Automated Driving with Real-Time Processing

Maria Malshakova*, Sebastian Manger*, Valentin Lindorfer, Thomas Schweiger
Höhere Technische Bundeslehr- und Versuchsanstalt Wiener Neustadt
Higher Technical Federal Teaching and Research Institute Wiener Neustadt
Department of Computer Science
2700 Wiener Neustadt, Austria
*Corresponding author's email: malshakova.maria@student.htlwrn.ac.at
manger.sebastian@student.htlwrn.ac.at

***Abstract*—This paper explores the future of technology and robotics, with a focus on automated driving. Automated driving is an emerging technology that aims to improve the efficiency and safety of transportation. A lack of automation in current robotic systems often leads to operational failures, some of which can have serious consequences. It is also a more convenient type of transportation. It would be beneficial to people who are unable to do day-to-day tasks due to health issues or very busy schedules. Although it cannot achieve a lot, a robot programmed for automated driving can be used for tasks such as package delivery, bringing coffee or tea, and more. Automated technology is not a completely new idea. It traces back to Tesla, Mercedes, and Nuro's delivery vehicles. They use autopilot to increase comfort and decrease the accident rate during traffic. However, since automated driving is a developing technology, it faces several challenges, including path-finding failures and dynamic environments. This paper provides explanations of how automated driving will make daily mundane tasks more efficient, examples, and customized solutions to these issues.**

Key terms-automated driving, automated vehicles, nodes, dynamic environments, collision avoidance.

I. Introduction

In this paper, the prime example, as well as solutions, will be based on the experiments with Coffee Bot, otherwise known as HUSO(figure [1]). The ROSbot was created by the students James Gosling and Matthias Rottensteiner. Its task is to use nodes and the map they provide to navigate through obstacles and deliver the coffee to the end goal. It uses a map and nodes to navigate itself through the dynamic environment that is new to the robot. Its main objective is to avoid collision with any obstacles and ensure successful coffee delivery. This research aims to enhance the navigation efficiency of autonomous robots by testing the robots performance in dynamic environments and proposing AI-driven solutions to improve obstacle avoidance and route optimization.

II. Experiment Setup

Experiments will be carried out on the ROSbot(figure [1]), to test the efficiency and capabilities of the robot. After every test, there will be a breakdown of what happened and how it was able to achieve that. There will also be explanations of the map itself that the robot will use. General tests will be carried out with the robot placed in the hopper. The robot will receive an instruction to get to a point in the environment. It will receive the map and coordinates and get to its destination using the most efficient and safe route. The number of successful and failed navigation attempts will be recorded, along with the safest and most efficient routes chosen by the robot. It will also be tested on how well it adapts and navigates new environments.

Figure 1: The ROSbot, front left side, close up with no changes or edits to how the original robot looks like

A. Nodes

The ROSbot uses nodes to locate itself in the environment, but what are nodes? Well, it's quite simple, really. The definition of a node is a unit of computation; it is a tool for communication and carrying out tasks within the system. In ROS, it is a fundamental tool used as a basis for everything. To understand them more clearly, it is important to note that a node can only be used for one task—for example, handling devices or computing algorithms. Although that negative effect resolves itself due to the fact that nodes can communicate with one another. A node can be programmed in multiple languages, such as C++, Java, and Python. It can be a simple or complicated program, depending on the situation and the task it has to fulfill.

B. Goal

The robot will have a certain goal that it needs to achieve with the least chance of getting into an obstacle. The robot will initialize, analyze the received mini-map, and determine its

location before navigating to its destination. The environment will be fluid, as we will test in different locations multiple times to see if it is better at locating itself on clearer fields or on more obstacle-heavy fields. The measurements that will be taken are accuracy and error rate. These will be put into a graph and then further analyzed and broken down into sections, such as navigation capabilities, obstacle avoidance, and route choice. The robot uses nodes [1], specifically two nodes, to locate itself and the goal on the map. These nodes carry coordinates that are scaled down on the map and then up again for the real-life situation. This experiment requires a specific ROSbot robot and a VPN connection over a laptop. In order to receive the map and see what the robot sees, as well as set an end goal, Gazebo needs to be installed [2]. It is a simulation that allows the user to see a simplified map of what the robot has to work with, as well as areas that it considers an obstacle or not. The map has color-coded areas that have a specific cost of going there. The closer the route is to the obstacle, the more it is a danger, and it will try to avoid it. Ideally, the robot's route should be efficient, cost-effective, and collision-free [3].

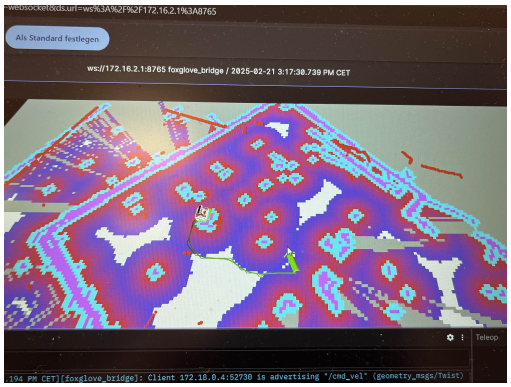

Figure 2: The map utilising nodes and creating a digital environment, in which one can set destination nodes for the robot

III. Implementations

Now, a test run will be conducted to evaluate how the test robot perceives itself and navigates through different environments. [4]. The series of tests will be run based on difficulty, efficiency, and cost-effectiveness. Difficulty levels are subjective; therefore, we categorized them based on the number of obstacles and the complexity of the path (table [I]).

A. Static Environments

In the first experiments, the robot was observed interacting with a static environment. It drove down a straight path with an obstacle in its way, allowing the tests to evaluate its ability to detect obstacles and navigate the most efficient route around them. The difficulty level for the static environment had been graded by the number of obstacles in its way. From that, it had been assessed based on the efficiency of the route and collisions. As one can see in Figure [2], it chooses the fastest way around the obstacle to get to the destination safely. The accuracy of the avoidance had been flawless (figure [3]), even with the higher chance of cost-effectiveness. If this was done with many small objects, the robot might try to find a way around if the gaps between them were too narrow. Should this not be the case, the robot will try to fit between the objects, which can lead to collisions and loss of drive. Following the completion of the experiments, it has been recorded that the robot's performance during the experiments produced the same results, even with slight fluctuations. The results were divided into five categories of difficulty.

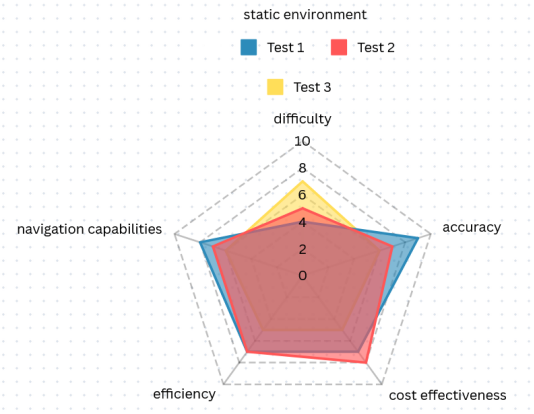

Figure 3: The diagram reflecting tests run in the static environment

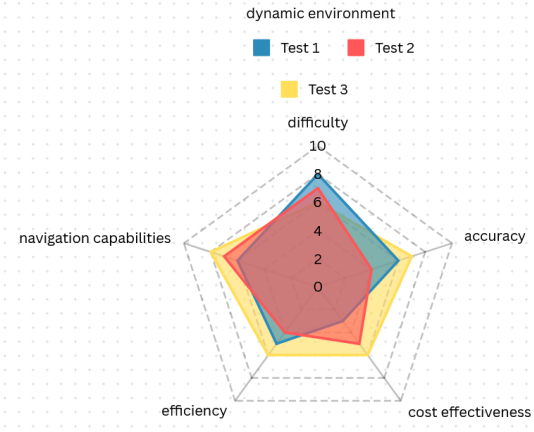

Figure 4: The diagram reflecting the tests done in a dynamic environment

B. Dynamic Environments

For the following set of experiments, tests had been run to evaluate how the robot adjusts to ever-changing, dynamic environments. The experiments had been conducted on a straight path with different environmental changes that could occur at any time. For example, a person walking in front of it or a small object, such as a can, being dropped in front of it. The difficulty had been measured by how unpredictable and close the obstacles were. The difficulty was measured based on the unpredictability and proximity of the obstacles. The

data gathered (figure [4]) from these experiments show that the robot can react to slow-moving objects and adapt its path accordingly. However, it is not able to avoid fast-appearing obstacles, especially if the obstacle appeared suddenly from the side. Due to its slower reaction time, it often spilled the coffee it was transporting. This issue could be addressed through future improvements in navigation algorithms and reaction speed, which will be further tackled in the proposed solutions.

Metric	**Definition**
Navigation capabilities	How closely he follows the predetermined path
Efficiency	How fast HUSO reached the goal
Cost effectiveness	Does he take the shortest possible path
Accuracy	Does he run into obstacles
Difficulty	How difficult it was for him to get to the goal

Table I: The direct depiction of all the information to the (figure 3 and 4) diagrams and further definition and explanation of the robots navigation course.

IV. Proposed Solutions

The results in this paragraph are reproducible and applicable to real-world robotic designs. Therefore, they are purely based on the factors observed during the experiments conducted during testing. There will be little to no sources, as these are observation-based solutions. Not every possible solution will be covered, as there are quite a few, due to automated driving being a fairly new technology. Additionally, some solutions are self-explanatory and are not an issue with the robot itself but rather with how people implement it into their daily lives. Furthermore, this section will not cover any modifications to the coffee cup the robot will be carrying, as this should have a universal solution applicable to other automation systems. Many of the proposed solutions would work best in combination, as they help eliminate the chances of collisions and inefficient routes.

A. Predictive Modeling

Predictive modeling will utilize a model for the object that acts as a dynamic obstacle and position it in front of the robot. It will also use AI to predict the object's course of action, increasing the chances of avoiding it, even if its movements appear unpredictable [5]. The ideal implementation would involve running the modeling process simultaneously after both nodes have been activated. It will take all obstacles into consideration and, as the robot operates, provide real-time feedback and updates on the object's movements, allowing the robot time to calculate a new route. This process will be applied to all dynamic objects, ensuring that the end route is continuously updated. Predictive modeling also works with patterns. It will learn and store common obstacle patterns, such as doors opening or a crowd of people walking in one direction. The robot can then reuse this information to save time by applying existing solutions instead of generating new ones from scratch.

B. More Sensors

The idea is to add sensors all around the robot so that it does not have to rely solely on cameras. This way, new obstacles coming from different directions will not pose as much of an issue. The sensors will cover blind spots (figure [5]) and provide more information for the robot to work with. Additionally, they will improve the robot's ability to orient itself by enhancing its environmental awareness. The sensors can be of different types [6] to detect forms of motion that regular sensors cannot. For example, an infrared sensor can detect objects even in low visibility, allowing the robot to navigate without issues in a dark room. The sensors can also work alongside the camera to improve reaction speed over time. One drawback is that an increased number of sensors could overwhelm the system with excessive data input. However, with more advanced programming, the robot can filter the gathered information, extracting only the most relevant data without delays that could slow down efficiency. Another advantage of having more sensors is the presence of backup options. If one sensor malfunctions, others can compensate by gathering the necessary data. A sensor does not even need to completely fail—if one takes longer to relay information to the robot, another can retrieve and transmit it faster. This will help prevent the robot from needing to stop and wait for data processing.

C. Preemptive Path Planning

To avoid the robot waiting for an obstacle to appear, it can instead have a dynamic navigation strategy in the form of preemptive path planning [7]. It will constantly reevaluate and adjust the situation, rather than being forced to make last-minute decisions. This approach allows the robot to consider the entire environment and find the most efficient route without needing significant adjustments when encountering obstacles. It will scan ahead to analyze the path in real-time and adapt to the situation without requiring last-second changes. Another positive effect of such an algorithm is the ability to create alternative routes. For example, in an office environment, areas with high traffic, such as main doors and meeting halls, could be predicted as obstacles. The robot could take a safer route that avoids these high-traffic areas, thereby preventing collisions or obstacles in general.

D. Buffer Zone

Adding a buffer zone is the most common method used to avoid obstacle collisions. Cars such as Tesla use this in their architecture. As seen in Figure [7], there are different layers of

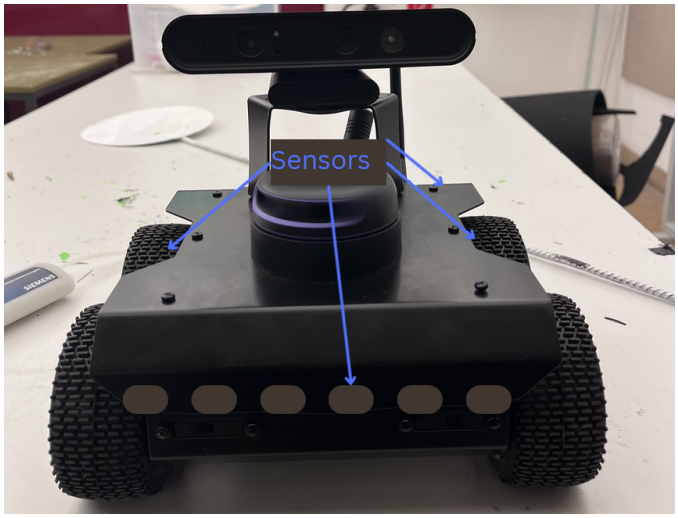

Figure 5: Further sensors surrounding the robot, one on all the bottom sides in a row, to predict unexpected obstacles on its way.

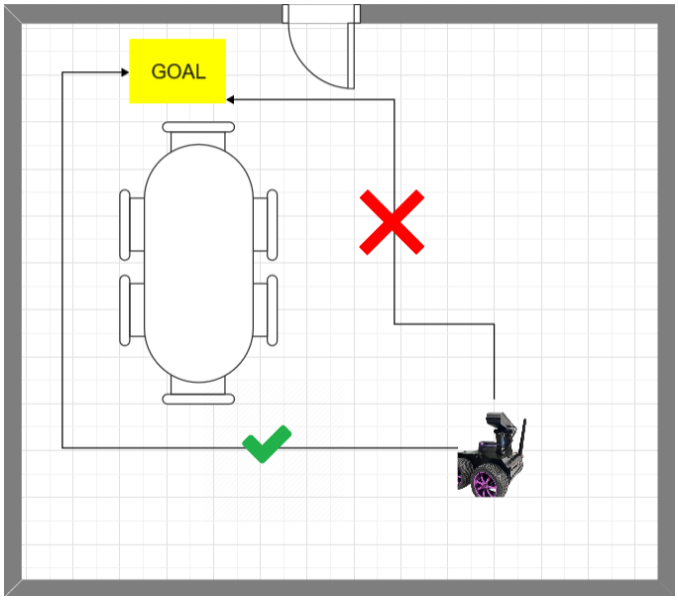

Figure 6: The robot creating a new solution due to a path being blocked by incoming traffic

the buffer zone, all of them monitored by cameras in a smaller robot, sensors will be enough). Tesla's buffer zone is quite large and ensures that zones are detected depending on the distance. The closer the object is to the car, the higher priority it has. If there is an object outside or right on the border of the zone, it will not be taken into account as quickly as when the danger is closer [8]. The buffer zone provides an area around the robot that acts as a safety border. It will detect when an obstacle is located in the buffer zone and avoid direct collision with the vehicle. For example, the coffee bot would slow down or move to the side to avoid fully halting or spilling the coffee. The buffer zone works with already pre-established sensors, therefore minimizing the cost. It also provides more predictable behavior for humans interacting with the automated bot. The buffer can be adapted depending on how cost-effective the route or area is. If the cost-effectiveness is high, the buffer zone will be large, for example, when passing through doors or crowded halls. A larger buffer zone is required for a cost effective route, due to the bot needing more time in figuring out if there needs to be a change of path. If the buffer zone is decreased, there are more chances of crashes. However, if it is a very non-cost-effective route, a small buffer zone can fully handle the situation. This will also work as an extra layer of protection in combination with other methods.

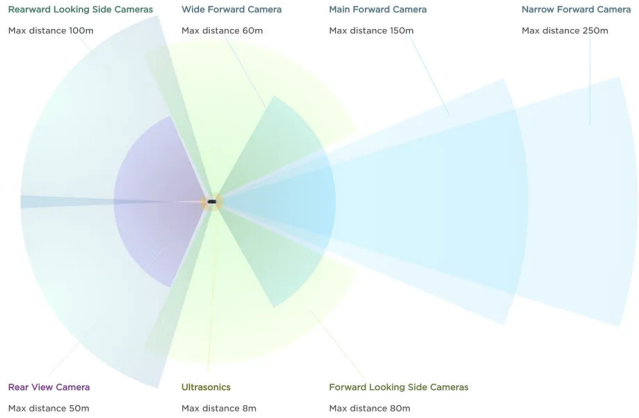

Figure 7: Teslas implementation of the buffer zone, in their cars, colourcoded by importance [9]

E. Priority Obstacle Handling

An algorithm can sort obstacles into separate categories and priorities. Depending on the priority, the obstacle will be regarded and dealt with first. For example, high priority would include fast and sudden-moving obstacles, such as a rolling can from a blind spot or a falling object. The algorithm uses nodes to determine how to handle the object, which direction to move, or if there is a need to slow down. This is often used in technology that is not even fully automated; for instance, drones with collision avoidance assign walls a higher priority than lesser dangers such as leaves or branches. However, the issue arises when the environment changes suddenly and frequently, which can cause low-priority obstacles to be overlooked and lead to collisions. That is a rare occurrence but not something that can be entirely avoided, especially in high-paced environments. The solution is to have a different algorithm for each priority. A simplified version would be [10]: low = slight path adjustment, medium = reduces speed, high = stops completely. This can be known as the traffic light algorithm.

F. Node Storage

The robot will use multiple nodes that utilize a system containing coordinates. With the first node, it locates itself in the environment, receiving its own coordinates. The second node is set by the user and is the destination location. The other nodes will be there as static points that the robot has memorized during older runs. This allows the bot to find the coordinates of the location and determine the quickest path to the destination.

The nodes, also known as waypoints, are in the system prior to the robot moving from its location. The nodes use the map (figure [2]) to consider the route before actually driving. This prevents the need to constantly create a new path, and it can adapt to changes. However, sometimes it is not quick enough. Now, how would that look if the path was not constant? If it had less time to figure out what to do in a collision? It is actually quite simple. If the destination and original location are the same every time, instead of following the same path, it could adapt to unexpected changes. By recalling the locations of high traffic as well as the most common obstacle collisions, it can detect shortcuts or safer, non-cost-effective routes (Figure [8]). This small change can ensure that the bot will not spill the drink. Another solution would be the addition of more nodes. They could serve as additional waypoints to enhance navigation. If AI is integrated into the system, the surrounding detection has the chance to adapt on the fly with less need for constant storage. It can store the more common danger areas and then rely on AI to handle sudden issues.

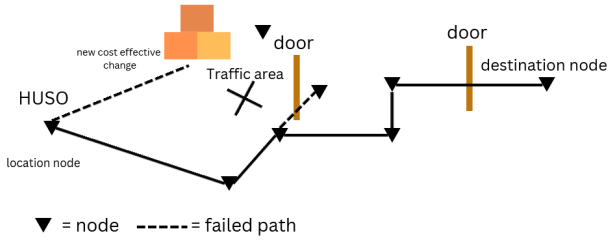

Figure 8: navigation through node way points, and substitute of check points if one of them is unreachable

G. Results

The proposed Solutions were put to the test and the following data had been collected. The success rate, otherwise known as how many times had the robot achieved its destination, is an observation done through collection of data in experiments. The reaction time consist of the speed that the robot detects and avoids obstacles, as well as the comprehension of the environment. Although the collisions are pretty self explanatory, it is good to note that they are measured per a singular test. Therefore, the data is more precise. The following table II reflects all the proposed solutions and gives back the outcomes of the tests.

Test Condition	Success Rate (%)	Average Reaction Time (s)	Collisions per Test
Baseline (No AI)	40%	1.2	3 per test
Predictive Modeling	85%	0.8	1 per test
More Sensors	75%	1.6	0 per test
Preemptive path planing	55%	1.2	1 per test
Buffer Zone	70%	1.0	2 per test
Priority Obstacle Handling	80%	1.1	2 per Test
Node Storage	70%	1.3	3 per Test

Table II: The results of testing all the solutions

V. Conclusion

To conclude, the best way to eliminate most issues that arise in automated driving is to use a combination of the solutions provided. The best possible outcome would be to integrate AI in predictive modeling with the addition of extra sensors and nodes. This will give the robot the highest chances of being efficient and avoiding collisions. However, this depends on the specific situation and environment in which the robot is being used. A simple setup at home would not require as many integrated add-ons, as they would only make it more expensive. However, if you are planning to have a coffee bot or multiple bots in a busy office atmosphere, it is best to invest in as many safety precautions as possible. Failing to do so could end up costing the company more.

VI. Acknowledgement

The authors would like to thank Dr. Michael Stifter for helping with this project and giving us constructive criticism, which greatly helped us improve our work. We would also like to thank Ing. MSc. Eichberger for supporting us and helping us stay on the right track. We couldn't have done this without their guidance. A thank you also goes out to James Gosling and Matthias Rottensteiner for allowing us to conduct experiments with their robot, HUSO, and for providing us with information about how it worked.

References

[1] rafal gorecki, "rosbot github." https://github.com/husarion/rosbot-autonomy.
[2] H. sp. z o.o., "Staring huso and seeing the map." https://husarion.com/tutorials/howtostart/rosbot2pro-quick-start/#remote-access-over-the-internet-vpn.
[3] N. Adiuku, "Learning-based navigation systems." https://www.mdpi.com/1424-8220/24/5/1377.
[4] R. R. . A. Kos, "Towards goal-directed navigation through combining learning based global and local planners." https://www.nature.com/articles/s41598-024-72857-3.
[5] X. Z. . Y. G. . L. Guan, "reinforcement learning (drl)." https://pmc.ncbi.nlm.nih.gov/articles/PMC6339171/.
[6] Quarz, "Ultrasonic sensor pinout." https://quartzcomponents.com/blogs/electronics-projects/obstacle-avoiding-robot-using-l298n-h-bridge-motor-driver.
[7] P. Li, "Preemptive-level-based cooperative autonomous vehicle trajectory optimization." https://www.mdpi.com/2079-9292/14/1/71.
[8] Tesla, "Tesla automated driving." https://www.tesla.com/support/auTesla.
[9] Tesla, "Teslas implementaion of the buffer zone." https://www.tesla.com/support/autopilot.
[10] A. G, "obstacle avoiding robot." https://www.instructables.com/How-to-Make-Smart-Obstacle-Avoiding-Robot-Using-Ar/.