Mobile Robot Environment for SLAM

Albin Nátrán
Alba Regia Technical Faculty
Óbuda University
Székesfehérvár, Hungary
natran.albin@stud.uni-obuda.hu

Károly Széll
Alba Regia Technical Faculty
Óbuda University
Székesfehérvár, Hungary
szell.karoly@amk.uniobuda.hu
ORCID: 0000-0001-7499-5643

***Abstract*— The paper introduces a mobile robot system in a virtual environment. The mobile robot can travel autonomously and implement SLAM by using sensor data. The system is prepared for later hardware implementation for benchmarking different SLAM algorithms.**

Keywords—mobile robot, SLAM, virtual

I. Introduction

One of the main areas of research in recent decades has been the navigation of mobile robots. Its main criteria are good and precise localization. The most common localization technology is GNSS (Global Navigation Satellite System). This system ensures the absolute position of the Earth with excellent accuracy. However, the accuracy of a system is greatly influenced by environmental factors such as a cave, city, or tunnel. The resulting error can result in inaccuracies of up to meters, which is not acceptable for autonomous navigation of a mobile robot. Therefore, they need to represent their environment in some form. The starting point for this can be a 2D map. This map can consist of geometric features, but even more complex objects. With this consistent map, the mobile robot will be able to detect free spaces, obstacles and landmarks.

This type of localization approach is called simultaneous localization and mapping (SLAM). SLAM is the process by which a robot or robotic system uses sensors to create a map of its surroundings and at the same time estimate its position. From this information, the mobile robot can plan its own route without human intervention.

SLAM can be created with several complex algorithms that create different maps. However, there is no standard for comparing maps. Therefore, it would be worthwhile to develop a method that allows the comparison of different algorithms. [1]

II. Hardware

The basis of the robot is an iRobot Roomba 605. Roomba is very good at creating a robust platform with useful accessories. The systems they provide may operate for years. It is outstandingly good in terms of parts support. The Roomba 605 is an omni robot. It has a very good weight distribution. Its wheels are driven by two separate motors and can be turned by a spherical wheel. It has a built-in IR and bumper sensors. Can be integrated into ROS. Numerous research and hobby developments have already has been done with this model. There is also a model called Create which is designed specifically for robotic developments. [4]

Roomba alone is not able to meet the defined criteria. Therefore, a Lidar-based sensor must be added. In this project the RPLIDAR A1M8 had been chosen.

There is only one essential component left which is the controller The Raspberry Pi 3 B + is the most appropriate. It supports Ubuntu and ROS. It has built-in Wi-Fi which can be used as communication protocol (see Fig. 1.). [2]

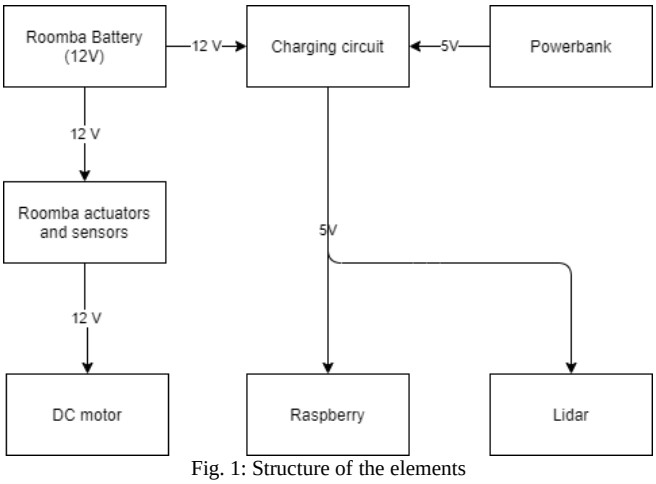

Fig. 1: Structure of the elements

III. Simulation

Due to the high computational demands of the simulation, a personal computer was chosen as the basic hardware. ROS partially supports Debian, Gentoo, Mac OS X, Arch Linux, Android, Windows, and Open Embedded. However, ROS's primary operating system is Ubuntu. The three most popular distribution is the Indigo, Kinetic and Jade Turtle. [3]

ROS Indigo and Kinetic are currently widely supported distributions. Indigo is the most stable version; however, essential packages are missing from it. Jade Turtle is the latest version, but because of that it is least stable. The kinetic distribution of ROS offers a trade-off between the two versions. Therefore, the best choice is the Ubuntu 16.04 (LTS) with ROS Kinetic distribution. [3]

CoppeliaSim proves to be a remarkably better choice than its peers. It has an integrated development environment and fast algorithm development capabilities. [5]

The simplest way to describe this simulation environment is a three-tier model. The hardware is in the first layer. The second layer is the operating system that manages the hardware and the application layer which in the simulation runs. It is important to note that the simulation must be designed in such a way that it can be considered a real robot by another program. Therefore, no other layer or communication protocol is required in that case.

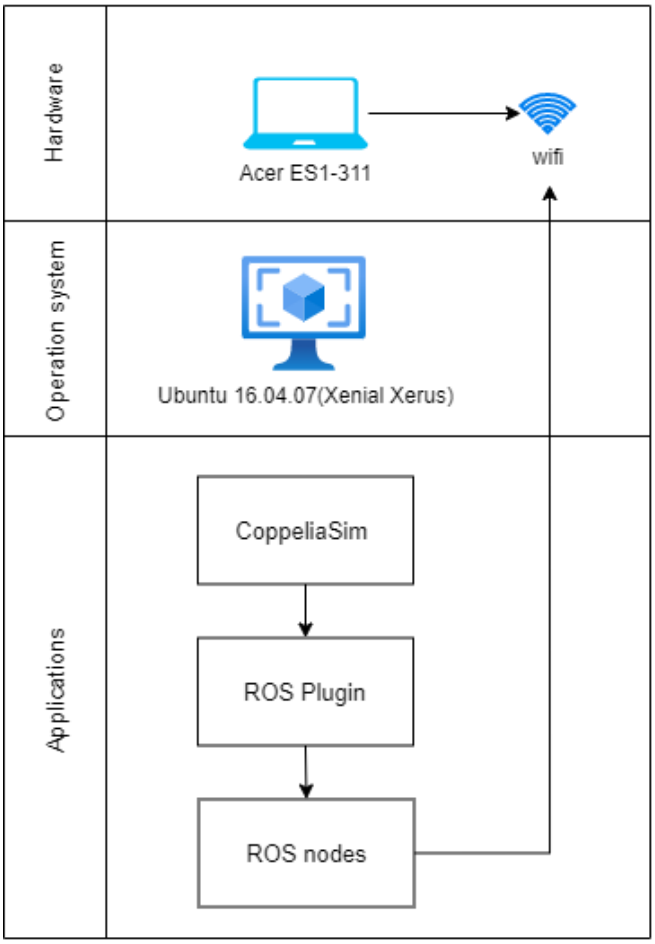

Fig. 2: Structure of the simulation

Following the example of the physical mobile robot, a virtual mobile robot was built. It's easy to create a new model, but it's important to keep in mind that the goal is to create a model that looks good, provides a well-optimized, and stable simulation.

The scene objects were divided into two groups. The first group includes the objects needed to build a mobile robot. The second group is environmental objects, which include walls, obstacles, and other objects.

In a new scene one opportunity is to model individual shapes from primitive shapes. In this case, shapes are well optimized and can be handled dynamically by the simulation program. However, these shapes do not reflect the shape of the robot. Therefore, the already completed 3D models were used. This makes it as accurate as possible externally. After importing, the scales are needed to be set and each shape named.

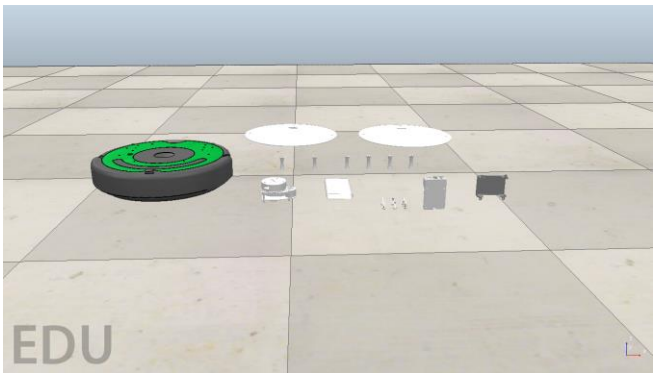

Fig. 3: Model elements

All the shapes required for the external appearance of the mobile robot are visible. To easily handle forms, grouping is required. There are three groups. The first is Roomba, the second is Platforms and the third is Lidar. After compilation and grouping of all the forms, the model is complete. However, it is not yet suitable for simulation. Pure shapes must be created from these shapes.

After creating the pure shapes, the model became very complex in structure. Each triangle in the simulation requires a calculation therefore the program becomes very slow. The model needs to be simplified. The number of triangles must be reduced. This can be done by using the Decimation function. There are also triangles inside the model, these are not necessary elements for simulation, which can be deleted by the Extract inside of selected shape function.

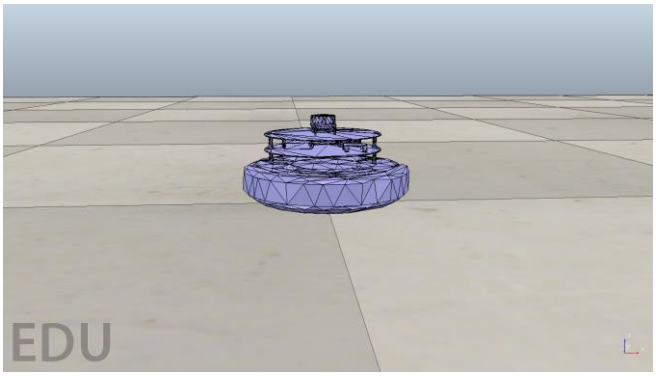

Fig. 4: Simplified model

The original model contained about 25,000 triangles. After simplifications, that number dropped to only 5,000. Further simplification is also possible using the Shape edit mode. However, the goal is not to oversimplify.

To get nearly the same point cloud when testing the virtual and real mobile robots, a test track is necessary. For the first tests a common apartment was chosen.

The test environment is a 56 square meter panel apartment, the layout of which can be seen in Figure 5. The CoppeliaSim model library contains almost all the elements needed to model the apartment.

Fig. 5: Environment

The scene consists of three main rooms. The first is the living room and dining room, followed by the hallway and finally the bedroom. For the mobile robot to be able to detect walls and obstacles, each object is defined as a dynamic element. When the simulation starts, the doors open automatically at a 90-degree angle.

Hierarchy design is an important element of simulation. It facilitates to make a logical connection between each element. The Roomba is located at the top followed by the platforms and finally the Lidar.

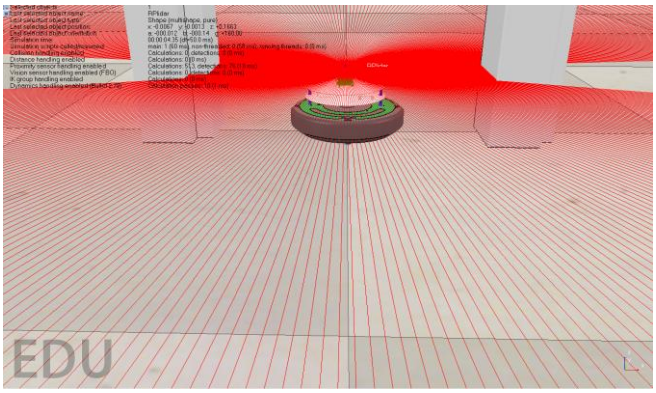

Fig. 6: Simulation

IV. Control Algorithm

Once the real and virtual mobile robot is complete, the design of the control module followed. It was realized in Matlab environment. Control of the robot can also be solved on the personal computer, which is running the simulation, however, its resources were not sufficient to perform both tasks at the same time. Therefore, a dedicated PC was used as a ROS master. The PC is running Windows 10 operating system. Supported by Matlab.

The program code is divided into four parts. In the first part, the blocks responsible for ROS communication can be found. The second part is the collision avoidance block. The third part is the Navigation block and the last is the SLAM implementation blocks.

Avoiding obstacles is one of the key issues for mobile robot systems. It must include some kind of collision avoidance. It is necessary to implement an algorithm that detects the obstacle and stops the robot. It then avoided the obstacle and progressed towards his goal. Topics provide the necessary information to develop the algorithm.

A custom block was also implemented. It has a total of five input parameters. Two of these cannot be linked to ROS. The first is the maximum speed. Basically, between -1 and 1. Speed values can be specified. However, with this solution we can also reduce and increase this value. The other is the emergency stop, in case of unwanted operation of the robot we can stop immediately. The other inputs are the known Topics. At the output, the Topics are transmitted necessary for the robot to move and the motion ban, which indicates if there is an obstacle near the robot. The emergency stop can take two values. 0 does not require a stop, the robot is working properly. A value of 1 indicates when a stop is required. [6]

If there is no collision, control is passed to the navigation block. Basically, the logic uses permutation of input values to detect and avoid collisions. An example of this is if the right bumper sensor has sent a value of 1 but another sensor has zero then there is an obstacle on the right side of the robot. In this case, the robot shifts to reverse and begins to turn left. After a value of 0 is repeated for all sensors, it transmits control. [6]

If the mobile robot were equipped with an accurate location system, there would be no need for a navigation block. The robot would always know its position. However, this is not possible for the reasons mentioned.

There are several options for navigation. The most common method is to use localization points. In this case, the robot knows its surroundings and has an accurate map. To get from the starting point to the specified destination, a route planning algorithm is needed that calculates the optimal route.

Another method is to focus only on avoiding a collision. The robot is constantly moving forward, if it detects an obstacle, it will dodge and continue its journey. The biggest disadvantage of this is that the crawl time cannot be determined. It can easily turn out that the robot is unable to find a way out of a room and gets stuck. Therefore, this method can only be used effectively in open spaces.

It is important for the robot to be able to map even unknown environments. Thus, a navigation form was necessary that facilitates it. As a result, an algorithm was chosen that follows the walls.

The algorithm is based on odom Topic. Based on the Lidar data and orientation, the distance to the wall can be determined and the degree of orientation error can be determined from this. If the error rate is 0, the robot travels parallel to a wall. If this value is non-zero, a correction is required. The Lidar has 0 ° and 360 ° viewing angles that can be narrowed down dynamically to get the best results. After determining the orientation error, it must be defined which wall the robot will follow. The easiest way is to find the nearest wall. [6]

Complementing the two indices with the necessary orientation. The difference between the measured and the required orientation gives the amount of error that has not yet been evaluated. The two sub-systems contain the necessary data structural changes. Based on the obtained data, a Matlab function block calculates the required speed and direction.[6]

Matlab is a great help in creating SLAM. There is an official Matlab example program that can implement SLAM from Lidar point cloud data. However, to do this, the full point cloud data structure must first be established.

There are two possible solutions to this problem. First, the Matlab example program can be used with different SLAM algorithms on a pre-saved point cloud. In this case, the map can only be compiled after the robot has gone through the entire test track. It also follows that its usability can only be evaluated after that. Since the robot uses a ROS framework, the other option is to use standalone ROS Nodes. This solution is more efficient as different algorithms can be applied simultaneously.

V. Conclusion

The paper introduced a simulation environment that is based on an iRobot Roomba hardware platform. The system is based on ROS framework, CoppeliaSim simulation environment and Matlab. The implemented environment is prepared for the hardware implementation with the purpose of testing relevant SLAM algorithms.

Acknowledgment

The authors thankfully acknowledge the financial support of this work by the Howmet Aerospace Foundation, the Arconic Foundation, and the project no. 2019-1.3.1-KK-2019-00007 implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the 2019-1.3.1-KK funding scheme.

References

[1] L. Joseph és J. Cacace, Mastering ROS for Robotics Programming - Second Edition, 2018.

[2] „What is Raspberry Pi?," [Online]. Available: https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/ (accessed on 25.10.2021)

[3] G. Staples, „ROS Distributions," 11 06 2020. [Online]. Available: http://wiki.ros.org/Distributions (accessed on 25.10.2021)

[4] „iRobot Roomba 605," 2021. [Online]. Available: https://www.irobotthailand.com/en/shop/irobot-roomba-605/ (accessed on 25.10.2021)

[5] „Coppeliaslim User Manual," Coppeliaslim, 2021. [Online]. Available: https://www.coppeliarobotics.com/helpFiles/ (accessed on 25.10.2021)

[6] „Robotic System Toolbox," 2021. [Online]. Available: https://www.mathworks.com/products/robotics.html (accessed on 25.10.2021)

Automation of EDM Machines

Márk Liszi
Alba Regia Technical Faculty
Óbuda University
Székesfehérvár, Hungary
liszi.mark@stud.uni-obuda.hu

Károly Széll
Alba Regia Technical Faculty
Óbuda University
Székesfehérvár, Hungary
szell.karoly@amk.uni-obuda.hu
ORCID: 0000-0001-7499-5643

***Abstract*— The automation of metal working machines is usually done for cycle time reduction and for the increase of capacity and precision. This project was started mainly to reduce the downtime between the installation of parts onto the machine. Also, the working conditions next to an electric discharge machine can be harmful to the human body in a long-term period. Collaboration of human and industrial robot will be used to create a semi-autonomous production cell of four electric discharge machining (EDM) machines.**

Keywords—robotics, manufacturing, automation, industry 4.0, electric discharge machines, industrial robot, robot cell

I. Introduction

With the use of electric discharge machining (EDM)[1], [2] we must agree with the fact, that the material remove rate is far not as fast as we are used to with CNC machines. Consequently, companies trying to fit as many workpieces into the machines as possible and they are making a mistake there. It is self-evident that they are trying to maximize their productivity but with the oversized fixtures they make the pallet and workpiece change longer. Especially if they cannot run another pallet because the short of space.

With the creation of an autonomous robot cell, we will try to reduce our downtimes caused by the change of the workpiece and the fixture. Dual education gives us the opportunity to use the resources of the university and also the industrial environment [11]. Thus, our solution can be implemented in simulation and in real-world as well. This paper focuses on the simulation environment.

II. Machined Parts

The workpieces are made of nickel-based superalloys. The electrodes what we are using for material removal are made of graphite. During the machining process relatively huge amount of metal is removed. Also, a thin layer of graphite is burned down from the electrode in each discharging process. The volume of the waste material creation under a full machining session creates about a bucket (5 liters) of sludge.

These workpieces can be found more than twenty different shapes within the area of the factory. For our project we will care about the three most identical. The smallest, the largest and the heaviest composition (see Fig. 1.).

TABLE I. Main Parameter of Workparts

Workpieces' parameters			
Part name	***Overall dimension***	***Weight/piece***	***Pieces / fixture*** [a]
Part1	156x79x30 mm	1,1 kg	24
Part2	170x367x244 mm	40 kg	2
Part3	488x610x235 mm	53 kg	1

[a] According to the current fixtures.

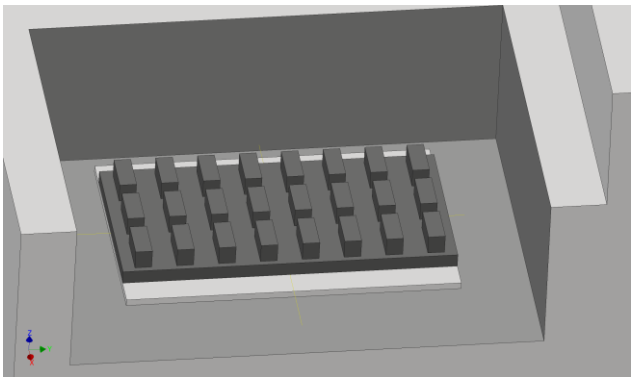

Fig. 1. The model of the largest fixture installed onto the EDM machine

According to the previously mentioned sludge accumulation the use of a conventional zero-point system is impossible. Previous experiences show that most type of commercially available zero-point sets are not sealed enough to handle this amount of waste material. While they are properly separate the dielectric liquid and all solid particles in it if the two segments are attached. As soon as the spigot is moved out of the clamp, sludge and dust can enter the clamping system (see Fig. 2.). Furthermore, the manual cleaning process were not precise enough and small amount of solid particle could get inside the clamping unit from the spigots.

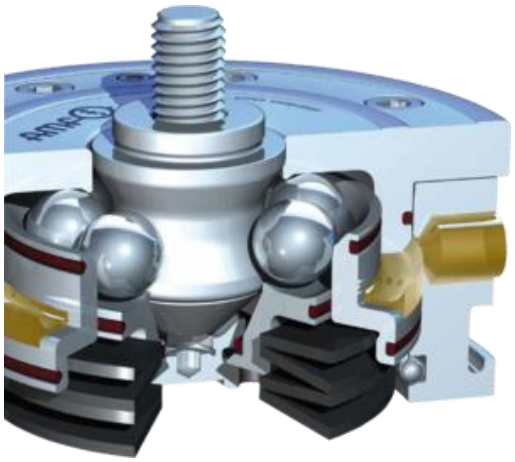

Fig. 2. Cross section of an conventional spigot-clamp assembly [9]

III. Downtimes

Currently the change of parts in the fixture happens while the fixture is installed onto the EDM machine. After the finished parts has been taken out from the fixture, the operators have to clean up its place for the next unmachined part.

In case of installing another fixture to the machine the operators first must have the zero point of the pallet.