p-ISSN 2301–4156 | e-ISSN 2460–5719

Improving the Adaptive Monte Carlo Localization Accuracy Using a Convolutional Neural Network

Riza Agung Firmansyah[1], Tri Arief Sardjono[2], Ronny Mardiyanto[3]

[1,2,3] Department of Electrical Engineering Faculty of Intelligent Electrical and Informatics Technology Institut Teknologi Sepuluh Nopember, Surabaya, 60111 INDONESIA (tel.: 031-5994251-54, ext. 1206/031-5947302; fax: 031-5931237, email: [1]7022211015@mhs.its.ac.id, [2]sardjono@bme.its.ac.id, [3]ronny@elect-eng.its.ac.id)

[Received: 30 March 2023, Revised: 14 June 2023]
Corresponding Author: Ronny Mardiyanto

ABSTRACT — This paper explains the increase in localization system accuracy of the adaptive Monte Carlo localization (AMCL) in robots utilizing a convolutional neural network (CNN). The localization system in robots is defined as the position recognition process of robots within their working environment. This system is essential as it allows robots to navigate and map efficiently and accurately. Without appropriate localization, robots cannot operate effectively and can encounter troubles such as losing direction or bumping into objects. AMCL is a popular localization system and is widely applied in robots. This method utilizes the changes in the robots' position and light detection and ranging (LiDAR) sensor reading as input. Reading of robot position changes is susceptible to error due to slips or wheel deformations. The inaccuracy of reading the robots' position change results in the inaccuracy of the robots' position prediction by AMCL, so improvements are required. Novelty in this paper includes providing compensation values from AMCL results for the error to be small. These compensation values were obtained from the CNN training results; hence, the proposed method was dubbed AMCL+CNN. Inputs given to the CNN were the changes in wheel odometry values and distance reading by the LiDAR sensor. CNN outputs were compared to the target data in the form of the robots' actual position from observation results. Network training was conducted for as many as 200 epochs to achieve the lowest validation loss. Testing was done on a robot installed with a robot operating system (ROS). Training and testing datasets were obtained from rosbag data when the robot traversed the testing area. In straight and turn scenarios, obtained AMCL+CNN algorithms had fewer errors than the regular AMCL and Monte Carlo localization (MCL). Results obtained are also superior in terms of positional error metrics when compared to several other comparison methods.

KATA KUNCI — Robot Localization, LiDAR, AMCL, CNN.

I. INTRODUCTION

Mobile robots have been utilized in some sectors like agriculture [1], health [2], hospitality [3], and tourism [4]. In doing the work, robots have equipment installed, such as grippers [5], tanks on the AGV robot [6], and health measurement equipment to measure persons' vital signs [7]–[9]. Robots must perceive their actual position to work correctly. If not, they cannot work as per previously given plans or works [10]. This process is commonly dubbed robot localization.

To support localization systems indoors, generally, robots use 2D LiDAR scanner sensors [10]-[12]. Those sensors have a pretty good ranging capability in all directions (360°). With rapid development in robotics, 3D LiDAR sensors have become prevalently employed [13], [14]. The ranging capability of 3D LiDAR is better than that of 2D LiDAR, but its price in markets is rather high. Visual-based sensors, such as RGB camera, has the ability to recognize features around robots [15], [16]. However, the RGB camera's ranging capability is not as good as that of LiDAR, so this sensor still needs to be paired with other sensors. In addition, feature recognition cannot work optimally in featureless environments. Other visual-based sensors, such as RGB-D cameras, have ranging capabilities based on depth [17]–[19]. However, its range is limited to around 87° in the horizontal direction.

The localization system is one of the essential stages in robotic research as it is the basis for the navigation system and map building [20]. The accuracy and efficiency of the localization system play a crucial role in the robots' performance when carrying out their work. Therefore, it is necessary to conduct research on improving the accuracy of measurement and efficiency. The efficiency in question is computational efficiency and cost efficiency. An increase in performance can be carried out using good-quality sensors or a combination of several sensors or sensor fusion [4], [21], [22].

One frequently used localization method is the particle filter-based method, namely the Monte Carlo localization (MCL). Despite its need-to-be-addressed shortcomings, this method can effectively predict the robots' location. MCL requires numerous accurate and up-to-date sensor data and environmental information to generate an accurate estimation. If sensor data are unavailable or inaccurate, the MCL results will be very unsatisfactory. Furthermore, this method is extremely sensitive to environmental changes, for example, layout modifications or new object additions. It implies that the algorithm must be regularly updated to guarantee accurate results. MCL also necessitates longer computation and time to compute an accurate location probability, which may lead to a delay in robot responses. Therefore, it is essential to find an alternate solution to resolve these shortcomings in the MCL implementation to the robots.

An adaptive MCL (AMCL) is another variation of the MCL algorithms used to determine the position and heading of robots in an unknown environment. The AMCL combines the Kalman filter and MCL to adjust the particle distribution and reduce the required number of particles to achieve the same accuracy. It adapts the required number of particles in response to changing environmental conditions. This way enables AMCL to optimize the particles needed and increase efficiency. Generally, AMCL is used to ascertain the robots' position and heading in an unknown environment more rapidly and accurately than the regular MCL. The AMCL combines the MCL technique and the Kalman filter to improve the accuracy and robustness of the robots' position estimation. Nevertheless,

p-ISSN 2301–4156 | e-ISSN 2460–5719

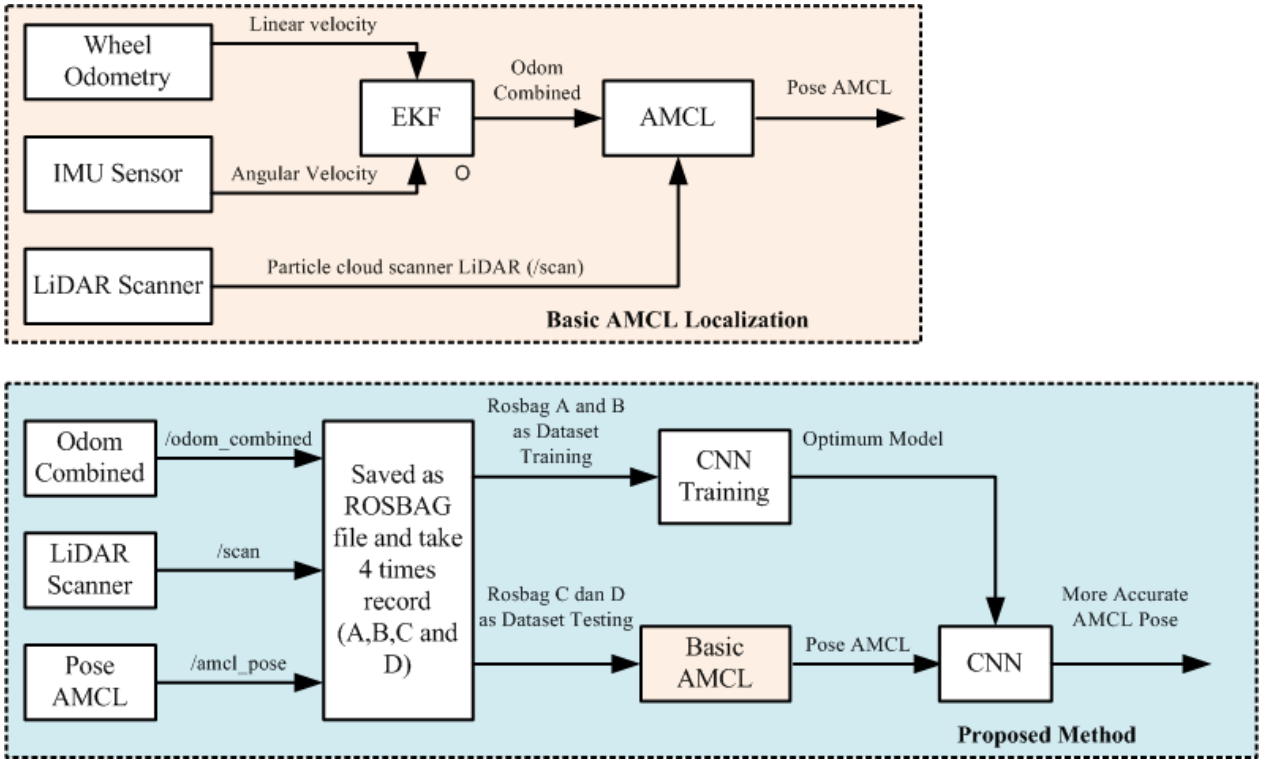

Figure 1. Block diagram of the proposed robot localization.

both algorithms are still affected by error potentials during the motion update process. This motion update is derived from reading changes in the robots' position based on sensor data of wheel odometry. The most common error that occurred is induced by wheel slips and deformations.

Technique on robot localization using 2D LiDAR has been conducted [23]. This technique employed LiDAR-based localization with an occupancy grid map combined with a camera. This camera functioned to read texts placed around the robots' working area. In the indoor localization, the results were pretty good. Another technique used in robot localization was 2DLaserNet [24]. This technique employed a convolutional neural network (CNN) to identify rooms, corridors, and doors in the robot's working area. Identification was carried out based on the reading of 2D LiDAR. A 2D LiDAR with neuron networks has also been used [25].

The accuracy of reading AMCL can also be increased using implicit representation-based Monte Carlo localization (IR-MCL) [26]. This technique performs position estimation based on 2D LiDAR data. The robot's working environment was represented in the form of a neural occupancy field (NOF) created using an artificial neural network (ANN). NOF was generated using a 2D LiDAR scan as input, with the output being the occupancy map's probability of suitability. This technique can generate a more accurate estimate of the robot's position and heading. The robot localization system using machine learning has several advantages compared to the AMCL method. Machine learning systems have the ability to automatically learn and adapt to their environment, enabling them to generate more accurate location estimates in changing environments.

This paper proposes to increase the accuracy of the AMCL localization system using CNN. In the proposed method, CNN functioned to predict possible errors arising in the robot's position and heading. The resulting error prediction value was used to compensate for the AMCL output on the x-axis and y-axis positions and the z-axis (yaw) rotation. If the accuracy of the robot's position increases, the overall performance of the robot improves. The proposed method was run on a robot equipped with a robot operating system (ROS).

The following sections of this paper describe the research methodology and result obtained. Section II covers the research methodology that have been carried out, starting from robot design, CNN design, testing, and evaluation metrics. Part III discusses the results of the tests that have been done and Part IV shows the conclusions based on the test results.

II. METHODOLOGY

In this section, steps to achieve the desired objectives are explained. The first step involved the preparation of the mobile robot that had ROS installed. After that, the ROS node was prepared to support the system to be built. On the basis of the LiDAR scan and wheel odometry inputs, an artificial neuron network with CNN architecture was made to determine the number of optimal particles. Once the robot, ROS framework, and CNN model were ready, datasets for the training process were collected. These datasets were in the form of rosbag files, from which sensor reading data were extracted offline. The training proses was conducted based on these extracted data. After the optimal model was obtained, it was then implemented to the robot localization system that had been built. The optimal value was measured based on the lower error pose values and error heading than the regular MCL and default AMCL in ROS. The process of the proposed method is exhibited in Figure 1. The addition of an inertial measurement unit (IMU) sensor and wheel odometry to the LiDAR sensor used distinguishes the proposed method from several prior methods [24], [25].

A. HARDWARE DESIGN OF THE MOBILE ROBOT

The mobile robot used was built with a differential drive mobile robot configuration. It was made with a length of 300 mm, a width of 200 mm, and a height of 100 mm. The wheels had a diameter of 65 mm and were attached at the back. The

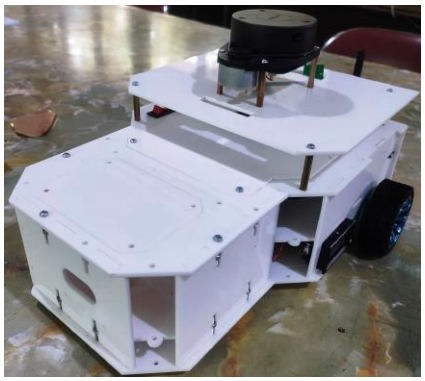

Figure 2. Visual form of the robot used.

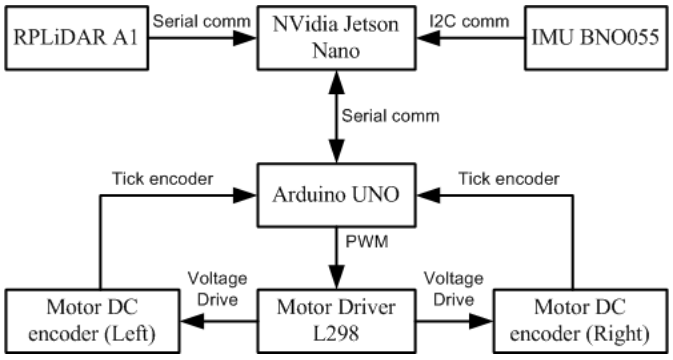

Figure 3. Block diagram of the connection between the components.

distance between the wheels was 250 mm. Castor wheels were attached at the front with a distance of 170 mm to the rear axle. A 2D LiDAR sensor was installed above the castor to support the localization system. The robot used is shown in Figure 2.

The robot made had several supporting electronic devices whose connections between components are shown in Figure 3. NVIDIA Jetson Nano was used as the central processor in which ROS was installed. The general-purpose input-output (GPIO) pin on the Jetson Nano was connected to an IMU BNO055 sensor. The two devices were connected by inter-integrated circuit (I2C) communication. The IMU sensor was connected to the ROS system using the /imu_node node, which generated the /imu_data topics. In order to control actuators, Arduino Uno was connected to NVIDIA Jetson Nano using universal serial bus (USB) communication. In ROS, the communication was done using the rosserial_arduino (/serial_node) node. Arduino was responsible for regulating the speed of the DC motor in accordance with the Jetson Nano via topic /cmd_vel commands. The pulse width modulation (PWM) signal from the Arduino was connected to the L298 motor driver so that the motor speed could vary. When the motor rotated, the rotary encoder sensor sent pulses to the Arduino pins, which were then processed into speed data (/left_speed and /right_speed) and the number of wheel rotations (/left_ticks and /right_tick topics).

B. ROS FRAMEWORK DESIGN

ROS is a meta-operating system run on the robot made. It has several interconnected nodes to support the robot's performance. Using topics, each node can be connected to other nodes. Topics comprise messages containing data on sensor reading, the robot's position, and so on. This research used several default nodes or default ROS, namely /rpLiDAR_node, /AMCL, /movebase, and /SLAM_gmapping. In addition to default nodes, other nodes to support the robot's performance were also created, including /ekf_odom_pub, which functioned

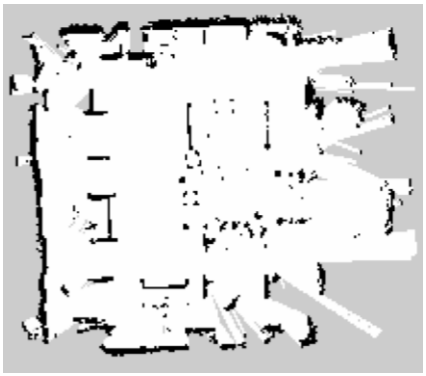

Figure 4. Results of the occupancy grid map from SLAM gmapping.

to run the extended Kalman filter (EKF). Then, /datalogger node functioned to extract sensor data previously stored in the rosbag.

The /rpLiDAR_node node is one of the essential nodes in the ROS-based robot navigation system. RPLIDAR is a 2D LiDAR sensor that uses laser beams to measure the distance and angle of surrounding objects. In ROS, the /rpLiDAR_node node functions to read data from the RPLIDAR sensor and generates /scan topics later used by other nodes in ROS. The /scan topics generated by the RPLIDAR node contain each point's angle and distance data around the LiDAR sensor. These data are accessible by reading /scan topics on the child msg.ranges[0-719], of which a value of 0-719 represents a 0.5° increase in each angle. These data are then represented in the form of arrays so that they can be utilized for environmental mapping or robot navigation autonomously. Moreover, data from /scan topics can be used to prevent the robot from colliding with nearby obstacles or objects. Users can utilize the ROS package provided by the RPLIDAR producer to run the /rpLiDAR_node node. Once it operates, the /rpLiDAR_node node will perpetually read data from the sensor and periodically generate /scan topics as per predetermined frequencies. It enables the robot to continuously update information on its surroundings and make precise decisions when navigating.

The subsequent node is the /ekf_odom_pub node, which operated EKF on ROS. EKF can be used to fuse data from the rotary encoder and IMU sensors. The combination of the two sensors aims to improve the odometry system of the robot. The rotary encoder on the wheel was used to measure the distance traveled by the robot linearly. In contrast, the IMU was used to measure the acceleration and angular velocity of the robot. The data from these two sensors were combined using the EKF filter to estimate the position and heading of the robot more accurately. In this node, several parameters needed to be set, including the diameter of the wheels used, the distance between the wheels or the wheelbase, and the number of encoder pulses when the robot moved one meter.

The SLAM gmapping algorithm was used to obtain the occupancy grid map of the room. The SLAM gmapping process was initiated by subscribing data from the LiDAR sensor and robot position (from /ekf_odom_pub topics). Then, the sensor data were converted into a gridmap by placing each particle scan of the LiDAR sensor on a grid that corresponded to the robot's current position. The grid was then updated to reflect each newly received scan. Subsequently, the grid was matched with the results of the previous scan. It was used to determine the most probable transformation between the two scans when updating the robot's position. Furthermore, the map grid data were converted into an occupancy grid which stated the

p-ISSN 2301–4156 | e-ISSN 2460–5719

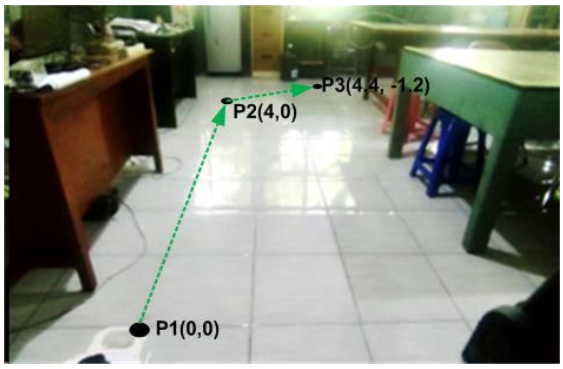

Figure 5. Rosbag robot workspace during rosbag collection.

probability of whether the room in the grid was occupied. After the map was created, gmapping published the resulting map via the appropriate ROS topics so that it could be used by other nodes in the system. The occupancy grid map results are shown in Figure 4.

The /AMCL node operated by acquiring data from the LiDAR sensor and the occupancy grid map, then generating the robot's poses (position and heading) within the environment. The /AMCL node required data from the LiDAR sensor and the occupancy grid map to generate robot poses. Data from the LiDAR sensor were used to detect objects and walls around the robot, while the occupancy grid map was used as a reference to determine the robot's position. Following the collection of the LiDAR sensor and the occupancy grid map, the /AMCL node processed the data to produce the robot poses. The poses were generated in the form /AMCL_pose, which could be used to navigate the robot autonomously.

C. ROSBAG DATASET COLLECTION

After the entire nodes were connected, the process proceeded by retrieving the dataset in the form of a rosbag file. Rosbag is a built-in tool in ROS (default tools) used to record topic data in a file. It was first necessary to determine the topics to be recorded and how long the recording would take to record data using a rosbag. Then, the recording was started by running the "rosbag record" command and providing a file name to store the recording data. Once the recording was complete, the rosbag file could be used to play back the recording using the "rosbag play" command.

The dataset was collected using scenario by running the robot from the start position or point P1 to point P2. Next, the robot rotated 90° counterclockwise in place. Then, the robot was moved to point P3 and rotated 90° counterclockwise. The position of these points in the test area is shown in Figure 5. In this scenario, the robot was moved in straight and turn positions, each of which was carried out twice. Therefore, from this scenario, the robot data would be obtained when the robot moved straight and turned. These data served as a reference for the accuracy of the proposed algorithm. Each scenario was run four times and stored in a different rosbag, resulting in four rosbags that were extracted into training and testing datasets.

D. ROSBAG EXTRACTION FOR TRAINING DATASETS

Collecting training datasets is a crucial step in creating a quality machine-learning model. This CNN machine learning used training and testing datasets in text form of CSV format. Meanwhile, from ROS, datasets were stored in a rosbag form, hence a rosbag to CSV conversion was needed for the training and testing datasets could be used. Based on these needs, a node /datalogger was created. The node converted the rosbag into a

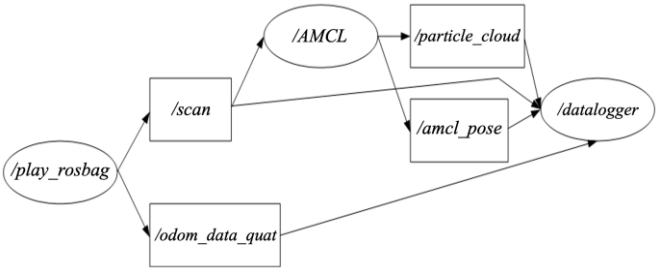

Figure 6. Connection between nodes when extracting a rosbag file.

CSV file, beginning from the start until the robot finished pivoting 90° at point P3. The stored data were in from of /AMCL_pose, /odom_combined, and /scan topics. The connection between nodes when running this process is shown in Figure 6.

The /AMCL_pose topics were served as a reference or comparative data after the correction process with CNN was complete. The /odom_combined topics were used to determine the robot's movement within a particular sampling time. These topics contained robot position and heading data based on the EKF process from the wheel odometry and IMU sensors. By comparing values at the current time (*t*) with the previous time (*t-1*), the robot's displacement value can be determined. In the training dataset created, the data stored were only the robot's displacement with respect to the x-axis, y-axis, and angular displacement with respect to the z-axis or yaw. As a result, in this training dataset, the robot's displacement value required three columns of data. The displacement equations are expressed in (1) to (3), where the OC(*s*,*t*) value is the topic value of /odom_combined on the s-axis. The notation *s* in the equation denotes the linear odom axis when written using lowercase letters, while the capital letters denote the angular axis.

The /scan topics were necessitated to find the difference in the values of the distance readings for each time sample. The difference in values that appears can represent the movement of the robot in the x-axis and y-axis. These difference values were inputted into CNN input because they were expected to correct the robot displacement values obtained from /odom_combined. The total differences in scan values (*Δscan*) in the dataset were 720 data columns. The *Δscan* value can be calculated using (4). The *Δscan* value is the value of the difference between the readings of the LiDAR sensor distance, while *n* is the direction or angle of the LiDAR sensor reading with a value of 0-719. The value of *n* represents an increase for every 0.5°. The val_{scan} value was obtained from reading the /scan topics in the msg.range[*n*] message, while *t* is the current time and *t-1* is the previous sampling time.

$$\Delta odom_{linear}(x) = OC(x,t) - OC(x,t-1) \quad (1)$$

$$\Delta odom_{linear}(y) = OC(y,t) - OC(y,t-1) \quad (2)$$

$$\Delta odom_{angular}(z) = OC(Z,t) - OC(Z,t-1) \quad (3)$$

$$\Delta scan(n) = val_{scan}(n,t) - val_{scan}(n,t-1). \quad (4)$$

The CNN architecture created was supervised-machine learning, so target data were needed. The target used was the difference between the position and the robot's actual heading based on the direct observation results with the /AMCL_pose value obtained from the rosbag file.

p-ISSN 2301–4156 | e-ISSN 2460–5719

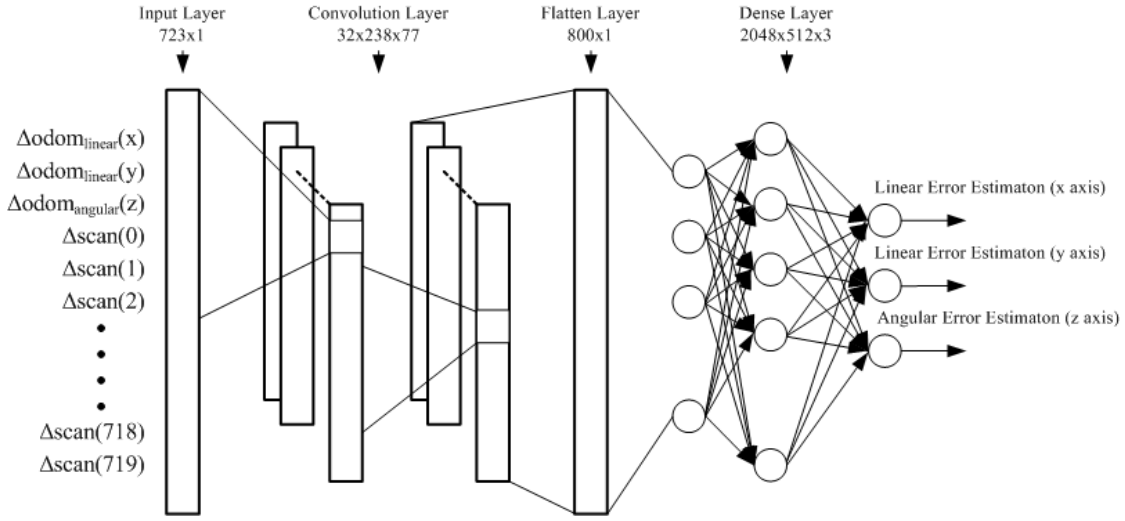

Figure 7. Proposed convolutional neural network architecture.

E. DESIGN OF THE CNN ARCHITECTURE

The proposed design of the CNN architecture was made using TensorFlow, involving several stages. The first stage was layer initialization which functioned to input training data into TensorFlow and created basic layers from CNN. Layer initialization is essential because it ensures that all data and layers used in the training process are available and ready to use. The next stage was to create a convolution layer that performed convolution operations on each part of the input data. The convolution layer makes capturing important features in the input data possible, which are then passed on to the next layer.

After the convolution layer, a pooling layer was created to reduce the size of the feature map, making it easier to pass on to the next layer. The pooling process also helps reduce overfitting and speeds up the training process. Following the pooling process, a flattening layer process must be carried out, namely changing the pooling results into a longer array. The flattening layer process aims to prepare data before being forwarded to the fully connected layer. The number of nodes in this layer was 800 nodes. The final stage was to create a fully connected layer, which was the layer that performed the final classification and predicted the model output. This fully connected layer consisted of two hidden layers with a total of 2,048 and 512 nodes, respectively. Meanwhile, the output layer consisted of three nodes, each generating error estimates on the linear x-axis and y-axis and the angular z-axis. The CNN architecture used is depicted in Figure 7.

F. NETWORK TRAINING

To obtain the optimal model, CNN created was given a dataset prepared to carry out the training process. The dataset was further divided into a training dataset and a validation dataset to measure the performance of the model during training. In the training used, the datasets for training were 75% of all datasets, while the remaining 25% were used for validation datasets. The test datasets, on the other hand, were datasets that were excluded from the training procedure.

After the datasets were ready to be read in the training stage, the model was compiled using the Adam optimizer. Because the distribution of the datasets and targets ranged from -1 to 1, the activation function employed was tanh. The loss function and evaluation matrix used were the mean squared error (MSE). The learning rate used was 0.01, which was the default value of the Adam optimizer. The training process was carried out until the optimal weight was obtained. In this training, the weight is considered optimal if the lowest validation loss/MSE value is obtained.

G. EVALUATION MATRIX

It is necessary to calculate the robot's total position error. The robot's position was represented using the cartesian axes with an assumption that the value of z = 0 since the robot was on a flat surface. The total position error was calculated based on the difference in the robot's target and position toward the x-axis and y-axis. This value was calculated using (5) and (7).

$$P_E(x,y) = \sqrt{(E_X)^2 + (E_Y)^2} \quad (5)$$

$$E_X = Act_X - Res_X \quad (6)$$

$$E_Y = Act_Y - Res_Y \quad (7)$$

where $P_E(x,y)$ is the total position error in meters, E_X is the error on the x-axis, E_Y is the error on the y-axis, Act_X is the actual value of the robot on the x-axis, Act_Y is the actual value of the robot on the y-axis, and the Res_X and Res_Y values are the results of the robot's position from the algorithm being tested according to the axis in question. The algorithms tested include regular MCL, ROS built-in AMCL, and AMCL+CNN which are the proposed algorithms.

Heading error shows the difference between the target heading and the actual heading. This heading was measured on the z-axis or the yaw of the robot. It was done because the robot was assumed to be on a flat surface. The heading error value was obtained using (8).

$$H_E = Act_{YAW} - Res_{YAW} \quad (8)$$

where H_E denotes the heading error on the z-axis or yaw in degree (°), the Act_{YAW} value is the actual yaw value from the observation, and the Res_{YAW} value is the yaw value obtained from applying the algorithm tested.

III. RESULT AND DISCUSSION

This section explains the testing results of the localization system applied to the robot. The testing began by creating an occupancy grid map and running the regular MCL localization algorithm with various particle numbers. The number of particles tested was 1,000 up to 5,000, with an interval of 1,000

p-ISSN 2301–4156 | e-ISSN 2460–5719

TABLE I
POSITION AND HEADING ERROR OF THE REGULAR MCL

Number of Particles	Position Error (m)		Heading Error (°)	
	Straight	Turn	Straight	Turn
1000	0.0277	0.0236	12.49	12.28
2000	0.0558	0.0826	13.98	7.79
3000	0.0243	0.0139	12.60	11.45
4000	0.0249	0.0164	12.62	10.80
5000	0.0287	0.0155	12.68	11.79
Average	0.0323	0.0304	12.87	10.82

particles. It was then followed by the default AMCL algorithm with particle values ranging from 100 to 5,000. The last was testing the AMCL+CNN algorithm, which is the method proposed in this paper. In addition, the test results were also compared with that of earlier study that proposed IR-MCL [26]. Several testing were then compared to determine the average error level that occurred. The lower the error occurs, the more accurate or better the localization system.

A. MAKING OF OCCUPANCY GRID MAP AND ROSBAG

Prior to the localization system testing, the occupancy grid map of the operating area was created. This map was created using the SLAM gmapping algorithm. The SLAM gmapping worked by subscribing the LiDAR sensor data (from scan rpLiDAR topics) and robot poses (from ekf_odom_pub topics). The robot operated slowly to obtain a good map since there was slip potential at high speed. In this testing, the robot was operated at a speed of 0.1 m/s.

During the process of creating the map, the robot was placed in the P1 position at the coordinate of (0,0). Subsequently, the robot was operated manually using the /teleop_twist_keyboard node. It was operated throughout every corner of the room that the LiDAR scan could reach. After completing this step, the map was saved using the /map_saver node. The resulting map is depicted in Figure 5. Once obtained, the map saved with /map_server was used for later testing.

Rosbag recording was done by operating the robot in position P1. Before that, the "rosbag record" command must be executed first. Next, the robot was moved to point P2 and rotated 90° in place. The robot was then moved to the position of point P3, where it continued by spinning 90° in place. Next, the rosbag recording was stopped and continued with the rosbag storage command. Storage of the rosbag was carried out four times, with each test being named "A", "B", "C", and "D."

B. TESTING WITH THE REGULAR MCL

After the occupancy grid map was created, the map, as shown in Figure 5, was used to test the localization system. The test was carried out by running the stored rosbags. Each rosbag was played back and simultaneously ran the regular MCL algorithm. The /AMCL node from the default ROS was configured to use a fixed number of particles and disable particle updates in order to produce regular MCL. The number of particles used was 1,000 to 5,000, with an interval of 1,000 particles, resulting in five variations in the number of particles, as presented in Table I.

In testing with a straight scenario, 3,000 particles produced the smallest position error results of 0.0243 m in the straight scenario and 0.0139 m in the turn scenario. Meanwhile, the largest position error was generated by the number of 2,000 particles, namely 0.0558 m in the straight scenario and 0.0826 m in the turn scenario. However, the number of 2,000 particles produced the smallest heading error in the turn scenario, which was 7.79°. Whereas, in the straight scenario, the smallest heading error occurred when the number of particles was 1,000.

According to the regular MCL testing, the error position average that occurred when the robot moved straight was 0.0323 m. This error value was recorded when the robot moved from P1 to P2 and from P2 to P3. The error heading average that occurred in that scenario was 12.87°. The error position and heading values in the turn scenario were obtained when the robot turned pivotally on P2 and P3 by 90°. The position error when turning was 0.0304 m, and the heading error was 10.82°.

TABLE II
POSITION AND HEADING ERROR OF THE DEFAULT AMCL IN ROS

Rosbag	Position Error (m)		Heading Error (m)	
	Straight	Turn	Straight	Turn
A	0.0262	0.0112	11.50	10.75
B	0.0262	0.0106	11.50	10.41
C	0.0229	0.0381	12.81	15.87
D	0.0434	0.0322	13.18	17.90
Average	0.0297	0.0230	12.25	13.73

C. TESTING WITH THE DEFAULT AMCL IN ROS

During testing with default AMCL in ROS, the number of particles was set to 100 up to 5,000 particles. Data testing was carried out four times on different rosbag files. It was carried out in the same manner as regular MCL testing. However, this testing updated the number of particles in the /AMCL node. Table II presents the results of the default AMCL or default ROS testing. The average position error of the four rosbags was 0.0297 m in the straight scenario and 0.023 m in the turn scenario, while the heading error was 12.25° in the straight scenario and 13.73° in the turn scenario.

Based on these data, AMCL provides a better robot position value when compared to MCL. It is shown in Table II, as evidenced by the smaller average position error values. However, for heading errors, AMCL actually produced a higher error value when turning, whereas when going straight, the heading error value only dropped below 1°. The two data obtained from MCL and AMCL were corrected using the addition of CNN, and then compared to the data generated by AMCL+CNN.

D. TESTING WITH THE AMCL+CNN

The AMCL+CNN testing was carried out by correcting the AMCL output value to be close to the actual value based on previously trained data. The training process was performed using the model as described in subsections II.E and II.F. The dataset for training contained 200 data with 723 inputs and 3 outputs. The CNN network training was carried out ten times with a maximum variation of epochs. The number of epochs tested ranged from 50 to 500 with 50 intervals. Throughout training, the validation loss value served as the reference metric observed. The smaller the validation loss value, the better the training results. The best results in this training were at a value of 200 epochs, resulting in a validation loss of 0.0148.

The most optimal weight of the training was then tested on the testing dataset. This dataset had a total of 114 data with a total of 723 inputs and 3 outputs. The outputs in this test dataset were used as a reference for the robot's position and heading.

p-ISSN 2301–4156 | e-ISSN 2460–5719

TABLE III
POSITION AND HEADING ERROR OF THE AMCL+CNN

Rosbag	Position Error (m)		Heading Error (m)	
	Straight	Turn	Straight	Turn
A	0.0076	0.0059	4.03	4.06
B	0.0076	0.0075	4.03	3.95
C	0.0255	0.0280	11.74	12.17
D	0.0470	0.0328	12.00	14.87
Average	0.0219	0.0186	7.95	8.76

TABLE IV
ERROR COMPARISON FOR EACH ALGORITHM

Algorithm	Average Position Error (m)	Average Heading Error (°)
Regular MCL	0.0314	11.85
AMCL ROS	0.0264	12.99
NMCL [26]	0.1369	2.37
IR-MCL [26]	0.0687	1.19
AMCL+CNN	0.0202	8.35

The output of the AMCL+CNN algorithm represents the actual position and heading of the robot. Based on these two values, the position error and heading error values could be identified.

Based on the comparative data that had been obtained, a comparison was made between the test data. From Table III and Table IV, it can be seen that the position error and heading error values for AMCL+CNN in the straight and turn scenarios are always smaller than the regular MCL and AMCL ROS. The results of the average position error of the four rosbags with the AMCL+CNN algorithm was 0.0219 m in the straight scenario and 0.0186 m in the turn scenario. In contrast, for the heading error, the straight scenario obtained a value of 7.95° and 8.76° for the turning scenario gets value.

Additionally, the results of conducted research [26] were used. In this testing, the estimated position of the robot was tested using the Monte Carlo localization + exploiting text spotting (NMCL) and implicit representation-based MCL (IR-MCL). Compared with these methods, the proposed method, AMCL+CNN, yielded the smallest position error, which was 0.0202 m (the average of straight and turning scenarios). The NMCL algorithm in testing produced a position error value of 0.1369 m, while the IR-MCL produced a position error of 0.0687 [26]. However, the value of the heading error from AMCL+CNN is the biggest heading error, so it still has the potential for improvement.

IV. CONCLUSION

The improvement of the AMCL algorithm accuracy using CNN for robot in indoor conditions has been realized. By using AMCL+CNN instead of the default AMCL or regular MCL, the error levels were successfully reduced. In the going straight scenario, the resulting positional error downed to 0.022 m compared to the default AMCL or regular MCL with 0.0297 m and 0.0323 m, respectively. At the same time, the heading error when going straight dropped to 7.95° from the default AMCL value of 12.25° and the regular MCL value of 12.87°. In the turning scenario, the resulting position error dropped to 0.0186 m compared to the default AMCL and regular MCL, which were 0.023 m and 0.0304 m, respectively. The heading error in the going straight scenario decreased to 8.76° compared to the default AMCL value of 13.73° and regular MCL value of 10.58°. With a lower error value, the robot's performance when running the navigation system or other work will be more optimal. The proposed method has a better positional error than the AMCL, NMCL, and IR-MCL methods.

CONFLICT OF INTEREST

The authors declare that there is no conflict of interest with any parties during the research and writing of the article entitled "Improving the Adaptive Monte Carlo Localization Accuracy Using a Convolutional Neural Network."

AUTHOR CONTRIBUTION

Robot making, Riza Agung Firmansyah; testing, Riza Agung Firmansyah; analysis, Riza Agung Firmansyah; advanced analysis, Tri Arief Sardjono and Ronny Mardiyanto; article writing, Riza Agung Firmansyah, Tri Arief Sardjono, and Ronny Mardiyanto.

REFERENCES

[1] F. Rovira-Más, V. Saiz-Rubio, and A. Cuenca-Cuenca, "Augmented Perception for Agricultural Robots Navigation," *IEEE Sens. J.*, Vol. 21, No. 10, pp. 11712–11727, May 2021, doi: 10.1109/JSEN.2020.3016081.

[2] S. Karmore *et al.*, "IoT-Based Humanoid Software for Identification and Diagnosis of Covid-19 Suspects," *IEEE Sens. J.*, Vol. 22, No. 18, pp. 17490–17496, Sep. 2022, doi: 10.1109/JSEN.2020.3030905.

[3] P.-Y. Yang, T.-H. Chang, Y.-H. Chang, and B.-F. Wu, "Intelligent Mobile Robot Controller Design for Hotel Room Service with Deep Learning Arm-Based Elevator Manipulator," *2018 Int. Conf. Syst. Sci., Eng. (ICSSE)*, 2018, pp. 1–6, doi: 10.1109/ICSSE.2018.8520030.

[4] B.P.E.A. Vasquez, R. Gonzalez, F. Matia, and P. De La Puente, "Sensor Fusion for Tour-Guide Robot Localization," *IEEE Access*, Vol. 6, pp. 78947–78964, Dec. 2018, doi: 10.1109/ACCESS.2018.2885648.

[5] Y. Peng *et al.*, "Research Progress of Urban Dual-Arm Humanoid Grape Harvesting Robot," *2021 IEEE 11th Annu. Int. Conf. CYBER Technol. Automat. Control, Intell. Syst. (CYBER)*, 2021, pp. 879–885, doi: 10.1109/CYBER53097.2021.9588266.

[6] D. Shi, H. Mi, E.G. Collins, and J. Wu, "An Indoor Low-Cost and High-Accuracy Localization Approach for AGVs," *IEEE Access*, Vol. 8, pp. 50085–50090, Mar. 2020, doi: 10.1109/ACCESS.2020.2980364.

[7] R.A. Firmansyah, Y.A. Prabowo, and T. Suheta, "Thermal Imaging-Based Body Temperature and Respiratory Frequency Measurement System for Security Robot," *Przegląd Elektrotechniczny*, Vol. 98, No. 6, pp. 126-130, 2022, doi: 10.15199/48.2022.06.23.

[8] R.M. Ñope-Giraldo *et al.*, "Mechatronic Systems Design of ROHNI-1: Hybrid Cyber-Human Medical Robot for COVID-19 Health Surveillance at Wholesale-Supermarket Entrances," *2021 Glob. Med. Eng. Phys. Exch./Pan Amer. Health Care Exch. (GMEPE/PAHCE)*, 2021, pp. 1–7, doi: 10.1109/GMEPE/PAHCE50215.2021.9434874.

[9] A. Carlucci, M. Morisco, and F. Dell'Olio, "Human Vital Sign Detection by a Microcontroller-Based Device Integrated into a Social Humanoid Robot," *2022 IEEE Int. Symp. Med. Meas., Appl. (MeMeA)*, 2022, pp. 1–6, doi: 10.1109/MeMeA54994.2022.9856407.

[10] L. Garrote, T. Barros, R. Pereira, and U.J. Nunes, "Absolute Indoor Positioning-aided Laser-based Particle Filter Localization with a Refinement Stage," *IECON 2019 - 45th Annu. Conf. IEEE Ind. Electron. Soc.*, 2019, pp. 597–603, doi: 10.1109/IECON.2019.8927475.

[11] M.-A. Chung and C.-W. Lin, "An Improved Localization of Mobile Robotic System Based on AMCL Algorithm," *IEEE Sens. J.*, Vol. 22, No. 1, pp. 900–908, Jan. 2022, doi: 10.1109/JSEN.2021.3126605.

[12] S.J. Dignadice *et al.*, "Application of Simultaneous Localization and Mapping in the Development of an Autonomous Robot," *2022 8th Int. Conf. Control Automat., Robot. (ICCAR)*, 2022, pp. 77–80, doi: 10.1109/ICCAR55106.2022.9782658.

[13] A. Ehambram, L. Jaulin, and B. Wagner, "Hybrid Interval-Probabilistic Localization in Building Maps," *IEEE Robot., Autom. Lett.*, Vol. 7, No. 3, pp. 7059–7066, Jul. 2022, doi: 10.1109/LRA.2022.3181371.

[14] K. Żywanowski, A. Banaszczyk, M.R. Nowicki, and J. Komorowski, "MinkLoc3D-SI: 3D LiDAR Place Recognition with Sparse Convolutions, Spherical Coordinates, and Intensity," *IEEE Robot., Autom. Lett.*, Vol. 7, No. 2, pp. 1079–1086, Apr. 2022, doi: 10.1109/LRA.2021.3136863.

p-ISSN 2301–4156 | e-ISSN 2460–5719

[15] W. Shen, Y. Jia, J. Zhu, and X. Qian, "A Fast Monocular Visual–Inertial Odometry Using Point and Line Features," *2022 7th Int. Conf. Signal, Image Process. (ICSIP)*, 2022, pp. 591–595, doi: 10.1109/ICSIP55141.2022.9886829.

[16] J. Li and A. Hamdulla, "A Research of Visual-Inertial Simultaneous Localization and Mapping," *2022 3rd Int. Conf. Pattern Recognit., Mach. Learn. (PRML)*, 2022, pp. 143–150, doi: 10.1109/PRML56267.2022.9882205.

[17] J. Yuan, S. Zhu, K. Tang, and Q. Sun, "ORB-TEDM: An RGB-D SLAM Approach Fusing ORB Triangulation Estimates and Depth Measurements," *IEEE Trans. Instrum., Meas.*, Vol. 71, pp. 1–15, 2022, doi: 10.1109/TIM.2022.3154800.

[18] J. Liu, X. Li, Y. Liu, and H. Chen, "RGB-D Inertial Odometry for a Resource-Restricted Robot in Dynamic Environments," *IEEE Robot., Automat. Lett.*, Vol. 7, No. 4, pp. 9573–9580, Oct. 2022, doi: 10.1109/LRA.2022.3191193.

[19] R. Long *et al.*, "RGB-D SLAM in Indoor Planar Environments with Multiple Large Dynamic Objects," *IEEE Robot., Automat. Lett.*, Vol. 7, No. 3, pp. 8209–8216, Jul. 2022, doi: 10.1109/LRA.2022.3186091.

[20] Y. Chen *et al.*, "Submap-Based Indoor Navigation System for the Fetch Robot," *IEEE Access*, Vol. 8, pp. 81479–81491, Apr. 2020, doi: 10.1109/ACCESS.2020.2991465.

[21] K. Tian and K. Mirza, "Sensor Fusion for Octagon – an Indoor and Outdoor Autonomous Mobile Robot," *2022 IEEE Int. Syst. Conf. (SysCon)*, 2022, pp. 1–5, doi: 10.1109/SysCon53536.2022.9773827.

[22] C. Li, S. Wang, Y. Zhuang, and F. Yan, "Deep Sensor Fusion Between 2D Laser Scanner and IMU for Mobile Robot Localization," *IEEE Sens. J.*, Vol. 21, No. 6, pp. 8501–8509, Mar. 2021, doi: 10.1109/JSEN.2019.2910826.

[23] N. Zimmerman *et al.*, "Robust Onboard Localization in Changing Environments Exploiting Text Spotting," *2022 IEEE/RSJ Int. Conf. Intell. Robots, Syst. (IROS)*, 2022, pp. 917–924, doi: 10.1109/IROS47612.2022.9981049.

[24] B. Kaleci, K. Turgut, and H. Dutagaci, "2DLaserNet: A Deep Learning Architecture on 2D Laser Scans for Semantic Classification of Mobile Robot Locations," *Eng. Sci., Technol. Int. J.*, Vol. 28, pp. 1-13, Apr. 2022, doi: 10.1016/j.jestch.2021.06.007.

[25] G. Spampinato, A. Bruna, I. Guarneri, and D. Giacalone, "Deep Learning Localization with 2D Range Scanner," *2021 7th Int. Conf. Automat. Robot., Appl. (ICARA)*, 2021, pp. 206–210, doi: 10.1109/ICARA51699.2021.9376424.

[26] H. Kuang *et al.*, "IR-MCL: Implicit Representation-Based Online Global Localization," *IEEE Robot., Automat. Lett.*, Vol. 8, No. 3, pp. 1627–1634, Mar. 2023, doi: 10.1109/LRA.2023.3239318.