Intelligent Mobile Robot for Workspace Mapping

Vladimir Mitić, Miloš Petković, Vladimir Sibinović, Darko Todorović, Goran S. Đorđević, *Faculty of Electronic Engineering, University of Niš*

***Abstract*—A mobile robot that determines the shape and size of an unknown workspace is presented in this paper. The robot is wheeled, and equipped with a controller and cliff sensors. The main goal of this paper is to demonstrate the potential use of neural networks for workspace mapping. A workspace can be a table or any other surface with cliff boundaries, assuming that the robot can reach all of its boundaries. The detection of the workspace cliff boundaries is accomplished using infrared sensors that are pointed downwards. The body of the robot is designed and manufactured using a 3D printer. Robot uses differential drive to move, and odometry for position sensing. The motion control algorithm is explained in detail. Controller is based around Raspberry Pi 3. The description of the neural network that determines the shape and size of the workspace is presented, along with test results. The robot is tested for determining the size and shape of a rectangular workspace with the dimensions of 70 cm x 50 cm.**

***Index Terms*—Workspace mapping, differential drive, odometry, artificial neural networks.**

I. Introduction

A mobile robot can be defined as a mobile and manipulative physical system that autonomously moves in unstructured space, while interacting with human beings or autonomously executing a task instead of them [1]. A machine can be considered mobile if it meets the set requirements:

- Mobility: The robot must be able to move freely inside the workspace.
- Autonomy: Human interaction with the robot is minimal and limited to its task.
- Perception: To be able to successfully work in the environment, the robot must possess sensors which would detect the changes happening inside of it.

Taking into consideration above mentioned requirements, a mobile robot that maps an unknown workspace has been designed. The robot control application is located directly inside the robot, on a Raspberry Pi 3. This allows it to autonomously map the workspace without any human interaction needed. The obtained map can then be transferred from the robot to a more powerful computer for further analysis. It can also be used to ensure that, once the robot determines the map, it never goes near the boundary of the workspace. A workspace can be a table or any other surface with cliff boundaries, assuming that the robot can reach all of its boundaries.

The drive of the robot was chosen to be differential drive, because of the possible maneuverability that it provides with simple motor control [2]. The drive of the robot consists of two motored wheels that are located on the left and right side, and two ball caster wheels located on the front and back. The negative side of this configuration is that it is not suitable for uneven floors, because unwanted direction shifts can occur.

The localization of the robot is implemented using odometry on the two motored wheels. For that purpose, motors with integrated incremental encoders where chosen. The position is calculated in respect to the robot's initial position upon starting the application. Odometry calculation is done by calculating the traveled distance of each wheel for a certain period, and then using that distance to determine the length and radius of the arc which robot describes on the map. If the calculated radius is infinite, the robot is moving in a straight path. Odometry is convenient because it doesn't require any changes done in the workspace. The problem when using odometry is with the constantly increasing error that accumulates because of the slippage of the wheels and the fact that there is no reference point present, except the starting point. Therefore, odometry is mostly used as a correction factor to an absolute positioning system to improve accuracy [3].

The cliff boundary detection is accomplished using four infrared sensors that are pointing downwards and are located on the corners of the robot. The sensors used have digital outputs that represent if the sensor is detecting the surface or not. The activation threshold for each of the sensors is defined manually via four trimmer resistors located on the PCB.

Summing up all defined characteristics, we can see that the task is to determine the workspace map, using unreliable localization system, and with limited information of the workspace boundaries. Having this in mind we wanted to see if it is possible to obtain enough useful information using these methods, with limited computing power. To achieve this, motion control algorithms needed to be carefully designed, because only with good path planning it is possible to overcome some of the shortcomings of the system. The data obtained from the sensors is then passed to a neural network that determines the workspace map. The neural network, through training, tries to filter the data and extract useful information from it.

Vladimir Mitić is with the University of Niš, Faculty of Electronic Engineering, Serbia (e-mail: vladimir.mitic@elfak.ni.ac.rs).

Miloš Petković is with the University of Niš, Faculty of Electronic Engineering, Serbia (e-mail: milos.petkovic@elfak.ni.ac.rs).

Vladimir Sibinović is with the University of Niš, Faculty of Electronic Engineering, Serbia (e-mail: vladimir.sibinovic@elfak.ni.ac.rs).

Darko Todorović is with the University of Niš, Faculty of Electronic Engineering, Serbia (e-mail: darko.todorovic@elfak.ni.ac.rs).

Professor Goran S. Đorđević is with the University of Niš, Faculty of Electronic Engineering, Serbia (e-mail: goran.s.djordjevic@elfak.ni.ac.rs).

II. Robot construction and control application

The design process of the robot included 3D modeling of the robot body, as well as control algorithms design. There were a couple of major issues that had to be overcome:

- Component placement and, most importantly, sensor placement;
- Robot movement algorithms that ensure enough relevant data is obtained from the workspace;
- Encoder reading, and localization calculation;
- Neural network architecture definition;
- Synchronization of all the subsystems.

The dimensions of the robot are limited by the dimensions of the workspace that needs to be determined. Because the sensors are pointing downwards the simplest workspace setup presented a rectangular table with the dimensions of 70 cm x 50 cm. This means that the robot dimensions shouldn't be over 20 cm. The components that are needed for robot control can be divided into four groups: Single-Board Computer (SBC) for high level control, PCB for motor control, PCB for sensor reading and PCB for encoder reading. Taking into consideration the component size as well as the robot dimensions, the components needed to be placed in three levels.

A. Robot body

The robot body was designed in a 3D modeling software and made on a 3D printer. The robot is symmetrical in order to simplify the motion control algorithms and ensure that the workspace boundaries would be detected in the same manner when moving in either direction. The robot body consists of a rectangular plate with rounded short sides. The longer sides are extended with straight segments that stretch beyond the body surface so that the sensors could be placed far enough from the wheels and the robot has room to stop without crossing the boundaries. On the middle of the longer sides the motored wheels are attached, and on the middle of the shorter sides the caster wheels are attached. The battery is placed beneath the body plate and is attached to it with zip ties.

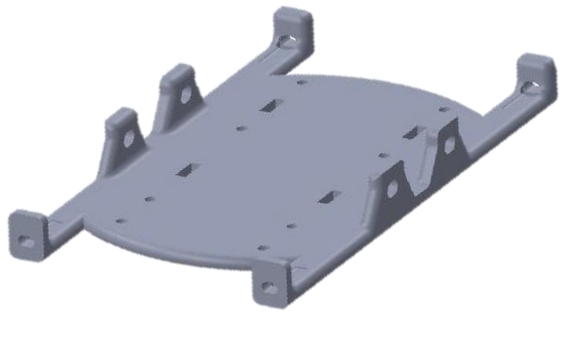

Fig. 1. 3D body plate design. The robot length is 20 cm, and the width is 16 cm.

The mounting holes for the three-layer component placement are also envisioned into the design. The four infrared sensors are attached to the robot body on the extended sides using an *L* shaped 3D printed holder. The holder provides the ability to adjust the distance between the sensors and the surface, as well as the width of the sensors coverage area.

B. Component placement

In order to retain the small dimensions of the robot, because every peripheral has its own PCB, the component placement had to be done in three levels (Fig. 2.). The bottom layer contains the PCB for the encoder reading. It was not possible to connect the encoders directly to the Raspberry Pi 3, so a PCB with a microcontroller that reads the encoders and forwards that information via UART was designed [4]. The second layer contains only the Raspberry Pi 3 that controls all the peripherals. Raspberry Pi 3 uses 3.3 V logic levels on all the pins and they are not fully TTL compatible. Therefore, logic level shifters were connected to all pins. The top layer contains the motor driver and the PCB for sensor reading. The motor driver is a standard H-bridge driver *L298N*. Because all the peripherals have their respective PCBs which operate on different logic levels, the wiring of the robot presented a significant challenge.

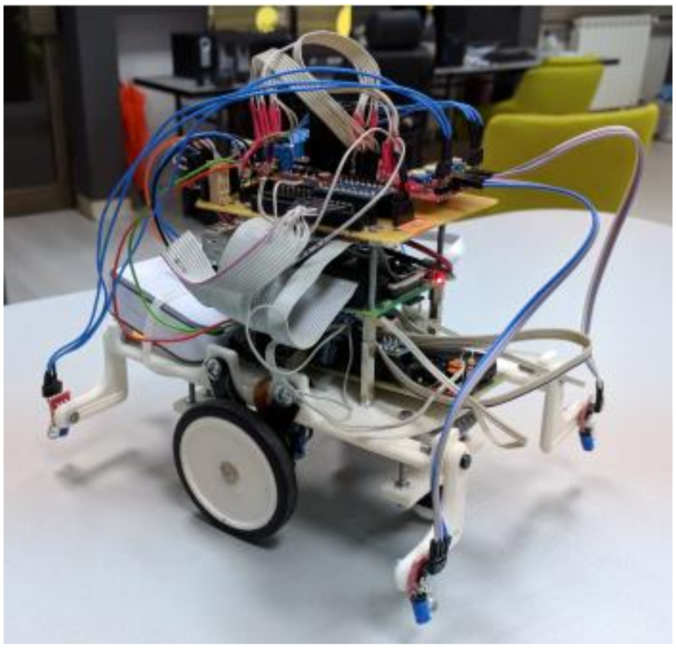

Fig 2. Assembled robot with all the necessary components and wiring.

C. Robot control application

Robot control application is developed in C++ programming language. The application consists of four parts that are separated as four classes. The first class is named *Drive* and it realizes the motion control algorithms, keeping the robot inside the boundaries of the workspace. The second class is named *Mailer* and it handles the communication with the microcontroller for encoder reading. The traveled distance for each wheel is also calculated here. The third class, named *Positioner*, calculates the robot coordinates in respect to its initial position. The last class is the main class, and it handles the data synchronization between all three classes. It also generates data for the neural network and determines when the data set is large enough so the robot can stop moving. When the data set is obtained, it is passed to a neural network, created using *Fast Artificial Neural Network Library* (FANN). After training the network using this data set, a test data set is run through the network and the results are exported to a file. The training data set, as well as the neural network parameters are also exported to files for later analysis and processing.

D. Motion control algorithms

Designing, and later fine tuning, the motion control algorithms presented a challenging task because the algorithm needed to take into consideration all the imperfections of the 3D printing and mounting, as well as transport delays that are present in the system. The motion control can be divided into two parts: speed profile and direction determination, and speed ramp realization.

The motor is controlled with a PWM signal, where the duty cycle of the signal is directly proportional to its speed. When changing the speed, a speed profile is defined to avoid wheel slippage. A trapezoid speed profile with programmable acceleration was chosen for the motor control. This minimized the wheel slippage which is a crucial point because the position relies solely on odometry. When accelerating, the angle of the ramp is allowed be lower because slower acceleration doesn’t present an issue, and this ensures that the wheel doesn’t slip. When decelerating, the angle of the ramp must be higher to ensure timely stopping without crossing the boundary. Breaking wasn’t implemented on the motor because that would increase the chance of wheel slippage. The angle of the ramp can be dynamically changed for different stopping scenarios.

Speed profile and direction determination was designed in that way so the robot moves near the edge of the workspace because the most relevant data for this system is located here. The algorithm is designed in a way that the boundary of the workspace stays on the left side of the robot. Also, the algorithm has a preferred direction of movement which is, in this case, going forwards. What this means is that the robot will go to the boundary, then stop, back up a little turning to the left, and start going forwards again. This kind of movement ensures that the robot can’t enter an endless loop of forwards-backwards motion on a single path. It also minimizes the time needed to collect enough data for the workspace mapping because the robot is going around the workspace, near the edge, in a clockwise direction.

When the application starts, the robot goes forward in a straight path until it reaches the boundary of the workspace. When a sensor detects the cliff, robot stops and waits for the conformation to continue moving. Confirmation signal comes from the synchronization thread that ensures that the coordinates of the workspace boundary are stored when the robot is still. After the coordinates have been stored, the algorithm choses the desired speed and direction for every motor based on the sensor readings. Because the preferred direction of movement is forwards, if the sensors readings suggest that the robot needs to go backwards, it would do so for a maximum of 550 ms. After that, the robot stops and starts moving straight forward again. If, in the meanwhile, any of the rear sensors is activated, the robot will stop and choose a desired forward curved path, but it will keep this setting for a maximum of 1 s. After 1 s the algorithm changes the path to straight again, to avoid entering an endless loop. Also, speed is decreased after 1 s of movement because a greater speed is needed to overcome static friction, but afterwards a slower speed will ensure safer stopping. After an enough amount of data is collected, the main thread initiates a stop command, which breaks the loop and exists the drive mode.

To achieve the movement that was described in the previous paragraph, the robot needed to be able to perform different maneuvers. This set of maneuvers included straight paths, different radius curved paths as well as rotating in place. Rotation in place is especially important for getting out of corners in the workspace. To achieve this, 5 possibilities for the desired speed were defined in respect to the highest motor speed: 100 %, 90 %, 88 %, 84 %, and 80 %. The speed profile and motor direction selection is influenced by a couple of factors. Current speed, direction, and travelling time are some of them, but the main factor that determines the desired speed profile is sensor reading i.e. if the boundary is detected or not. We used only 4 infrared sensors with digital outputs, where logical 1 represents boundary detection. This means that there are 16 possible combinations for the sensor readings which reflect on 16 states. Only 9 of 16 states are considered valid because these are the states that the robot can autonomously resolve. Figure 3. shows the enumeration of the sensors for better understanding of the defined speed profiles.

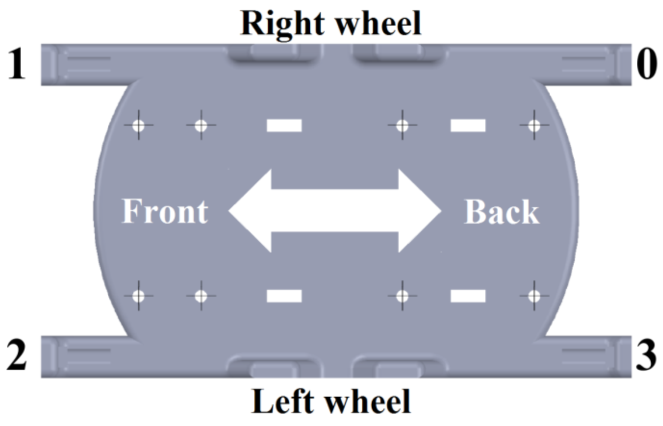

Fig. 3. Sensor enumeration and direction of movement definition.

Every state in which the sensors placed diagonally are activated indicates that the robot is stuck and can’t move on its own. These states represent the following sensor combinations: 0-2, 0-1-2, 1-3, 0-1-3, 0-2-3, 1-2-3, 0-1-2-3.

When only the sensor 0 is activated, the left motor speed is set to 84%, the right motor speed is set to 100%, and the direction of the movement is forwards. This is the case when the robot has detected the boundary on its right back side, while backing up. Therefore, it needs to make a hard-left turn so that the boundary is located to its left side.

When only the sensor 3 is activated, the left and the right motor speed is set to 88%, and the direction of the movement is forwards. This is the case when the robot has detected the boundary on its left back side, while backing up. Because backing up is done in a curved path, the robot goes forwards which ensures that it will move on along the edge.

Activation of both rear sensors (0-3) indicate that the robot is facing directly away from the workspace boundary, and it mostly happens when the robot is turning in place to get out of a tight corner. Therefore, the left motor speed is set to 80%, the right motor speed is set to 100%, and the desired direction of the movement is forwards. This way, the robot turns to the boundary with its front side.

If the sensor 1, or the combination 1-2 is activated, the left motor speed is set to 80%, the right motor speed is set to 100%, and the direction of the movement is backwards. This combination indicates that the robot approached the boundary with the front side, but not at the desired angle, so a smaller radius left turn is chosen.

If only sensor 2 is activated, the left motor speed is set to 84%, the right motor speed is set to 100%, and the direction of the movement is backwards. This indicates that the robot approached the boundary in a desired manner, therefore, a larger radius left turn is chosen.

If the activated sensor combination is 2-3, the robot will rotate in place. This sensor combination occurs only then the robot is in the corner of the workspace, which is located on its left side. In order to get out of this situation, the left motor speed is set to 80%, the right motor speed is set to 90%, the left motor direction is forwards, and the right motor direction is backwards. This movement stops only when sensor 0, the combination 0-3, or any of the invalid combinations are activated. In the first two cases the robot will go forwards as described in the previous paragraph.

If the activated sensor combination is 0-1, we have a similar case like in the last paragraph. The only difference is that the corner is now located at the robot's right side. The left motor speed is set to 90%, the right motor speed is set to 80%, the left motor direction is backwards, and the right motor direction is forwards. The stopping logic is the same as in the previous case. This case is rarely encountered, because of the way the robot moves, keeping the boundary to its left side.

After the algorithm determines the desired speed profile, a function that changes the PWM duty cycle according to that profile is called. From the selected speed profiles, it is obvious that every sensor combination had to be thoroughly analyzed for cause and effect, to ensure that the robot moves in a desired manner inside the workplace.

E. Odometry reading and position calculation

The incremental encoder reading is implemented using an integrated circuit HCTL-2032 connected to a microcontroller PIC18F4431. The microcontroller obtains the number of pulses from the HCTL-2032 with the direction of movement, and forwards that information to the Raspberry Pi 3 via UART communication.

Every time, the encoder reading information is refreshed, a new robot position is calculated based on the traveled distance of each wheel. When the application starts, the current robot position and orientation are used for defining the reference coordinate system, and all the coordinates are calculated in respect to that coordinate system. This means that, every time the robot is used to map the same workspace, the workspace boundaries will have different coordinates. The workspace shape and dimensions, on the other hand, stay the same and they can be later used for calculating the absolute robot position inside it.

F. Neural network

The neural network was designed to calculate the probability of a point being inside or outside the workspace. If the point is inside the workspace, the neural network should give an output of 0. If the point is outside the workspace, the neural network should give an output of 1. This basically means that the neural network has two inputs (one for each coordinate) and one output. The coordinates are passed in cm in order to speed up the network. The network represents a conventional three-layered curve fitting neural network. It has 4 neurons in the hidden layer. The activation functions for the neurons in the hidden layers are sigmoid, while the activation function of the neuron in the output layer is linear. Training algorithm used is a standard Backpropagation algorithm. To obtain satisfying training results with a reasonable data set, the algorithm needed up to 1000 training epochs. Many times, the algorithm finished even before the 400. training epoch.

G. Obtaining workspace information data

After we have defined the robot motion control algorithm along with the position calculation, workspace information data needed to be formatted in a way that is suitable for neural network training. Because the neural network has two inputs and one output, one training data sample contained three parameters. For every calculated robot position, the sensor readings were used to define the desired output of the network. If the sensors weren't detecting, the output would be 0, and in the other case the output would be 1. While obtaining the data set there were a couple of important issues that need to be resolved. For the network to provide good results, the data set should be large enough, without any conflicting points. Also, with every new position, odometry error increases and in that manner the robot movement time should be as minimal as possible. Finally, the data set should contain the equal number of points for both inside and outside the workspace. The motion control algorithm solves the last problem buy guiding the robot to move along the edge. The solution to the first two problems we found in adding "fake" points. The added points were "fake" because the robot didn't necessarily pass through them, but we can predict whether they are inside or outside the workspace. If the robot is inside the workspace, and we draw a line between its position and the coordinate center, we can claim that all the points on that line between the robot's position and the coordinate center are inside the workspace. So, we take 5 equidistant points on that line and add them as "fake". In that manner, it the robot is on the workspace boundary, and we draw a line between its position and the coordinate center, we can claim that all the points on that line further from the robot's position are outside the workspace. We also take 5 equidistant points on that line and add them as "fake". This is only possible if the mapped workspace is represented with a convex figure. After, a large enough data set has been obtained, a function that removes duplicate points and resolves conflicted points is executed. Duplicate points appear when robot passes through the same point more than one time. Conflicted points appear on the boundary of the workspace due to positioning error. A point can in one case be interpreted as inside and in the other as outside the workspace. In this case, the function removes the point that is declared as inside in order to enlarge the boundary data points set.

III. Test Results

The robot was tested for mapping a rectangular table with the dimensions of 70cm x 50 cm. To achieve good results, the network required 2000 points inside the workspace and 1200 point outside the workspace. This data set includes the "fake" points with duplicates and conflicted points. After removing the duplicates and conflicted points, the data set was decreased to 1959 points in total. The mean-square error of the training was 0.043 after 1000 training epochs. Another way of calculating the error is in the number of outputs that differ from the desired output by a certain margin. If we set that margin to be 0.49, the number of false outputs is 121, which represents 6.18 % of the training data set. The training of the neural network lasts a couple of seconds depending on the data set. This is why the neural network training must be done after the robot has stopped moving.

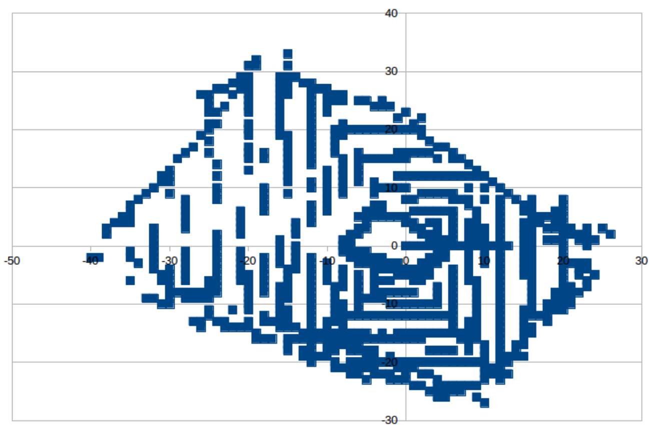

Fig. 4. Training data set for points inside the workspace, including "fake" points.

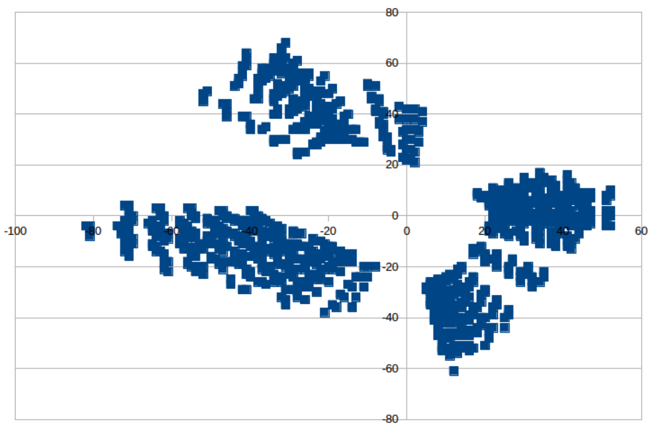

Fig. 5. Training data set for points outside the workspace, including "fake" points.

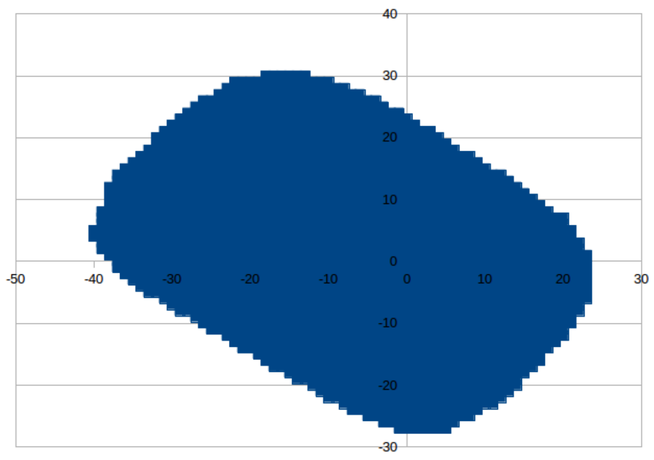

Fig. 6. Neural network output for points in the range of -50 cm to 50 cm. Blue points indicate points inside the workspace.

The results show that the neural network successfully determined the size and shape of the tested workspace. The output of the neural network is rounded before plotting on this figure in order to have a clear separation of the points inside and outside the workspace. The real output suggests the probability of a point being inside the workspace, and with that kind of data, better mapping of the workspace can be done. A workspace map of this kind can now be easily used for a robot motion control algorithm that ensures that the robot never gets near the boundary of the workspace.

IV. Conclusion

Analyzing the results we can come to a conclusion that it is possible to use neural networks for workspace mapping. However, because of the large amount of time that is needed to train the neural network, it seems that it would be more applicable if the robot only obtains the data. That data can then be forwarded to a more powerful computer that would calculate the map and send it to the robot. Because of the limited computing power, the simplest network architecture was chosen for this robot, and this needs to be upgraded to achieve better results for various workspace configurations. For further work we plan to change the architecture of the neural network and the data acquisition algorithm in order to be able to map nonconvex workspaces as well. The localization based on odometry showed that even with a constantly increasing error, useful information can be obtained from it. However, on a larger scale, where the workspace size can be a couple of times larger than this one, this kind of localization would have unacceptable errors. Therefore, for the next development, an absolute localization system should be selected, e.g. a camera based system.

References

[1] J. Defever B. Francis and T. Geerinck, "Mobile Robots with Shared Autonomy", Vrije Universiteit Brussel, Brussels, Belgium, 2004

[2] Dudek G., Jenkin M. „Computational Principles of Mobile Robotics 2nd edition", 2010.

[3] Borenstein J., Everett H. R., Feng L. „Where am I? Sensors and Methods for Mobile Robot Positioning", 1996.

[4] V. Sibinović et al. "Mobile Robot Odometry Using High Resolution Incremental Encoders and Single Board Computers", 2017