Industrial Intralogistics – Improving an Autonomous Mobile Robot Learning Platform

Matija Markulin, Daniel McGuiness (supervisor), Benjamin Massow (supervisor)

***Abstract*—This paper shows an implementation of Autonomous mobile robots (AMRs) in industrial logistics and their use in improving MCI's AMR learning platform. This is achieved by using *Turtlebot4* mobile robots made by *Clearpath Robotics* in a model of an industrial logistics center and controlling them by remote PC using ROS2 (Robot operating system) infrastructure and its navigation libraries. The robots are programmed in *Python* using ROS2 APIs (application program interfaces) which allow for easily discernable code for teaching this subject. The produced source code for robot operation, as well as detailed instructions for system setup, are assembled in appropriate teaching material for MCI's learning platform.**

***Index Terms*—ROS2, Turtlebot4, Autonomous mobile robots, SLAM, navigation**

I. Introduction

Intralogistics, management and delivery of the materials within the industrial complexes, is an important part of modern warehouse management systems. It enables the seamless movement of goods within the system and raises the overall production efficiency. Lately, autonomous mobile robots (AMRs) are becoming a more and more important part of any intralogistics system. AMRs are the area of robotics that has been quickly developing in recent years due to further developments in the miniaturization of computing components making it possible to incorporate all the necessary components into a mobile system. With the autonomous navigation capabilities of AMRs, it becomes possible to introduce automated logistics solutions even in dynamic environments where conventional mobile robots with predefined paths would be impossible to implement. With the adaptability of AMRs, it is possible to automate the existing warehouses with minimal expenses for adapting the space itself or even to have a combined workspace where humans and robots can work alongside each other without any danger to either.

Complexities of the AMR systems require experts in robot programming to set up the system like this. These experts need to be taught during their studies and that requires a robust learning platform.

The setup of one such system on the model of the logistics center will be shown in this paper, as well as its arranging in a learning platform for future generations of students.

II. Previous work

In previous work [1], the basic setup and usage of robots are explained for the beginner robot programmers to get familiar with programming the robots with ROS. It explains basic robot commands and the structure of the ROS2 communication network but does not explain the usage of localization or navigation libraries.

There are also official tutorial pages [3] for the usage of the robots, but they provide tutorials for a few different setups of the robot, and this learning platform will focus on the one present in MCI's laboratories. Official tutorials also only briefly touch upon the topic of navigation and only with one robot, and this project also goes into the navigation of multiple robots at the same time.

III. Objectives

The objective of this project is to create a model of an industrial logistics center as shown in figure 1 and program the robots to autonomously navigate to the predefined coordinates in space. Once the setup is done it is explained in detail and made into a comprehensive learning platform to be used at MCI in the future

IV. Methods

This project is realized using Turtlebots, cost-effective modular mobile robots equipped with all of the necessary sensors for autonomous navigation. The robots are programmed in *Python* using ROS2 (Robot operating system) communication framework. ROS2 is an open-source system that provides communication between PC and robots.

Fig. 1: Model of an industrial logistics center

A. *Turtlebot*

Turtlebots, shown in figure 2, are AMRs developed by *Clearpath robotics* whose fourth iteration was used in this project. Turtlebot4 is equipped with an RPLIDAR-A1 LiDAR system, an OAK-D-PRO computer vision camera, and an IR proximity sensor on the front bumper. The robot also has an integrated Raspberry Pi 4 computer for control of the robot. According to [2], the robot's max payload is 9 kg, but with slight modifications to the robot and its software, it can be increased to 15 kg.

Fig. 2: Turtlebot4 [2]

B. *ROS2*

ROS2 [5] is a second iteration of the open-source robot programming framework. It is built on the foundation of ROS1 with improvements in real-time applications and intercommunication of all the systems. Due to the incorporation of a distributed communication framework, ROS2 can seamlessly support the work of multiple robots and computers.

The main way in which ROS operates is through subscribers and publishers. Each of the programs running on the ROS network is called a node, and each of the nodes can have multiple subscribers and publishers. The publisher broadcasts its data on the network on its *topic*. Unlike ROS1, in ROS2 publishers broadcast the data on the *topic* only once when instructed in the code and have no traffic in the meantime. Subscribers on the other hand "listen" to the *topic* and each time it receives the message a callback function is called, that can process the received message according to the requirements of the program.

Other than topics ROS nodes can also communicate

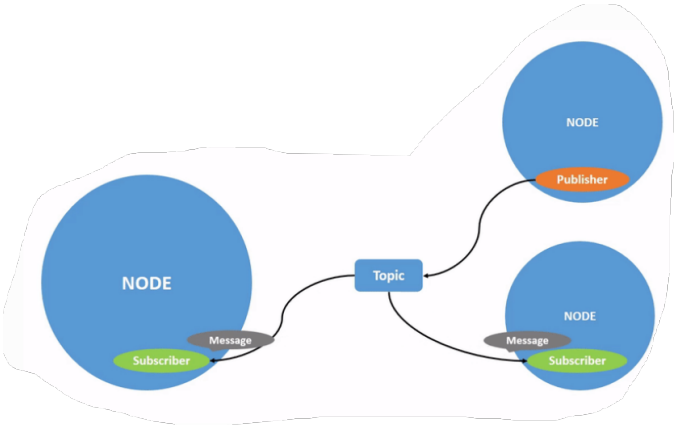


Fig. 3: Example of publisher and subscriber communication [1]

using services and actions. Services are setup in the way of server and clients, where a client sends the request to the server, the server executes needed commands and sends the client the response

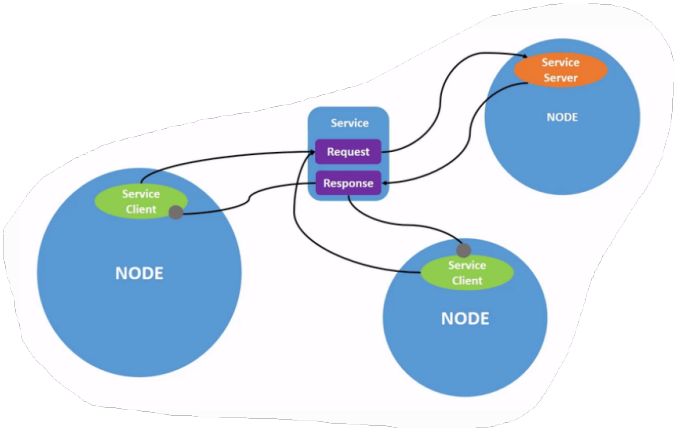


Fig. 4: Example of service [1]

Actions also work on the principle of server and client but with an even more complex communication structure.

A client asks the server to do something, and the server then confirms the acceptance of the task, carries it out, and confirms the execution along with returning the data produced by the action to the client (if there is any).

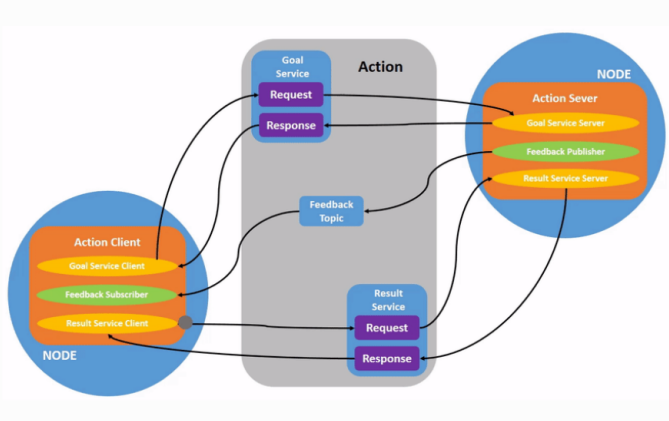


Fig. 5: Example of action [1]

ROS2 distribution used in this project is Humble Hawksbill, LTS (long time supported) version at the time of realizing this project[1]

C. SLAM and localisation

SLAM (simultaneous localization and mapping) is a technique used for mapping an unknown area while simultaneously estimating the robot's position within the environment.

Most SLAM algorithms implement probabilistic calculation methods. At first robot's location has no uncertainty and the first features of the environment have uncertainty coming from the LiDAR. Then as the robot moves its location becomes more uncertain due to uncertainty in wheel motor encoders and each new feature of the environment has even larger uncertainty that comes from the product of robot position uncertainty and sensor uncertainty. Then when the sensor registers the first feature after further driving the uncertainty of the robot's position can be reduced by comparing it to the uncertainty of the feature, and because of that can reduce the uncertainty of the position of all other features.

From this, it can be seen that SLAM is a hard problem to solve, but it has been a topic of research for many years and there are now libraries that can quite efficiently solve this problem.

These libraries can then be used to map the environment, as shown in figure 6, to be used later in the working of the robot because there is no need to run resource-heavy SLAM algorithms if the environment is mostly static with some dynamic elements (ie. shelves and workstations are fixed but there could occasionally be a pallet of the goods on the floor). In cases like this localization library can be used to find the robot's position in the environment and to see the unexpected obstacles but the base map that was created before can be loaded into the program.

D. Navigation

For the navigation *Nav2* package was used. *Nav2* is a modular robotics navigation stack based on plugins that can be developed independently and changed to the version appropriate for the current application.

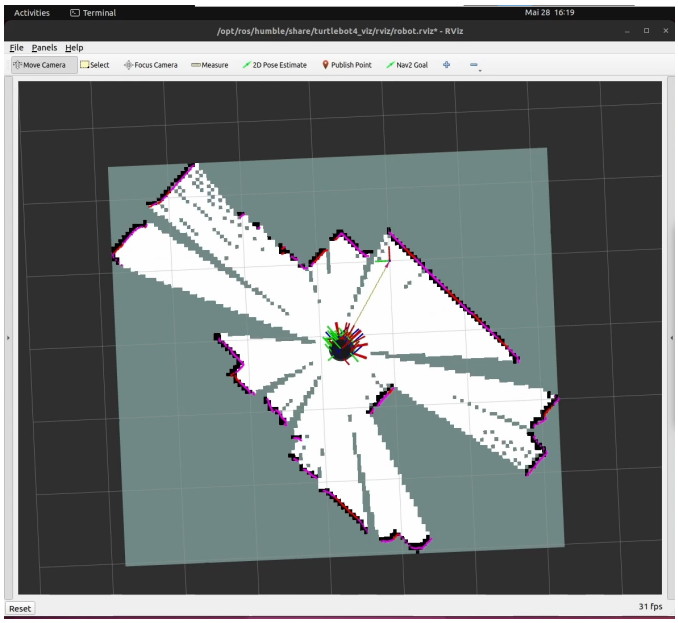
Nav2 is used to calculate the optimal path through the environment to get to the goal with regard to a map of the environment (whether loaded from file or generated by SLAM) and input from the sensor alerting to new obstacles in the way. The calculation of the path is conducted with the use of a map that is structured in several layers.

The first layer is the dark purple one in figure 7 representing the static map of the environment. The pink lines are the current readings of the obstacles from the LiDAR and are also taken into account during path planning if they do not align with the static map. An example of that is pink spots on the map which are caused by other robots that are docked around the parameter but were not there when the map was recorded. The next layer is the inflation layer which consists of light blue and pink/blue areas around it. The blue area of the inflation layer is space that the robot is not supposed to go in and is used to take in account the size of the robot to prevent it from hitting the walls. The pink/blue area is a slow-down zone that is used to slow down the robot in the proximity of the walls

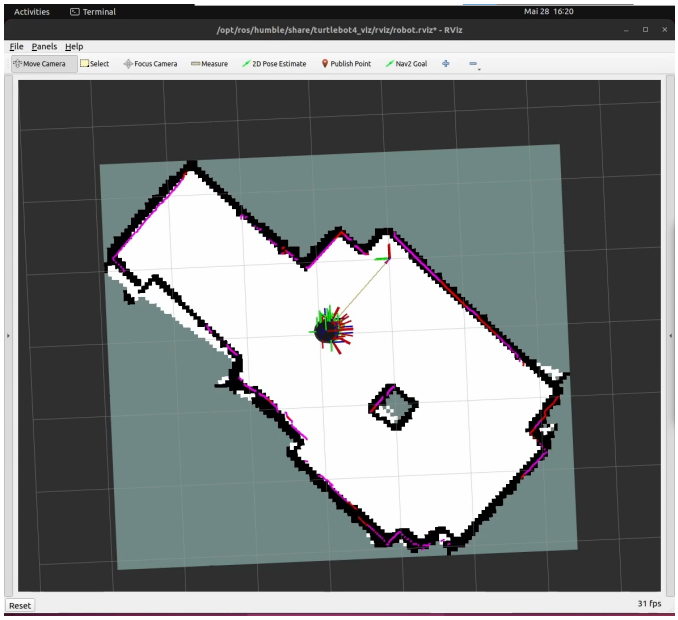
E. Robot operation nodes

Finally with the use of all of the aforementioned systems ROS2 nodes for robot operation can be programmed. Two scripts are created, one to take care of the navigation of individual robots and the master

[1]New LTS version (Jazzy Jalisco) was released during the making of this project but the support for Turtlebot4 for this version is not yet available.



(a) Begging of mapping



(b) End of mapping

Fig. 6: Mapping the work environment using SLAM

node to monitor them and give out tasks to said robots. Individual robot script is run multiple times creating ROS2 nodes for the control of each of the robots, and the master script is run only once and handles the operation of the swarm.

In an ideal case robots could communicate with each other easily thanks to ROS2 architecture, but the bandwidth of the Turtlebot's WiFi antenna is very limited and

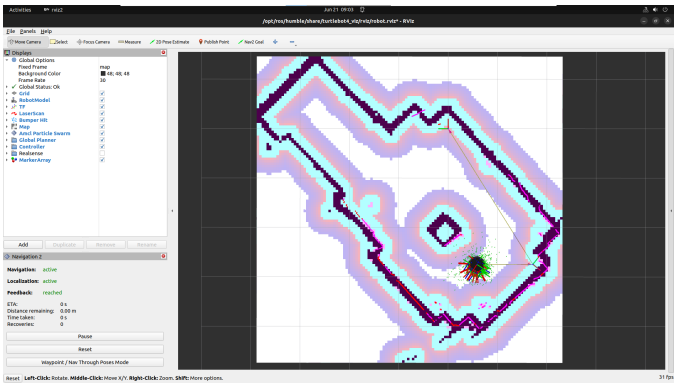


Fig. 7: Nav2 map layers

even a couple of robots running on the same network can make the connection between PC and robot to slow for real-time control [4]. Due to that, the network is set up in a way that each robot can communicate only to the master node, but not to the other robots which greatly hampers Turtlebot's collaboration and swarm capabilities.

V. Solution Setup

A. Robot setup

The first step in the development of a model of the logistics center is to set up the robots and their connection to the ROS2 framework. While a similar setup was done in [1] due to updates to the ROS2 and Turtlebot4 integrations it needs to be brought in line with the current version and to have all of the robots on the same edition of the software which was not the case before. As stated previously Humble Hawksbill distribution of ROS2 is used and the support packages for it need to be installed on the Turtlebot. That is done by following the instructions from [3]. The easiest way to update all of the robots to the current versions of the software would be to simply update all of their Debian packages, but the network the robots are connected to is not connected to the internet so that is not an option. Instead, the latest Turtlebot4 disc image was flashed to the SD card of each of the robots bringing them to the current software version.

B. PC setup

This robotics learning platform is primarily intended for use in a Linux environment. Recommended OS for use with ROS2 is Ubuntu Linux - Jammy Jellyfish (22.04). Ubuntu can be installed either natively or within a virtual machine. Proper installation of Ubuntu would offer a bit more computing power, but if the user of the

learning platform does not have it already installed, a virtual machine offers much easier access to the use of the Linux platform

ROS2 packages for Linux can be installed either as a Debian package or from source. We chose to use the Debian package installation because it is more convenient than the installation from the source. Since this thesis is developing the learning platform it will not be concerned with editing the ROS2 source code so there is no need to install it in that way. The Debian installation offers a much easier start to using ROS2 with the installation in just a few terminal commands. Turtlebot4 specific packages can be then installed in the same way. The exact process of installation is shown on the learning platform. After all the needed packages are installed robots can be programmed with PC over ROS2 system.

C. Environment setup

While most of the features of the Trutlebots can be used and learned in a simulated environment, the use of real robots and all of the challenges that come with that offer a much better understanding of robots and all of the problems a future engineer might run into. For that, a safe and controlled environment was needed.

The location for the robot's working area is the aula of the MCI 4 building. There an area was fenced in, as shown in figure 8, so that the robots can't get everywhere in the aula and get in the way. A few obstacles were placed in the area to showcase how robots can navigate around. Inside the area, there are a couple of obstacles to showcase the robot's ability to dodge obstacles.

Fig. 8: Robot's working area

VI. Navigation solution

A. Using RVIZ for navigation

The robots and the PC are now set up and the working environment is prepared so it is time to start navigating the robots in that environment. For that, we will use the map created like it is explained in section IV-C. Now that there is a map another thing that is needed for the localization is the robot's approximate initial position. The initial position can be given either through the RVIZ tool by hand or as a part of the navigation code. For the localization and navigation with RVIZ, those nodes first need to be started. Through RVIZ initial position can be set by use of the 2D pose estimate tool in the toolbar. The tool can be used as shown in the figure 9. The beginning of the arrow represent the location of the robot and it points in the way the robot is orientated. It can be seen that at this point RVIZ reports some errors in the system but those are due to the robot's unknown position so RVIZ can't place it on the map and thus throws an error, as soon as the initial position is given and the robot is localized the errors will disappear.

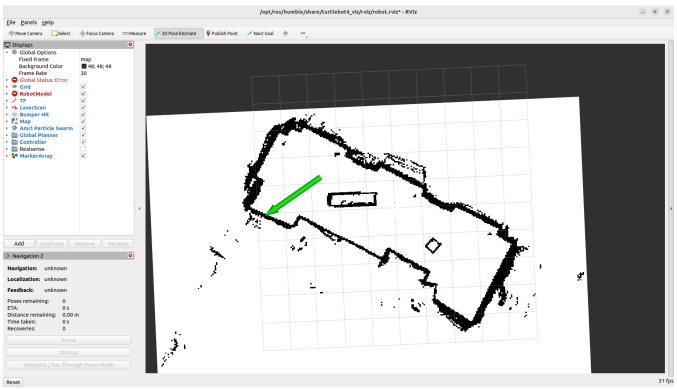


Fig. 9: Setting the initial pose

Once the robot is successfully localized the navigation can start. For that Na2 goal tool form the same toolbar can be used. It functions in the same way as the 2D pose estimate tool for marking a pose. Once the pose is given with this tool Na2 stack will calculate the optimal path to the goal and navigate the robot there.

B. Naviagtion of single robot

After we have seen how to navigate the robot "by hand" using RVIZ lets have ROS2 do it all for us. For that the package Turtlebot4Navigator is used. That package contains a child class to the Nav2's simple navigator class, with some Turtlebot4 specific functions, like docking and undocking. For the navigation the object of said class is created. Then robot's approximate coordinates are given to that object to localize the robot. After the successful localization, the goal coordinates can be given and the Nav2 stack will take care of navigating the robot to the appropriate location. There are multiple ways to instruct the robot motion, for example

only giving one point to navigate to or giving multiple points along the way that will define the trajectory.

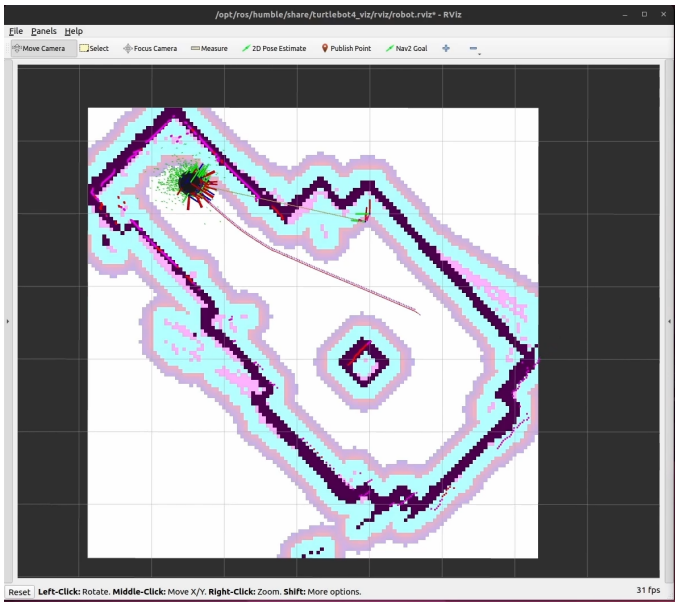


Fig. 10: One robot navigating the environment

C. Navigation of multiple robots

The problem of navigation of multiple robots becomes somewhat more complex if any level of cooperation is expected between them. If there is no cooperation each robot could just separately run single robot navigation as explained in the previous section without any concern for the others. But for cooperation, a more complex structure is needed. That structure will require separate ROS2 nodes to handle the navigation and communication. One will be run for each robot individually and will take care of controlling that robot and processing its data. The other node will collect the data from all of the individual robot's nodes, process it, and coordinate the task assigned to each of the robots.

For this setup the master node chooses the robot for the next task based on its docked status and its battery charge. The robot deemed appropriate for the task is then given the next set of coordinates to navigate to. In this example, the master node has a simple set of tasks for robots and a simple decision basis for which robot to assign the task to, but this setup can be easily modified for much more complex tasks without major changes to the overall setup.

VII. Learning platform

In the end all of the code developed here is assembled into an online learning platform for future students, a few

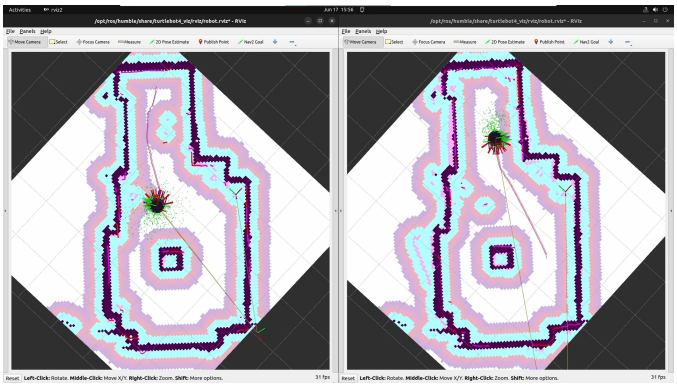
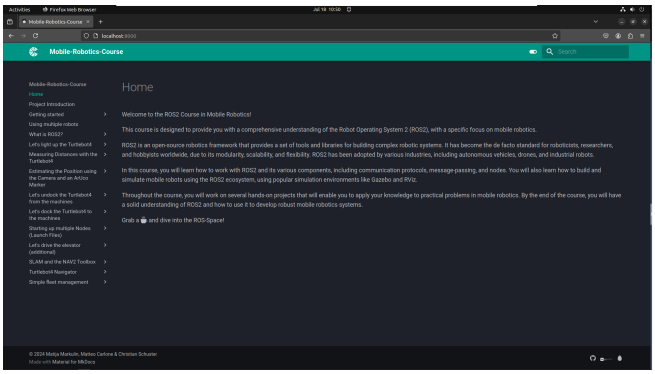


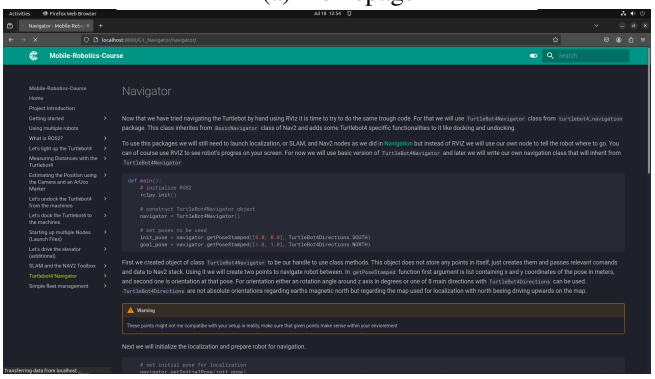
Fig. 11: Two robots navigating the environment

example pages of the learning platform are in figure 12. The basis of the platform was created in [1] and this project is expanding on it with the topics of autonomous navigation which is a crucial part of creating a working intralogistics model.

New parts of the learning platform go over the topics of PC setup in order to use multiple robots at once, how SLAM and Nav2 work, how to map the environment, how to navigate the single robot, and how to navigate multiple of them.



(a) Homepage



(b) Navigator class explanation

Fig. 12: Learning platform example pages

Each of the topics explains the theoretical background and exact method how to use any of the functions. The code needed for any of the steps is broken down and explained step-by-step to allow maximal understanding of it.

VIII. Conclusion

The problem of intralogistics has presented an interesting angle to teach the students how to use autonomous mobile robots. This application of AMRs offers a great combination of various robotics skills that are needed to realize this solution successfully. While multiple problems and difficulties were encountered during the development of this application they all offer an insight into the engineering method and a crucial part in learning how to create robotic solutions.

Coupling the ROS2 framework with Turtlebot4 creates the perfect environment for the learning of mobile robotics. Turtlebot4 is easy to setup AMR that incorporates all of the needed functionalities while also making it very modular and adaptable to the application at hand. It has all of the needed sensors already integrated into the robot, and with RasperyPi on board, it can easily be adapted to any job it might be needed for.

References

[1] Christian Schuster, "Industrial Intralogistics – Developing an Autonomous Mobile Robot Learning Platform using the ROS2 Framework", Master's thesis

[2] https://clearpathrobotics.com/turtlebot-4/

[3] https://turtlebot.github.io/turtlebot4-user-manual/

[4] https://github.com/turtlebot/turtlebot4/issues/408

[5] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," Science Robotics, vol. 7, no. 66, p. eabm6074, 2022. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074