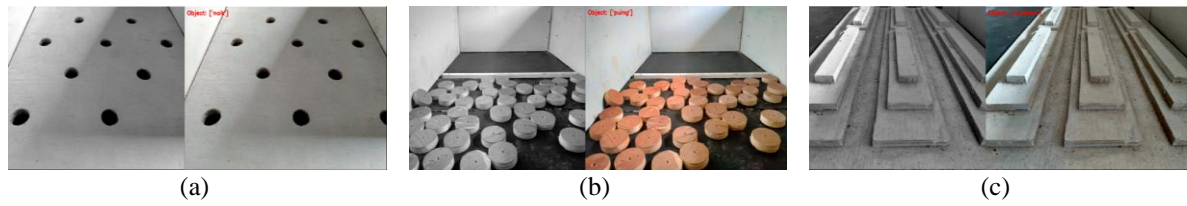


Figures 10. Obstacle detection experiment at a distance of 10 cm; (a) up and down obstacles, (b) debris obstacles, and (c) pyramid step obstacles

Table 6 shows that the rising obstacle is detected as an increasing obstacle for debris obstacles at a distance of 5 cm. The debris obstacle is seen as a debris obstacle. Still, for the step obstacle, a debris obstacle is detected; at a distance of 10 cm, all obstacles are detected well, and at a distance of 15 cm, they are seen well.

There was a detection error at a distance of 5 cm, namely at the step obstacle, as in the previous experiment, there was an error in detecting the obstacle. Therefore, the researcher carried out an analysis of

the obstacle that was incorrectly identified. In the first step, GLCM feature extraction is done for the detected obstacles, as shown in Table 6.



Figures 11. Obstacle detection experiment at a distance of 15 cm; (a) up and down obstacles, (b) debris obstacles, and (c) pyramid step obstacles

Table 5. Obstacle detection test results based on robot distance

Distances (cm)	Obstacles	Obstacle detected
5	Up and down obstacles	Up and down obstacles
	Debris obstacles	Debris obstacles
	Pyramid step obstacles	Debris obstacles
10	Up and down obstacles	Up and down obstacles
	Debris obstacles	Debris obstacles
	Pyramid step obstacles	Pyramid step obstacles
15	Up and down obstacles	Up and down obstacles
	Debris obstacles	Debris obstacles
	Pyramid step obstacles	Pyramid step obstacles

Table 6. GLCM feature extraction results for detected obstacles

Distances	5 cm			10 cm			15 cm		
	Up and down	Debris	Pyramid step	Up and down	Debris	Pyramid step	Up and down	Debris	Pyramid step
Energ_0	0.001273	0.00035 3	0.000311	0.001563	0.00025 5	0.000362	0.001224	0.0003 7	0.000311
homogeneity_0	201.710.0 76	616.769 .377	510.382. 259	128.211. 245	65.970. 577	812.479.26 9	188.893. 639	650.36 0.359	539.681. 841
Entropy_0	7.493.996	8.574.5 29	8.597.93 7	734.307	8.706.7 31	8.693.339	7.425.95 8	8.518.8 53	8.578.89 9
Contras_0	200.710.0 76	615.769 .377	509.382. 259	127.211. 245	65.870. 577	811.479.26 9	187.893. 639	649.36 0.359	538.681. 841
energy_45	0.001119	0.00030 4	0.000276	0.001342	0.00020 7	0.000304	0.001035	0.0003 43	0.000267
homogeneity_4	317.513.1 13	987.410 .689	523.036. 084	218.788. 518	100.272 .069	103.420.48 5	248.451. 485	980.40 9.077	543.695. 269
entropy_45	7.604.155	8.766.1 02	8.686.06 9	7.478.76 1	8.926.5 88	8.823.258	7.577.13 4	8.668.0 45	8.695.61 4
Contras_45	316.513.1 13	986.410 .689	522.036. 084	217.788. 518	100.172 .069	103.320.48 5	247.451. 485	979.40 9.077	542.695. 269
Contras_90	0.001255	0.00034 2	0.000393	0.001439	0.00024 2	0.000413	0.001141	0.0003 86	0.000361
homogeneity_9	200.926.9 19	689.547 .921	320.002. 707	176.376. 907	670.882 .136	651.503.69 1	16.273.6 59	621.25 6.582	323.706. 508
entropy_90	7.485.536	8.670.1 58	8.385.82 8	7.431.43 3	8.793.1 22	8.564.934	7.485.44 5	8.548.9 39	8.423.70 5
Contras_90	199.926.9 19	688.547 .921	319.002. 707	175.376. 907	669.882 .136	650.503.69 1	16.173.6 59	620.25 6.582	322.706. 508
Contras_135	0.00111	0.00028 5	0.000298	0.00136	0.00020 5	0.000322	0.001018	0.0002 93	0.000289
homogeneity_1	319.826.6 48	10.436. 479	656.622. 171	237.777. 172	106.766 .836	107.698.38 8	264.046. 004	104.67 5.386	688.887. 842
entropy_135	7.660.904	8.810.9 71	8.649.63 7	7.528.47 3	8.932.9 18	8.790.166	7.623.41 8	8.745.5 58	864.189
Contras_135	318.826.6 48	10.426. 479	655.622. 171	236.777. 172	106.666 .836	107.598.38 8	263.046. 004	104.57 5.386	687.887. 842
Detected	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes

Table 6 shows GLCM feature extraction results for detected obstacles. It can be seen that the feature extraction from the step obstacle at a distance of 5 cm had an error in its detection, whereas if you refer to Table 5, the obstacle was detected as debris. After extracting GLCM features for these obstacles, the next step is to look for the Euclidean distance metric based on GLCM feature extraction from training data and data on incorrectly detected obstacles. Following are the calculations. In this calculation, X represents the final training data, and Y defines the obstacle data so that:

$$d_{(X,Y)} = \sqrt{(energy0_x - energy0_y)^2 \dots + (contras135_x - contras135_y)^2} \quad (4)$$

from (4), there is an Euclidean distance from the training and obstacle data; a poll will be conducted to determine the type of obstacle detected. Before completing the vote, the following table of the Euclidean distance calculation results is shown in Table 7.

After getting the results of the Euclidean distance calculation, a poll is carried out on the data where the data to be polled is 3 Euclidean distance data, which has the shortest distance; this is based on the selection of neighbor values in the previous KNN algorithm which is 3. So, we get the obstacle with the shortest distance, namely the debris obstacle, step obstacles, and debris obstacles because most polls are debris obstacles; therefore, the system detects these obstacles as debris obstacles. The analysis concluded that a distance of 10 cm and a distance of 15 cm are ideal distances for detection using the KNN algorithm.

Table 7. Data result of calculating the Euclidean distance between training data and obstacle data

Feature extraction	Debris obstacles	Pyramid step obstacles	Debris obstacles	Up and down obstacles	Pyramid step obstacles
Energ_0	0.000677	0.000627	0.000514	0.000808	0.000453
homogeneity_0	496.537.340	407.669.291	597.492.987	372.860.544	334.278.728
Entropy_0	8.091.818	8.237.896	8.264.116	7.782.769	8.351.820
Contras_0	495.537.340	406.669.291	596.492.987	371.860.544	333.278.728
energy_45	0.000606	0.000599	0.000468	0.000765	0.000400
homogeneity_45	626.802.406	516.395.375	704.223.634	521.310.993	582.935.024
entropy_45	8.203.666	8.303.369	8.358.180	7.849.869	8.532.961
Contras_45	625.802.406	515.395.375	703.223.634	520.310.993	581.935.024
energy_90	0.000971	0.000903	0.000722	0.000979	0.000470
homogeneity_90	280.741.326	217.858.083	314.421.506	284.704.909	429.642.163
entropy_90	7.804.565	7.866.032	7.984.941	7.631.175	8.401.191
Contras_90	279.741.326	216.858.083	313.421.506	283.704.909	428.642.163
energy_135	0.000617	0.000598	0.000473	0.000779	0.000382
homogeneity_135	626.803.150	520.120.590	722.124.558	480.067.332	573.162.254
entropy_135	8.222.099	8.298.807	8.369.625	7.836.752	8.588.029
Contras_135	625.803.150	519.120.590	721.124.558	479.067.332	572.162.254
Euclidean distance	163.647.585	281.640.172	299.129.233	320.418.604	327.373.636

4. CONCLUSION

Several conclusions are drawn based on the results of the experimental test and analysis. First, from testing using the test dataset, it can be concluded that the system achieved the highest accuracy level in detecting debris obstacles, reaching 80% for rising obstacles, 100% for debris obstacles, and 90% for step obstacles. These results indicate that the system is more effective in recognizing debris obstacles than others.

Furthermore, analysis of the test error values shows that the debris barrier has the lowest error value, only around 20%. This is obtained based on calculations from test data that has been carried out. Meanwhile, debris obstacles have no error value, and step obstacles have an error value of 10%. This indicates that the system achieves the highest level of accuracy when detecting debris obstacles. In addition, the analysis also reveals that distance greatly influences obstacle detection. Test results show that the ideal space to detect all obstacles is around 10 cm and 15 cm. It is essential to understand the operational limitations of robots in recognizing obstacles well.

Finally, the KNN method was proven to classify obstacles for debris, pyramid steps, and up-down obstacles. However, the accuracy of the results depends on the data collection and the distance from the data collection to the obstacle to be detected. This could also potentially impact feature extraction from the GLCM used in the analysis. Therefore, dataset selection and data collection distance are critical factors in improving the performance of obstacle detection systems. This research has limited data collection time and uses a low-resolution camera. Future research will be carried out using cameras that have higher resolution and collect more data.





ACKNOWLEDGEMENTS

This research was carried out with financial support from the Ministry of Education, Culture, Research, and Technology of the Republic of Indonesia through a 2023 Fundamental Research Grant based on Decree Number 0217/E5/PG.02.00/2023 and Agreement/Contract Number 061/E5/PG.02.00.PL/2023.





REFERENCES

- [1] D. Zeng, Y. Liu, C. Qu, J. Cong, Y. Hou, and W. Lu, "Design and human-robot coupling performance analysis of flexible ankle rehabilitation robot," *IEEE Robotics and Automation Letters*, pp. 1–8, Nov. 2023, doi: 10.1109/LRA.2023.3330052.
- [2] T. H. Nasution, A. Simon, F. Fahmi, K. Tanjung, M. Zarlis, and N. Noer, "Analysis of the use of CMUcam5 Pixy camera in wheeled soccer robots," *IOP Conference Series: Materials Science and Engineering*, vol. 851, no. 1, p. 012034, May 2020, doi: 10.1088/1757-899X/851/1/012034.
- [3] S. L. Chen and L. W. Huang, "Using deep learning technology to realize the automatic control program of robot arm based on hand gesture recognition," *International Journal of Engineering and Technology Innovation*, vol. 11, no. 4, pp. 241–250, Aug. 2021, doi: 10.46604/IJETI.2021.7342.
- [4] S. A. Dewangga, H. Tjandrasa, and D. Herumurti, "Robot motion control using the Emotiv EPOC EEG system," *Bulletin of Electrical Engineering and Informatics*, vol. 7, no. 2, pp. 279–285, Jun. 2018, doi: 10.11591/eei.v7i2.678.
- [5] B. Kusumoputro *et al.*, "Guide to the 2022 Indonesian Robot Contest (KRI) (in Indonesian: *Panduan kontes robot Indonesia (KRI) tahun 2022*)," Pusat Prestasi Nasional, Kementerian Pendidikan, Kebudayaan, Riset dan Teknologi, 2022. [Online]. Available: [https://kontesrobotindonesia.id/kri-2022.html#/. Accessed: Dec. 30, 2022.](https://kontesrobotindonesia.id/kri-2022.html#/)
- [6] R. Sheh, S. Schwertfeger, and A. Visser, "16 years of RoboCup rescue," *KI - Kunstliche Intelligenz*, vol. 30, no. 3–4, pp. 267–277, Oct. 2016, doi: 10.1007/S13218-016-0444-X/FIGURES/8.
- [7] J. Hao and T. K. Ho, "Machine learning made easy: a review of Scikit-learn package in Python programming language," *Journal of Educational and Behavioral Statistics*, vol. 44, no. 3, pp. 348–361, Jun. 2019, doi: 10.3102/1076998619832248.
- [8] S. Bhattacharya, S. Kundu, B. Mukhopadhyay, and S. Bhattacharya, "Finding a baseline machine learning model in Sci-Kit Learn package in terms of accuracy and efficiency," *2021 IEEE Pune Section International Conference, PuneCon 2021*, 2021, doi: 10.1109/PUNECON52575.2021.9686536.
- [9] I. J. Koenka, J. Sáiz, and P. C. Hauser, "Instrumentino: an open-source modular Python framework for controlling Arduino based experimental instruments," *Computer Physics Communications*, vol. 185, no. 10, pp. 2724–2729, Oct. 2014, doi: 10.1016/J.CPC.2014.06.007.
- [10] F. Al Islama, A. Putra, F. Utaminigrum, and W. F. Mahmudy, "HOG feature extraction and KNN classification for detecting vehicle in the highway," *Indonesian Journal of Computing and Cybernetics Systems (IJCCS)*, vol. 14, no. 3, pp. 231–242, Jul. 2020, doi: 10.221146/IJCCS.54050.
- [11] I. U. W. Mulyono *et al.*, "Parijoto fruits classification using K-nearest neighbor based on gray level co-occurrence matrix texture extraction," *Journal of Physics: Conference Series*, vol. 1501, no. 1, p. 012017, Mar. 2020, doi: 10.1088/1742-6596/1501/1/012017.
- [12] K. Akila and P. Pavithra, "Optimized scale invariant HOG descriptors for object and human detection," *IOP Conference Series: Materials Science and Engineering*, vol. 1119, no. 1, p. 012002, Mar. 2021, doi: 10.1088/1757-899X/1119/1/012002.
- [13] M. Bansal, A. Goyal, and A. Choudhary, "A comparative analysis of K-nearest neighbor, genetic, support vector machine, decision tree, and long short term memory algorithms in machine learning," *Decision Analytics Journal*, vol. 3, p. 100071, Jun. 2022, doi: 10.1016/J.DAJOUR.2022.100071.
- [14] A. A. Fauzi, F. Utaminigrum, and F. Ramdani, "Road surface classification based on LBP and GLCM features using kNN classifier," *Bulletin of Electrical Engineering and Informatics*, vol. 9, no. 4, pp. 1446–1453, Aug. 2020, doi: 10.11591/EEI.V9I4.2348.
- [15] T. H. Nasution and U. Andayani, "Recognition of roasted coffee bean levels using image processing and neural network," in *IOP Conference Series: Materials Science and Engineering*, 2017, doi: 10.1088/1757-899X/180/1/012059.
- [16] T. Subramani, V. Jeganathan, and S. K. Balasubramanian, "An effective supervised machine learning approach for indian native chicken's gender and breed classification," *Proceedings of Engineering and Technology Innovation*, vol. 24, pp. 73–86, Apr. 2023, doi: 10.46604/PETI.2023.11361.
- [17] X. Zhang, X. Su, Q. Yuan, and Q. Wang, "Spatial-temporal gray-level co-occurrence aware CNN for sar image change detection," *IEEE Geoscience and Remote Sensing Letters*, vol. 19, 2022, doi: 10.1109/LGRS.2021.3110302.
- [18] R. Laganier, *OpenCV 2 computer vision application programming cookbook*, First. Birmingham: Packt Publishing, 2011.
- [19] J. Sigut, M. Castro, R. Arnay, and M. Sigut, "OpenCV basics: a mobile application to support the teaching of computer vision concepts," *IEEE Transactions on Education*, vol. 63, no. 4, pp. 328–335, Nov. 2020, doi: 10.1109/TE.2020.2993013.
- [20] J. S. Sheu, C. K. Tsai, and P. T. Wang, "Driving assistance system with lane change detection," *Advances in Technology Innovation*, vol. 6, no. 3, pp. 137–145, May 2021, doi: 10.46604/AITI.2021.7109.
- [21] R. Betancourt and S. Chen, "Pandas library," *Python for SAS Users*, pp. 65–109, 2019, doi: 10.1007/978-1-4842-5001-3_3.
- [22] I. Stancin and A. Jovic, "An overview and comparison of free Python libraries for data mining and big data analysis," *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2019 - Proceedings*, pp. 977–982, May 2019, doi: 10.23919/MIPRO.2019.8757088.
- [23] M. Rocha and P. G. Ferreira, "An introduction to the Python language," *Bioinformatics Algorithms*, pp. 5–58, Jan. 2018, doi: 10.1016/B978-0-12-812520-5.00002-X.
- [24] S. Kumar, K. Sharma, G. Raj, D. Datta, and A. Ghosh, "Arduino and ESP32-CAM-based automatic touchless attendance system," *Lecture Notes in Electrical Engineering*, vol. 851, pp. 135–144, 2022, doi: 10.1007/978-981-16-9154-6_14/COVER.
- [25] V. Surendar, A. A. Pillai, K. R. A. Rakhav, D. Kirthiga, and V. Kavin, "Automation in lab diagnosis of smears using Arduino," *2nd International Conference on Sustainable Computing and Data Communication Systems, ICSCDS 2023 - Proceedings*, 2023, pp. 1102–1106, doi: 10.1109/ICSCDS56580.2023.10104872.





BIOGRAPHIES OF AUTHORS

Tigor Hamonangan Nasution     is a lecturer in electrical engineering at the Universitas Sumatera Utara. He has been a lecturer since 2014 until now. He is also a Senior Member of IEEE. He completed his masters in electrical engineering in 2014 and is currently studying for a Doctorate at the Department of Computer Science, Universitas Sumatera Utara. He is active in research in the field of computer engineering, especially in the fields of embedded systems, and artificial intelligence. He can be contacted at email: tigor.nasution@usu.ac.id.



Arza Muhammad Prihandoyo     is an undergraduate at the Department of Electrical Engineering, Universitas Sumatera Utara. He is also the former Universitas Sumatera Utara Student Robotics Unit Chair. He actively helps several lecturers with research in the field of computer engineering. He can be contacted at email: mrarza2566@gmail.com.



Seniman     is a lecturer at the Department of Information Technology, Universitas Sumatera Utara. He has done extensive research in the field of computer science and has produced several scientific publications. He is also active in guiding student robotics competition activities. He can be contacted at email: pakniman@usu.ac.id.