HOCHSCHULE SCHMALKALDEN
UNIVERSITY OF APPLIED SCIENCES

Hochschule Schmalkalden
Fakultät Maschinenbau

Project Work Report

Autonomous Logistic Mobile Robot in the Framework of MATLAB/SIMULINK & ROS

Presented By

Joe Francis
Matrikelnummer: 312660

in Study Program
Mechatronics & Robotics

Master of Engineering

Supervisor 1: Prof. Dr.-Ing. Frank Schrödel
Supervisor 2: Prof. Dr.-Ing. Andreas Wenzel

Schmalkalden, den 24.03.2022

ACKNOWLEDGMENT

I would like to thank warmly Prof. Dr-Ing. Frank Schrödel for all kind of supports provided by his side. He supported me by all relevant technical knowledge that were decisive in this project, in addition to providing me with an efficient overall approach in tackling this field. Prof. Schrödel was not only just a supervisor for me, he was really a mentor and big motivator for me on all levels.

I would like also to send special thanks to Prof. Dr-Ing. Andreas Wenzel for all supports, feedbacks and cooperation been provided. They were really valuable and important. Moreover, I would like to thank Prof. Wenzel for giving us the opportunity to work on one of the important research and educational mobile robotic platforms, the Pioneer 3-DX robot. I am very grateful to have the opportunity to work on this platform, which helped me to gain more valuable experience.

Finally, I would like to thank also Mr. Norbert Greifzu for his cooperative support and for his assistance in providing the necessary details.

ABSTRACT

The main focus in this report will be on the commissioning work conducted on the Pioneer 3-DX robot. The main objective was to prepare the robot to be compatible within the development platform of Matlab/Simulink and ROS. Simulink was chosen as the main environment for algorithm developments and ROS as middleware for communication with the robot and a target for algorithm deployment.

The report tackled the robot's technical specifications, additional fitted hardware, communication flow diagram between different components & environments, detailed schematic diagram of the overall system. Then, an implementation and testing approach of the algorithm based on the V-model was introduced. This approach consists mainly of algorithm development phase in Simulink, testing simulation in Gazebo as virtual environment and then testing on the physical P3-DX robot operating by mean of ROS.

Further in the report, algorithm development within Simulink was tackled in more details, in addition to the implementation in Gazebo and then on the physical robot. More in depth details were introduced on how to establish the robot to operate and work within this development platform. In further sections, ROS environment was introduced with its necessary tools and its relation with the Pioneer robot. Later the method of Simulink-ROS bridging was discussed, with the major important tools and features that can be used within this bridged platform. In the end, a simple automated test drive, for demonstration test purpose, was achieved on the robot using Simulink-ROS. In the Future scope, the aim is to develop and implement more advanced algorithms for autonomous applications in more challenging operational domains.

TABLE OF CONTENT

ACKNOWLEDGMENT I
ABSTRACT II
INTRODUCTION 1
EVOLUTION OF AUTONOMOUS MOBILE ROBOT 2
Comparison AMR vs AGV 4
PROJECT DESCRIPTION 6
USER REQUIREMENTS & TECHNICAL SPECIFICATIONS 7
MORPHOLOGICAL CHART 8
DECISION MATRIX & PAIRWISE COMPARISON 9
MAIN TECHNICAL SPECIFICATIONS OF THE ROBOT 10
1.1 Physical Dimensions 10
1.2 Power Supply (Batteries) 10
1.3 Perception Setup 11
1.4 Driving Unit 12
1.5 Embedded Controller 13
ADDITIONAL HARDWARE EQUIPPED ON THE ROBOT 14
2.1 Master Single On-Board Computer 14
2.2 Intel RealSense Camera & IMU 15
2.3 HMI Display 18
2.4 Wireless Keyboard 19
2.5 DC-DC Converter 19
COMMUNICATION FLOW DIAGRAM 20
DETAILED HARDWARE WIRING SCHEMATIC DIAGRAM 22
DIFFERENTIAL DRIVE KINEMATIC GENERAL OVERVIEW 23
3.1 Differential Drive Important Equations 24
3.2 Differential Drive Robot Motion Behavior 26
3.3 Pose Computation from Wheel Encoders 26
ALGORITHM IMPLEMENTATION & TESTING STRATEGY 29
Testing Approach 29
ALGORITHM DEVELOPMENT WITHIN SIMULINK 32
Main Simulink Model 35
SIMULATION TESTING IN VIRTUAL ENVIRONMENT GAZEBO 43
Simulink-Gazebo Coupling 44

ALGORITHM IMPLEMENTATION ON THE PHYSICAL ROBOT 48
Working with Aria 48
Working with the Robot Operating System (ROS) 52
ROS Workflow Diagram Implemented in the Project 52
ROS Main Structure & Components 54
ROS Bags 56
rqt 58
Rviz 59
ROS with Pioneer 3-DX Robot (RosAria) 59
ROS client package Demo for the Pioneer 3-DX Robot (rosaria_client_interface) 61
Bridging ROS and Simulink 63
ROS Bags Usage in Simulink 66
Sonar ROS Bag in Simulink 66
Intel RealSense ROS Bag in Simulink 68
Pose ROS Bag in Simulink 71
Simulink-ROS Algorithm Implementation for a Simple Automated Drive 74
CONCLUSION & FUTURE SCOPE 81
LIST OF FIGURES IX
LITERATURE XIII

INTRODUCTION

Increasing safety, speed, ensuring precision and improving operational efficiency are main challenges for many factories, warehouses, healthcare institutions, agriculture sector, smart cities and public sectors. All are investing heavily in innovative ways to tackle all these challenges. Autonomous mobile robots known by AMR are becoming major key player in these domains [1].

A recent report by Tractica Research estimates that the worldwide sales of warehousing and logistics robots will reach $22.4 billion by the end of 2021, with robot unit shipments reaching 620,000 units per year by 2021 [2]. According to a report posted by IDTechEx research, the market for mobile robots in logistics, warehouses and delivery is likely to reach $83 Billion in 2032 and $334 Billion in 2042 [3]. All these statistics and future projections prove that the sector of mobile robots is very promising and it will play a vital role in the major and essential sectors [Fig.1].

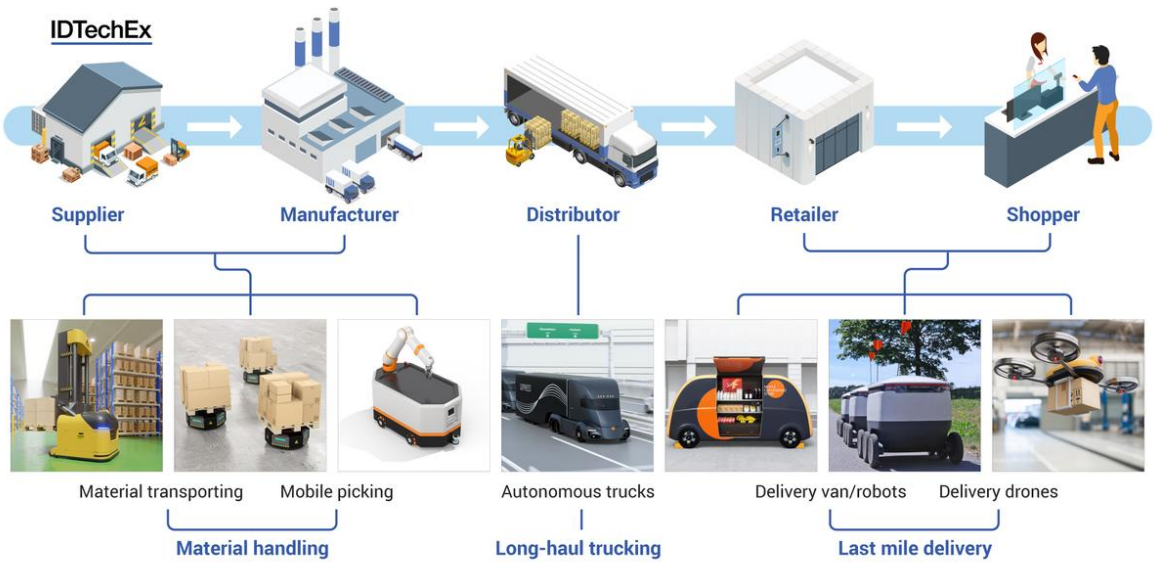


Fig.1 Autonomous Robots in Supply chain from Supplier to Customer [3]

This report will focus mainly on the commissioning phases implemented on a mobile robot, in order to make it ready for reliable autonomous applications in outdoor unstructured environments. The intended use of this mobile robot is for goods transportation from a location A to B. The future aim is to work on expanding the operation design domain of autonomous mobile robots, which will lead to a paradigm shift in the logistic sector.

EVOLUTION OF AUTONOMOUS MOBILE ROBOT

The field of mobile robots in logistics has evolved heavily in the past decades. First, warehouses were relying on manual movement of carts by humans from location A to B. Then, the adoption of fixed structure such Cartesian robots. Due to the limitations of these solutions, the warehouses shifted to the usage of automated guided vehicles (AGV). These vehicles [Fig.2] consist of some control systems for driverless operation and they are back to 1950s [4]. Most of them use a simple control system to move from location to another one, they are mostly following some predefined paths such as lane markings or wires embedded in the floor that produce a magnetic field. The AGV are equipped with sensors to detect the magnetic field generated by the wire in order to follow it. Moreover, they are equipped with other types of sensors that can detect obstacles through the predefined path in order to stop only. Those types of vehicles still in use today due to their reliability and when the involved situation does not require complex systems.

Fig.2 Automated Guided Vehicle (AGV) using Magnetic Strips Embedded in the Floor [5]

Nowadays, it is not efficient to rely only on the AGVs in logistics due to the increased demands and the needed flexibility in the operation. The vehicles need to adapt to changes quickly, make decisions by its own in order to prioritize some tasks, avoid obstacles and not to be restricted to a predefined path. Moreover, it should collaborate with humans safely and deals with increased system complexity such as fusion with other robotics applications. Based on that, the autonomous mobile robots known as AMR [Fig.3] were introduced in the warehouses and many other application fields in order to deal with that arising complexity.

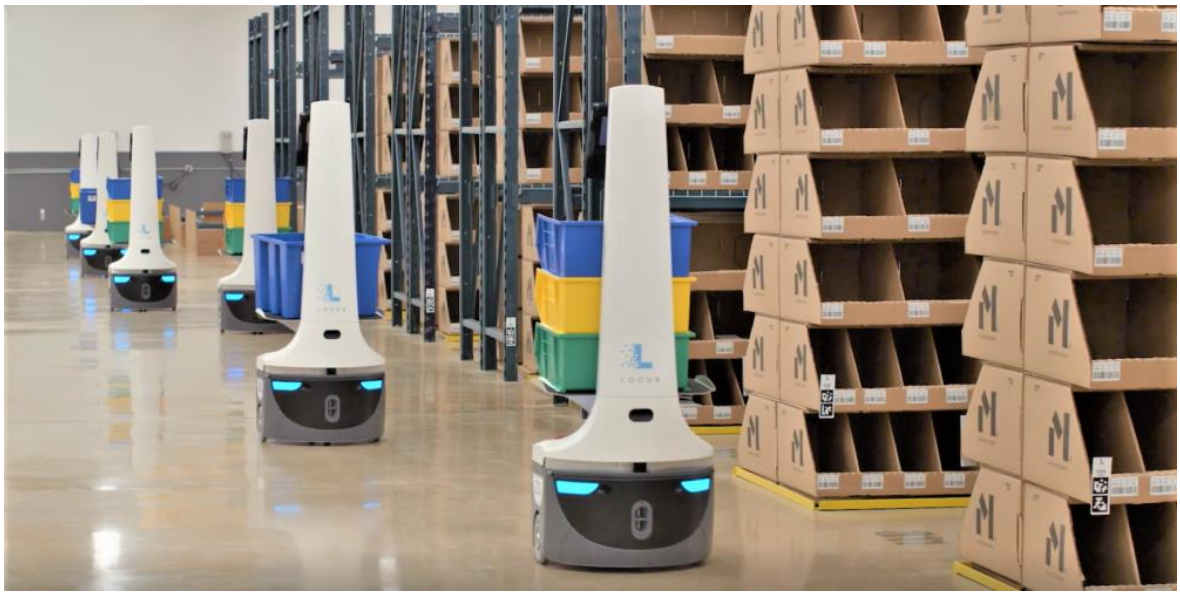


Fig.3 Locus Robotics' Autonomous Mobile Robots (AMR) | Credit: Locus Robotics [6]

The AMRs are designated to work in more demanding and challenging environment with more flexibility and adaptability to change. For example, they can move around a warehouse by selecting the most efficient path and by interacting with other robots and humans efficiently and in same time avoiding all obstacles, then rejoining the desired path safely [4]. The AMRs are becoming more and more popular in different applications either for indoor or outdoor environments [Fig.4]; they are becoming an essential factor in the current industrial and mobility transformation.

Fig.4 Honda Electric Autonomous Work Vehicle (AWV) for Material Transportation [7]

Comparison AMR vs AGV

As stated before AGV rely on a simple control structure. It relies mostly on a conventional control feedback-loop [Fig.5]. The controller is of classical types PID, State Flow… It generates a manipulated variable and it is applied on the robot, then the controlled variable (output of the plant) is been feedback to the controller where in its turn allow the system to follow the setpoint/target value been set initially which can be the target destination location.

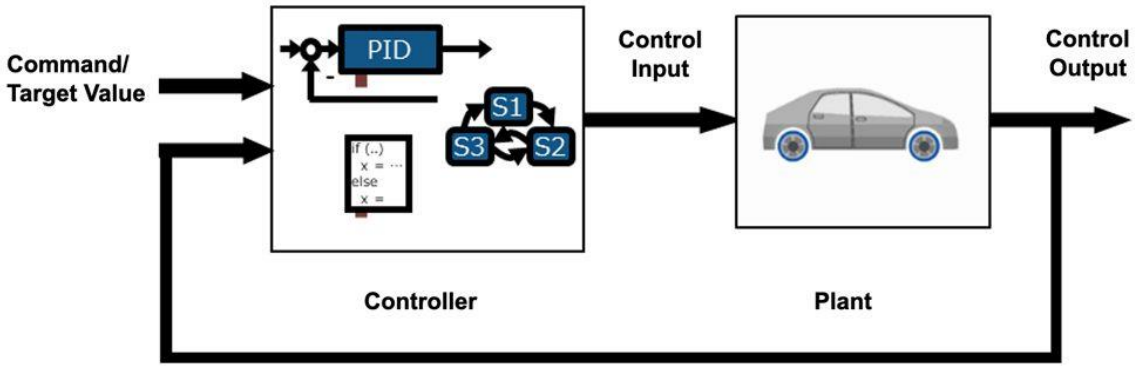


Fig.5 Conventional Control Feedback-Loop [8]

To achieve the complex and challenging tasks required from an AMR in comparison with AGV. The autonomous mobile robots rely on more complex functional architecture and control structure. The Generic functional architecture [Fig.6] incorporate fusion of different types of sensors (Camera, Lidar, Radar, ultrasonic, GPS, IMU,..) in order to perceive the environment and localize the robot precisely.

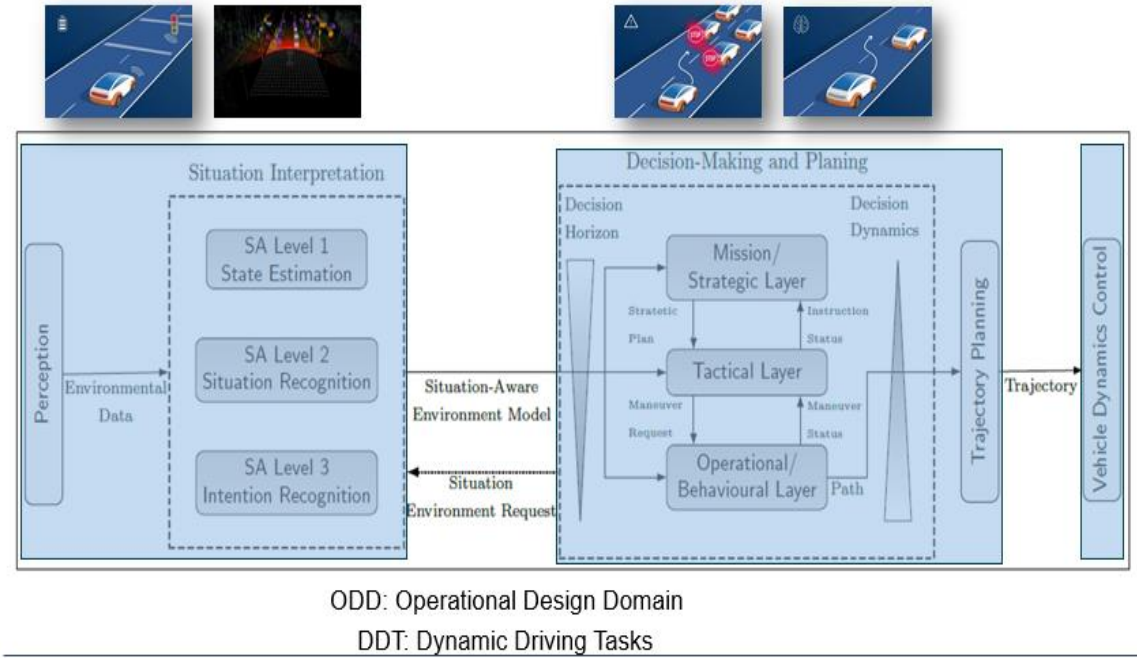


Fig.6 AD Generic Functional Architecture [9]

Based on the environmental data received from the sensory setup, an environmental model is being generated, by mean of situation interpretation, in order to be used later on by the decision making layer on request. After that, a valid trajectory is being generated and the vehicle dynamics parameters of the vehicle will be controlled accordingly [Fig.6]. By adopting this kind of functional architectures, the mobile robot can navigate autonomously from location A to location B by selecting the most efficient path and by avoiding all statics and dynamics obstacles that may be encountered during its route. The AMRs as stated can operate with more flexibility and adaptability to change of behavior and environments. Moreover, AMRs can rely on more advanced controller than a classical PID such as a model predictive control (MPC). It is important to mention that the autonomous mobile robots depend heavily on the operational design domain (ODD), in other terms where the robot will operate if in indoor, outdoor, unstructured, crowded environments, or on public roads… Also AMRs depend on another factor, known as the dynamic driving tasks (DDT) such as accelerating, decelerating, yielding, driving at constant speed, following another vehicle, overtake, avoiding an obstacle and rejoin the desired path…

Both ODD and DDT can highly affect the development of the functional architecture [Fig.6] and the algorithm design, where the latter can be simplified a lot or become more complex depending on the situation as discussed.

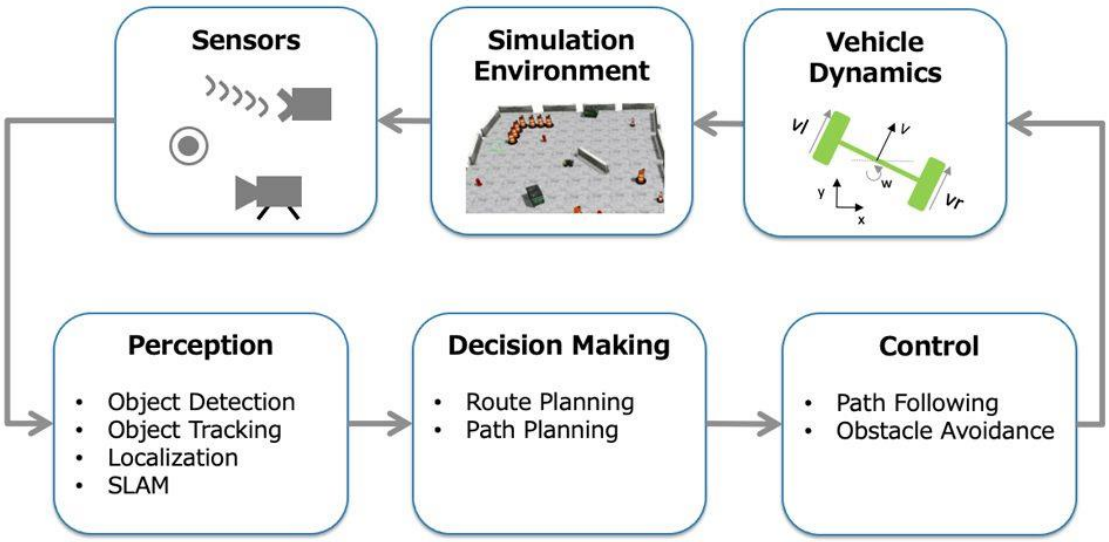


Fig.7 Advanced Functional Blocks Architecture Loop [8]

The generic functional architecture presented in [Fig.6] can be visualized more clearly as a simplified closed loop in [Fig.7]. The different functional blocks are connected in series from perception, decision making, control, vehicle dynamics to implementation either in virtual simulation environment or on a real vehicle.

PROJECT DESCRIPTION

The main aim from this project is to prepare a mobile robot and conduct a complete commissioning in order to make it ready for an autonomous logistic solution for operation in an unstructured outdoor environment. The mobile robot in use is a Pioneer 3dx using a differential drive type. The development of the functional architecture algorithm was done within Matlab/Simulink and ROS. In this report, the example of food delivery such as Pizza in outdoor environment will be as use-case. The robot can be adapted to transport any types of goods or materials.

Fig.8 Pioneer 3-DX Robot with Delivery Thermo-Box | Credit: Pioneer Robot [10]

- The Robot is designated for an autonomous delivery System.
- The User will send his/her request through a mobile application.
- The Robot will receive the request and navigate autonomously toward the delivery shop / pickup point.
- The employee will insert the requested order in the robot's thermo-box.
- The robot will return back to the customer to hand in his/her order.
- The Robot should avoid autonomously all static and dynamic obstacles through its trajectory.
- ADAS functionalities will be implemented on the robot.

USER REQUIREMENTS & TECHNICAL SPECIFICATIONS

The project was initiated by collecting the user wishes and requirements, then translating them into technical specifications. The complete set of user requirements and technical specifications can be found on the following link:

- https://padlet.com/francisjoe7711/t1wgu35ewiinmma6

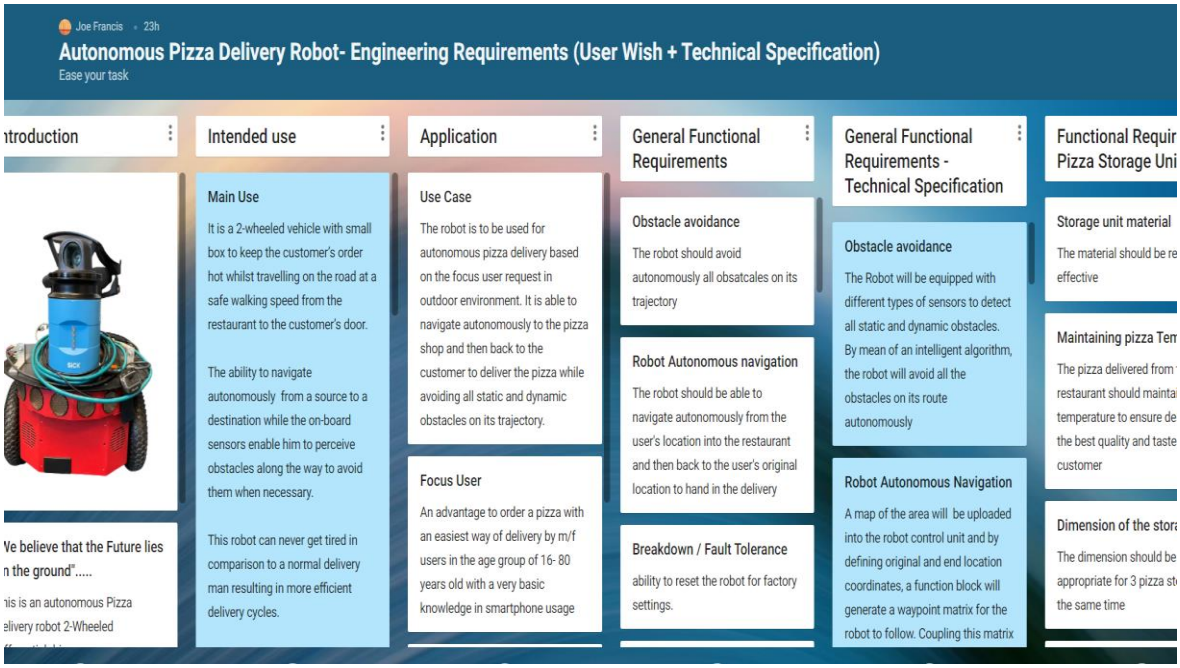


Fig.9 Caption Part of the Engineering Requirements (User Wish & Technical Specifications) Developed in Details on Padlet

MORPHOLOGICAL CHART

After setting the user requirements and technical specifications, a morphological chart was been developed for investigating different potential solutions for each functional module, in order to combine different selected solutions and obtain finally three main different concepts.

The detailed morphological chart, solution concepts and components comparison such as sensors ,driving types… can be found on the following link:

- https://padlet.com/francisjoe7711/3tni9wnnh0dcm7qo

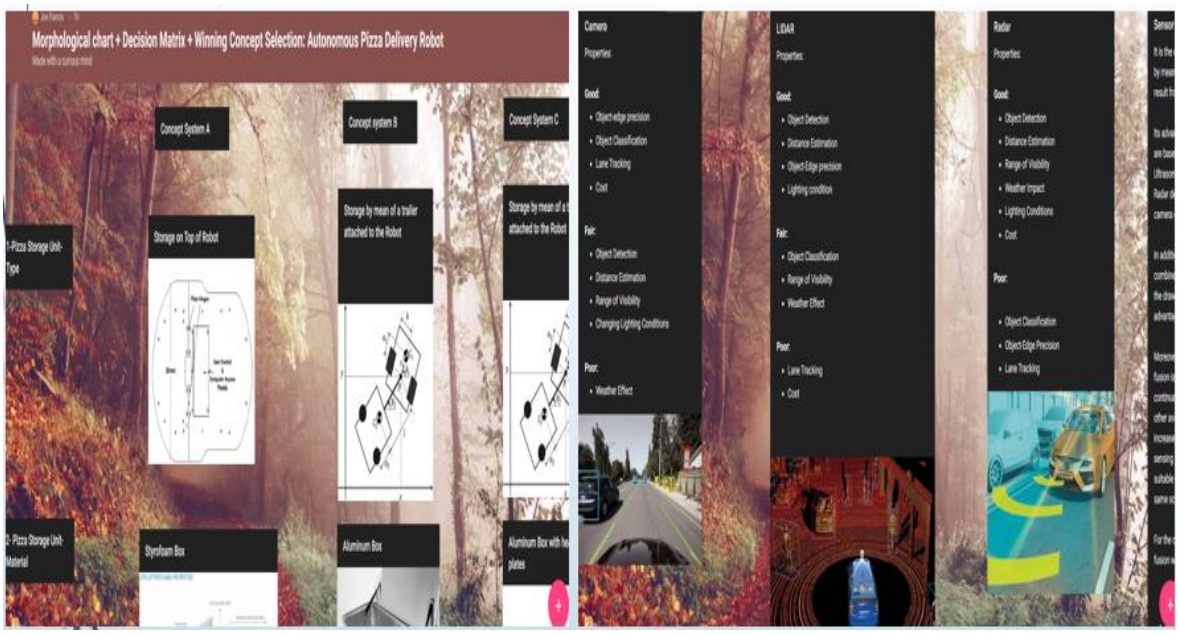


Fig.10 Caption Part of the Morphological Chart Developed in Details on Padlet

After combining different solutions of different functional modules, three solution concepts were obtained. Concept system A, B and C are shown in details on the Padlet link provided above. In the end, one solution concept will be the winning one after the decision matrix.
The sensor comparison, done in the Morphological chart [Fig.10], between Camera, Lidar and Radar was retrieved from [11].

DECISION MATRIX & PAIRWISE COMPARISON

The full excel spreadsheet of the decision matrix & Pairwise comparison with the selected winning concept for design are shown in details on the same Padlet link:

- https://padlet.com/francisjoe7711/3tni9wnnh0dcm7qo

	Weighing %	Criteria	Taste	Cost	Comfort	Reliability	Robustness	Effectiveness	Result	Result %	
	36%	Taste		5	5	3	5	3	21	20/90 = 23.33%	5- Hot like in restaurant/ 3- Med
	16%	cost	1		3	1	5	3	13	14.44%	5- Low / 3- Medium / 1- High
	12%	comfortable operation	1	3		1	5	1	11	12.22%	5-Easy operation from the phon
	12%	Reliability	3	5	5		5	5	23	25.56%	5- No faults or maintenace requ
	12%	Robustness	1	1	1	1		3	7	7.78%	(Explained Below table) 5- Rigid
	12%	Effectiveness	3	3	5	1	3		15	16.67%	5- Complete the full day cycle w
Total	100%								90	100%	

Pairwise Comparison		Points system	Meaning
		5	Criteria X more important than Y
		3	Criteria X is same importance as Y
		1	Criteria X is less importance than Y
Comparing X Horizontal to Y Vertical criteria			

	Taste :	Maintaining Pizza Temperature and Condition
Robustness:	5- Rigid overall construction and can tackle any terrain / 3- Some parts are constructed with less rigidity and can tackle some specified terrains/ 1- Major parts are Less rigid and the robot can tackle only very fe	

Fig.11 Caption Part of the Decision Matrix Developed in Details as an Excel Spreadsheet in Padlet

The decision matrix and pairwise comparison shown in [Fig.11], were developed based on different criterias such as maintaining the taste of delivered food, cost of the concept, comfortable operation, Reliability, Robustness, effectiveness with different weighing according to their importance by priority.

After evaluating the final results of each concept, the concept A was the winning concept for design. The details of solution concept A are shown on the Padlet of the morphological chart.

MAIN TECHNICAL SPECIFICATIONS OF THE ROBOT

As stated previously the Pioneer 3-DX robot is used in this project. It is a two-wheeled robot, relying on differential drive type. The robot is well-known by its reliability, robustness and it is widely used by the educational institutions and laboratories. In the following, some standard properties of the robot will be presented as per the manufacturer without any modifications.

1.1 Physical Dimensions

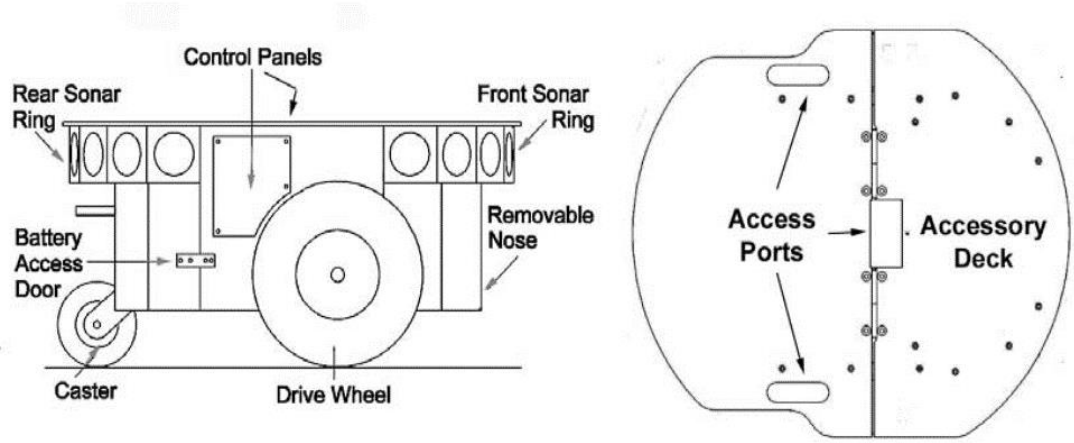


Fig.12 Pioneer 3-DX Robot's Layout [10]

The robot, shown in [Fig. 12], has a size of 445mm x 393mm and was made from tough Aluminum body. It weighs only 9Kg and can carry a load up to 25Kg [10].

1.2 Power Supply (Batteries)

The robot is equipped with three 7Ah (21Ah in total) lead acid batteries delivering 12V DC voltage with a total energy of 252 Wh [Fig.13]. The Pioneer 3-DX can run continuously for 8-10 hours and up to 4 hours with onboard computer. It can takes around 12 hours for a full standard recharge and only 2.5 hours with the 4A increased high capacity charger [10].

Fig.13 Batteries Storage Compartment

1.3 Perception Setup

In this section, only the Sonar sensor system will be introduced as it is originally equipped on the robot. The Pioneer can support up to four Sonar arrays with a total of sixteen transducers [Fig.14]. The robot, in use, holds only eight transducers at the front, which are separated by 20^0 intervals [Fig.15]. The front Sonar arrangement can provide a 180^0 sensing coverage, from the most right to the most left. The Sonar setup provides object detection and range information for collision avoidance. It is important to mention that the Sonar transducers are multiplexed, only one disc per array is active at a time but all available arrays fire one transducer simultaneously. The Sonar ranging acquisition rate is adjustable and it is normally set to 25 Hz (40 ms per transducer per array). The equipped sonar sensors have a sensitivity range from 0.1 m to over 4 m (depending on the ranging rate) [10].

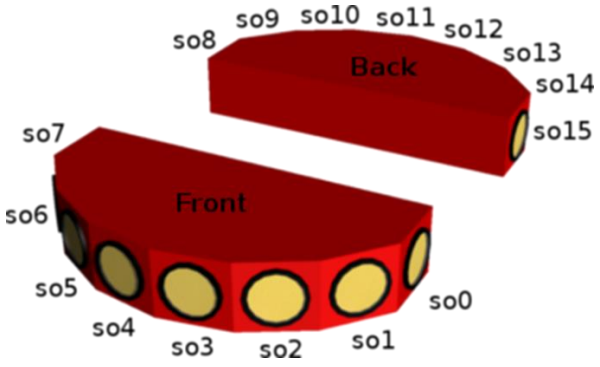


Fig.14 Front and Back Sonar Arrays [10]

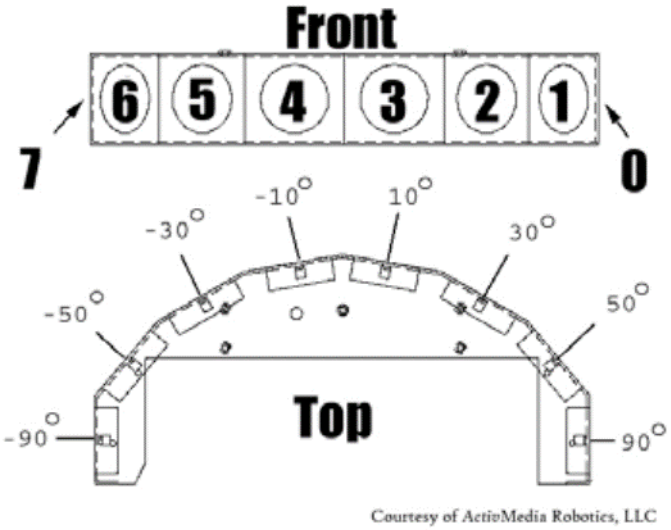


Fig.15 Front Sonar Arrangement [10]

1.4 Driving Unit

The Pioneer 3-DX is equipped by two foam-filled wheels with knobby treads having a diameter of 195.3 mm and 47.4 mm as width [10]. The steering of the robot is achieved by differential type kinematic. The driving unit is composed of two high-torque reversible 12V DC gear-motors, one on each side with a gear ratio of 38.3:1 [Fig.16]. Moreover, the motors are equipped with 500 counts high-resolution optical quadrature shaft encoders for precise position and speed sensing. They can achieve 19150 counts per wheel revolution, 128 counts per mm of linear distance travelled by the robot and 33140 counts per 360^0 rotation. The robot can achieve a maximum linear speed of 1.4 m/s and rotational speed of 300 deg/s. The pioneer 3-DX can overcome a step of 20 mm and a gap of 89 mm. It can drive over a slope with a maximum grade of 25% and it is suitable for the wheel-chair accessible paths [10].

Fig.16 Inner View of the Robot Showing One of the 12V DC Gearmotor

The integrated motor unit shown in [Fig.16], is composed of the gearbox (on the left) attached to the DC motor (in the middle) and by its turn connected to the optical encoder (on the right).

1.5 Embedded Controller

The P3-DX robot is equipped originally with an embedded SH-2 microcontroller [Fig.18]. It comes with ARCOS firmware and supported by the ARIA library built originally in C++ language. This library can be used to develop algorithms either fully in C++ (or Python with less flexibility) in Microsoft Visual Studio for example, or it can be used within ROS (Robotic Operating System) middleware using the node RosAria.

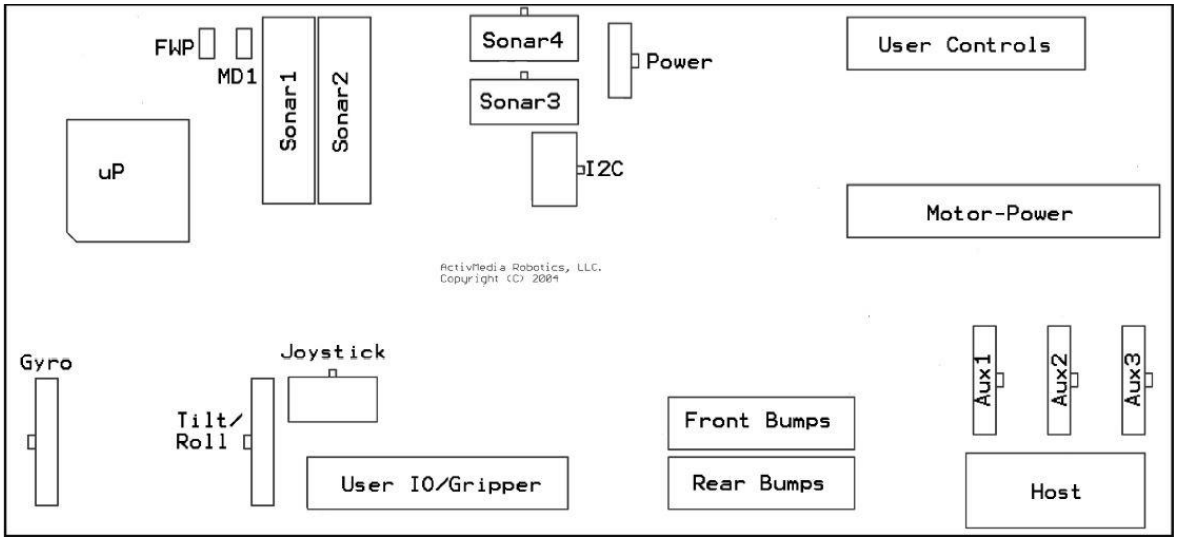


Fig.17 SH-2 Microcontroller with Connector Locations Equipped Originally on P3-DX [10]

It is important to mention that the SH-2 microcontroller cannot work independently and operate the robot. It needs a constant communication connection through the host port [Fig.17] with a master computer that can be a laptop/PC or any other mini-computer board, which can run the main algorithm. The SH-2 is considered more, as a low-level microcontroller that receive the data from sensors such as Sonar and send action commands to the actuators such as driving motors. The SH-2 is mainly manipulated by the master computer device as it is a master-slave communication type protocol.

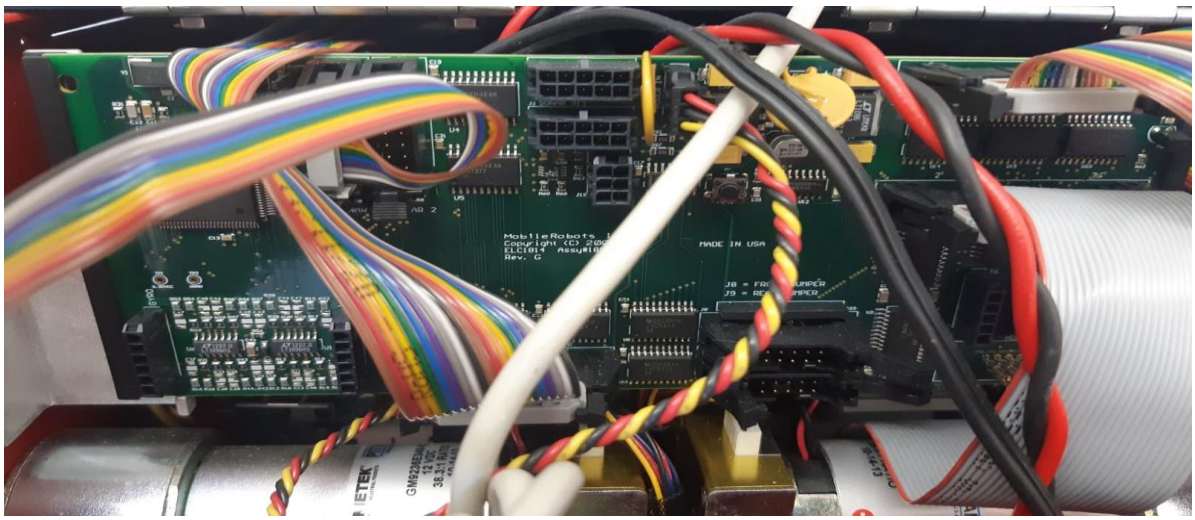


Fig.18 SH-2 Microcontroller Board Installed in the Robot's Frontal Inner Compartment

ADDITIONAL HARDWARE EQUIPPED ON THE ROBOT

In this section, the additional components installed on the robot will be presented. These components are not equipped originally on the robot; they were added additionally in the framework of the commissioning, to make the robot more suitable with challenging autonomous applications.

2.1 Master Single On-Board Computer

As discussed in the previous section, the P3-DX robot needs a master computer to operate. The decision was made to use the Nvidia Jetson Nano as a single on-board master computer on the robot [Fig.19]. It can empower the robot and make it more suitable for the complex autonomous functionalities. The Jetson Nano board cannot only run classical autonomous functionalities, but it is highly capable and efficient to its compact size to run AI algorithms. The board has a 4GB of RAM 64-bit LPDDR4 at 25.6 gigabytes/second, quadcore ARM® A57 CPU and 128-core NVIDIA Maxwell™ architecture-based GPU [12]. It has a dedicated slot to insert an external memory card that contains the image of the operating system and acts as a storage memory. The Jetson board operates on Linux system with Ubuntu distribution. This development board is special because it combines CPU and GPU capabilities in one compact board, where the latter is essential in running computer vision and AI algorithms that become crucial in modern autonomous robotics applications. In summary, it is a computational efficient single board computer at an affordable price below 100 Euro [12].

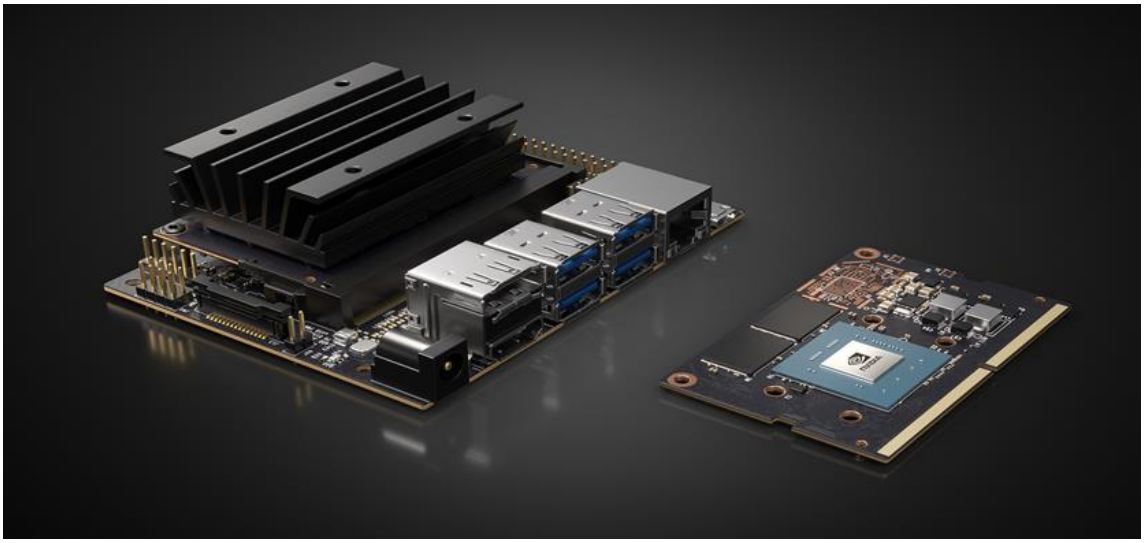


Fig.19 NVIDIA Jetson Nano Development Kit [12]

As discussed, the Jetson Nano board [Fig.19] will be the master computer on the P3-DX robot. It will run the main algorithm, the generic three-layer autonomous functional architecture shown in [Fig.6] and [Fig.7].

2.2 Intel RealSense Camera & IMU

In addition to Sonar sensors in the perception setup, an Intel RealSense Camera was equipped additionally on the robot. The camera plays an important role in the perception setup as it is considered somehow the eye of the robot and helps to reinforce the environmental data and by its turn help in building a better environmental model. Moreover, the camera play a pivotal role in a sensor fusion algorithm relying on different sensory setup, where the drawback of one sensor type can be compensated by the advantage of another sensory type. Also, setting the priority of which type of data to receive and its frequency, make the perception functionality more efficient. The use of the camera with a computer vision algorithm by profiting from the power of the NVIDIA board introduced in the previous section, has several advantages on other types of sensors such as good object classification, object-edge precision, lane tracking… The choice was on the Intel Real Sense D435i depth stereo camera shown in [Fig.20], is due to its reliability and advanced functionality usage in many robotics applications and especially in mobile robots. It is very efficient in outdoor environment, which is the main domain of operation of the project. It is highly robust against several light conditions effects as it contains some advanced features that can be used easily in the algorithm for automatic filtering. Some of them are decimation filter, threshold filter, temporal filter, disparity…

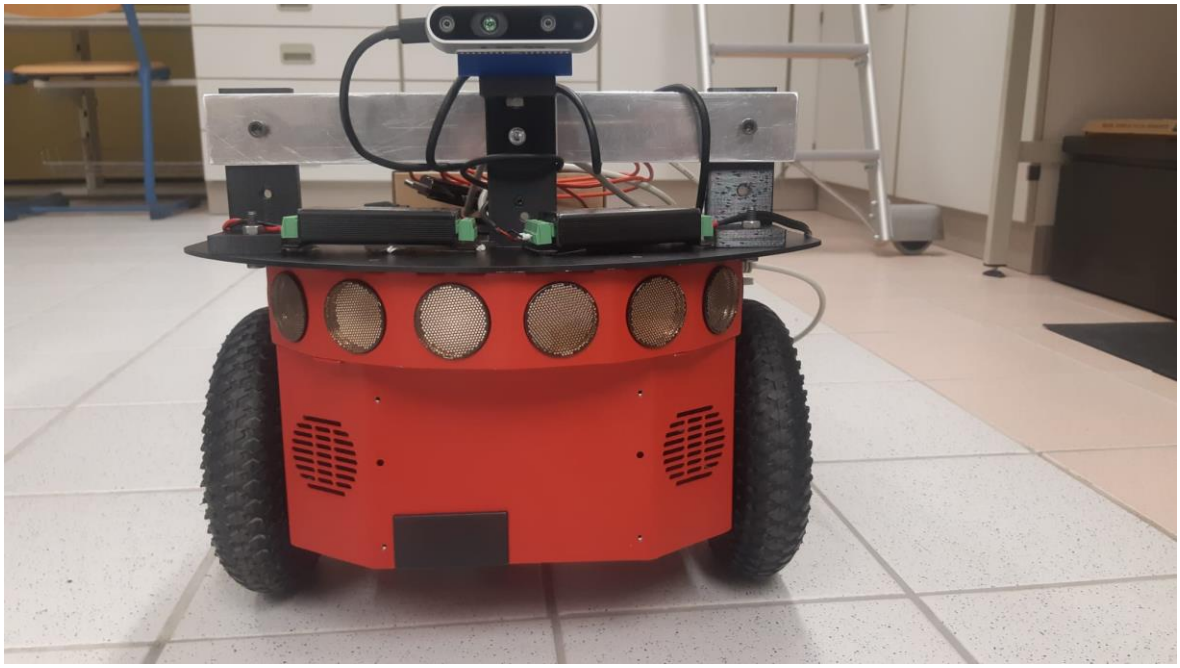


Fig.20 Intel RealSense D435i Camera Equipped on the Top of the P3-DX Robot

The advantage of using this camera in the project is for its compatibility with ROS middleware that is been used in the development of the overall algorithms, where several ROS topics are available for use. More information on ROS in the coming chapters.

It is important to mention that the Intel RealSense D435i is composed mainly of the stereo depth module and the vision processor [Fig. 21], both integrated in one compact device.

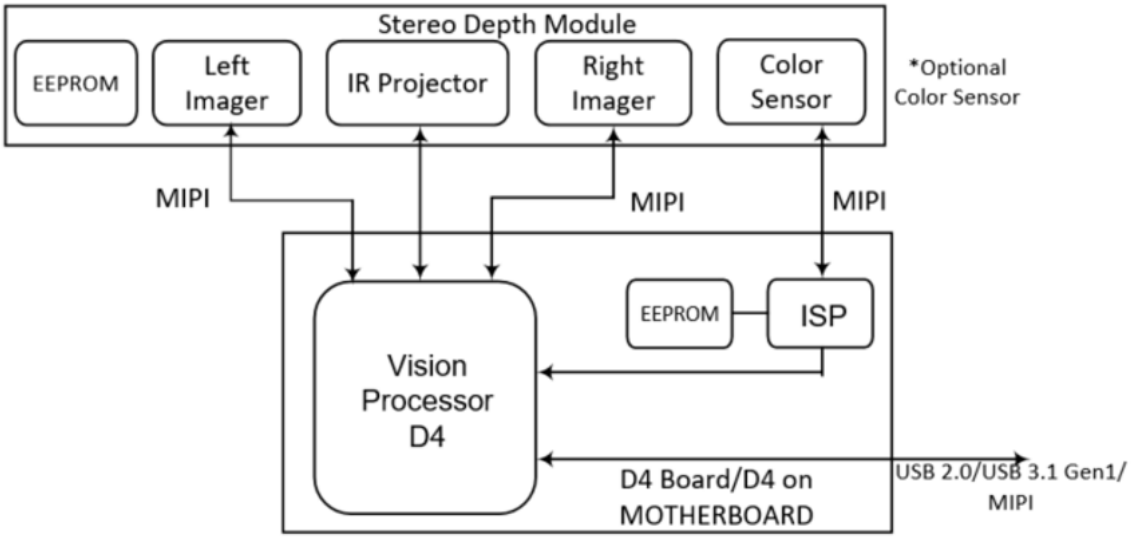


Fig. 21 Intel RealSense System Block Diagram [13]

From [Fig. 21], the stereo depth module is composed of the color sensor to collect the RGB image and from two imager to receive the depth information. Moreover, there is an IR projector that can improve the accuracy of the data for depth information and especially in low light conditions. All the information are forwarded to the vision processor unit that send the processed data to the connected device (PC) in order for the data to be used in the algorithm formation or been simply visualized on the Intel RealSense viewer application. Moreover, the D435i camera has an in-built inertia measurement unit (IMU) with six degree of freedom measurement capability. It is composed of 3-axis accelerometer and 3-axis Gyroscope, with an acceleration range of ±4g and gyroscope range of ±1000 deg/s [13]. The integrated IMU is very important not only for camera stabilization issues in computer vision algorithms, it is important to be used in the localization of the mobile robot and been merged in a fusion algorithm with the odometry data of the wheel encoders for precise localization, as the robot tend to drift over time when relying only on wheel encoders data. Thus, the integrated IMU has many benefits and allow neglecting the usage of a separate unit on the robot, which lead to space saving, less wiring connections and less ports usage.

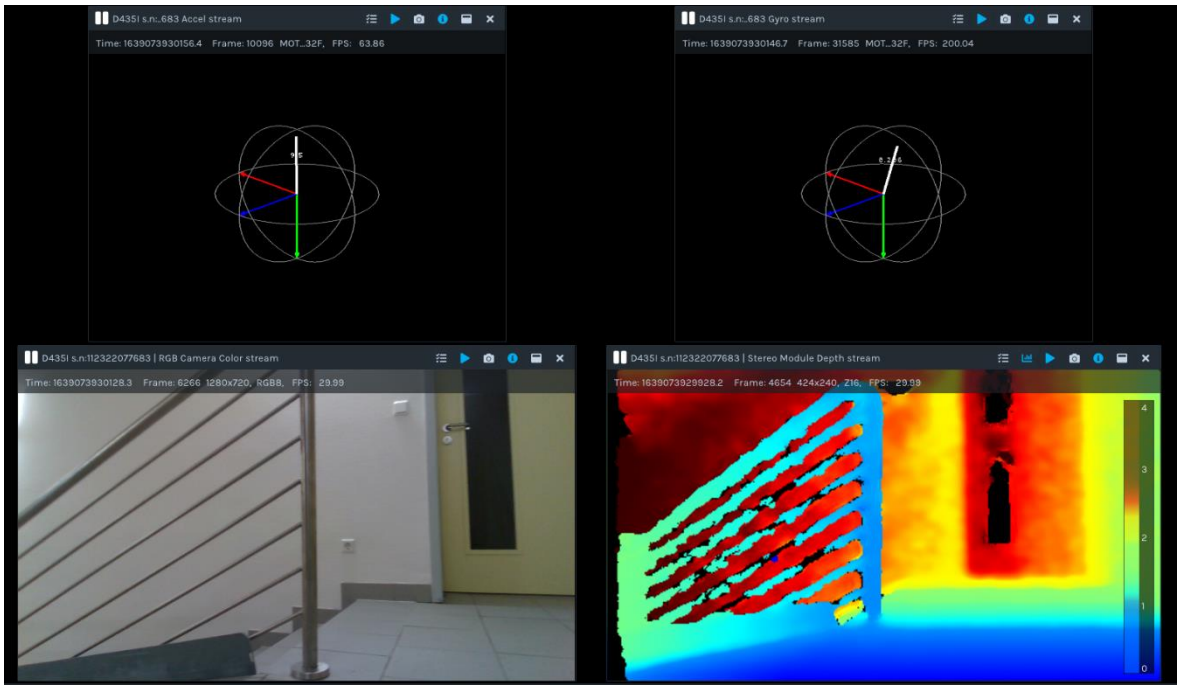


Fig.22 Output of the D435i Camera mounted on the Robot inside Building D at HSM

From the Intel RealSense D435i camera, four main data types can be extracted and used further in algorithm development. The [Fig.22] shows the extracted output from the camera in Intel RealSense Viewer. At the top left side, the accelerometer data are streamed. At the top right side, the gyroscope data are streamed. At bottom left side, the normal RGB color image is displayed. At the bottom right side, the Depth stream is displayed. In that image, the depth information of each object can be known according to the color, where blue is the closest and red the farthest from the camera mounted on the robot. The depth information can range from 0.3m to 3m [13].

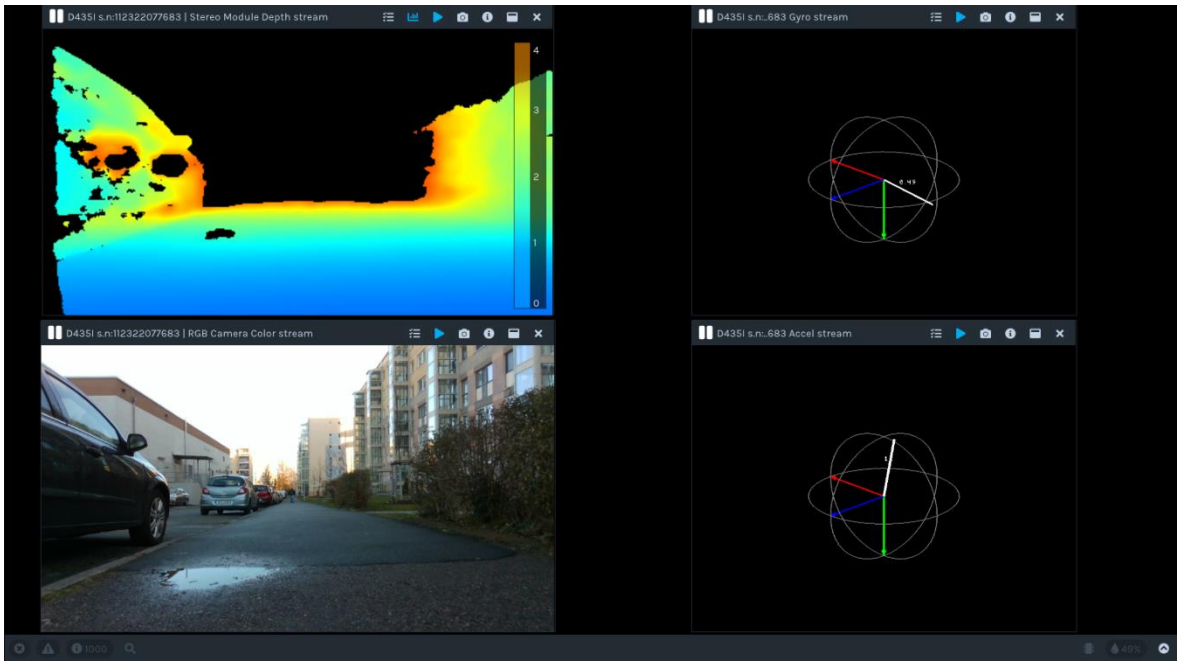


Fig.23 Output of the D435i Camera mounted on the Robot in Outdoor Environment (Gera)

Despite of the several advantages of the camera use and in particular the Intel RealSense D435i in many mobile robotics applications, there is some drawbacks for the camera usage in general. Two major problems were been experienced while achieving the commissioning and testing on the P3-DX robot. The first problem is that the camera is not able to detect the glass as an object. From [Fig.22], it can be noticed that the glass in the door is not being detected and the glass is displayed as black in the depth image. This is a major problem as the robot can crash into big glass doors or facades if only relying on the camera for perception, from here the usage of an additional sensor that relies on a different physical principal is essential. In this case, the Sonar sensors are been used with the camera in the perception block. The second problem resulting from the camera use, is that water spot on grounds cannot be detected. As seen in [Fig. 23], that the water spot on the road appearing in the RGB image, it appears as black spot in the depth image. This issue, can cause a problem in operation as the robot can slip due to aquaplaning or can get damaged by water splashes in big water spots. It should be taken in consideration while designing the algorithm.

2.3 HMI Display

Fig.24 Waveshare 11.6 Inch LCD Display Connected to NVIDIA Jetson Nano [14]

To allow monitoring and interfacing with the robot, an 11.6-inch LCD capacitive touch display from company Waveshare is used as an HMI for the robot's system [Fig.24]. The choice of this particular display monitor is due to the fact that is fully functional with touch ability with the NVIDIA Jetson board. It has a resolution of 1920x1080 accessible through the HDMI port with IPS toughened glass display. It has a viewing angle of 178^0. To enable the touch functionality, the monitor should connected by standard USB cable to the Jetson board. The display is powered by a 12V DC source. The Waveshare display has two in-built Hi-Fi speakers with 3.5 mm jack for audio output. It is equipped at the back with M4 mounting holes for 75x75mm wall mount. Additionally, it has back buttons for several commands and settings [14].

2.4 Wireless Keyboard

For efficient teleoperation control, the robot was equipped with a mini wireless keyboard with an integrated mouth touchpad working via Bluetooth communication protocol with the onboard computer. By this keyboard, the robot can be driven manually as desired, select features or stop the robot in case of emergency. The keyboard in use is Rapoo E2700 [Fig.25], it is operating on 2.4 GHz wireless band [15], it has a working range up to 10 meters from the Nano USB receiver plugged to the computer on the robot.

Fig.25 Rapoo E2700 Wireless Keyboard [15]

2.5 DC-DC Converter

The robot is equipped with two QSKJ DC-DC converter [Fig.26], They are very useful in terms of providing adjustable regulated voltage for different additional accessories that will be mounted on the robot. The converter in use, can accept an input voltage range between 6V to 32V and can deliver an output voltage between 0.8V to 28V. The desired output voltage can be adjusted precisely by rotating an upper screw on the case. Moreover, the converter can provide a maximum current of 15A and maximum power of 150W. It has eight input and output terminals, maximum allowed current per terminal is 6A and multiple ports in parallel should be used when larger current [16].

Fig.26 QSKJ DC-DC Converter [16]

COMMUNICATION FLOW DIAGRAM

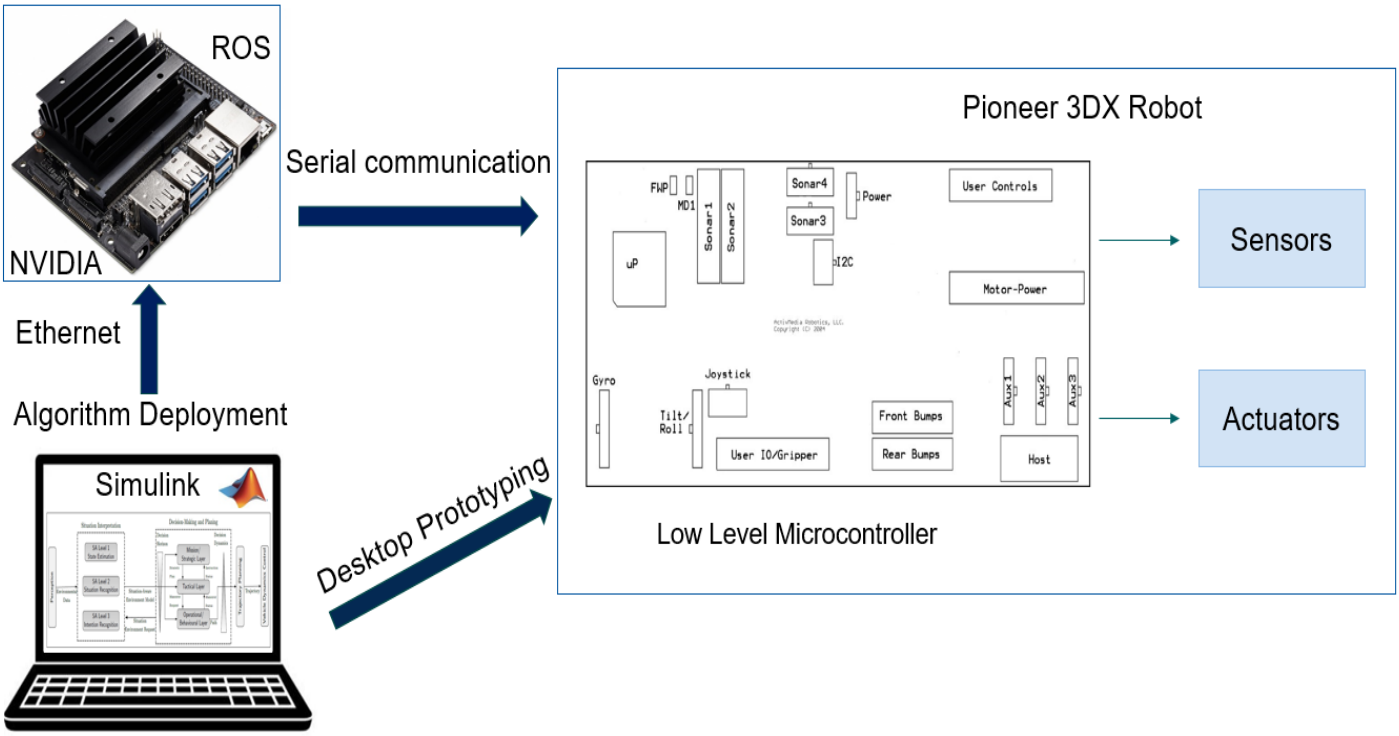


Fig.27 Overall Communication Diagram for Algorithm Deployment on the Robot

As shown in [Fig.27], there is one master computer which is the NVIDIA Jetson board to run the main algorithm. By its turn, the master computer is connected by serial communication, using a RS-232 to USB converter connector [Fig.28], to the low-level microcontroller (SH-2 disscussed in section 1.5) located inside the P3-DX robot. The SH-2 low level microcontroller will control the motor power board and calculate accordingly the left and right wheel velocities as per the main algorithm deployed on the master computer. Additionally the SH-2 microcontroller will receive the data signals from the Sonar sensors and transfer them to the NVIDIA board. Moreover, additional specific sensors, that are compatible with the P3-DX, can be equipped and connected directly to their specified slots on the SH-2 microcontroller; such as Gyro, Tilt/roll sensors, additional Sonar at the back, front and rear bumps that can detect if the robot touch an object. This microcontroller has an internal host port, which can be used for connecting directly to the master computer, rather than using the external RS- 232 port on the side panel of the robot. As seen in [Fig.27], a laptop/PC can be used for rapid desktop prototyping, which will be useful in algorithm design phase and testing in order to directly monitor and tune parameters on the spot. This PC running the Simulink model, can be connected directly to the robot's SH-2 microcontroller, using the same serial communication protocol discussed above. In this case, Aria library must be installed and compiled on this laptop/PC and having ROS middleware installed on the same laptop/PC by using a virtual machine with Linux OS-

Ubuntu distribution. Another way to use the rapid desktop prototyping method, is to connect the laptop/PC to the NVIDIA board by Ethernet communication protocol by using the standard RJ-45 connector cable [Fig.29]. In this case, only the Simulink model will run on the laptop/PC, ROS-middleware and Aria library are installed on the NVIDIA Jetson board. By using the second method, there is benefit from the computational efficiency of the Jetson board and it is in the same time a Hardware-in the loop testing (HIL) for the NVIDIA board under the algorithm deployment. Finally, in the same way, the algorithm can be deployed as a standalone ROS node on the NVIDIA board, where the algorithm can run independently on the master computer-board without the need of the laptop/PC. Simulink, ROS and algorithm deployment will be discussed more in details later in the next sections.

Fig.28 USB to Serial Converter Adapter [17]

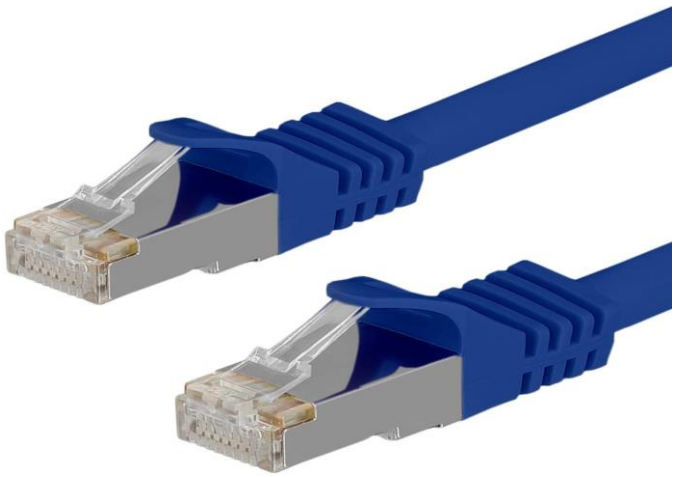


Fig.29 RJ-45 Ethernet Connector Cable [18]

DETAILED HARDWARE WIRING SCHEMATIC DIAGRAM

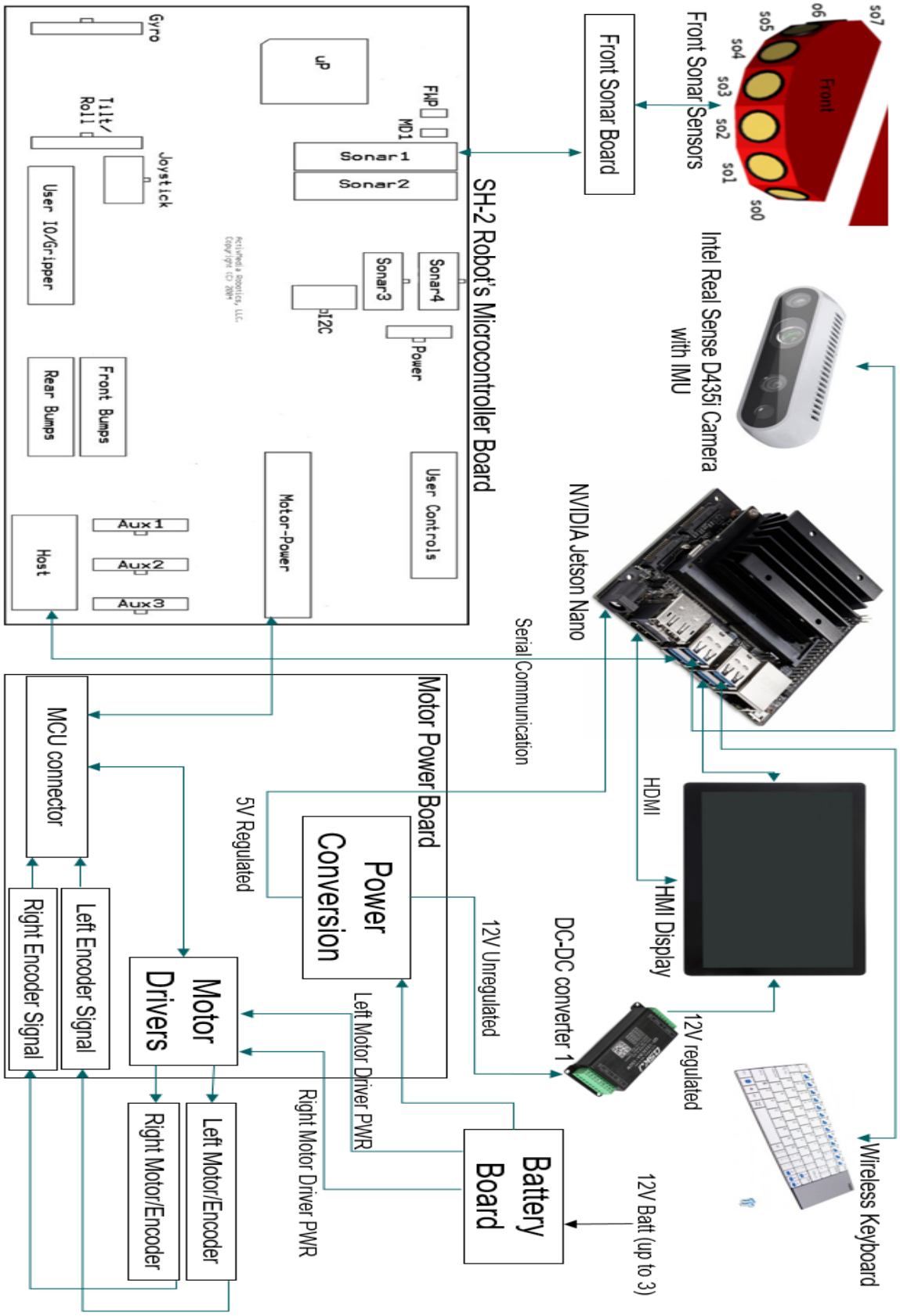


Fig.30 Complete Hardware Wiring Schematic Diagram

More details can be found in [10] about the robot's original system pre-interconnections

DIFFERENTIAL DRIVE KINEMATIC GENERAL OVERVIEW

As discussed in a previous chapter, the kinematic model of the P3-DX robot is a differential drive type [Fig.31] consisting of two driving wheels and one central rear caster.

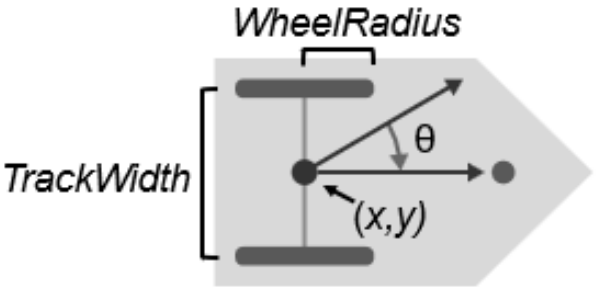


Fig.31 Differential Drive Kinematic [19]

As shown in [Fig.31], the vector $[x\ y\ \theta]^T$ is designated as the pose of the robot; where (x,y) describe the position of the robot in Cartesian coordinates and the angle theta describes the heading angle or orientation of the robot. Moreover, there are two more parameters while describing the differential drive kinematic. They are the wheel-radius and the track-width of the robot (the distance between the two driving wheel).

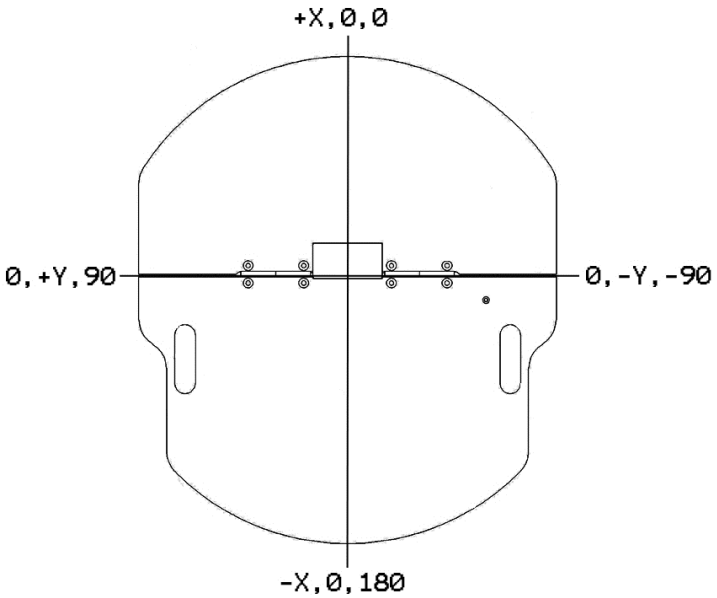


Fig.32 Internal Coordinate System of the P3-DX Robot [10]

In order to interpret more clearly the pose vector $[x\ y\ \theta]^T$ on the P3-DX robot, the internal coordinate system [Fig.32] will be used in the overall algorithm development and data interpretation. From [Fig.32], the frontal area of the robot including the Sonar system is in the direction of the coordinate (+X, 0, 0). The origin of the coordinate system (0, 0, 0) is where the axis intersect at the small opening rectangle in [Fig.32]. Using the coordinate system in [Fig.32] is

very crucial, because based on that the desired robot's movement direction can be set correctly. Moreover, the data related to pose extracted from the wheel encoders and the data extracted from the Sonar sensors are based on that coordinate frame. The Sonar system consisting of eight sensors discs, each of them gives the detected obstacle coordinates (x, y) with respect to the coordinate system [Fig.32]. It is important to notice that the detected distance by a Sonar sensor, it is not the considered distance from the sensor disc but the distance from the origin of that coordinate frame.

3.1 Differential Drive Important Equations

It is important to mention in this section, some of the important equations for the differential drive type. These equations will be used in the algorithm functional development, specifically in the vehicle dynamics functional block [Fig.6] & [Fig.7] in order to control the robot's longitudinal and lateral motions.

The state of the differential drive robot is described by $[x\ y\ \theta]^T$. Deriving the robot's state with respect to time, the robot's state_dot is obtained as $[\dot{x}\ \dot{y}\ \dot{\theta}]^T$ with: [20]

$$\dot{x} = \frac{R}{2}\left(\dot{\phi}_r + \dot{\phi}_l\right)\cos\theta \qquad \textit{Eq. 3.1}$$

$$\dot{y} = \frac{R}{2}\left(\dot{\phi}_r + \dot{\phi}_l\right)\sin\theta \qquad \textit{Eq. 3.2}$$

$$\dot{\theta} = \frac{R}{d}\left(\dot{\phi}_r - \dot{\phi}_l\right) \qquad \textit{Eq. 3.3}$$

$\dot{x}$	Linear velocity in x	[m s-1]
$\dot{y}$	Linear velocity in y	[m s-1]
$\dot{\theta}$	Heading rate	[rad s-1]
θ	Heading angle	[rad]
$\dot{\phi}_r$	Right ang.wheel speed	[rad s-1]
$\dot{\phi}_l$	Left ang.wheel speed	[rad s-1]
R	Wheel radius	[m]
d	Track width	[m]

The equations 3.1, 3.2 and 3.3 are based on the notations as per [Fig.31].

For the unicycle model [Fig.33], the following equations describe the state_dot: [20]

$$\dot{x} = V\cos\theta \qquad \textit{Eq. 3.4}$$

$$\dot{y} = V\sin\theta \qquad \textit{Eq. 3.5}$$

$$\dot{\theta} = \omega \qquad \textit{Eq. 3.6}$$

$\dot{x}$	Linear velocity in x	[m s-1]
$\dot{y}$	Linear velocity in y	[m s-1]
$\dot{\theta}$	Heading rate	[rad s-1]
θ	Heading angle	[rad]
V	Linear forward speed	[m s-1]
ω	Heading rate	[rad s-1]

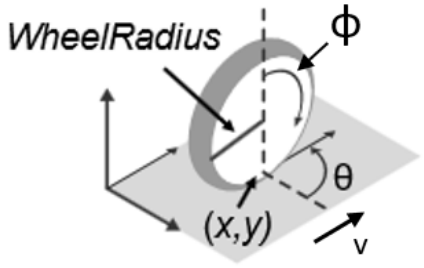


Fig.33 Unicycle Model [21]

Dividing Eq. 3.1 by 3.4 and Eq. 3.3 by 3.6 then after rearranging, the following equations are obtained for differential drive type:

$$\dot{\phi}_r = \frac{V + \omega\,\frac{d}{2}}{R} \qquad \textit{Eq. 3.7}$$

$$\dot{\phi}_l = \frac{V - \omega\,\frac{d}{2}}{R} \qquad \textit{Eq. 3.8}$$

$\dot{\phi}_r$	Right ang.wheel speed	[rad s-1]
$\dot{\phi}_l$	Left ang.wheel speed	[rad s-1]
V	Linear forward speed	[m s-1]
ω	Heading rate	[rad s-1]
R	Wheel Radius	[m]
d	Track width	[m]

The Eq. 3.7 and Eq. 3.8 are the main equations used to describe a differential drive type robot [Fig.31] , they will be used in the Simulink and ROS algorithm functional development; where V and ω (known also as Yaw rate, rotation around vertical z-axis) are the main parameters for

longitudinal and lateral robot's motion control. According to the designed algorithm and after a valid trajectory is generated, V and ω are calculated in the master on-board computer and forwarded as bus signal to the low-level robot's microcontroller. The latter will calculate the desired right and left angular wheel speeds according to the equations 3.7, 3.8 and to the robot parameters properties.

3.2 Differential Drive Robot Motion Behavior

Considering $\dot{\phi}r$ and $\dot{\phi}l$ are the right and left angular wheel velocities respectively, the robot's motions can be summarized as follow:

I- If $\dot{\phi}r = \dot{\phi}l$ and both are positive values => The robot will move straight forward

II- If $\dot{\phi}r = \dot{\phi}l$ and both are negative values => The robot will move straight backward

III- If $\dot{\phi}r = -\dot{\phi}l$ => The robot will turn to the left at place

IV- If $\dot{\phi}l = -\dot{\phi}r$ => The robot will turn to the right at place

V- If $\dot{\phi}r > \dot{\phi}l$ and both are positive values => The robot will follow a curvature to the left

VI- If $\dot{\phi}l > \dot{\phi}r$ and both are positive values => The robot will follow a curvature to the right

The desired robot's motion is achieved, by computing the corresponding angular wheel velocities in the low-level SH-2 Microcontroller according to equations 3.7, 3.8 and after the computed V and ω are received from the master computer.

3.3 Pose Computation from Wheel Encoders

The wheel encoders as discussed before, are an easy source from where the robot's pose three-elements vector can be extracted. This basic method is widely used in robotics and it is known also by wheel odometry. The following equations will summarize how a pose vector $[x\ y\ \theta]^T$ can be extracted from the wheel encoders in order to be used in robot's localization functional block:

Assuming:

- D_c is the arc distance travelled by the center of the robot (origin of the internal coordinate frame [Fig.32])
- D_r is the arc distance travelled by the right wheel of the robot and measured by the right wheel encoder
- D_l is the arc distance travelled by the left wheel of the robot and measured by the left wheel encoder

According to [22], the following equation is used to compute the distance travelled either by the left or right wheel by counting the number of ticks of the appropriate wheel encoder:

$$D_{r,l} = 2\pi R \ \frac{\Delta\, ticks_{r,l}}{N_{r,l}} \qquad \textit{Eq. 3.9}$$

$D_{r,l}$	Right or left arc distance travelled by the left or right wheel	[m]
$N_{r,l}$	Right or left total number of ticks counts per 1 revolution	[counts/rev]
R	Wheel radius	[m]
$\Delta\, ticks_{r,l}$	Right or left ticks counts difference between new and old counts	[counts]

As most of the encoders count accumulated ticks from the start, $\Delta\, ticks = ticks' - ticks$ with $ticks'$ is the new current ticks count and $ticks$ is old previous ticks count. Both of them are extracted at the instant interval when the travelled distance $D_{r,l}$ is needed. By knowing D_r and D_l, the travelled distance of the center of the robot D_c is calculated as follow:

$$D_c = \frac{D_r + D_l}{2} \qquad \textit{Eq. 3.10}$$

To extract the current pose vector of the robot $[x'\ y'\ \theta']^{\mathrm{T}}$ from the wheel encoders and by knowing the previous pose vector $[x\ y\ \theta]^{\mathrm{T}}$ where at start can be $[0\ 0\ 0]^{\mathrm{T}}$ which then will be the first computed pose vector and so on, the following equations are used:

$$x' = x + D_c \cos\theta \qquad \textit{Eq. 3.11}$$

$$y' = y + D_c \sin\theta \qquad \textit{Eq. 3.12}$$

$$\theta' = \theta + \frac{D_r - D_l}{d} \qquad \textit{Eq. 3.13}$$

x'	Current x-position	[m]
y'	Current y-position	[m]
θ'	Current heading angle	[rad]
x	Previous x-position	[m]
y	Previous y-position	[m]
θ	Previous heading angle	[rad]
D_c	Robot center trav.dist	[m]
D_r	Right wheel trav.dist	[m]
D_l	Left wheel trav.dist	[m]
d	Track width	[m]

As can be noticed from this section that the current pose vector is computed relative to the previous one and relative to the origin coordinate frame, this can be problematic over long

distances. The wheel odometry tend to drift over time and it is highly dependent on several factors, some of them are the road surfaces, wheel slippage and wheel radius which can be affected by tire pressure. Thus, relying on the wheel odometry alone is not sufficient for robot localization. There is a necessity to consider fusing the wheel odometry data with other types of sensors such as IMU from which the pose vector can be extracted. Moreover, GPS and compass can be used in the fusion process for outdoor applications, as it exists several possibilities. Using a sensor fusion algorithm can insure to get a precise robot localization with a high degree of fidelity.

ALGORITHM IMPLEMENTATION & TESTING STRATEGY

The development of the robot's algorithm for the autonomous application was being developed following the functional architecture introduced in [Fig.6] and [Fig.7]. The strategy of the algorithm development is to begin with a smaller functional algorithm and then expand it as necessary according to the AD generic architecture [9]. This strategy allow to reach more reliable result step by step. The algorithm development was done mainly in Matlab/Simulink. The selection of this development environment is due to many reasons, the most important one is that Simulink is very efficient to be used by engineers especially from control background, instead of relying solely on pure programming languages such as C++ and python that need high programming skills and especially when dealing with complex autonomous algorithms. It is important to mention that C++ and python are been used also during the algorithm development, but in an integrated way within Matlab/Simulink and ROS which will be discussed later. Matlab/Simulink offer a high degree of flexibility during development, as it offers several important toolboxes, several ready functions blocks to be used and it offers the ability to create custom functions by programming language within Matlab and then merge it with the main algorithm in Simulink. Moreover, Simulink is characteised by the hardware support for several devices and development board and also by a very important feature known as automatic code generation into C, C++,… Additionally, Matlab/Simulink can be coupled easily with other platforms; one of the most important is the Robotic operating system. ROS is very beneficial to be used with Simulink, as many robotics hardware are not directly supported by Simulink such as the case of the Pioneer 3DX-robot. On the opposite side, many other hardware and development board are supported directly by Simulink such as Arduino, Raspberry Pi and Lego Mindstorms vehicles… Herewith, it is necessary to have ROS as middleware, on the NVIDIA master robot's computer, between Simulink and the P3-DX robot's hardware. ROS offers the direct implementation of the Simulink model on the physical robot or on a digital twin of the robot in a virtual environment by using a Software called Gazebo as part of ROS middleware environment, These Topics will be discussed more in details in the coming sections.

Testing Approach

In order to validate an autonoumous algorithm and prove its reliablitly, it may take a huge amount of testing on the real vehicle. It may reach around 2 billion Kilometers ~ 10^{-9} /h to prove AD system reliability for autonomous cars at SAE level 5 [11]. Even though that the algorithm for P3-DX robot in this project does not need this big amount of testing, but it is still inefficient to directly test the algorithm and apply it on the physical P3-DX robot for several reasons. Some of them are, in this project for example some delays were occurred in the commissioning phase of the robot in order to let it operate. These problems are due to some communication issues

with the robot and finding the appropriate libraries and tools because the robot is no longer supported by the manufacturer, so here some difficulties occurred in order to solve that problem. Regarding that issue, it was not efficient to wait that the robot operates in order to begin designing the algorithm and test it. Moreover, in some cases there are many persons will work on the same project and on the same robot which will be not effective for each to wait to test the algorithm on the robot. Additionally, to test the physical robot in a big variety of real-cases scenarios will be a big hustle; also if there are some bugs or errors in the algorithm, it may cause a problem on the physical P3-DX robot.

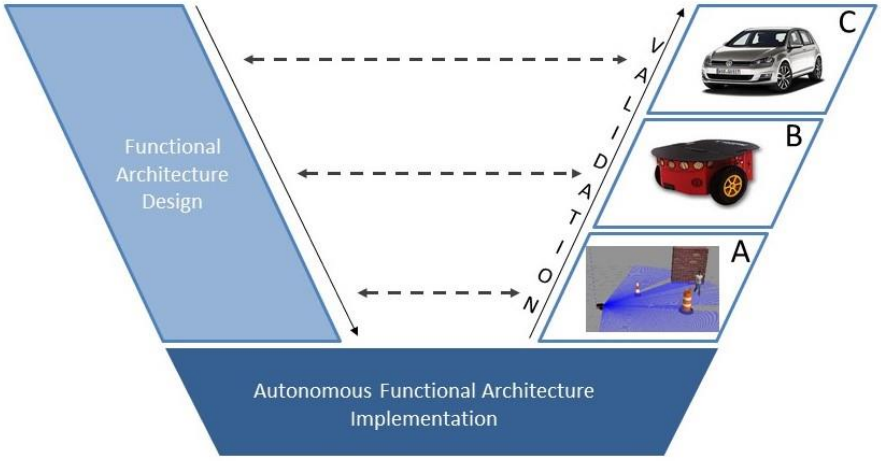


Fig.34 Three Layer Testing Approach based on the V-model

Due to all these mentioned reasons, a testing strategy must be set for a proper implementation of the algorithm. In this project, a testing approach based on the V-model [Fig.34] has been adopted. Beginning first by collecting all user requirements, technical specifications and evaluating different concept solutions as mentioned previously in the "Project Description" and the following chapter sections. Then, based on that the design of the functional architecture was launched by splitting into functional and sub-functional modules. After that, the algorithm functional architecture was implemented, where its development is conducted within Matlab/Simulink as mentioned before. In order to test the algorithm and start validating it, a three phase testing approach, shown on the right side of the V-model [Fig.34], was been adopted. It is important to mention that the third layer named C in [Fig.34] is ignored in this report as the final target vehicle of this project is the P3-DX robot shown in the second layer B [Fig.34]. The third testing layer on a full-scale vehicle will be considered in the future scope of this project. The three testing layers approach [Fig.34] consist first by testing the algorithm in a virtual environment, layer A, by using Gazebo software. In layer A, a digital twin of the real P3-DX robot was used with similar properties. This will allow a better test results and solid predictions of what would be the case on the real robot. In this first layer, the overall functional algorithm was tested from how the robot will detect the obstacles, how it will avoid them and in which way the robot

will behave to the changed parameters by tuning. The main focus in this layer was on decision making layer [Fig.6], [Fig.7] and on how a valid trajectory is generated to reach the target destination while avoiding all obstacles. In this first testing layer, when any inappropriate robot's behavior was noticed, directly a modification and tuning in the algorithm design was made and then implemented again for testing in this Layer A. It can be noticed, that the algorithm can be validated at a great extent in the first testing layer [Fig.34] before testing on the real robot. For this reason, testing the designed algorithm in a simulation environment is very crucial in the development phase, and it can save time, money and effort at a great extent. After achieving the required tests by simulation and receiving good results, the aim is to shift to the second test layer B [Fig.34]. This layer consist of real testing on the physical P3-DX robot, which will be in fact the final target vehicle in this project to achieve the task of the automated logistic delivery solution. In layer B, the overall algorithm was once again implemented on the robot, in order to observe the behavior of the robot under the applied algorithm in real world conditions and to take in consideration of some deviation that may occur in results from the previous testing simulation layer A. In this second testing layer, the focus was on the perception functional block, in order to take in account the errors in measurement from the sensory setup that cannot be noticed obviously in the simulation test. Some of the errors are related for example to the light reflections problems for the camera and the inability to detect glass facades as discussed in a previous chapter. The focus in layer B, was also on the vehicle dynamics functional block; in order to observe the behavior on the robot's longitudinal and lateral motions under different values for the velocity V and the heading rate ω. For sure this previous mentioned behavior was been monitored in simulation test, but in real tests it can be observed more clearly especially on different road surfaces conditions. For example in simulation test layer A, it was been recognized that the robot can roll over if V and ω were increased too much when the robot tries to achieve a small radius curvature. After excessive testing in layer B, the overall algorithm functional architecture will be ready to implement on this final target P3-DX robot to achieve the desired task as required by the user. The target in future is to shift for testing in the third layer C [Fig.34], on a full-scale vehicle in the aim of trying to have a close development platforms between robots and cars. In this testing layer, more selected precise and decisive tests will be conducted. It is important to mention that the full-scale vehicle in layer C can be a bigger scale mobile robot and not necessarily a car. It would be advantageous, if the same vehicle type characteristics are used in all three testing layers for more accurate and reliable testing results.

ALGORITHM DEVELOPMENT WITHIN SIMULINK

The algorithm for the P3-DX robot will be developed within Matlab-Simulink according to the sense-plan-act concept but it will be inspired more closely by the functional architecture [Fig.6] and [Fig.7]. The main target is to let the robot move from location A to location B while avoiding all obstacles. In order to start developing the desired algorithm in Simulink while knowing the main objective and the robot's type, MathWorks offers very nice documentations and solid foundation to start with. This is very important for anyone who has not any previous experience before with this development platform as well as for experienced users in saving time and effort. The decision was taken to begin with a simple Simulink model provided by Mathworks designated for controlling a differential drive robot and specifically the P3-DX robot [Fig.35]. The algorithm in intended only to let the robot move from initial settled position to a destination settled goal without any obstacle detection, avoidance or any other kinds of complex decision making algorithms. This Simulink model will all necessary explanations can be found in [23].

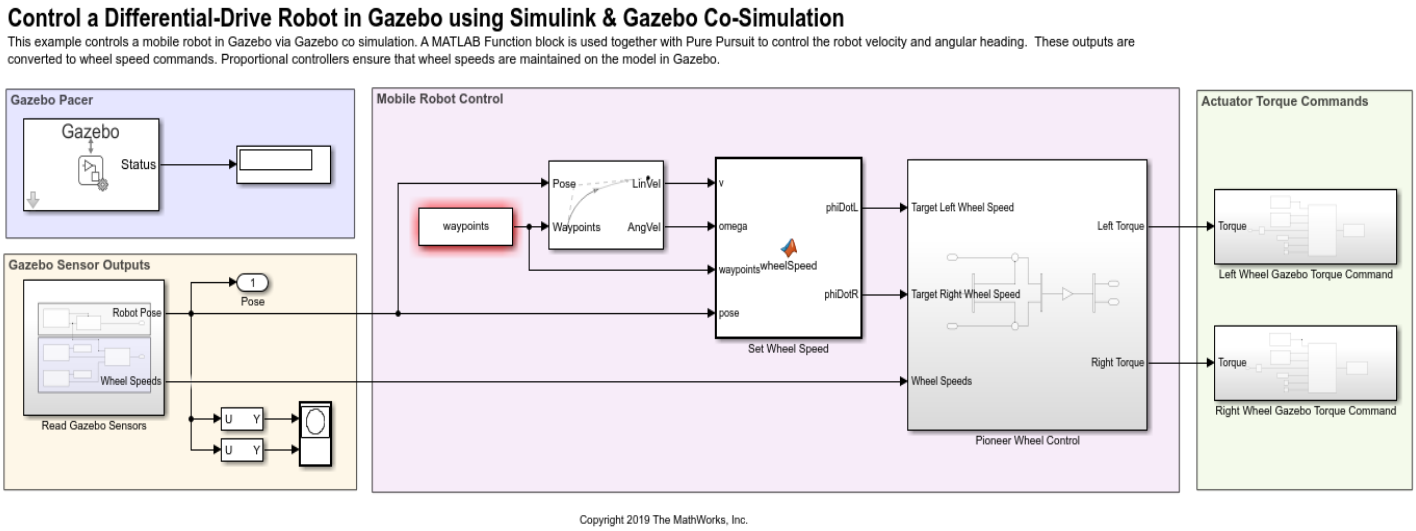


Fig.35 Simple Simulink Model for Controlling the P3-DX Robot [23]

The Simulink model [Fig.35] consists of several functional blocks. The two sub-functional modules at the two extremities, which are sensing and actuating are intended to be used with the simulation environment Gazebo as per the General functional-closed loop [Fig.7]. Coupling with Gazebo will be discussed in the next chapter. In the Mobile robot control sub-module [Fig.35], the main controller is chosen to be the pure-pursuit controller type which is very known in mobile robotics applications. It needs as inputs, the actual pose vector of the robot $[x\ y\ \theta]^T$ and a waypoints matrix shown in red outline in [Fig.35] as need to be defined in Matlab and saved in its workspace. The waypoints matrix consist of several desired (x,y) coordinates points for the robot to follow in order to generate a path from start to goal location. The Pure Pursuit controller requires setting of some more parameters such as the desired linear velocity

(m/s), maximum heading rate (rad/s) and the lookahead distance (m) [Fig.36]. Based on the mentioned required inputs and parameters, the controller will calculate the actual linear velocity (V) and heading rate (ω) in order to generate a valid trajectory for the robot.

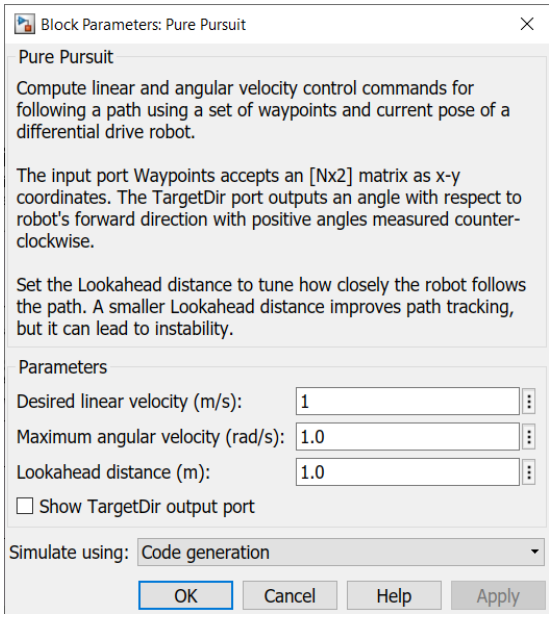


Fig.36 Pure Pursuit Block Parameters

The lookahead distance parameter [Fig.36] is a very crucial factor to be taken in consideration, it can affects heavily the robot's behavior.

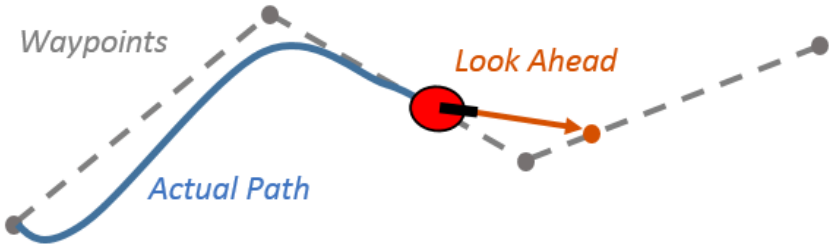


Fig. 37 Look Ahead Distance Principle [24]

The look ahead distance is basically how long the robot should look from its current location along the path in order to compute the angular velocities of the robot [24]. In [Fig.37], it is clearly shown the robot in a big red dot and the orange dot is the point that the robot is looking at along the path. In case, the look ahead distance is tuned to a small value, this will allow the robot to quickly regain the path between waypoints and as result the robot will move quickly toward the path [24]. A small look ahead distance has big drawback that will cause the robot to oscillate too much between waypoints leading to instability and overshooting the path [Fig.38]. This effect was clearly observed when implementing the algorithm with this parameter in simulation and on the physical robot that will be discussed in next chapters.

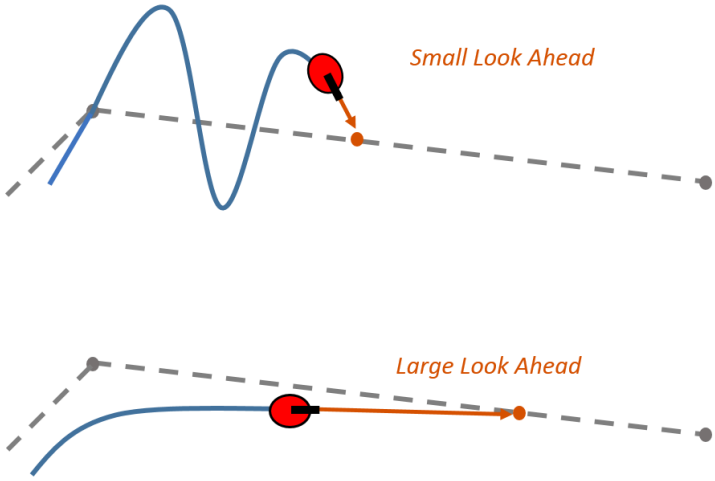


Fig.38 Effects of Small and Large Look Ahead Distances [24]

In order to solve the problem of path overshooting, the look ahead distance can be tuned to a larger value for a better path maintaining behavior as seen in [Fig.38]; but as much as the look ahead distance parameter increases, the waypoints shortcutting and curvatures near corners are bigger. For all these reasons, the parameters of the pure pursuit controller should be tuned very precisely according to the desired application. Moreover, the linear velocity and heading rate will affect also this behavior. Thus, it is very essential to tune these parameters first in simulation environment before implementing on the real robot. This method was been implemented during this project. It is found that a look ahead distance factor between 1 and 1.3 gives the robot a good behavior according to the waypoints, linear velocity and heading rate settled in this project which will be discussed later in a next chapter. According to [24], the main limitation of the pure pursuit controller is that the robot cannot follow exact direct paths between waypoints as seen in [Fig.37] and [Fig.38]. Parameters must be tuned as discussed to optimize the performance and to converge to the path over time. Back to the Simulink model of [Fig.35], after that V and ω are computed and with the pose and waypoints information; the Matlab function block (set wheel speed) [Fig. 35] computes the left and right angular wheel speeds of the robot according to [Eq.3.7] and [Eq.3.8]. Moreover, this Matlab function will allow to stop the robot at goal destination, when reaching a predefined threshold. After that, the Pioneer wheel control block [Fig.35] will compute the left and right wheel torque that will be used for robot's actuating in Gazebo. The torques are calculated by subtracting the calculated target angular wheel speed of each wheel from the measured wheel speeds from wheel sensors, then multiplied each by a gain factor of 0.8. The process of torque calculation can be seen clearly in the sub-module of the pioneer wheel control block [Fig.39].

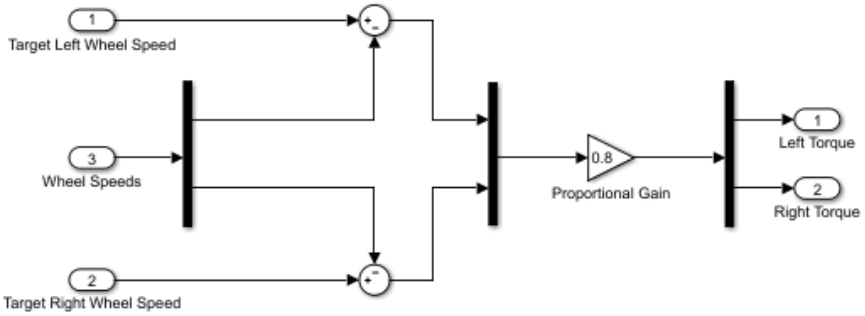


Fig.39 Left and Right Torque Computations from Wheel Speeds

For better signals arrangement, MUX and DEMUX blocks from Simulink library browser were used in [Fig.39].

Main Simulink Model

In this section, the main Simulink model that been developed for this project will be discussed [Fig.40]. It is based on the simple model [Fig.35], but with more expansion and development to match the required use case and including more functionalities such as obstacle detection and avoidance.

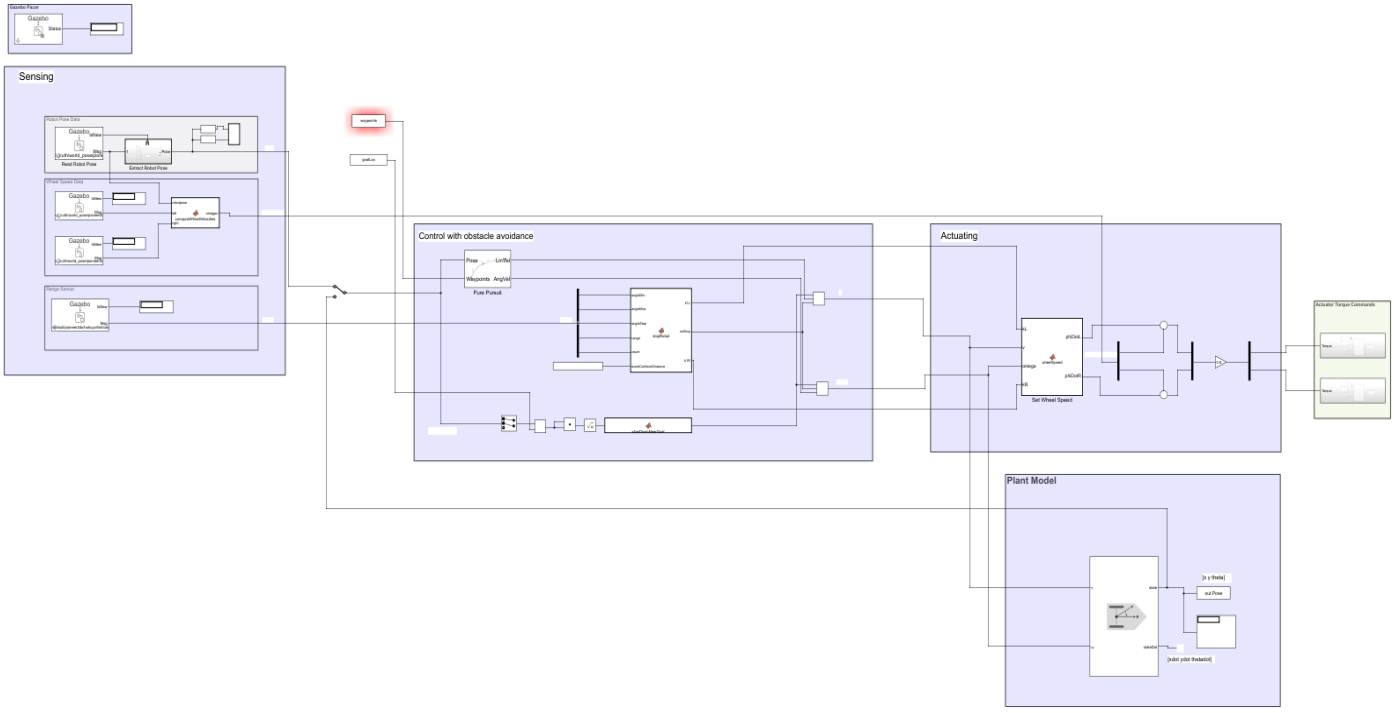


Fig.40 Main Applied Simulink Model for Automated Functionalities on the P3-DX Robot

The Simulink model [Fig.40] consists of several main functional blocks from Sensing, planning, decision making, actuating and an optional plant model that can be used in fusion algorithm to improve the data from the perception block. This Simulink model was mainly used in

virtual testing of the robot in Gazebo that will be discussed later, then in order to be implemented on the real robot. Each functional block of the Simulink model [Fig.40] will be discussed shortly in the rest of this section.

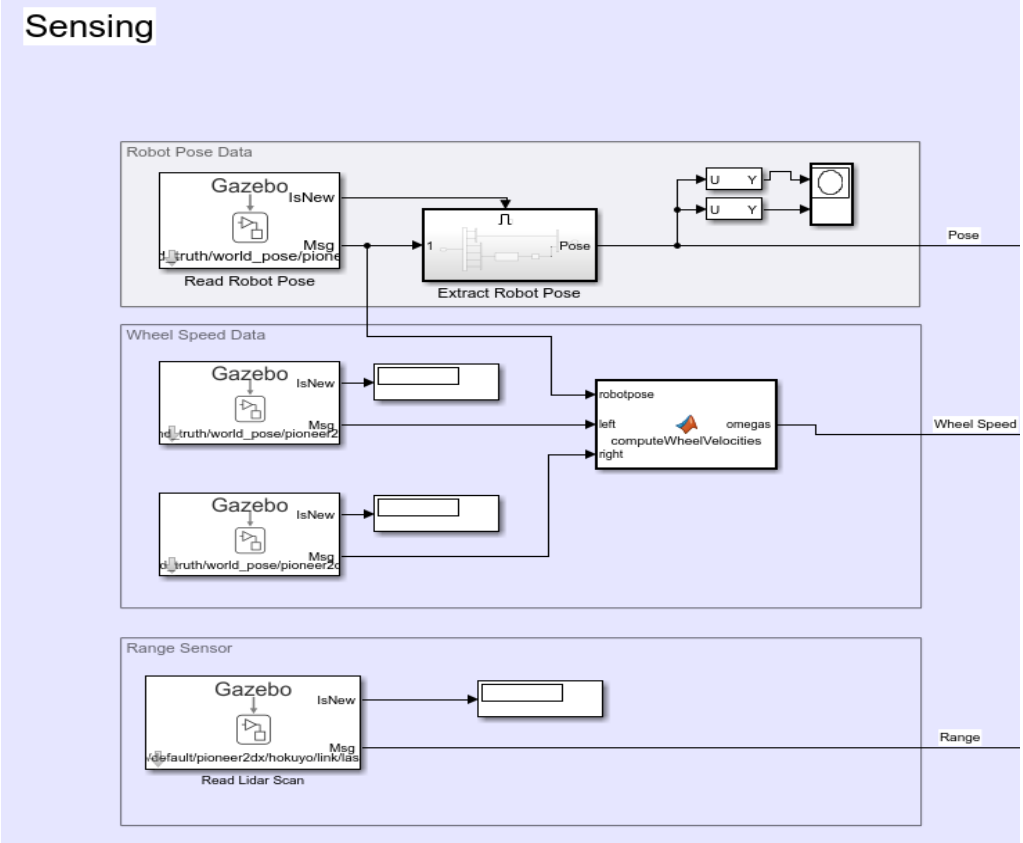


Fig.41 Sensing Block Module of the Main Algorithm

The sensing block [Fig.41] consists of extracting environmental data from the different sensors equipped on the robot, here the data are extracted from Gazebo environment. Mainly from wheel encoders, the pose and wheels speeds are extracted. For obstacle detection here, Lidar sensor was used only for simulation in Gazebo with the same concept usage of the Sonar on the real robot. The Lidar scan in Gazebo allow better visualization for obstacle detection. The robot pose is extracted from wheel encoders of from gazebo pose topic of the pioneer robot by using a coordinate transformation method. The pose information received from the robot are Quaternion data type that are difficult to deal with. For that reason, they are transformed into Euler angles ZYX representation inside the extract robot pose sub-block [Fig.41]. Fortunately, Simulink offer a ready coordinate transformation block in the library browser. Inside this block, the type of inputs coordinates and the desired output types can be specified. This block will take the signal of the robot's pose after using the bus selector block in order to specify the needed signals inside the bus and then transform into Euler representation [Fig.42], then using MUX block to get the desired pose vector $[x\ y\ \theta]^T$. Alternatively, it exists a manual way for coordinate transformation where all the mathematical code can be written in Matlab window and used as Matlab function in the Simulink model. This method will be not tackled in this report.

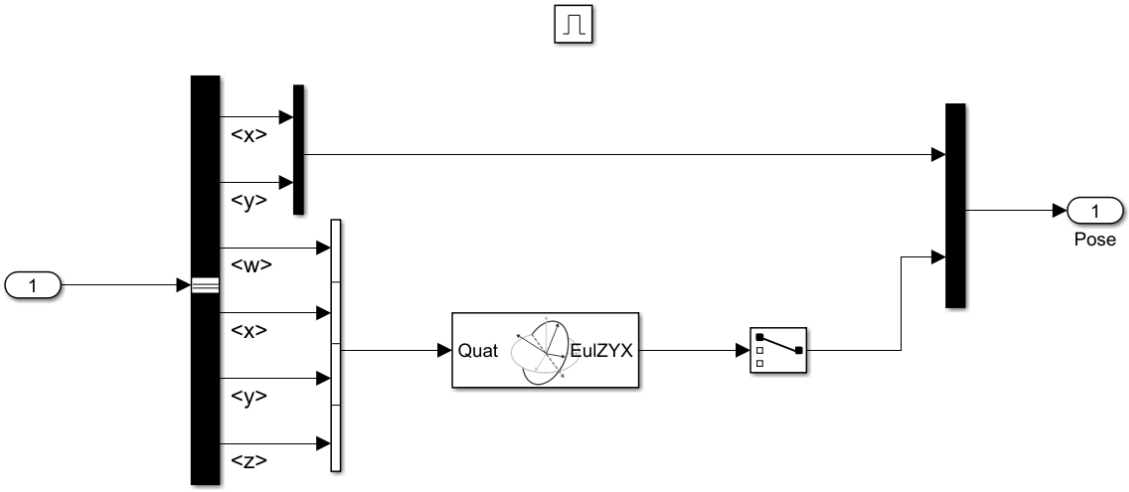


Fig.42 Coordinate Transformation inside the Extract Robot Pose Block

The wheel speeds are computed by using a Matlab function block named as compute wheel velocities in [Fig.41], it takes as inputs the pose information from the pose topic related to the chassis and in addition to the pose information from the pose topic related to left wheel and right wheel. In summary, the sensing block [Fig.41] gives after some kind of interpretation; the following outputs: pose, wheel speed and range from obstacles.

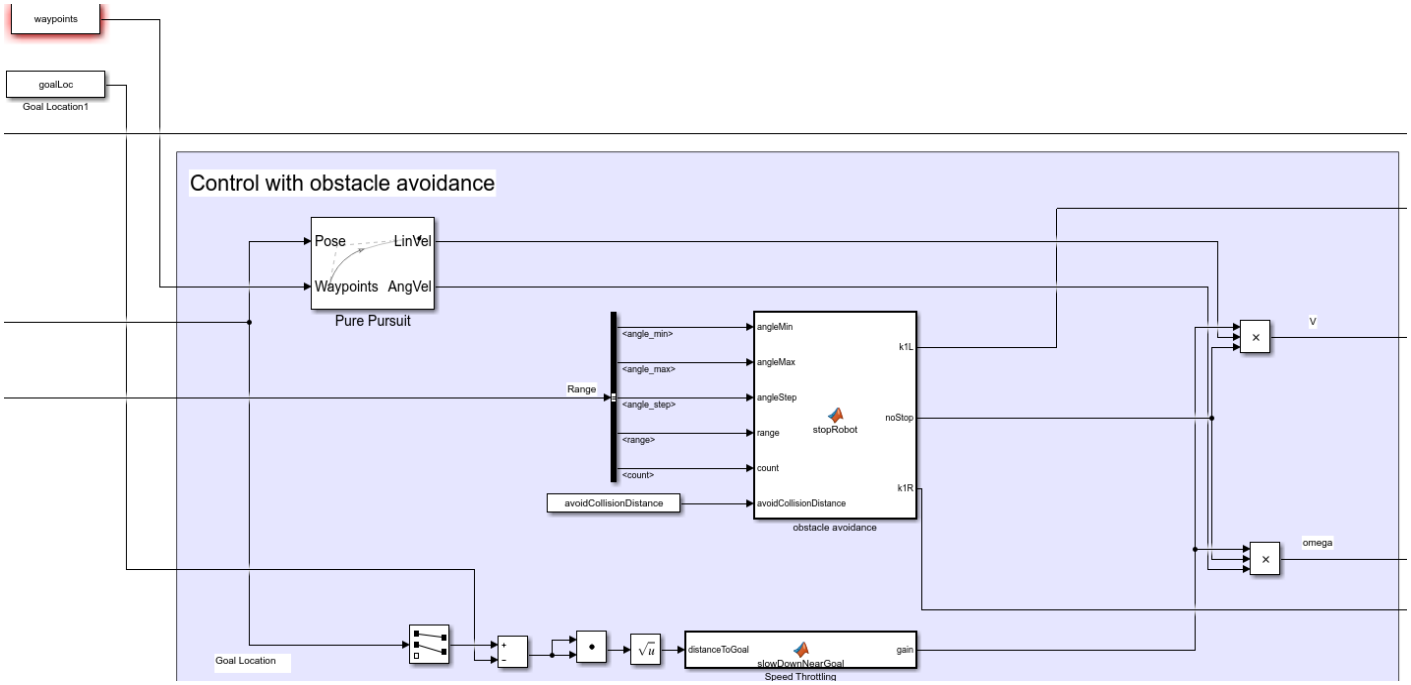


Fig.43 Control and Obstacle Avoidance Functional Block of the Main Simulink Model

After defining start, goal locations and set of waypoints for the robot to follow and after receiving the outputs data from the sensing blocks, these information are sent to the control and decision-making functional block [Fig.43] per request. Based on the pose and waypoint information, the linear velocity and heading rate are calculated by the pure pursuit controller as discussed before. Then, the range information from the Lidar scan or alternatively the Sonar, is

forwarded to Matlab function block named obstacle avoidance in [Fig.43]. In this Matlab function, based on the range reading, the decision is taken how to avoid the obstacle and at which distance. This block will give two outputs factors K_{1L} and K_{1R} that will be used to control the robot's motion. The functional obstacle avoidance block in [Fig.43] has the following Matlab code:

```
*
function [k1L, noStop,k1R] = stopRobot(angleMin, angleMax,
angleStep, range, count, avoidCollisionDistance)

angMin = -pi/10;
angMax = pi/10;
k1L= 1;
k1R= 1;
noStop = true;
avoidCollisionDistance = 2;


if count ~= 0
    %create a lidar scan object from the range readings
    scan = lidarScan(range, angleMin:angleStep:angleMax);

    rangeScope = scan.Ranges((scan.Angles > angMin) & ...
                             (scan.Angles < angMax) & ...
                             (scan.Ranges ~= 0));

    if rangeScope < avoidCollisionDistance
        k1L = -1;
        k1R = 1;
    else
        k1L= 1;
        k1R= 1;
    end
end

end
*
```

Basically this function was intended first to stop the robot when detecting an obstacle for first trials. Then, it was developed more in order to serve in obstacle avoidance not just stopping at an obstacle. The function called 'stopRobot' which is currently used for obstacle avoidance, it takes as input the bus elements found in the range bus signal coming from the Lidar scan in a similar concept with the Sonar array. Using a bus selector block, the necessary bus elements are selected. From these elements, the detecting range angle for the Lidar scan was settled between [-pi/10; pi/10] radians. Then, the function takes another input; the avoid collision distance which is a constant parameter. In this application, this parameter was settled to 2m; so when an

obstacle is detected in the range of 2m, the robot will start with the avoidance maneuvering. In the same Matlab code, a Lidar scan object is created to store readings from the Lidar sensor and then been converted to range value in meters called range scope. The important part of this simple code, is that when the range scope value from Lidar is less than the settled avoid collision distance parameter that means that an obstacle has been detected and the robot should avoid it. For that reason, the parameters K_{1L} and K_{1R} are settled to -1 and 1 respectively. Later in the actuating functional block, will explain how these two parameters affects the robot's motion to achieve the obstacle avoidance maneuvering. On the other side, if the range scope value is greater than the avoid collision distance this means there is no obstacle in the defined range and the robot should continue normally the path with both K_{1L} and K_{1R} are settled to 1. Another important block inside the control with obstacle avoidance block [Fig. 43] is the speed-throttling block. This block serves to slow down the robot progressively when approaching the target and then stop. This will allow the robot to stop smoothly especially when delivering items to the users. Moreover, this block will eliminate the possibility of instability or keep fluctuating when reaching the target. This Matlab function, will take as an input the actual pose vector subtracted by the goal location coordinate, then the dot inner product is applied on the resulting vector and after that, the square root is applied to get the norm of the distance to goal value. This value will give the information of how far is the robot from reaching the destination location in order to take the appropriate decision in slowing down and stopping. The speed throttling Matlab function will always give a gain of 1 when the robot sill far away from the destination, so the velocity and heading rate remain both unaffected. Inside this function, a slowdown threshold parameter is been set to 1, so when the distance to goal is less than 1 which means the robot is approaching the final target. Here, the robot start to decelerate progressively by setting the gain value equal to ((distance to goal) / (slowdown threshold)). Additionally, a stop threshold is been set to 0.5. When, the distance to goal is less than 0.5; the gain is set to 0. This means, when the robot is at 0.5 m radius from the final location, it will stops totally after having decelerated. Finally, it can been noticed from [Fig.43] that there are three main blocks affecting the V and ω of the robot, which are the pure pursuit controller, obstacle avoidance and speed throttling block. The outputs of this blocks are combined into two block products, one for V and one for ω. By this way, initially the pure pursuit controller will compute the linear velocity and heading rate then multiplied by the gain values coming from the other two blocks in order to stop robot, decelerate and so on… From [Fig.43], it is been recognized that K_{1L} and K_{1R} are forwarded directly to the actuation block which will be discussed next. As this Simulink model was intended to be tested first with Gazebo, so the decision was taken to directly control the joints which are the left and right wheels in Gazebo. For sure, it is possible for K_{1L} and K_{1R} to directly been involved in the block products of V and ω by changing their settled parameters

values; and this will be the case when implementing on the real robot where only the control of V and ω are done over the Bus communication.

As seen from [Fig.40] the main Simulink model, there is also an actuating block shown more clearly in [Fig.44]. The actuating block will transform the linear velocity (V) and heading rate (ω) received from the functional block [Fig.43] into left and right angular wheel speeds which are useful to deal with especially in Gazebo as for joint controls. The process in actuating block will occur in the low-level SH-2 robot's microcontroller [Fig.17].

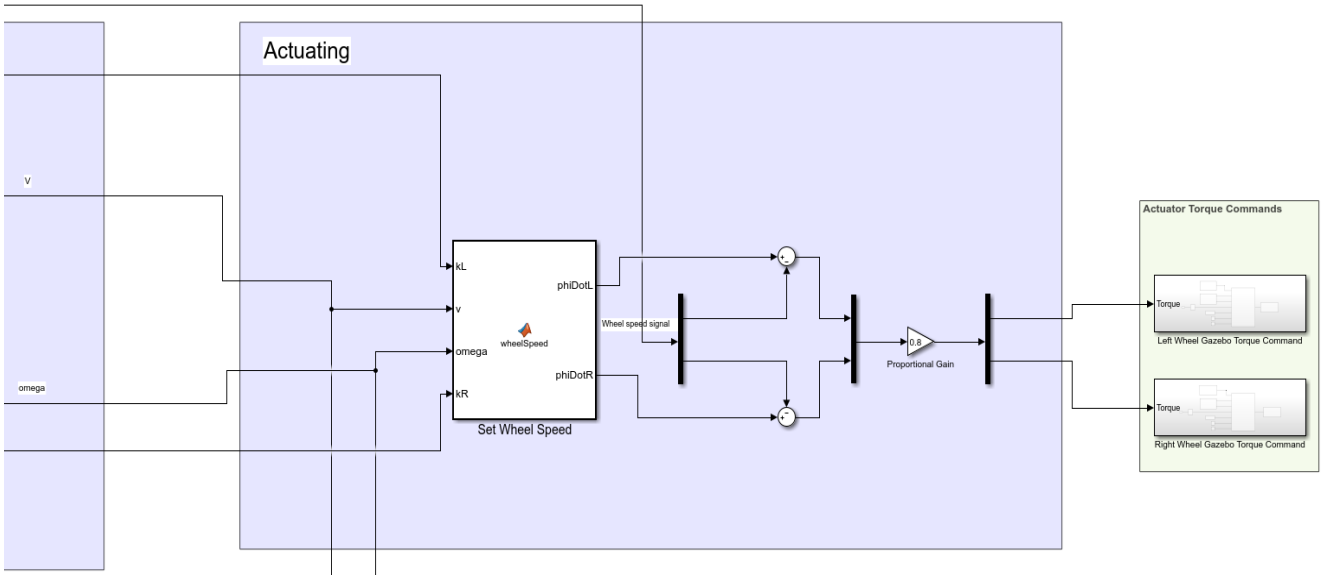


Fig.44 Actuating Functional Block

The actuating block [Fig.44] has main Matlab function named "Set Wheel Speed" that takes as input V, ω, K_{1L} and K_{1R}. Based on those parameters and the [Eq. 3.7], [Eq. 3.8]; the computation of left and right wheel speeds are done. They are designated by phiDotL and phiDotR in [Fig.44]. After the computation of the angular wheel speeds, they are transformed into left and right wheel torques as by the process described before and shown in [Fig.39]. Then, the computed wheel torques values are used in the actuator torque commands block designated for Gazebo joints control that are actually the left and right wheels of the robot. This will allow to visualize the robot's motion in Gazebo virtual environment.

The “Set Wheel Speed” Matlab function has the following code:

```
*function [phiDotL, phiDotR] = wheelSpeed(kL,v, omega, kR)
%%wheelSpeed computes the speed of the left and the right
wheel given the linear and angular velocities.

% Robot's constants
trackWidth = 0.381;
wheelRadius = 0.195/2;

% Convert velocity and heading angular velocity to wheel
speeds
phiDotL = kL*((v - trackWidth/2*omega)/wheelRadius);
phiDotR = kR*((v + trackWidth/2*omega)/wheelRadius);

end*
```

As seen from the Matlab code above, the wheel speed function compute the angular wheel speeds according to the inputs mentioned above and to the specific constants related to the P3-DX robot such as wheel radius and track width. When there is no obstacle detected, as discussed before K_L and K_R will be both equal to 1. In this case, phiDotL and phiDotR will have exactly the same equations as [Eq.3.7] and [Eq.3.8] and the robot continue normally to follow the desired path according to V and ω calculated from the pure pursuit controller. On the other side when K_L = -1 and K_R = 1, this means an obstacle has been detected as mentioned previously. In this case, the robot will steer in the aim to avoid the obstacle and when the obstacle is no longer in the defined range, the robot will rejoin and continue again the predefined path.

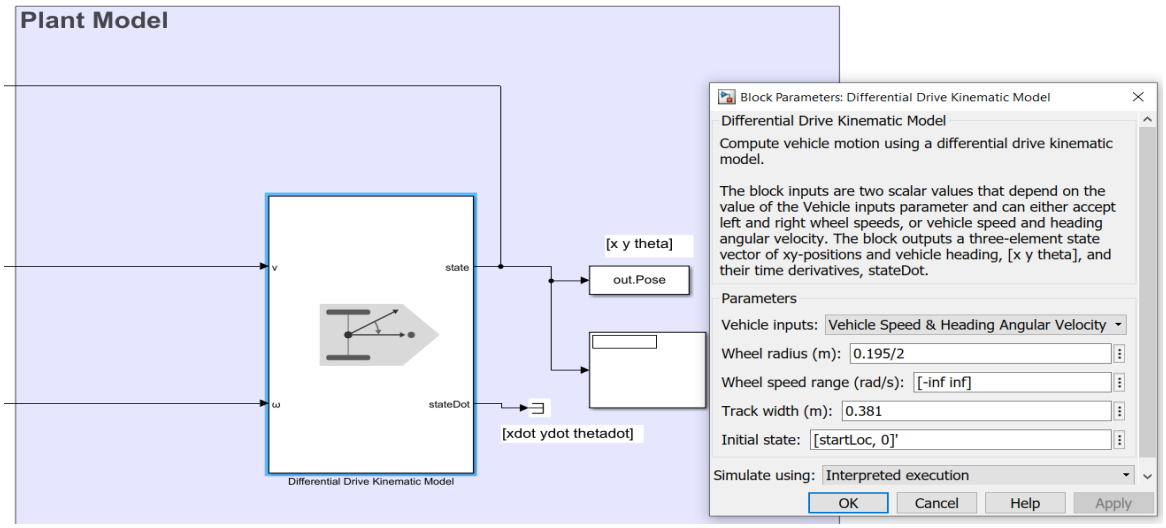


Fig.45 Plant Model for the P3-DX Robot

In [Fig.40] there is one additional optional Block named “Plant Model” as mentioned before. This block [Fig.45] is based on the differential drive Kinematic model. It takes as inputs the linear Velocity (V) and the heading rate (ω) calculated from the control and obstacle avoidance block of [Fig.43]. The robot’s constant parameters (wheel radius and track width) should be defined with the robot’s initial state. After that, the robot current state or pose is calculated with the time derivative also of the current state. The computed robot’s pose can be merged with the measured extracted pose from the wheel encoders in a fusion algorithm in order to compensate for the errors that can occur from the wheel encoders. For sure, it exists better ways for fusing data for localization such as merging different types of sensors with odometry like IMU and others…

SIMULATION TESTING IN VIRTUAL ENVIRONMENT GAZEBO

After the design and development of the main algorithm within Matlab/Simulink as development environment that been discussed in the previous chapter, now it is the turn for the first implementation test that will be the first layer according to the V-model [Fig.34]. The first testing layer will be a simulation test in a virtual environment by using Gazebo as a software tool. The decision to choose Gazebo for virtual testing is due to several important factors. It is wide spread in robotics applications and an open source that can be downloaded for free. It is highly supported by many plugins and additional tools. The most important factor is that Gazebo can be easily coupled with Simulink which can both form a development-test tight environment. By its turn, Simulink offers many useful ready Gazebo blocks that can be found in the library browser inside the robotics system toolbox [Fig.46]. These Simulink blocks for Gazebo were used in the Simulink models discussed in the previous chapter [Fig.35] and [Fig.40].

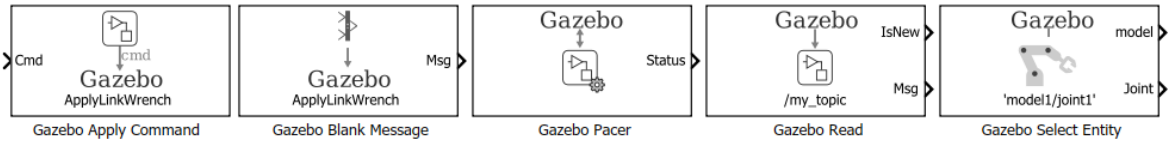


Fig.46 Available Simulink Blocks for Gazebo in the Robotics System Toolbox

Gazebo usually operates on Linux OS, but it is sufficient to be installed within a virtual machine with Linux OS on the same Windows PC that runs Matlab/Simulink. This method will allow for best coupling option between the two environments. Mathworks provides an open source link from where a preconfigured VM can be downloaded with ROS-Gazebo already installed on it. Moreover, this VM contains some predefined ready Gazebo world examples that be can launched directly with Simulink. This VM provided by Mathworks can be download on the following link [25] with all necessary instructions. In addition, Gazebo can be installed on any other Linux OS system PCs and specifically on mini-single board computers. Gazebo was also installed within the ROS package on the robot's master single-board computer, the Nvidia Jetson Nano. By this way, it can also be used solely within ROS or also coupled with Simulink with some communication setup that will be discussed later. Gazebo offers the possibility to create many real cases scenario in virtual environment, it allows to use many ready objects, import ready models or create a 3D CAD model and imported as URDF file to establish the specific testing scenario. In addition, it has the building editor functionality in order to build an environment based on a given 2D map of the area. Gazebo offers a great opportunity to create a digital twin of the real robot. The physical properties of the robot and environment conditions can be taken into account, such as robot's weight, center of mass, friction between joints and

between ground, way of robot's joint assembly and degree of freedom, speed of wind that the robot can face…

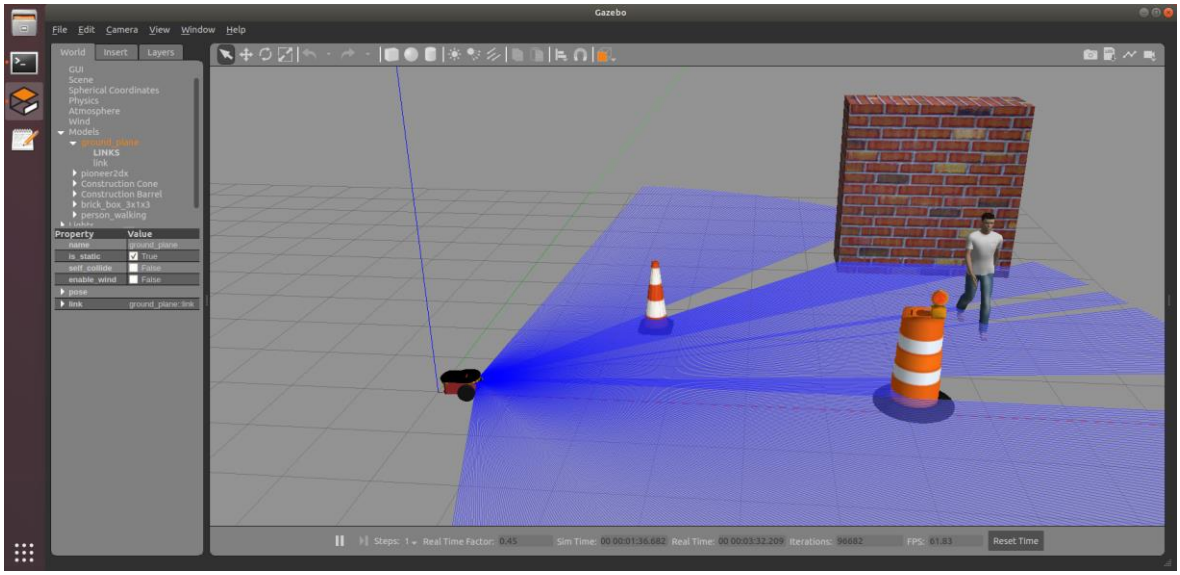


Fig.47 Testing Scenario Created within Gazebo used the P3-DX Robot

The pioneer robot used in the simulation in this project was mainly not equipped with any sensors. In Gazebo each model has what is called an SDF file in XML format, where there the model can be configured and changed. By modifying the robot's main SDF file and adding the SDF file of the Lidar Hukoyo sensor, now the robot is equipped with a Lidar scanner for object detection. The output of the Lidar scanner can be visualized by the blue rays in [Fig.47], where the dark blue rays indicate the obstacles been detected. The Hukoyo Lidar sensor was used in the same concept as the robot is equipped with the Sonar arrays, but the decision to use this Lidar type sensor in the simulation because it has good plugins that can work reliably with Simulink. For sure, similarly suitable plugins can be developed for the Pioneer's Sonar system or use the Hector Sonar SDF. It is important to mention that each square on the Gazebo ground grid [Fig.47] is actually **1m** in real coordinates for both x and y directions. The x-direction in Gazebo is in the direction of robot's travel orientation [Fig.47].

Simulink-Gazebo Coupling

In order to let Simulink and Gazebo work together, there is a useful instruction provided by Mathworks on [23]. On the mentioned link, Mathworks provides a Simulink model discussed previously [Fig.35] in order to test Gazebo with it. This test example is already configured to work with a virtual copy of a similar type of the pioneer robot. The main focus in the rest of this section is on testing the main designed Simulink model [Fig.40] with Gazebo. First step is to launch the Simulink model with the Gazebo world that is designed as needed by the use-case

scenario. In this Gazebo world [Fig. 49], the modified pioneer robot with Lidar sensor is used. After that, the communication between Gazebo and Simulink are done by using a Simulink block called "Gazebo Pacer" [Fig.46] in the main Simulink model [Fig.40]. Inside the "Gazebo Pacer" block, the main network communication is set over TCP/IP, if Gazebo is used on the same PC's VM, by setting the Hostname/IP address of the virtual machine and then connection test can be conducted for verification of proper communication. After that, the "Gazebo Read" Simulink blocks [Fig.46] are used in the sensing functional block of the main algorithm [Fig.40]. The "Gazebo Read" blocks seen in [Fig.41], serve to simulate real sensors such as wheel encoders and Lidar. These blocks will read a topic similar to ROS topics which will be discussed later. These blocks will output a bus signal message based on the selected topic and message from the Gazebo server and according to published data from the robot at a time instance, in other way it is what the robot is sensing. The topics are related to pose, laser scan, or… The same topic can be used several times like the "Gazebo Read" pose in [Fig. 41] to extract different information like actual pose or wheel speed according to a specific interpretation of the signal Bus message. On the other side in order to apply the output of the main algorithm [Fig.40] to visualize the robot's motion in Gazebo, the calculated torque values from the actuating blocks [Fig.44] are forwarded to the actuator torque commands block. This block contains two sub-blocks, one to control left wheel torque and the second to control right wheel torque in Gazebo [Fig.44]. Each of these two sub-blocks consist of Bus signal treatment [Fig.48].

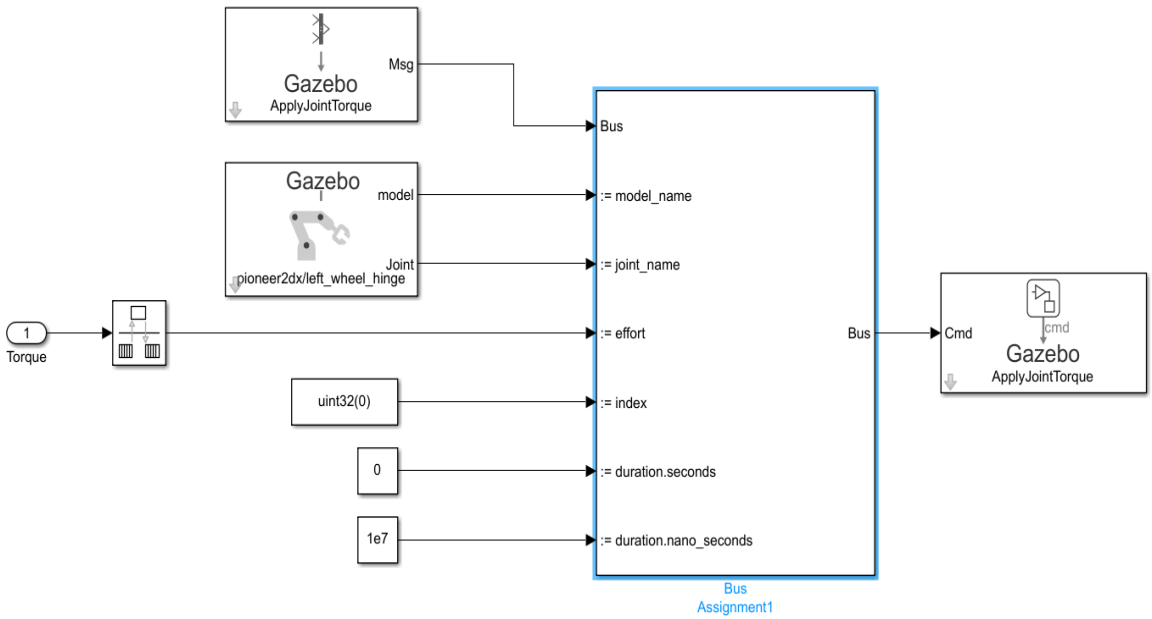


Fig.48 Gazebo Left/Right Wheel Torque Command

As seen in [Fig. 48], the Gazebo torque command uses mainly the "Gazebo Apply Command" block [Fig.46] in order to apply here the joint torque command and send it to Gazebo simulator

over the network. This block accepts a bus signal, for that reason a Bus assignment block is used to create this bus signal that accept several elements. The content of the bus signal are the value of the calculated torque, a gazebo entity such as link or joint available from the opened Gazebo world and here it is the left or wheel hinge in order to apply on in the torque command. To select the required gazebo entity, which can be also for example a robot arm joint in an industrial robot if used in Gazebo world, the "Gazebo Select Entity" block is used [Fig.46]. As a result, after Gazebo receive those torque commands, the robot's wheels will rotate accordingly allowing the robot to move according to the algorithm designed in Simulink. Thus, the coupling of Gazebo and Simulink affects the latter at the peripheries of the model and specifically at the sensing and actuating functional blocks. Gazebo environment by means of the appropriate Simulink blocks [Fig.46], serve as the virtual sensors and actuators of the robot for the Simulink model. The implementation of the main algorithm [Fig.40] in Gazebo as the first test layer according to the V-model [Fig. 34], is shown in [Fig. 49]. The complete video of this test with all necessary explanations can be found on the Mission Mint YouTube channel on the link [26].

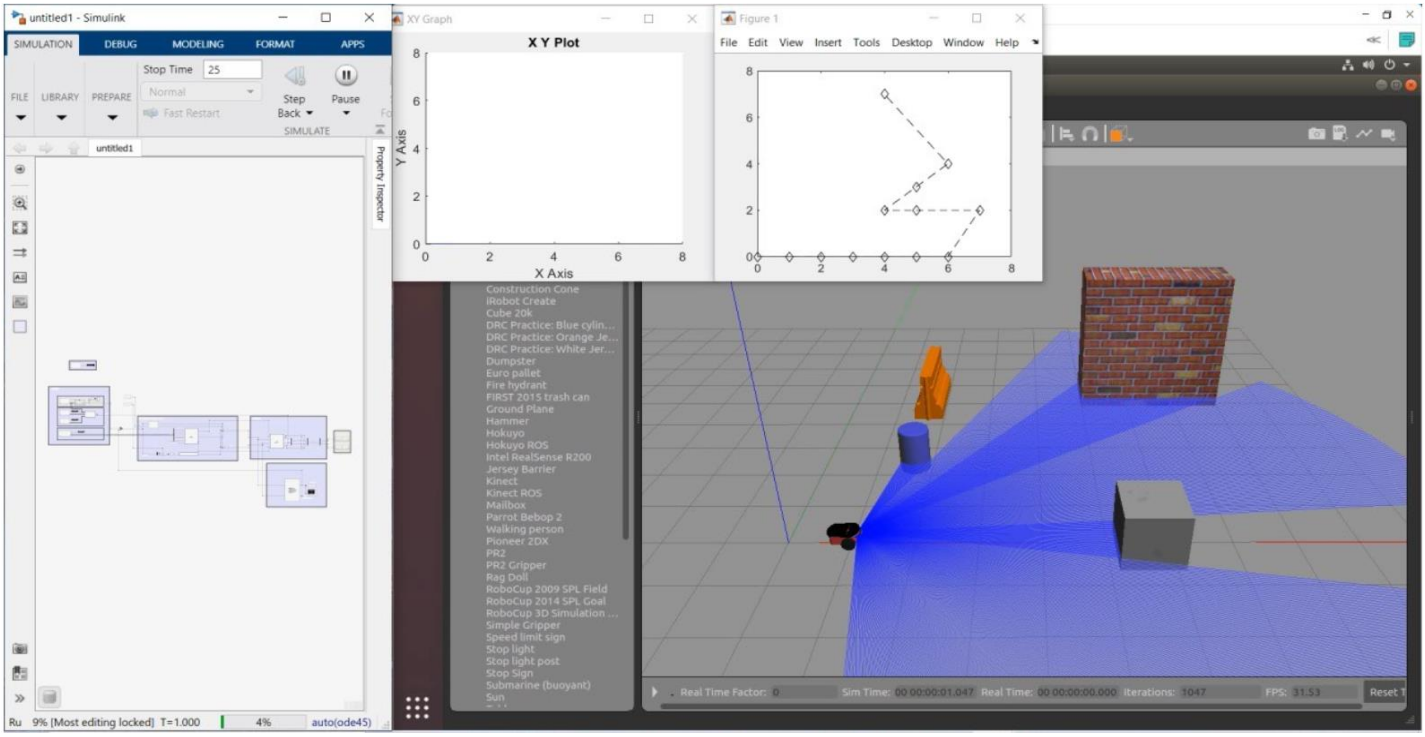


Fig.49 Coupling of Simulink and Gazebo for Algorithm Testing in Virtual Environment

On the left-hand side in [Fig.49], the main Simulink model [Fig.40] is tested in Gazebo on the right-hand side in [Fig.49]. On the top of [Fig.49], the predefined path for the robot to follow according to the desired waypoints and next to it, the actual path generated by the robot while moving in Gazebo will be displayed [Fig.50]. According to the main Simulink model, the robot

will achieve the desired movement from initial location to goal location while avoiding all obstacles as defined previously. This virtual testing offers the opportunity to visualize robot's behavior in different cases such as in obstacle avoiding maneuvers and the effect on the lateral and longitudinal motions with respect to various parameters tuning in the Simulink model.

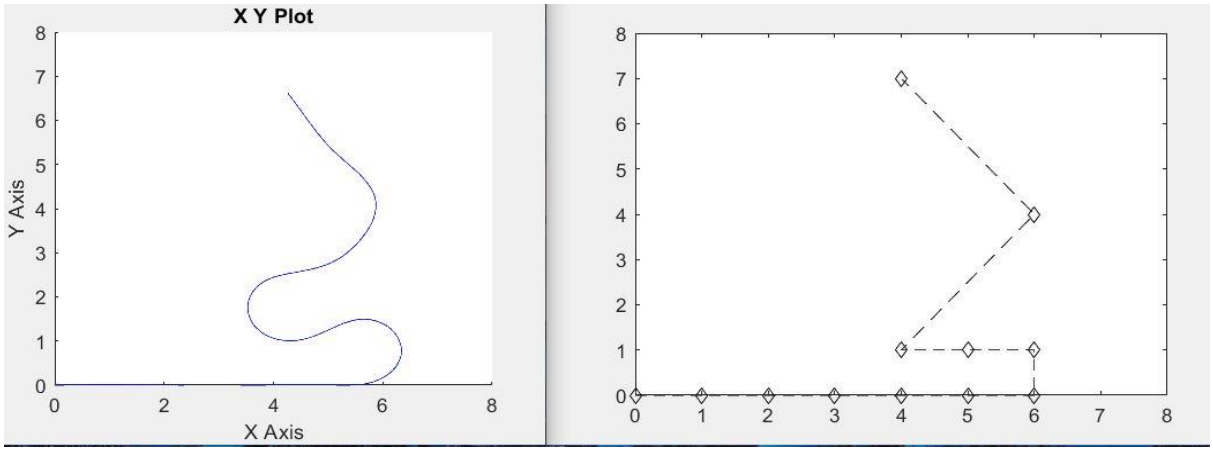


Fig.50 Actual Path Achieved by the Robot in Gazebo (left) Vs Predefined Desired Path (right)

In [Fig.50], it can be noticed that the achieved path by the robot in Gazebo from start to goal locations is quiet similar to the predefined path which was settled in the Simulink model by defining a Waypoints matrix starting from (0 , 0) to (4 , 7) as goal location. The deviation between the graphs is due to several facts, some of them are the nature of the pure pursuit controller in achieving the paths as discussed in previous chapter and also due to the fact that the robot has avoided some obstacles located at the points that are previously defined in the waypoints matrix for the desired path. The performance of the robot can be adjusted by tuning the parameters of the pure pursuit controller as discussed previously. Thus, testing the algorithm in virtual environment is very essential in the development process. It allows to validate the designed algorithm at a great extent and to apply the required modifications before the implementation on the real robot. This critical phase will contribute in saving time, cost and effort in the overall process.

ALGORITHM IMPLEMENTATION ON THE PHYSICAL ROBOT

After excessive testing in the virtual environment in the first testing layer according to the V-model [Fig.34], now the turn to shift up to the second testing layer which is the implementation on the real P3-DX robot. In order to implement the algorithm successfully on the P3-DX robot, it requires beforehand some commissioning and preparation work. This commissioning phase was the most challenging during this project due to lack of resources and support from the robot's manufacturer in term of finding the suitable supporting packages and libraries in order establish the required communication with the robot and to make it suitable to the development platform of Matlab-Simulink and ROS. The commissioning work has begun during the initial stages of the project and was been conducting during the overall project stages using parallel engineering method. It is not efficient to wait to finish from the first testing layer in virtual environment to begin with the commission work on the robot, as it is a lengthy process.

Working with Aria

In order to establish communication between the host PC and the P3-DX robot, Aria library must be installed on the host PC as the SH-2 robot's microcontroller has an Arcos firmware and works with the Aria library for the algorithm implementation [10]. Aria library is built in C++ language and need to be debugged and compiled first before been able to using it. Microsoft Visual Studio was used for the compilation process in order make ready executable files then can be launched directly using Windows-PCs. In this report, the compilation process of Aria will not be tackled as it is a lengthy process. The instructions can be followed in the "Read-me" file when downloading the Aria package. In order to get the Aria original package, the official Robot's website that contains these packages are down and it is no longer working. Many difficulties were been faced to find parts of Aria package and then combining them in one package for compilation. As a recommendation, a newer rebuild of the overall Aria package can be found on the link [27] for download. This rebuild is named "AriaCoda-master" and published on GitHub with all details and instructions on the mentioned link. After having the two packages, the original Aria and Aria-coda master, the decision was taken to work with the original Aria as the Aria-coda master is always in updating process to become more and more stable and been able to support the newer versions of interfaces with the other tools such as Matlab/Simulink as mentioned on the GitHub link. It is important to compile first the Aria library on a Windows-PC and use the provided examples by Aria after they become executable files. Those examples will be used to test the robot connection with the host PC in an easy and interactive way where the first robot's movement can be achieved by simple command from the connected host PC. Some examples provided by Aria are SimpleConnect, teleop, wander

and many more… Each example code can be launched with Microsoft Visual Studio and execute it for testing on the robot. For example, wander mode will allow to the robot to navigate randomly while using the Sonar system to avoid obstacles, the opened example in Visual Studio can be modified in C++ language to change the range of detection and how the robot will avoid the obstacles and other staffs. It is recommended to compile the Aria example named "Aria Demo", after compilation this example will become an application that can be opened directly in Windows PC. The advantage of Aria demo app is that can provide the possibility to use all examples provided by Aria in one interface from tele-operation mode, wander, sonar readings, position readings,.. In order to run Aria demo on the robot, it is important to make sure that the robot is connected to port COM 1 of the PC, this can be checked by going to "Device manager" in Windows OS and check the USB connections ports. A common mistake can happen, if the connected used port is not configured to COM 1, the Aria demo will shut down immediately after try to opening it. In case the robot is not physically available, it exists also a solution to check the Aria demo interface by installing MobileSim application by Omron Adept Mobile Robots [10]. This simulation application is compatible with Windows OS and run with the Aria Demo application [Fig.51].

Fig.51 MobileSim (left) with Aria Demo Interface (right)

From Aria Demo interface [Fig.51], any available mode can be selected by typing its suitable key. For example to drive the robot manually by mean of a keyboard, the teleop mode is selected by pressing the letter key 't'. In this mode, the current state of the robot from velocities and position are displayed in the black Aria demo interface [Fig.51]. As a result, the robot's movement is visualized in the MobileSim application [Fig.51]. A ready or custom map can be selected in MobileSim application. Alternatively, the Sonar mode can be selected by pressing 's'

and the sonar readings will be displayed in the Aria demo interface. It is important when launching this overall demo, that the MobileSim app should be launched first and then Aria Demo for successful TCP/IP connection between the two applications. In the same way, Aria demo can be implemented on the P3-DX robot directly by connecting first the robot to the laptop/PC by serial connection. After that, Aria demo application is launched and then the robot can be controlled by the various available mode [Fig.51].

Unfortunately, it is not sufficient to install only Aria on the Windows OS PC, because this mean that the overall algorithm should be developed solely in C++ language in Microsoft Visual Studio and the Simulink model developed on the Windows PC cannot be implemented on the robot. Aria supports Simulink up to version 2014. In this project, Simulink version R2020a was used. For this reason, ROS is chosen as middleware between the windows PC running Simulink and the P3-DX robot that uses Aria. ROS is installed on the VM on same windows laptop/PC running Simulink and on the on-board computer Nvidia Jetson Nano. Both environments run on Linux OS. For this reason, additional step must be achieved, is to install also Aria package on the Linux devices. The same downloaded Aria or AriaCoda-master package on Window OS PC, contains the necessary "Makefile" for Linux OS installation. For installation in Linux OS, the aria source directory must be opened in a terminal and then running 'make', more information can be found in the "Readme" file available in Aria main folder or in Aria-coda master that can be downloaded from [27]. After that, the Aria package is ready to be used on the Linux OS. In order to make sure that the Aria library is working fine and everything is installed correctly, it will be good to try some Aria example modes similar to those been executed on the Windows PC. These examples are again SimpleConnect to test if the PC can connect successfully with the robot, demo mode to test the overall Aria demo as in [Fig.51] and many others. In order to run one of these Aria examples in Linux OS, open Aria source directory in a new terminal and type the following:

```
export LD_LIBRARY_PATH=`pwd`/lib
  cd examples
  make simpleConnect
  ./simpleConnect
```

And instead of simpleConnect command, demo command can be written to launch the complete Aria demo as in [Fig.51] including tele-operation mode and others. The make command in front of each selected example (make simpleConnect) should be written only in the first time and after that, no need for the 'make' command when need to launch any example again [Fig.52]. It is important to mention on some hardware that runs on Linux OS, due to some root

settings, it is necessary to give permission for the connection to the port first and before launching the above command to run any examples. For example on the Nvidia Jetson Nano board, this permission must be given in order to establish connection with the robot. This permission is given first by switching on the robot and achieving the serial connection, then the following command must be written in a new Linux terminal :

```
sudo chmod -R 777 /dev/ttyUSB0
```

After typing such command, a password is asked to be entered and here the normal password used to log in to the device should be entered. After that, the permission is given to access the port and the connection can be established successfully with the robot by running the commands above for any example. In case after running any example, the port cannot be assigned automatically, it can be assigned manually next to the name of the mode in that way:

./simpleConnect -robotPort /dev/ttyUSB0	for simple connect mode
./demo -robotPort /dev/ttyUSB0	for Aria demo mode

As a USB-adapter is used to establish serial connection between the robot and the on-board computer Nvidia Jetson Nano, the port /dev/ttyUSB0 is been used.

```
user@ubuntu:/usr/local/Aria$  export LD_LIBRARY_PATH=`pwd`/lib
user@ubuntu:/usr/local/Aria$ cd examples
user@ubuntu:/usr/local/Aria/examples$ ./demo
Connnecting to robot using TCP connection to localhost:8101...
Could not connect to simulator, connecting to robot through serial port /dev/ttyS0.
Syncing 0
No packet.
Syncing 0
No packet.
Trying to close possible old connection
Syncing 0
No packet.
Syncing 0
No packet.
```

Fig.52 Running Aria Demo on the VM operating on Linux on the Same Windows OS-PC

After the Aria package has been installed on Linux computers, now the robot operating system (ROS) can be used with Matlab/Simulink in order to implement the main designed algorithm on the physical Pioneer 3-DX robot.

Working with the Robot Operating System (ROS)

In order to realize the implementation of the algorithm designed in Simulink on the Robot, ROS is needed to be installed on the master computer. ROS is considered as the main pillar of this project in term of implementation. It is a middleware environment between the Simulink as development environment and the robot itself. There are many reasons that affect the decision to use ROS in this project. The Pioneer robot microcontroller is not supported directly by Simulink and Aria support is very limited and not valid for latest Simulink releases. Moreover, the Simulink model have to run independently on the Nvidia Jetson board for the autonomous functionality and Jetson Nano is not highly supported by Simulink R2020a for code generation target hardware in a different situation with the Rasberry Pi boards. It is important to mention that Nvidia boards are getting more support in the last Simulink releases R2021a and above, but either in that case it remains challenging and not efficient to just generate a random C++ code from the Simulink model and run it on the Nvidia board. Also in that case, it requires many conditions that are difficult to satisfy them all for successful implementation on the robot. As a result, the decision was taken to use the Robot Operating System. ROS offer a high degree of flexibility in the development and implementation process, it can be installed on any hardware that runs on Linux OS. It offers the possibility that any Simulink model can be implemented on any robot with the condition that the latter is supported by ROS. It is sufficient after setting up ROS with Simulink, to generate a ROS node from the Simulink model to run directly on the target ROS hardware. It is a very efficient and reliable solution that will be discussed later. As known, ROS is wide spread and used in many applications on most of the well-known robots. Moreover, ROS offers many useful tools that can help in algorithm development and implementations that will be discussed in this section. ROS is a very big topic and in this report will only cover what is related to the project itself.

ROS Workflow Diagram Implemented in the Project

The workflow diagram [Fig.53] was mainly adopted in the project from algorithm development to implementation in virtual environment and on the physical P3-DX robot according to V-model [Fig.34]. As can be noticed from [Fig.53], ROS is the core component of this workflow diagram. As discussed before; the overall algorithm, according to [Fig.6] and [Fig.7], is developed in Matlab/Simulink. Then from [Fig.53], it can be seen that Simulink and ROS are coupled and bridged together. This bridging method will allow the implementation of the Simulink model in virtual environment (Gazebo that is part of ROS) and on the Pioneer 3-DX robot (which is also adapted to be a ROS hardware). From [Fig.53], there are two methods for Simulink-ROS implementation. It exists a method to directly connect the windows Laptop/PC, running the Simulink model, into ROS which can be by its turn running on the virtual machine of

the same PC or directly on the Jetson Nano by means of Ethernet communication. This "Desktop Prototyping" method will allow direct live implementation of the algorithm from Simulink window tab into the ROS Hardware, which will help in monitoring the overall implementation process, observing live signals, and tuning the required parameters directly on spot from the Simulink window tab while the algorithm is running on the Hardware device.

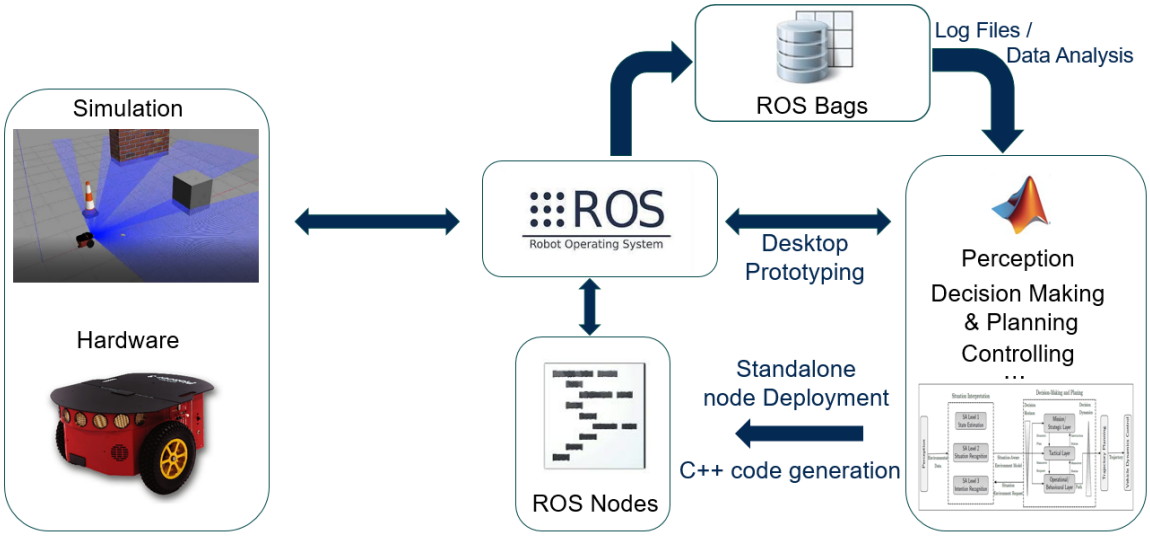


Fig.53 Process Workflow Diagram for Algorithm Deployment and Implementation

The second method for Simulink-ROS implementation is the standalone node deployment [Fig.53]. This method will allow to deploy the Simulink model into the target hardware that can be the Nvidia Jetson Nano. In this standalone deployment method, the Simulink model is transformed into ROS node containing C++ codes by the Simulink code generation tool. Then, they are directly transferred into the Catkin workspace of ROS on the target hardware. There, inside the ROS catkin workspace, this generated node can be extracted and built by "*catkin-make"* command in order to be an executable file and ready for use. By this method, the generated ROS node, consisting of the Simulink model, can be launched directly on the Nvidia Jetson Nano without the need of the Simulink Windows PC that been used in development and desktop prototyping. Thus, the Standalone node deployment method will be used in the final stage and after the necessary previous verification & validation steps have been achieved. By this way, the robot is able finally to operate autonomously to achieve the required tasks. Additionally, it can be noticed form the workflow diagram [Fig.53] that it exists a tool named "ROS Bags". This tool was very crucial and important in this project, it serves mainly for files logging and data analysis. ROS structure and components, ROS Bags and ROS-Simulink bridging, they all will be discussed in details in the upcoming sections.

ROS Main Structure & Components

To install ROS on a desired hardware, the best option is to install it from ROS WIKI on the following link [28]. There is a complete instruction step by step for each ROS installation version on the desired hardware and operating system. ROS WIKI is the official platform for all ROS documentations and information, where many useful instructions and tutorials can be found. In this project, ROS 1 Melodic was installed. It is very stable version and highly supported. It is recommended if there is enough space on the target hardware to install the desktop-full version as it include ROS, Gazebo and all useful tools such as *rqt* and *rviz*. After ROS installation, all the ROS files and future work will occur in a folder called Catkin-workspace which is a dedicated working space for ROS. Many workspaces can be created with different names. The Catkin workspace (catkin_ws) should be always sourced an opened in terminal when working in ROS.

ROS consists of several components [Fig.54]; the most essential one is the ROS master especially in ROS 1.

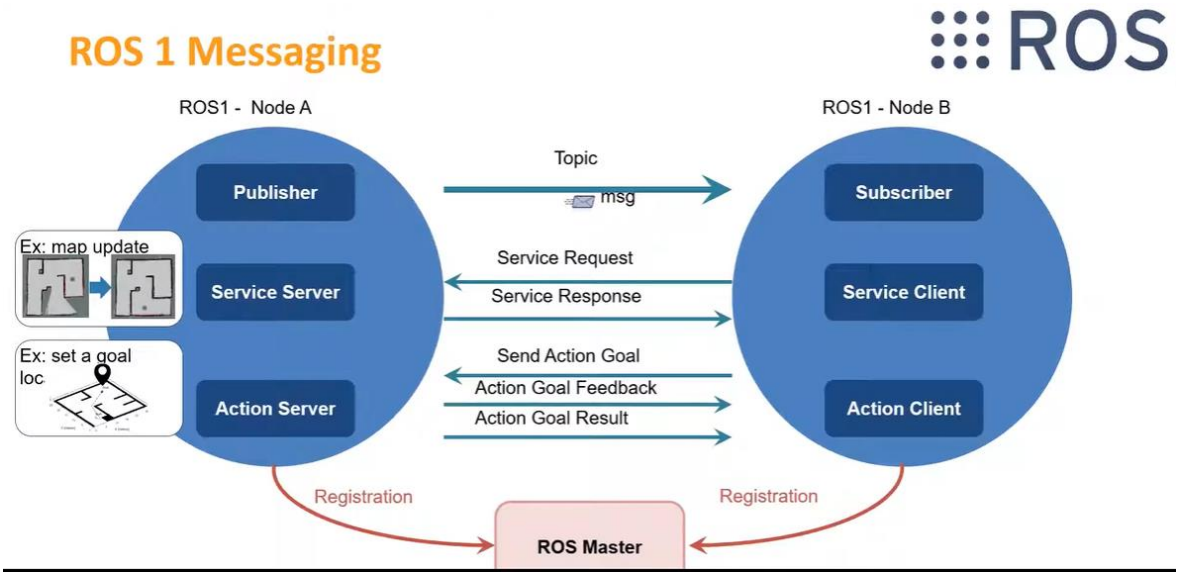


Fig.54 ROS 1 Structure [29]

As its name said, it the master in the ROS Graph, where all nodes will find each others, exchange messages and invoke services. ROS master should be launched first when starting a ROS project. ROS master is launched by typing the following command in a new Linux terminal: *roscore*. After ROS master has been launched, now it is ready to start the ROS project. Successful launch of ROS master means that ROS is working properly on the target hardware. Usually, ROS master need connection to the internet in order to be launched successfully.
In case ROS need to operate without internet, type the following commands in the terminal before launching the ROS master:

export ROS_HOSTNAME=localhost

export ROS_MASTER_URI=http://localhost:11311

roscore

A process running a ROS code is called a node. The ROS code can be for example a C++ or python code and the node can be the Simulink model. The number of nodes depend on the application. For example, the overall Simulink model can be a one node or it can be divided into several nodes [Fig.55] such as one node for perception, one node for navigation, one node for decision-making, one node for motion control etc.… Moreover, each hardware or sensor may have its own node such as one node for the P3-DX robot, one node for the Intel RealSense camera, one node for Lidar and so on… The advantage of using the hardware ROS nodes, is that they are usually provided by their manufacturers and can be obtained most frequently from a specified GitHub. Depending on the number of nodes that the algorithm use, it should be launched all in advance. To check the active nodes in the ROS network, type the following command in the terminal:

rosnode list

After that, all active nodes will appear in the terminal.

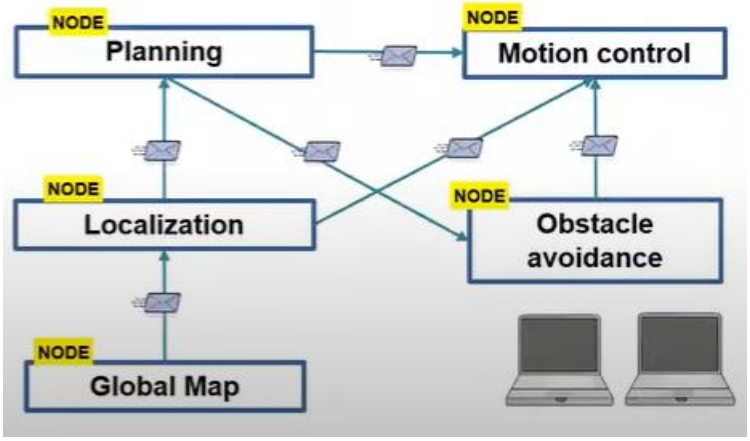


Fig. 55 Algorithm Split into Several ROS Nodes [29]

In order for the nodes to communicate with each other, it needs what is called “ROS Topic” [Fig.54]. These topics are set of specific data. For example, Sonar topic from where all the sonar range readings can be accessed. Another topic is the pose topic from where the pose readings can be accessed.

In order to see the active ROS topics on the ROS network, type the following command in the terminal:

rostopic list

It exists various type of topics depending on the applications and the nodes been used. In order for the nodes to access a specific topic, they use Publish/Subscribe Protocol [Fig.54]. For example, if the perception node needs to get data from the Sonar sensors, this node will **subscribe** to the available Sonar topic available on the ROS network. On the other side if the motion control node [Fig.55] needs to send velocity to the robot, it will **publish** the necessary data over the *cmd_vel* topic. Sometimes, it is not sufficient for the nodes to use the Publish/Subscribe model only as it is as a one-way transport [30]. Services are very useful; it establishes a server and client relation between the nodes for quick request/reply interactions such as occurrence of an update in the navigation Map [Fig.54]. Moreover, it exists a similar component in ROS called Action [Fig.54], they are useful in requesting and setting a specific action. For example, the planning node [Fig.55] will send new goal location to the global map node and the latter will set this goal location [Fig.54].

ROS Bags

ROS Bags are very useful tools in ROS. They are used mainly as logging files for recording purposes. It can be used later on for data analysis. When the ROS network is operating, any topic of interest can be recorded and then playing it back again to visualize all data. Moreover, all the topics that are active over the ROS network can be recorded. In order to record all topics while ROS is running, type the following command in terminal:

rosbag record –a

In order to record only a desired topic, type the following command in terminal:

rosbag record <topic-name>

In order to play back a recorded ROS Bag file, type the following command in terminal:

rosbag play <the recorded file.bag> [Fig.56]

While the ROS bag file is being played [Fig.56], all concerned topics are been displayed under the previous command *rostopic list* [Fig.57]. In order to see the data recorded by each topic [Fig.58], type the following command in the terminal:

rostopic echo <topic-name>

As seen ROS Bag feature is very important, it allow to work with data even if the actual hardware is not available. It sufficient to run the robot one time and record a bag file. Then, the

work with this file can be carried offline either for analysis and adjustment or in the development process of the algorithm. As the recorded ROS Bag files can be imported easily in the Simulink model and they take the role of the sensor data and the input to the algorithm, even if live sensor measurement is not available. ROS Bag in Simulink will be discussed later. One more feature that ROS bag offers and it may be useful is that it can record robot motion. When the bag file is played again, the robot will replicate blindly the same achieved motion.

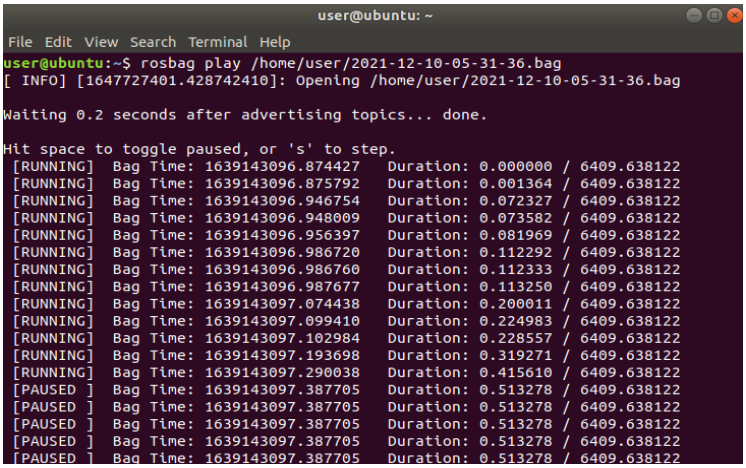


Fig.56 Playing Back a ROS Bag File in Ubuntu Terminal

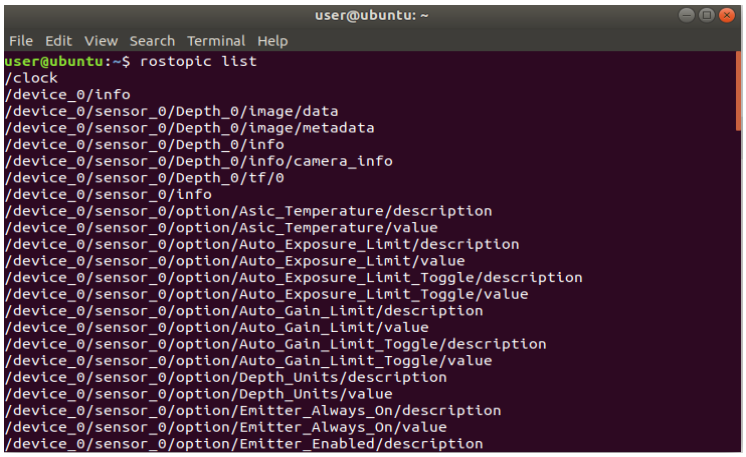


Fig.57 Some of the Available Topics from the Recorded Intel RealSense ROS Bag File

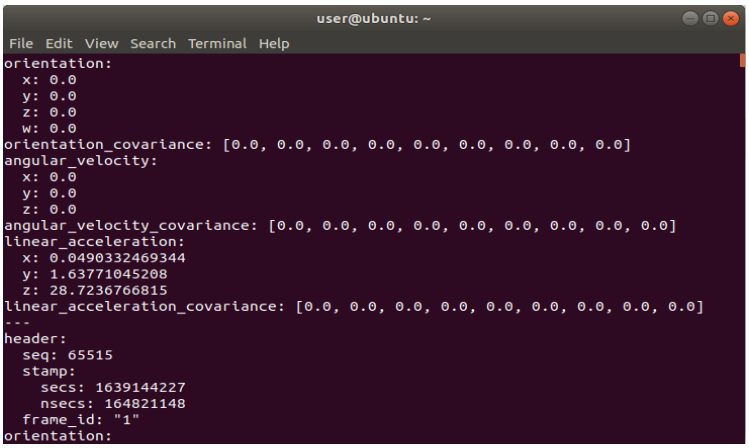


Fig.58 Playing Back IMU data from the Selected Topic from the Intel RealSense Bag File

rqt

Another useful tool provided by ROS is rqt. It is a mutli-purpose tool that serves mainly for monitoring of what happening over the ROS network and for diagnostics. It can display graphically, the connected active ROS nodes with the subscribed and published topics [Fig.59]. In rqt, dealing with ROS bags is more easier. Moreover, this tool provides a detailed topics monitoring and the possibility to publish easily on any desired ROS topics by a suitable command. The tool rqt offers many more useful features that can be explored when working with the interface [Fig.60].

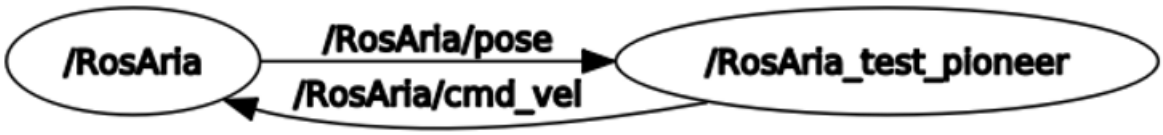


Fig.59 ROS Graph Visualized in rqt

The ROS Graph in [Fig.59] was recorded while working on the P3-DX robot, it show the original ROS node of the robot *(/RosAria)* connected the ROS node that was been generated from the Simulink model named */RosAria_test_pioneeer*. In this algorithm, the ROS node generated from Simulink extracts robot's pose by subscribing to the topic */RosAria/pose*. Then accordingly, the pure pursuit controller in the */RosAria_test_pioneer* node calculates the desired velocity and heading rate which are published to the topic */RosAria/cmd_vel*, leading to the desired robot's motion.

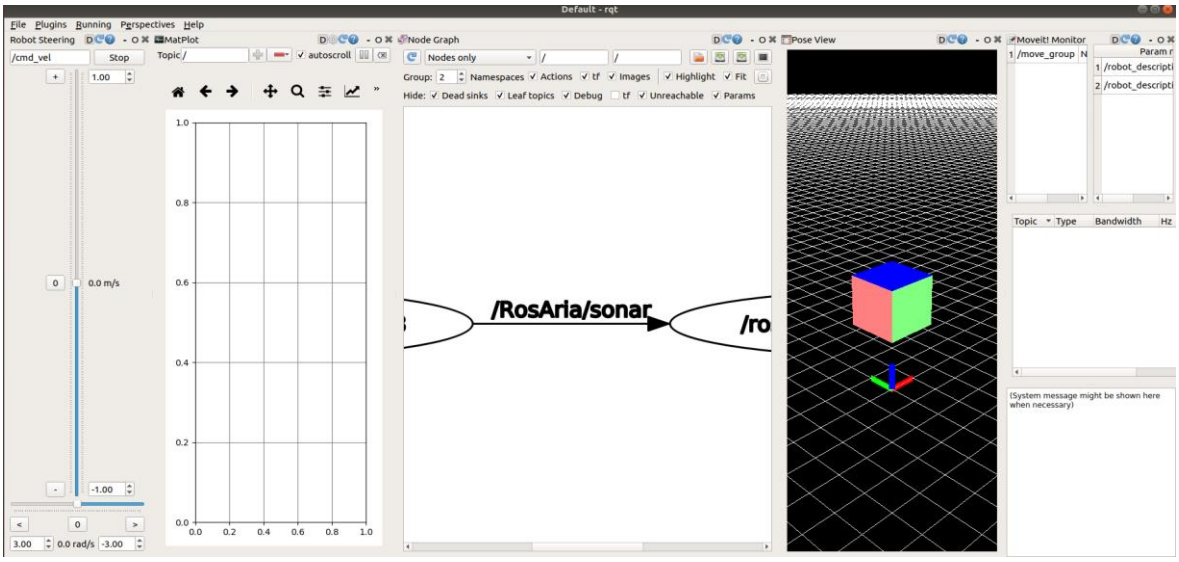


Fig.60 rqt tool Interface

Rviz

Rviz is an important tool within ROS, it is mainly intended for 3D visualization. It offers a view of the robot model, capture sensor information and replay capture data. Moreover, Rviz is important in the ability to display data from camera, laser from 3D and 2D devices including pictures and point clouds [31]. Rviz contains several other tools such as visualization of robot's pose and tracking vectors in a selected map that are important for tracking the robot's behavior and localization. This tool can be used also to recognize if the obstacles are correctly detected. Additionally, Rviz is a very important tool for SLAM algorithms.

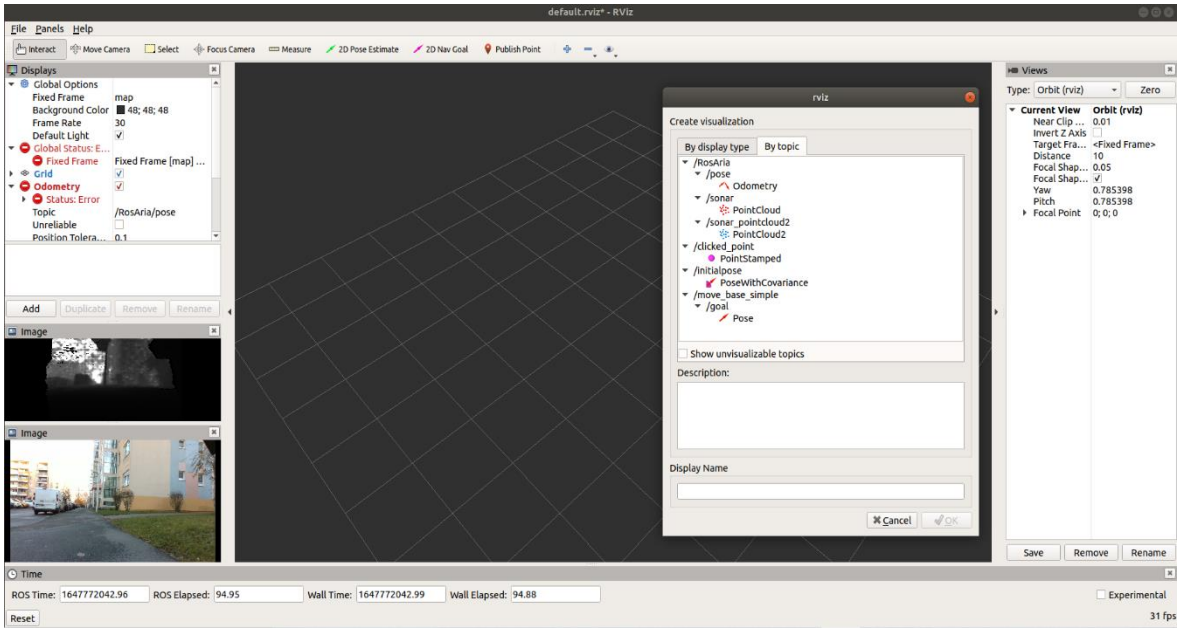


Fig.61 Rviz Interface with Intel RealSense and P3-DX Robot's Topics Visualization

ROS with Pioneer 3-DX Robot (RosAria)

As discussed in a previous section, in order to access the P3-DX robot, Aria package must be installed and configured. This is not enough in order to use the Robot with ROS for the algorithm implementation. For this reason, a specific package for the Pioneer robot must be installed on the ROS master computer. The dedicated ROS node for the pioneer robot is named *RosAria* which is mainly based and use the Aria library installed before. "rosaria" package can be installed by cloning the git into the catkin workspace from the following GitHub link [32]. All the installation procedure can be found on ROSWIKI on the following link [33]. After *RosAria* has been installed and configured on the master computer, now the overall robot setup become a ROS hardware. ROS node (*RosAria)* can be launched after launching first ROS master (*roscore*) as discussed before.

In order to run the *RosAria* node, type the following in a terminal:

```
cd catkin_ws
. devel/setup.bash                  //to access and source the Catkin Workspace
rosrun rosaria RosAria              //to access the rosaria package and run the RosAria node
```

After that, a connection to the robot via ROS is established by hearing an audible sound, the P3-DX ROS topics are available for usage by the algorithm. *RosAria* node should be launched first before running any other algorithms or nodes. The same procedure should be done for port settings and permission when connecting the robot to the Nvidia master computer as done in a previous section for running Aria examples on Nvidia Linux OS. After running the *RosAria* node, the original ROS topics for the P3-DX robot are available [Fig.62], such as for the pose, sonar, battery voltage, etc… these topics can be accessed as usual by the (*rostopic list*) command.

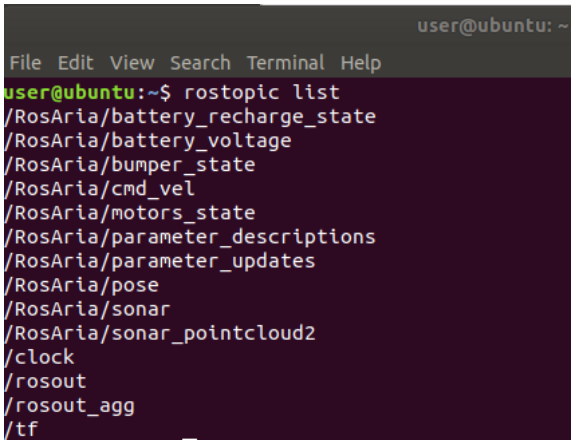


Fig.62 RosAria Available Topics List for the Pioneer Robot

[Fig.62] shows all the available topics when running the RosAria node, these topics are used by the algorithm in order to send and receive data from the robot. For example, the algorithm in the perception block can subscribe to the topic /RosAria/sonar [Fig.62] that gives the Sonar readings. This topic, will give the (x,y) coordinate of the detected obstacle by each Sonar sensor [Fig.63] according to the internal coordinate system frame of the P3-DX robot [Fig.32]. On the other hand, the topic /RosAria/sonar_pointcloud2 will give sonar readings in another point cloud matrix format [Fig.64]. In order, to extract the robot pose measured by the wheel encoders, the algorithm can subscribe to the topic /RosAria/pose [Fig.65] that give the gives the data in Quaternion coordinates [Fig.65] and need to be transformed to Euler coordinates which can be done in Simulink as discussed before. Moreover, if the algorithm needs to send velocity commands to control the robot, it can publish on the /RosAria/cmd_vel topic [Fig.62].

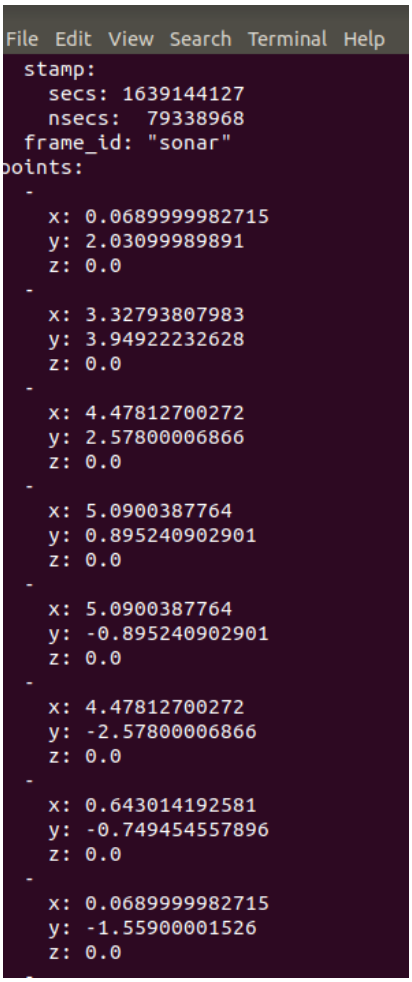


Fig.63 Data from /RosAria/Sonar

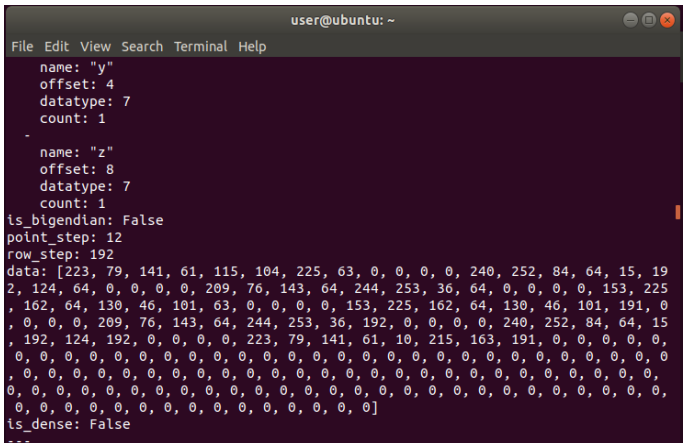


Fig.64 Data from /RosAria/sonar_pointcloud2 Topic

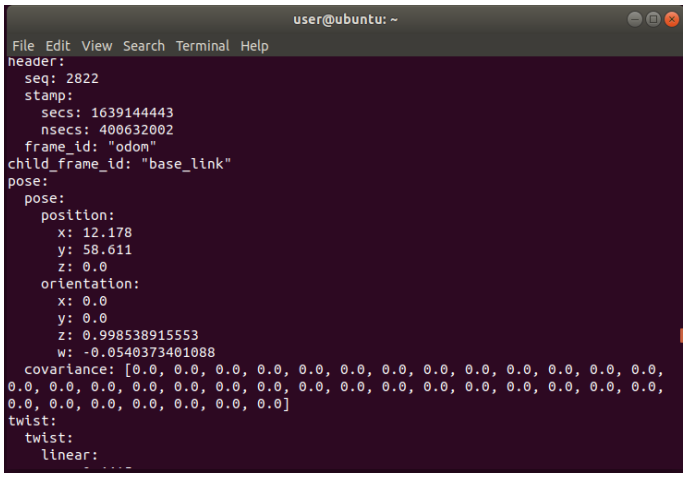


Fig.65 Data from /RosAria/pose Topic

ROS client package Demo for the Pioneer 3-DX Robot (rosaria_client_interface)

After installing the main ROS package (*rosaria*) for the pioneer robot. It exists additional package that can be installed named *rosaria_client_interface*. This node consists of several useful combined codes. When running the *rosaria_client interface* node, an interface pop up in the terminal that allows to select different modes by the suitable key. The most important mode in this node is the teleoperation mode which allow to control the robot by a keyboard. The advantage of this mode in comparison with previous Aria teleop example, is that the teleop mode inside the *rosaria_client interface* node will run in ROS environment. This will allow to visualize the ROS topics and benefit from the ROS bag feature while driving the robot. For example by this mode, the pose topic data can be recorded and these data can be used later in the construction of the waypoints for path planning.

To install the **rosaria_client_interface** package, the following git can be cloned to the source file inside the catkin workspace from the following GitHub link [34] with all necessary instructions. After the package is cloned to the source file inside the catkin workspace, the command *catkin_make* can be used in the terminal in order to make the package ready for use.

In order to launch the *rosaria client interface* node, it exists two methods. The first one is to launch manually the ROS master then rosaria and then rosaria_client interface nodes, by typing the following in each of the three terminals:

Terminal 1: roscore // ROS master is launched

Terminal 2:

```
cd catkin_ws
. devel/setup.bash
rosrun rosaria RosAria          // RosAria node is active
```

Terminal 3:

```
cd catkin_ws
. devel/setup.bash
rosrun rosaria_client interface  // the interface is active and the desired mode can be selected
```

Additionally, it exists an easier way to launch the client interface node by using a launcher file that launch the ROS master and the two other nodes automatically by one command. To run the client interface using the automatic launcher method, open one terminal and type the following:

```
cd catkin_ws
export ROS_HOSTNAME=localhost                    *
export ROS_MASTER_URI=http://localhost:11311     **
. devel/setup.bash
roslaunch rosaria_client rosaria_client_launcher.launch
```

The two commands lines (* and **), need only to be written in the case when using the *rosaria client interface* demo without internet on the master computer. When, there is an active internet connection on the ROS master computer, those two commands lines can be ignored.

Bridging ROS and Simulink

After ROS middleware is ready on the master computer, the P3-DX robot can now be manipulated directly by Simulink which is an important target in this project. Simulink offers very important blocks in the library browser inside the ROS toolbox [Fig.66]. They are used for the interaction with the ROS device.

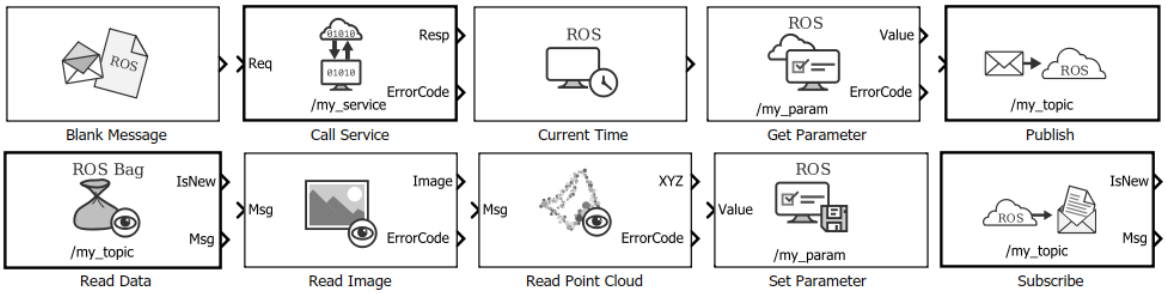


Fig.66 Simulink Blocks for ROS Applications

Beforehand, there is a small step to be done in Simulink in order to allow successful bridging with ROS. After launching Simulink, the Robot Operating System (ROS) should be selected from the APPS toolbar [Fig.67].

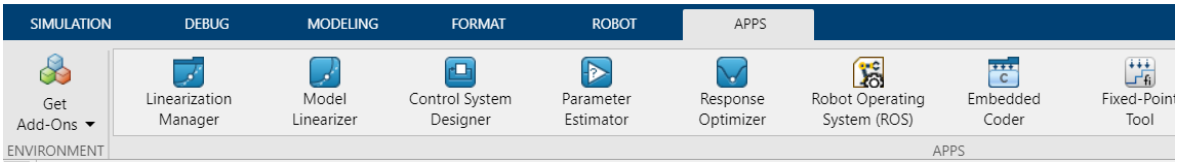


Fig.67 Simulink APPS Toolbar

Then from the upper tab, "Connect to Robot" should be selected [Fig.68]. After that, a box popup in order to connect to the ROS device. In that box [Fig.68], the device IP address should be entered with the username and password that are used usually to login to the ROS device.

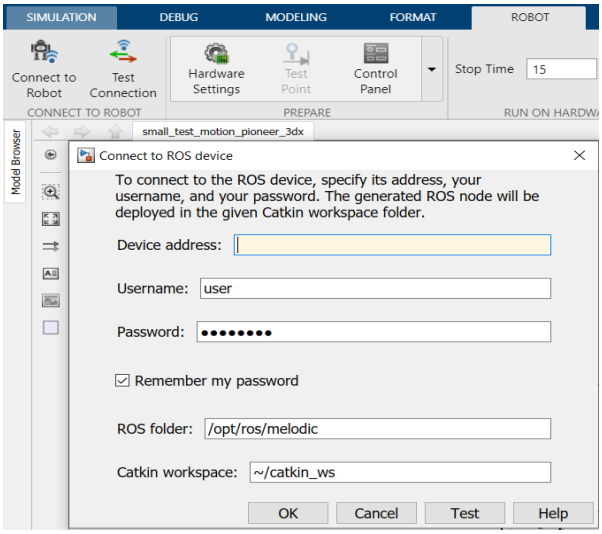


Fig.68 Connecting to ROS Device from Simulink

Moreover, the ROS folder according to the installed version must be specified with the correct name of the catkin workspace to let the algorithm to be deployed directly there. After the required setup, a test connection must be done for checking.

After successful connection with the ROS device, Simulink and ROS become bridged together. At this stage, a Simulink model can be deployed into the ROS hardware either by the desktop prototyping method or as a standalone node deployment on the target device. For this purpose, a small Simulink model has been implemented on the Pioneer 3-DX robot in order to test the Simulink ROS toolbox blocks and achieve first motions on the robot by this bridging method. This simple Simulink model [Fig.69] was created to only send some velocity commands to control the robot. This model will run directly on the ROS device by the desktop prototyping method, where directly a ROS node will be generated inside the catkin workspace on the target hardware after the Simulink model is compiled.

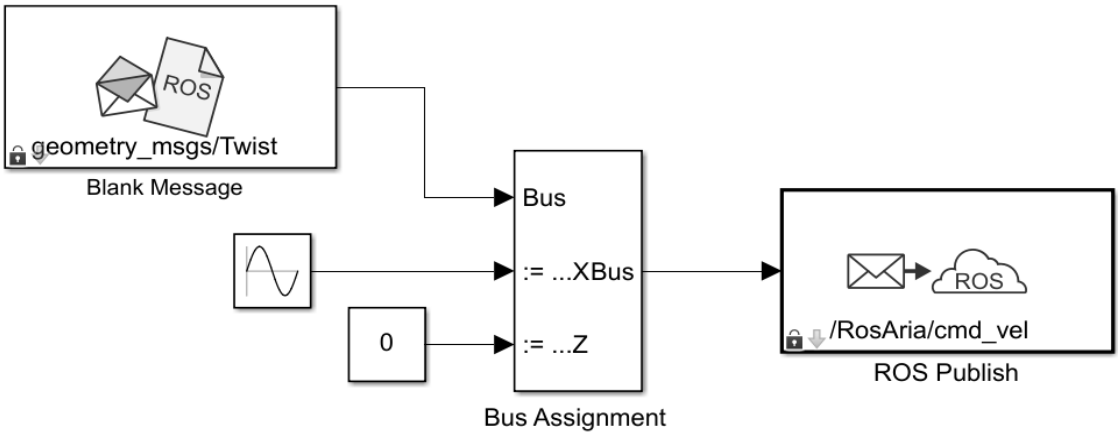


Fig.69 Simulink Model for Sending Velocity Commands over ROS for the P3-DX Robot

In the Simulink model [Fig.69], ROS Publish block is used to send velocity commands to the robot. Inside this block parameters box [Fig.70], the topics of the connected robot appears, and the suitable topic can be selected to publish on it. Here the topic */RosAria/cmd_vel* was selected. This topic uses the *geometry_msgs/Twist* type as seen in [Fig.70].

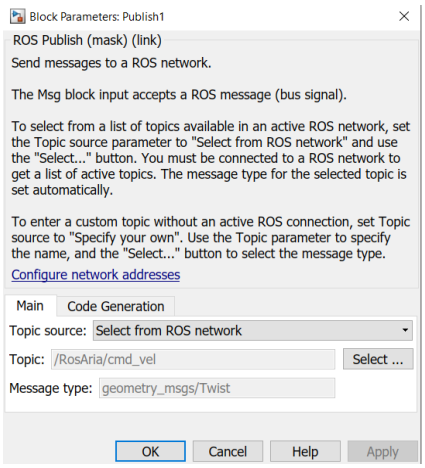


Fig.70 ROS Publish Block Parameters

Moreover, the ROS publish block accept only a Bus signal. For that reason, a ROS blank message block is used [Fig.69] to create a bus signal with a blank ROS message with the same type *geometry_msgs/Twist* needed for the publish block. Then, this blank bus signal will be assigned the needed value for the velocity command by mean of the Bus Assignment Block [Fig.69] that will output the required bus signal to the ROS publish block. In fact, this bus signal should contain the linear velocity (V) and the heading rate (ω) for the desired robot motion.

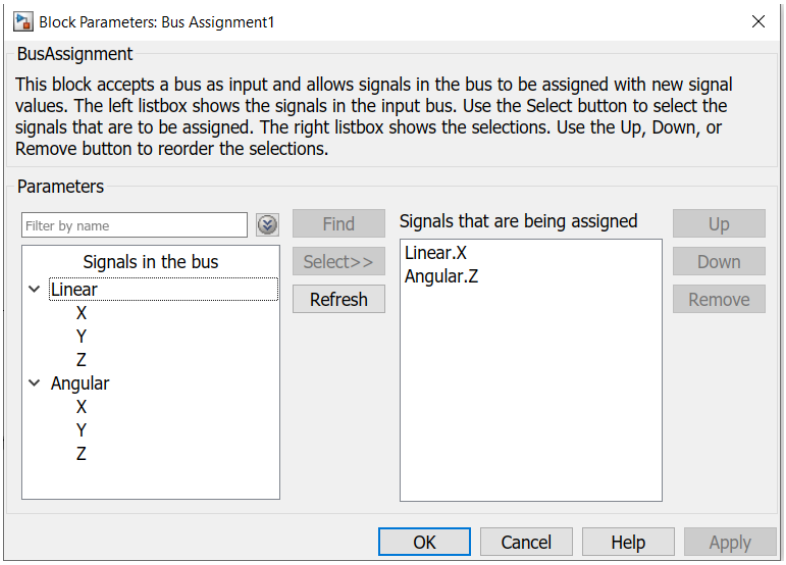


Fig.71 Bus Assignment Block Parameters

Generally, the *geometry_msgs/Twist* consists of several signals in the Bus. It consists of linear and angular velocities signals in (X, Y, Z) directions. Those signals are available in the Bus Assignment Block [Fig.71] in order to be selected for assigning the desired value. Since the longitudinal linear velocity (V) and the heading rate (ω) are the only interesting values for this motion control, the Linear.X signal for forward velocity (V) and the Angular.Z (angular rotation speed about Z-axis) signal for the heading rate (ω) are selected for assignment [Fig.71]. Depending on the desired motion, the suitable values for Linear.X and Angular.Z are assigned, then a bus signal is been created to be published on */RosAria/cmd_vel* Topic. For example, if need to achieve a forward motion, a positive constant value is assigned to Linear.X in the Bus Assignment block [Fig.69]. In case, a rotational motion is needed, a positive or negative constant value is assigned to Angular.Z depending on the desired rotational direction. In the example of [Fig.69], a sine wave test signal is assigned to Linear.X. In that case, the robot has achieved a back and forth periodical motion depending on the amplitude and frequency of the assigned sine wave signal. In another test, a ramp signal was assigned to Linear.X of the Bus Assignment block [Fig.69]. In that case, the robot has moved at a constant settled speed and then it started to accelerate depending on the settled activation ramp time and its slope in the ramp signal parameter block.

ROS Bags Usage in Simulink

As discussed before ROS Bag is a very important tool in the overall process workflow. Moreover, its usage with Simulink is very beneficial. It can play a vital role during the algorithm development. ROS Bags can be considered as the input data source for the algorithm, especially for the perception block when the possibility of the real-time data is not available. It can play the role of the Subscribe Block [Fig.66] in a virtual way. The latter block is used to access data for available hardware in real-time. To benefit from the advantage that ROS bags offers, it is sufficient to run a test drive on the real hardware and record the bag file from ROS environment. After that, this ROS bag file, containing the required data, can be used in Simulink for algorithm development as data source without the need to access the real hardware and the sensory setup. In this project, ROS Bags were used in Simulink in order to treat each data type independently for the proper adjustment at sub-level before combining in the main algorithm where ROS Bags will be replaced by Subscribe Blocks for direct access of the data from the real hardware and where the algorithm will be finally implemented. In this section, some of the ROS Bags recorded from the P3-DX robot during this project and implemented in Simulink, will be highlighted.

A ROS Bag block can be selected from the Simulink library browser inside ROS toolbox [Fig.66]. When adding the block to the Simulink model, the recorded ROS topics data will be available for selection [Fig.73] after loading the complete ROS Bag log file and selecting the topic of interest for data stream [Fig.72].

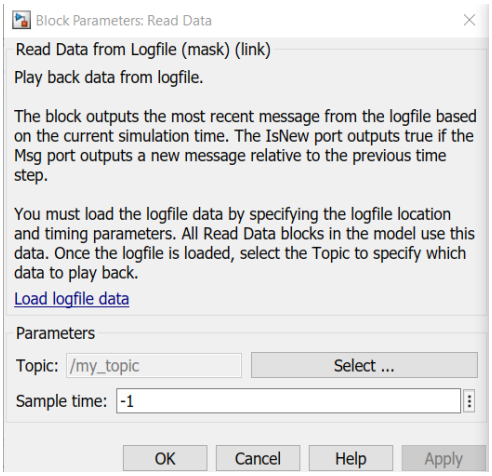


Fig.72 ROS Bag Block Parameters

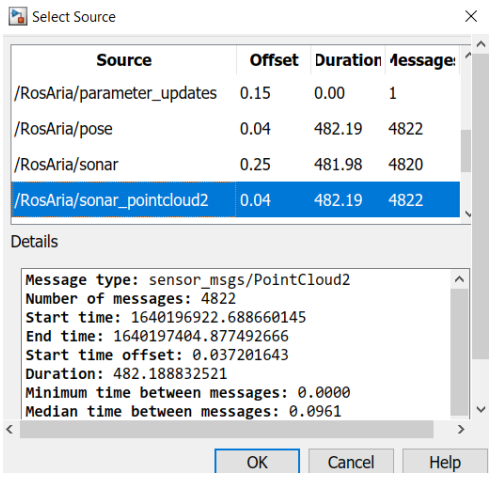


Fig.73 Available Topics for Selection

Sonar ROS Bag in Simulink

After loading a ROS Bag file recorded from the P3-DX robot while driving inside the university laboratory, the first implementation of the log file in Simulink was to extract the readings from the Sonar sensors. A simple Simulink model was been created for this purpose [Fig.74]. The aim from this implementation in Simulink was to access and display the Sonar readings in a

similar way as visualized in ROS terminal [Fig.63]. Moreover, a bus signal treatment will be applied to separate the bus elements from an array of sub-buses. After the bus signal treatment of the Sonar data, the readings from the eight different sensors can be achieved independently and without dealing with a complete merged Sonar point cloud array. By this way, the treated Sonar reading data can be used in building the environmental model from the environmental data as per [Fig.6] in order for to be used by the Decision making layer. Similarly, the independent eight Sonar sensors data can be used simply in a state flow algorithm for obstacle avoidance.

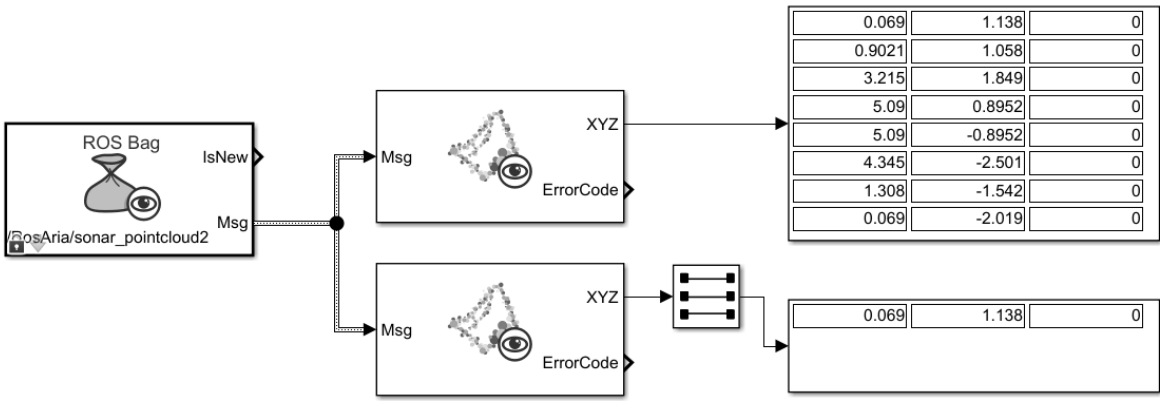


Fig.74 Simulink Model for Displaying P3-DX Sonar Readings from a ROS Bag file

As seen from [Fig.74], the */RosAria/sonar_pointcloud2* topic was selected for data sonar extraction. The ROS Bag block will give a bus signal message as an output consisting of an array of sub-buses of the (X Y Z) coordinates readings of the eight sensors in a point cloud format. Simulink offers a very useful block in ROS toolbox library, which is the "Read Point Cloud" block [Fig.66]. This block will transform any point cloud message received from a camera, Lidar, Sonar, etc… into (X Y Z) Cartesian coordinates of the points as [MxNx3] array. In the block parameters, the maximum point cloud size can be set to give only the interested points as an output. In this example the size was set to [8 1], in other way it means that only the readings from the eight first sensors will be displayed. In the upper display block [Fig.74], the (X Y Z) coordinates of the detect obstacles by the eight sonar sensors are displayed simultaneously. The first row is the reading from Sonar sensor number zero as per the Sonar array arrangement [Fig.14], until the last row which is the reading from the sensor number 7 [Fig.14]. The three columns in the display block [Fig.74] are the X, Y and Z coordinates respectively. It is important to mention that the points coordinates obtained are as per the internal coordinate system of the P3-DX robot [Fig.32]. For that reason, it can be noticed in the upper display block reading [Fig.74], that from the sensor number four, the Y-coordinates become negative. This fact is due to the location of the sensors from number four to seven on the negative side of the Y-axis according to the coordinate system [Fig.32]. Additionally in [Fig.74], another "Read Point

cloud" block was used with a "Selector" block in order to separate the signal of each Sonar Sensor and be displayed independently from the complete set. Inside the "Selector" block, the index and output size are indicated for the correct selection of signals. In the bottom display block [Fig.74], the signal from the first sensor number zero is displayed alone. Additional work is needed in term of bus separation signals in order to get the eight independent signals of the Sonar System. In [Fig.74], it was displayed in that way for demonstration purpose. After having the eight separate signals of the sonar system, each signal can be transformed into a vector and then its norm will be computed to obtain a nominal range value from each sensor.

Intel RealSense ROS Bag in Simulink

Another ROS Bag file recorded in ROS environment from the Intel RealSense D435i camera mounted on the robot [Fig.20], was used in Simulink. The ROS bag of the Simulink model [Fig.75], containing all the topics from the camera ROS node, it was been recorded in an out-door test in the city of Gera-Thüringen.

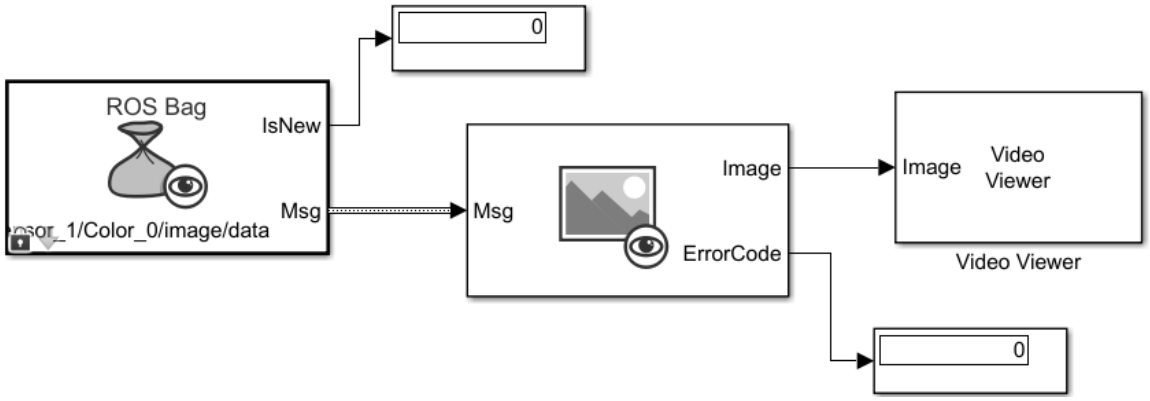


Fig.75 Simulink Model Designed for Displaying RGB Video from Intel RealSense ROS Bag

In the Simulink model [Fig.75], the used ROS Bag file contains all ROS topics available from the Intel RealSense ROS package. As discussed before these topics range from the color RGB image information, to depth information, IMU data from the integrated unit inside the camera and many more. The topic */device_0/sensor_1/Color_0/image/data* was selected to access the color RGB image/video data from the available topics in the Bag file [Fig.76]. Simulink offers a useful block "ROS Read Image" available in the ROS toolbox [Fig.66]. This block, added to Simulink model [Fig.75], will allow to extract image signal from ROS image message. In this block parameters box, the image encoding type can be selected and it was set to **rgb8** for normal color rgb image display. Moreover, the maximum image size can be configured for the correct display. After some trials, the suitable image size was found to be [540 , 960].

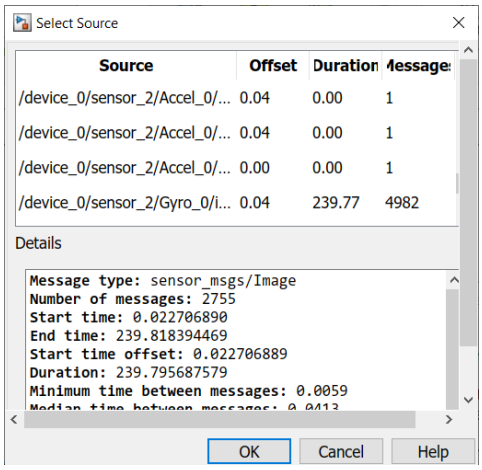


Fig.76 List of Available Topics from Intel RealSense ROS Bag

After the "ROS Read Image" parameters has been set in model [Fig.75], the output image signal can be used within a vision algorithm or simply its output can be displayed by using a "Video Viewer Block" form the computer vision toolbox within Simulink. This block allows to stream and display the video from the image signal, in a separate window within Simulink [Fig.77].

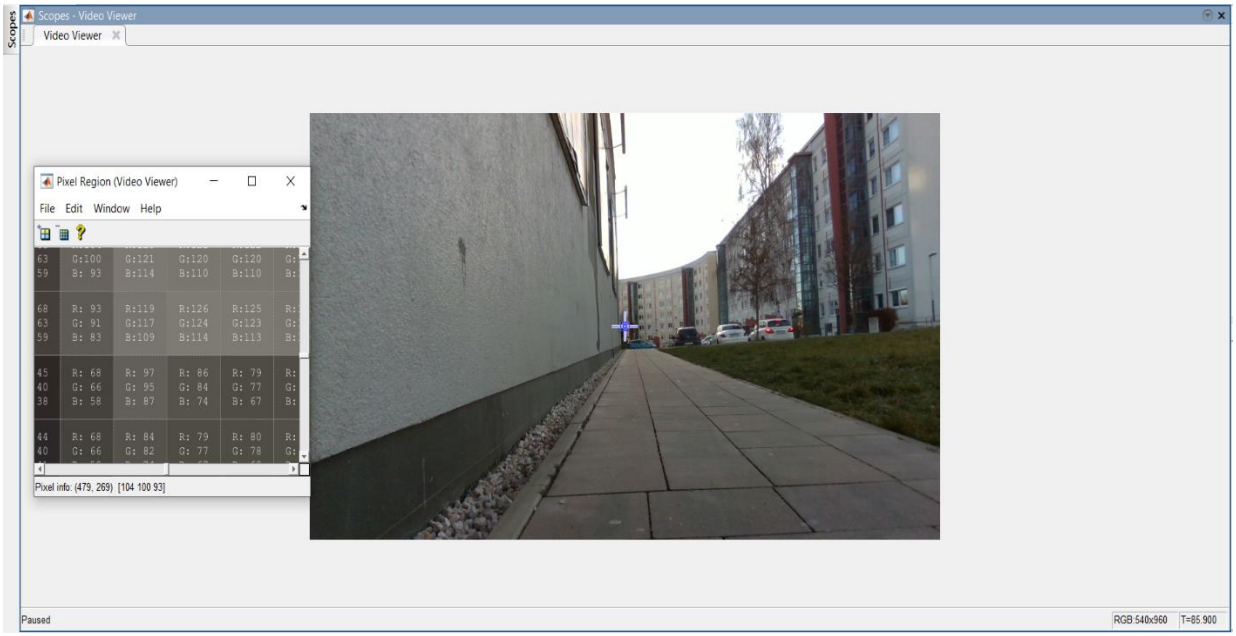


Fig.77 Gera Robot's Test-Drive Video Stream in a Dedicated Matlab Scope

In the same Matlab window streaming the video in [Fig.77], some tools of the computer vision toolbox can be used; such as the used "Pixel Region" viewer in [Fig.77]. Moreover, during the video stream, any image can be exported directly to the image tool where the image processing toolbox can be applied.

Using the computer vision toolbox inside Simulink, a very simple implemmation example was performed on a image extracted from ROS image message of the Bag file of Gera test. This example [Fig.78] created in Simulink, will use the Edge detection block from the mentioned

toolbox. This feature will apply Sobel algorithm to detect edges from the input color image which was separated into R, G, B signals. The Sobel edge detection block was applied into each of the three signals only for demonstration purposes.

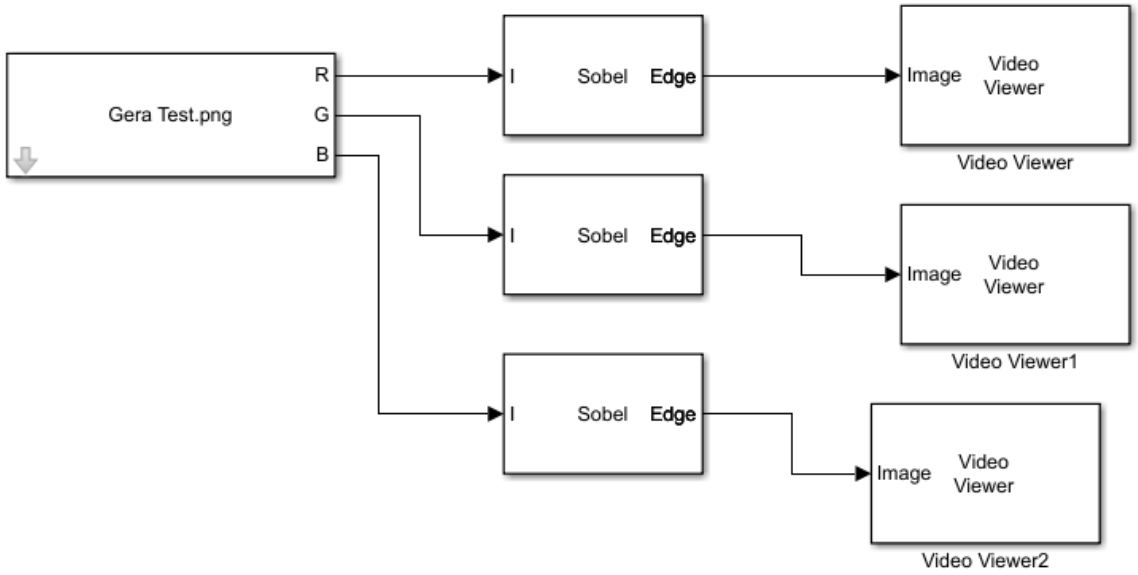


Fig.78 Sobel Edge Detection Algorithm Applied on an Image Extracted From ROS Message

The Sobel Edge detection block [Fig.78] will give a binary image with the detected Edges. The result can be visualized, by using "Video Viewer" block that will display the image in separate scope [Fig.80]. [Fig.79] is the original color image and [Fig.80] is the processed image resulting from the Red (R) signal of the original image. The detected edges of different objects of [Fig.79] are shown in the processed image [Fig.80], where the edges of road pavememts, clouds in the sky, bicycle, cars, etc... can be seen.

Fig.79 Original Image1 from Gera Test

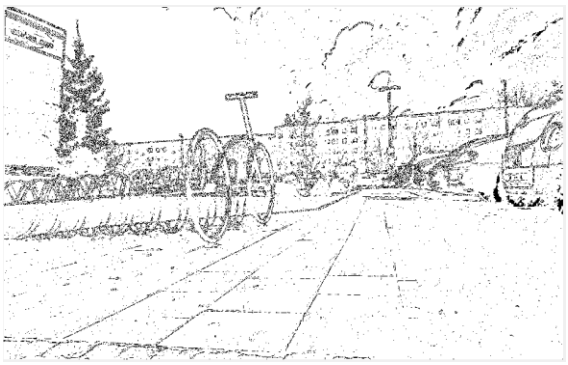


Fig.80 Processed Image1 from Gera Test

Similarly, another image [Fig.81] was processed in the same way [Fig.82]. In the processed Image [Fig.82], the edges of the sidewalk where the robot operates can be seen. Moreover, the

edges of different objects in [Fig.81] can be noticed in [Fig.82]; such as the cars parked next to the sidewalk, the buildings, etc…

Fig.81 Original Image 2 from Gera Test

Fig.82 Processed Image 2 from Gera Test

The algorithm for edge detection [Fig.78] is a very basic one. As can seen in [Fig.80] and [Fig.82], the result is not sufficient as not all the edges were detected clearly. This is due to several facts such as the condition of the original image where in [Fig.81] some parts of the sidewalk extremities are not visible. This affects the processed image in [Fig.82], where some line edges were not detected and missing. The overall process requires more filtering and the usage of more complex vision algorithms that can be done within Matlab/Simulink or other open sources platform such OpenCV. The algorithm of [Fig.78] was only used for demonstration purpose in order to show a simple image processing in Matlab/Simulink that can be done with an image extracted from a ROS message. A more advanced vision algorithm is planned in the future scope of this project. Finally, after a ROS image message was processed after being extracted from the Intel RealSense ROS Bag file, similar data process and manipulation can be done with other topics data inside the same Bag file [Fig.76] and not necessary related to vision topics. One example is using the IMU data from the same Bag file with an appropriate designed Simulink model for extracting necessary data for localization. This process will also be considered in the future scope of this project, where it will be used within a fusion algorithm for precise localization.

Pose ROS Bag in Simulink

Another implementation was done is Simulink using the ROS bag recorded from the P3-DX robot during a test drive inside the university laboratory. The ROS bag in this section has the topic contents from where the Sonar was extracted in a previous section. This time, pose data will be extracted from the bag file in a Simulink model. This manipulated data will allow to display the real pose vector $[x\ y\ \theta]^T$ of the robot at each time instant during this test. Moreover, these treated data will be used to generate a waypoint Matrix that can be used in the path planning block, instead of building these waypoints manually or import them from a map. This advantage will be discussed more in detail in the next chapter.

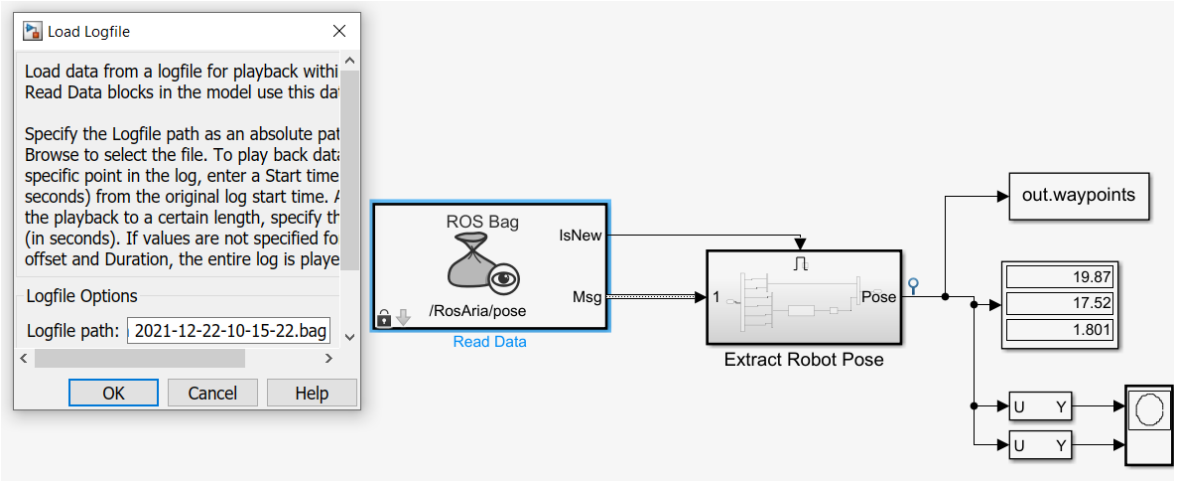


Fig.83 Simulink Model for Extracting Robot's Pose from a ROS Bag File

The ROS bag file recorded on 22/12/2021 in the university laboratory was used in the designed Simulink model [Fig.83] for Pose extraction. The */RosAria/pose* topic was selected in the main ROS Bag file. The Bag block will give a bus message signal containing all pose information. Similarly as done in the implementation in Simulink chapter before Simulation testing phase, will be done here. In the Sensing block [Fig.41], the pose data was gathered from Gazebo virtual data and the bus signal was treated as in [Fig.42] in order to get the data in Cartesian coordinates. The similar method will applied to the Simulink model [Fig.83] where here the data source is the Bag file recorded from real test. The Bus message signal from ROS Bag block will be treated inside the Extract Robot Pose block [Fig.83]. In this block, the same sub-components for bus signal selection and coordinate transformation are used similarly to [Fig.42]. The bus signal arriving to the Extract Robot pose block is treated in a bus selector block where the signals of interest are selected, similarly to [Fig.42]. Inside this bus selector block, the component of the bus signal are displayed on the left [Fig.84].

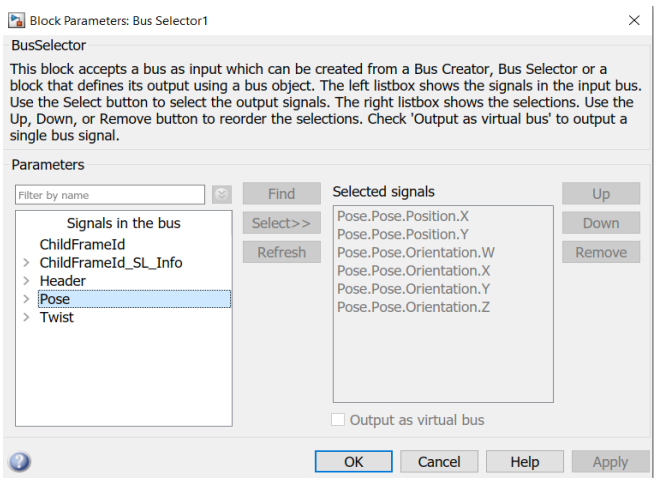
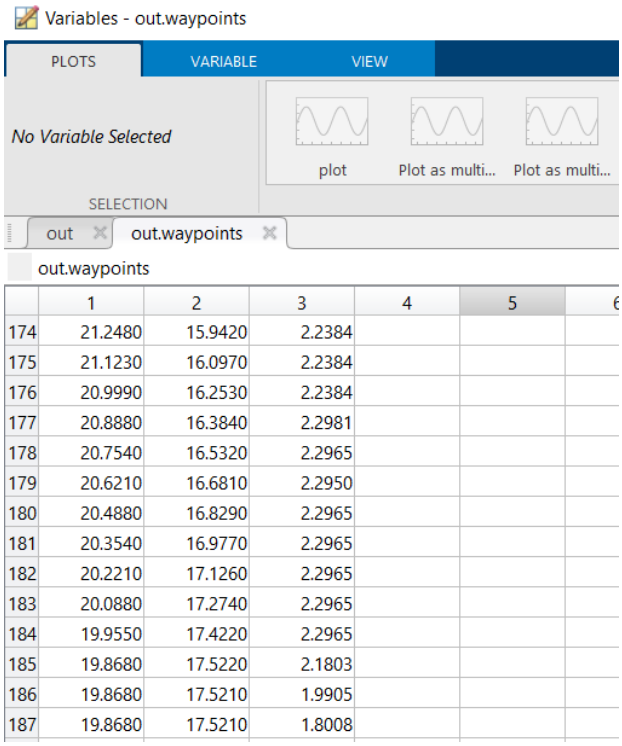


Fig.84 Bus Selector Block Parameters

The most important signals in the main Bus are Pose and Twist [Fig.84]. Pose as known is designated for position and orientation. Additionally, Twist is designated for linear and angular

velocities data. Inside the Main Pose signal in Left column [Fig.84], the following sub-signals are selected in the right column [Fig.84]. The first two sub-signals **Pose.Pose.Position.X** and **Pose.Pose.Position.Y** are selected. These sub-signals are directly robot position in (X,Y) Cartesian coordinates. The next four sub-signals [Fig.84**], Pose.Pose.Orientation.W, X, Y, Z** are Quaternion coordinates and need to be transformed to Euler (Z Y X) in order to get the rotation angle around Z-axis (Known as heading angle or Yaw angle). This coordination transformation are done by a specific block as in [Fig.42]. After that, the resulting three signals are merged together in MUX block to give as an output, the pose vector $[x\ y\ \theta]^T$. The pose vector from the Extract Robot Pose block can be visualized in a display block. For example in [Fig.83], the pose vector was displayed at an instant where (X= 19.87m,Y= 17.52m, θ= 1.801rad). In addition, an XY plot can be generated from the pose data by using selector blocks to identify the data to be plotted from the pose signal. It can be noticed from the Simulink model [Fig.83] that exists an additional block "out.waypoints". This block is very important, it allows directly to export all the pose data [x y θ] into the Matlab workspace. The exported data will be in the form [Nx3] matrix [Fig.85], where N is number of points recorded by three columns for [x y θ]. After that, the first two columns in [Fig.85] are used to form the (X,Y) waypoint matrix that will be imported into the path planning block in the main algorithm, without the need of manual definition or selection of these waypoints.



	1	2	3	4	5
174	21.2480	15.9420	2.2384		
175	21.1230	16.0970	2.2384		
176	20.9990	16.2530	2.2384		
177	20.8880	16.3840	2.2981		
178	20.7540	16.5320	2.2965		
179	20.6210	16.6810	2.2950		
180	20.4880	16.8290	2.2965		
181	20.3540	16.9770	2.2965		
182	20.2210	17.1260	2.2965		
183	20.0880	17.2740	2.2965		
184	19.9550	17.4220	2.2965		
185	19.8680	17.5220	2.1803		
186	19.8680	17.5210	1.9905		
187	19.8680	17.5210	1.8008		

Fig.85 Exporting Pose Data [X Y θ] into Matlab Workspace

Simulink-ROS Algorithm Implementation for a Simple Automated Drive

In this section, a designed Simulink model inspired by the model used in the simulation phase, will be implemented on the P3-DX robot as ROS hardware by using the bridging method discussed previously. Moreover, the components of the concerned algorithm [Fig.86] are based on the designed sub-models discussed in the ROS Bags in Simulink sections.

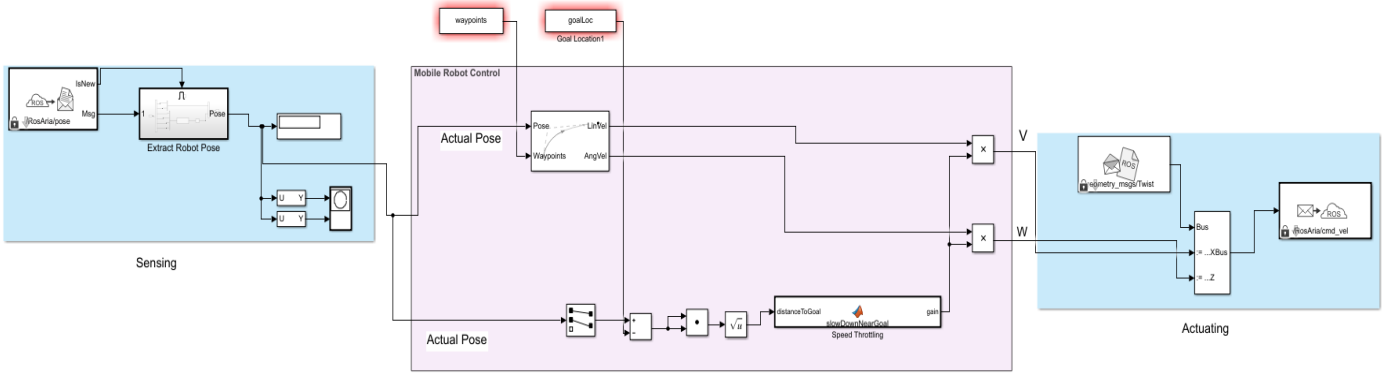


Fig.86 Simulink Model Deployed on the Robot as ROS Hardware for a Simple Automated Drive

The Simulink model [Fig.86] will be transferred into a ROS node named "RosAria_test_pioneer_node". This generated node will be deployed as a standalone ROS node on the target hardware, the Nvidia Jetson Nano mounted on the robot. But before the final deployment, the algorithm was been tested in the university laboratory by the desktop prototyping method [Fig.53], in order to observe the path plots in Matlab and tune the parameters when deemed necessary. The mentioned algorithm [Fig.86], will allow the robot to drive in automated way from Lab A to Lab B in the university laboratories building. The Simulink model [Fig.86] consists mainly of three main functional blocks; sensing, robot's motion control and actuating. The sensing block is intended mainly to extract the pose from wheel encoders for localization. The structure of this block is identical to the model used in Pose ROS Bag in Simulink section [Fig.83]. The only difference in the Sensing block of [Fig.86] is that the ROS Bag block is replaced by a ROS Subscrib block [Fig.66]. This subscriber block will let the access the topic */RosAria/pose* for direct live access of the pose data from the P3-DX robot. After the same procedure done before for the signal treatment and coordinate transformation, the pose vector in Cartesian Euler coordinates will be forwarded to the motion control functional block [Fig.86]. In the latter block, the actual pose information and the desired waypoints are fed to the Pure Pursuit controller that will calculate accordingly the actual linear velocity (V) and heading rate (ω) for the longitudinal and lateral motion control. The controller parameters for this test was set to 0.3 m/s for desired linear velocity, 0.8 rad/s for maximum heading rate and the look ahead distance factor was set to one. The latter factor was found after many trials to best at one. During testing on the real robot, it was recognized that look ahead factors below one, would cause the robot to oscillate too much between the waypoints leading in some cases

to complete loss of the desired path. Moreover, larger values has led to big shortcuts of the curves in the desired path. It was normal to encounter those effects and it was expected to happen in the real testing as the theoretical concept of pure pursuit controller was been highlighted in a previous chapter and specifically in [Fig.37] and [Fig.38]. Back to the Simulink model in [Fig.86], it exists additional function in motion control block; the speed-throttling block. The functionality of this block was discussed before as it the same that been used in the main algorithm for Simulation in Gazebo. The speed-throttling block will need the actual pose and goal location in order to track the remaining distance from arrival. This block will allow the robot to slow down before reaching its destination at predefined range and to stop completely when reaching the target at predefined proximity. The output of this block is a constant gain value that will be multiplied by V and ω for the desired motion control. After that, the continuously computed linear velocity and heading rate values will be forward to the final actuating block [Fig.86] for the controlling the P3-DX robot. The structure of the actuating block [Fig.86] is the same that was used in [Fig.69] in order to send velocity commands from Simulink to the robot over ROS. In the actuating block, a bus signal is created and the desired signals are assigned to it as in [Fig.71]. Then, the created bus signal is sent to the target ROS hardware by using a Publisher ROS block [Fig.66]. This block will publish the data to the */RosAria/cmd_vel* Topic. Then, the velocities are transformed to angular wheels speeds for achieving the desired motion from the robot. On the other side, it is important to mention that when deploying the algorithm into the ROS hardware with the standard Simulink settings, the robot behavior is not appropriate. From the first implementations with standard settings, it was observed that the robot achieve the desired path by a non-continuous motion. It drives step-by-step. This motion is undesirable and take too much time to achieve a small portion of the path. The reason for that behavior is that Simulink uses a fixed step solver in algorithm implementation on real hardware, in contrast with the variable step solver used with the algorithm in Simulation implementations in Gazebo. As known, the variable step solver cannot be implemented on a real embedded controller hardware. To solve that issue, the fixed step size must be changed from auto value to a defined value. After some trials and analyses, it is found that a value of 0.02 for the fixed-step size (fundamental sample time), is good to achieve a smooth continuous motion after the algorithm been implemented on the P3-DX robot. In order to set this parameter, the hardware settings must be accessed in the Robot upper Simulink tab. Then, in configuration parameters window [Fig.87] after selecting Solver tab, the desired value of 0.02 can be set in the solver details for the fixed-step size (fundamental sample time) [Fig.87].

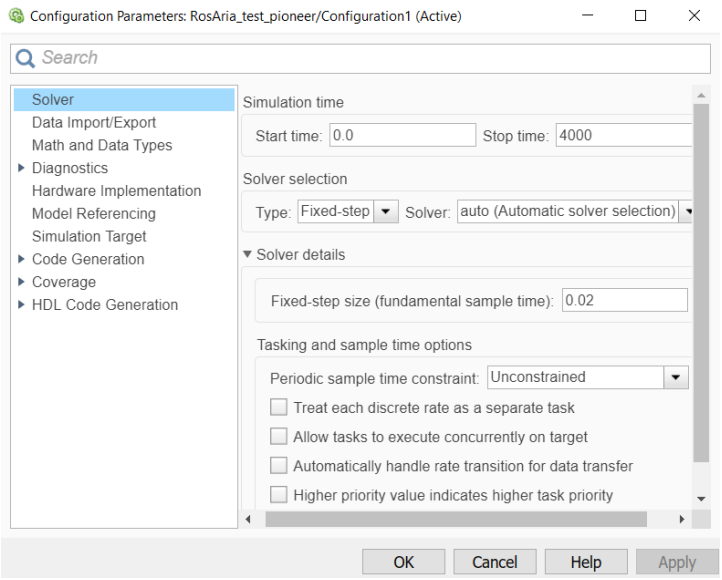


Fig.87 Configuration Parameters Window in Simulink

In the Simulink model [Fig.86], the definition of waypoints was done by two methods. First, the waypoints was defined manually in Matlab and saved to the workspace. To achieve that method, a origin point was set physically in the Lab A where the robot need to start, then the coordinates where measured physically toward the final destination point in Lab B in a relative way. After the measurement from the real operation domain is done and the waypoints coordinates are noted , they were saved in the Matlab workspace in order to be used by the algorithm for the path-planning functionality. The defined waypoints were plotted in Matlab in order to visualize the desired path from Lab A toward Lab B [Fig.88], inside the university laboratories building.

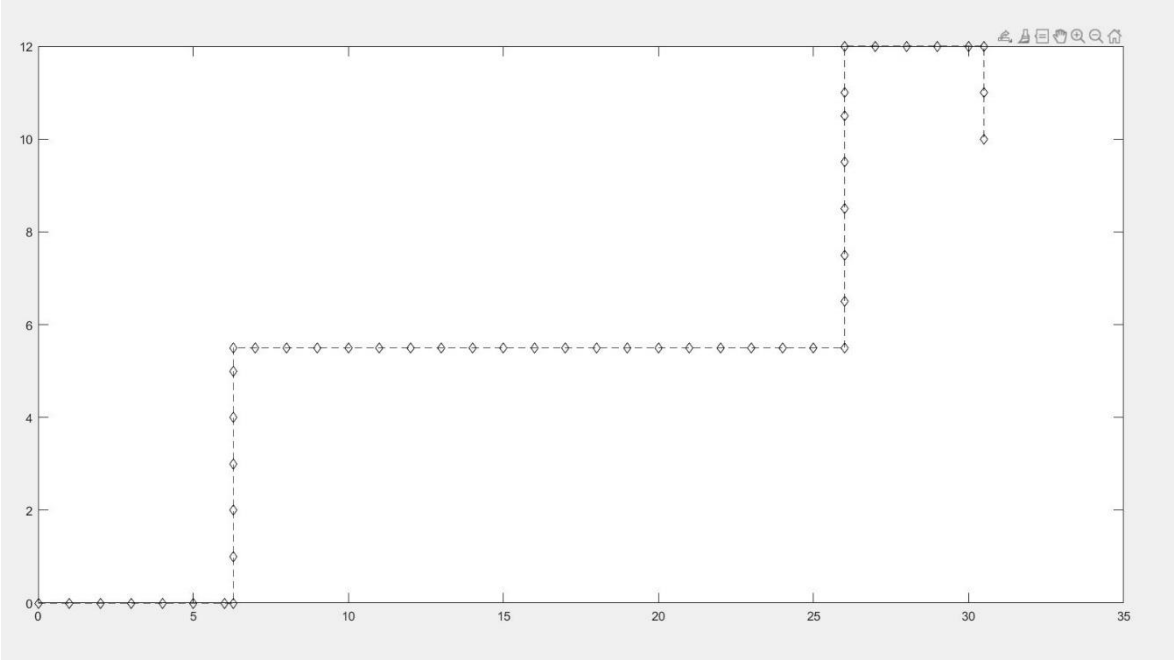


Fig.88 Desired Real Path Plot from Waypoints Coordinates (Y-axis versus X-axis) in Meters

After the waypoints matrix has been saved into the Matlab workspace, the algorithm [Fig.86] was deployed on the P3-DX robot. The pure pursuit controller in [Fig.86] calculates according to the waypoints [Fig.88] and other the defined parameters, the linear velocity and heading rate for generating the trajectory. During the implementation of the algorithm [Fig.86], the robot was observed to achieve the path [Fig.89] that was been visualized directly in Simulink.

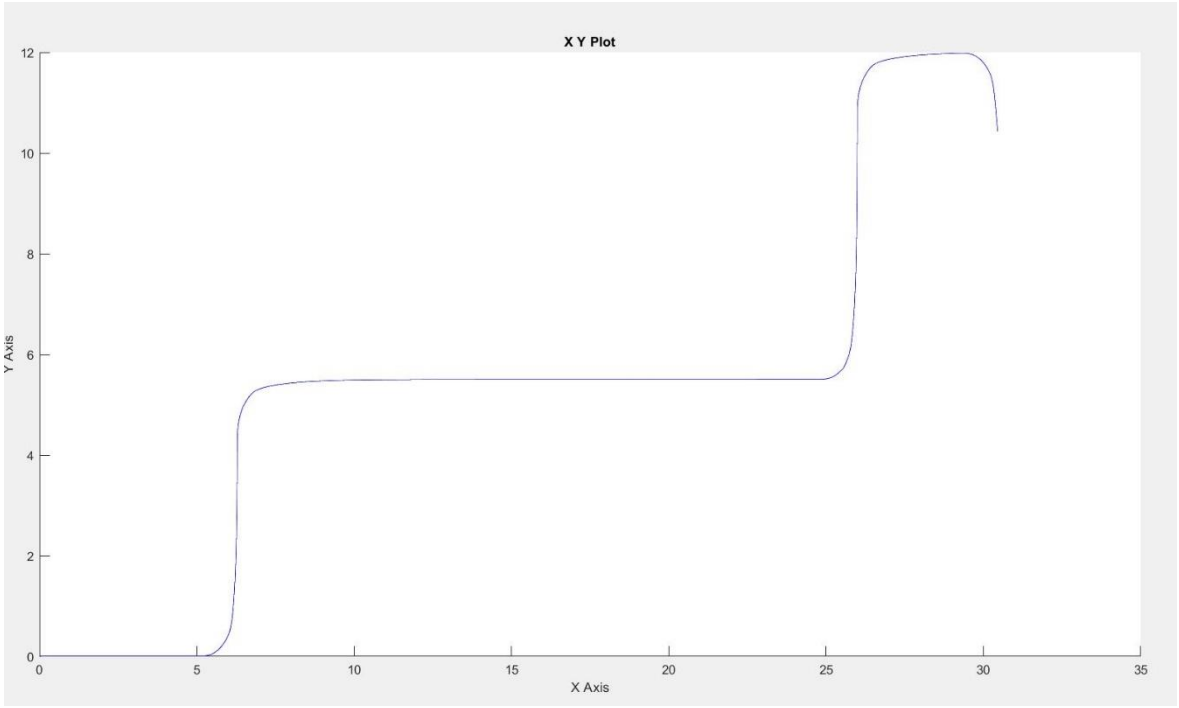


Fig.89 Achieved Path by the Robot from Lab A to Lab B using Pure Pursuit Controller

Comparing [Fig.88] to [Fig.89], the two plots are quiet similar. The only difference is the rounded curves at the corners [Fig.89] that are expected results from the Pure Pursuit Controller as discussed before.

When looking to the plots [Fig.88] and [Fig.89], it seems that the robot has achieved perfectly the path from Lab A to Lab B. In fact, the robot has failed to achieve the desired path between the two labs in an automated way. It has drifted many times along the path where manual intervention were needed to keep it on the right track to reach the destination goal location. The drifting behavior was not been noticed in [Fig.89] due to one main reason. As known wheel odometry tends to drift over time, the computation of the pose and done in relative way and according to the internal coordinate system of the robot, not global. In this way the robot do not recognize that there is drift from the real path, as in real world this drifting behavior can be observed clearly. In order to improve the robot behavior, by relying also on wheel odometry alone, another method was implemented in generating the waypoints rather than manual measurement from the real world. This method will rely on running the robot in a manual test drive from Lab A to Lab B and then back. To achieve this manual test using ROS environment, the Robot was driven manually using the *rosaria_client interface* node mentioned in a previous

section, while recording a ROS Bag file form the manual test-drive. After that the ROS bag file containing the pose data of the robot along the driven path, was used within a Simulink model [Fig.83] to extract robot's pose in the desired coordinates and then the *out.waypoints* block was used in the model [Fig.83], for exporting directly the pose data [x y θ] into the Matlab Work-space. From this data, only the first two columns were been saved in a new matrix in the work-space. This new waypoints matrix consisting of the (x, y) points coordinates, contains the real data including the drifting effects. The desired path for the robot to follow using this method, was been generated when playing back the ROS Bag file in model [Fig.83]. As a result, the desired predefined path was obtained in [Fig.90].

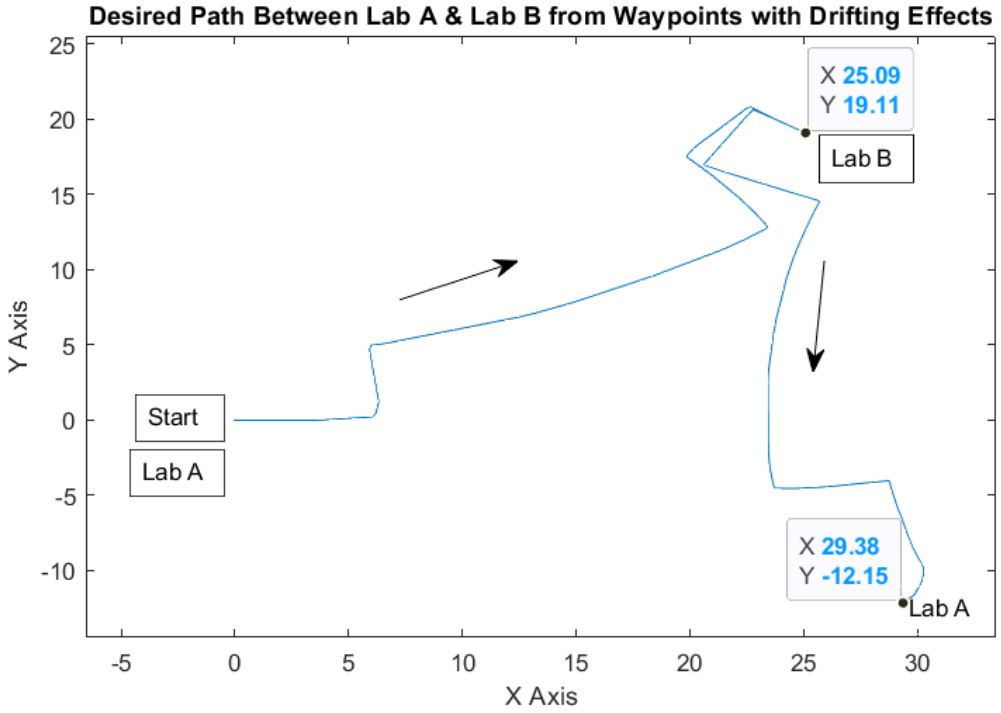


Fig.90 Desired Go-Return Path between Lab A & Lab B Constructed by Waypoints w/ Drift Effects

The predefined desired path plotted [Fig.90] in Matlab, was generated from the waypoints extracted from the pose recorded in the Bag file. These waypoints coordinates includes the drift effect resulting from the wheel odometry. Comparing the path only in the going direction from Lab A to Lab B [Fig.90] to the same path between same labs measured by scale in real world [Fig.88], it can be noticed that the path layout is the same but with drift in coordinates is obvious. For example, the real coordinates of the final point in Lab B is (30m , 10m) [Fig.88] ;while the same location measured by the robot is (25.09m ,19.11m) [Fig.90]. These difference between those two similar points is the drift that occurs by the robot's odometry measurement. As long as the robot move further, the drift will increase as the wheel odometry are relative accumulating measurements where the error will increase with time. It can be noticed that the second

section of path from Lab B to Lab A [Fig.90] which is the returning way, it has a similar layout, if rotated clockwise, to the path from Lab A to Lab B in the going direction [Fig.90]. This deviation is due to the increased drift in the returning direction, where the final point (29.38m, -12.15m) [Fig.90] is supposed to be at (0,0) again. The waypoints, with drifting effects [Fig.90], extracted from robot's pose data in the manual test-drive on the same desired path; they are fed into the Pure Pursuit controller in the algorithm [Fig.86] in order to achieve an automated drive from Lab A to Lab B and then return to initial location in Lab A again. The Pure Pursuit controller calculates the linear velocity and heading rate accordingly. The actual path achieved by the robot [Fig.91], during the automated test drive, was been monitored live within Simulink. As a result, the robot has achieved the actual path [Fig.91] similar to the predefined path [Fig.90], but the main difference is in the rounded corners in [Fig.91] instead of the sharped edges corners in [Fig.90] and this is due to the known behavior of the Pure Pursuit controller.

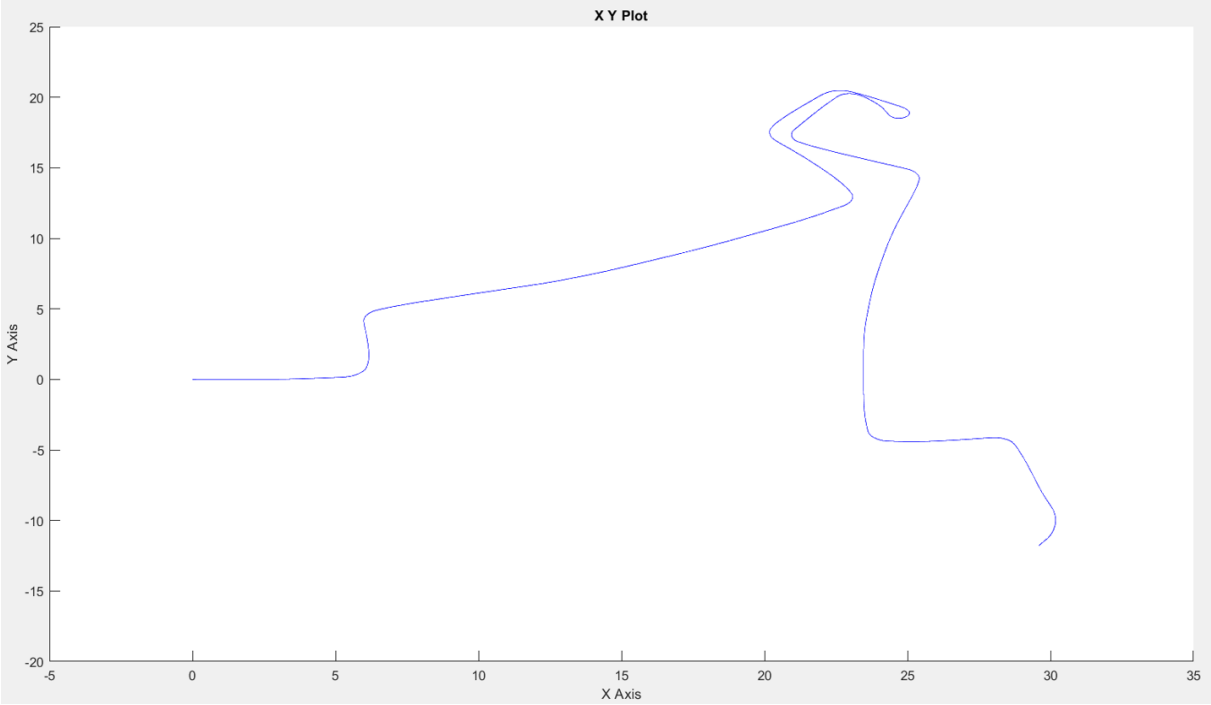


Fig.91 Actual Go-Return Path between Lab A & Lab B Achieved by the Robot by Mean of the Algorithm Based on the Waypoints w/ Drift Effects

It is a similar case to that between [Fig.88] and [Fig.89]. It is important to mention that the algorithm [Fig.86] with the waypoints including drifting effects [Fig.90], has allowed the robot to achieve the automated drive correctly from Lab A to Lab B in similar layout as in [Fig.89] in real world without major drifts been observed. It was noticed, in this method that the robot has drifted two times from the desired path on the way back (Lab B to A), but it is acceptable as compared by the first waypoint definition method where the robots has drifted several times from the first stages of the path. In summary, to achieve an acceptable automated drive relying

only on wheel odometry, it is better to use the waypoint definition method 2. In this method, as discussed the waypoints will be extracted from the recorded Pose data of a manual test drive on the same desired path rather than defining the waypoints manually or from a map. The extracted waypoints will be fed into the algorithm to achieve the autonomous functionalities. In fact, relying on wheel odometry alone is not sufficient and especially over long distances either in indoor or outdoor environments as discussed in several occasions before. Using wheel odometry as the only localization data source, is sufficient over few meters only where it can be used for short demonstrations purposes and testing of the designed algorithm. It is necessary to fuse the data with others sources such as IMU , GNSS,… to get better behavior and more precise localization. The fusion technic is being considered to be used in a more advanced algorithm in the future scope of this project.

When the algorithm [Fig.86] was deployed into a ROS node and ran on the target hardware, the ROS graph [Fig.59] visualized within "rqt" interface, was obtained. The graph shows how the ROS node named */RosAria/test_pioneer* ,generated from Simulink model [Fig.86], communicates with the original ROS node of the P3-DX robot named */RosAria*. In this case, the Simulink ROS node communicates with the robot's node by subscribing to the topic */RosAria/Pose* and publishing on the topic */RosAria/cmd_vel* as per the Simulink algorithm [Fig.86]. The realized automated test drive [Fig.92], of the algorithm [Fig.86], demonstration video inside the university building laboratories between Lab A and Lab B, can be seen on the Mission Mint YouTube channel on the link [35].

Fig.92 Caption from an Automated Test Drive inside the University Laboratories

CONCLUSION & FUTURE SCOPE

After achieving the commissioning tasks on the P3-DX robot, now it is a ready platform to be used within the framework of Matlab/Simulink and ROS. As a result, more complex algorithms can be designed in Matlab/Simulink for advanced autonomous functionalities and then the deployment will be done via ROS on the robot. This process of algorithm development and deployment is very efficient and flexible. The focus was to bridge Simulink and ROS together, as Simulink offers big flexibility and support in algorithm developments, whereas ROS is a very popular middleware with important tools and wide support on several layers that become essential in the variety of the robotics applications. Moreover, Simulink is a preferred platform for algorithm development by the majority of engineers, in contrast to overall algorithm development in a specific programming language that require extensive knowledge and experience. Bridging Simulink and ROS together is very important not only from the technical side but also by allowing people from different backgrounds to team up and tackle the robotics development field. The main challenge during that project, was to find the suitable packages and configurations to access the P3-DX robot in order to let it accessible by ROS, which is a very essential step for the Simulink models deployment on the robot. Adopting the V-model multi-layer testing approach, from algorithm design to simulation then implementation on the robot, in this project was very beneficial especially with the parallel commissioning work. The overall process was found to be efficient and allow reaching solutions to the problems in a convenient way. Finally, the P3-DX robot is now ready for more advanced algorithm developments to tackle more complex and challenging applications.

In the future scope of this project, it is planned to increase the sensory setup of the robot in order to work on a more advanced perception functional block, that is able to collect the required environmental data for the generation of a more concrete environmental model. This step is essential in order to cope with the planned advanced algorithm development in the next period. The aim from this development is to let the algorithm architecture has more concrete layers as [Fig.6] for the advanced autonomous applications. A concrete use-case of this project in the next period, will be an automated outdoor delivery solution in the city of Gera-Thüringen. The prepared P3-DX robot in this project will be used in the mentioned use-case. The robot needs to drive autonomously, for goods transportation, from a specific shopping center point to a specific residential complex within the city of Gera.

LIST OF FIGURES

Figure 1: Autonomous Robots in Supply Chain from Supplier to Customer [3] 1
Figure 2: Automated Guided Vehicle (AGV) using Magnetic Strips Embedded in the Floor [5] ... 2
Figure 3: Locus Robotics' autonomous mobile robots (AMR) | Credit: Locus Robotics [6] 3
Figure 4: Honda electric autonomous work vehicle (AWV) for material transportation [7] 3
Figure 5: Conventional control feedback-loop [8] 4
Figure 6: AD generic functional architecture [9] 4
Figure 7: Advanced Functional Blocks Architecture Loop [8] 5
Figure 8: Pioneer 3-DX Robot with Delivery Thermo-Box | Credit: Pioneer Robot [10] 6
Figure 9: A Caption of Engineering Requirements (User Wish & Technical Specifications) developed in details on Padlet 7
Figure 10: A caption of the morphological chart developed in details on Padlet 8
Figure 11: A caption of the Decision Matrix developed in details as an excel spreadsheet on Padlet 9
Figure 12: Pioneer 3-DX Robot's Layout [10] 10
Figure 13: Batteries Storage Compartment 10
Figure 14: Front and Back Sonar Arrays [10] 11
Figure 15: Front Sonar Arrangement [10] 11
Figure 16: Inner View of the Robot Showing One of the 12V DC Gearmotor 12
Figure 17: SH-2 Microcontroller with Connector Locations Equipped Originally on P3-DX [10] 13
Figure 18: SH-2 Microcontroller Board Installed in the Robot's Frontal Inner Compartment 13
Figure 19: NVIDIA Jetson Nano Development Kit [12] 14
Figure 20: Intel RealSense D435i Camera Equipped on the Top of the P3-DX Robot 15
Figure 21: Intel RealSense System Block Diagram [13] 16
Figure 22: Output of the D435i Camera mounted on the Robot inside Building D at HSM 17
Figure 23: Output of the D435i Camera mounted on the Robot in Outdoor Environment (Gera) 17
Figure 24: Waveshare 11.6 Inch LCD Display Connected to NVIDIA Jetson Nano [14] ..18
Figure 25: Rapoo E2700 Wireless Keyboard [15] 19
Figure 26: QSKJ DC-DC Converter [16] 19
Figure 27: Overall Communication Diagram for Algorithm Deployment on the Robot 20
Figure 28: USB to Serial Converter Adapter [17] 21
Figure 29: RJ-45 Ethernet Connector Cable [18] 21
Figure 30: Complete Hardware Wiring Schematic Diagram 22

Figure 31: Differential Drive Kinematic [19] 23
Figure 32: Internal Coordinate System of the P3-DX Robot [10] 23
Figure 33: Unicycle Model [21] 25
Figure 34: Three Layer Testing Approach based on the V-model 30
Figure 35: Simple Simulink Model for Controlling the P3-DX Robot [23] 32
Figure 36: Pure Pursuit Block Parameters 33
Figure 37: Look Ahead Distance Principle [24] 33
Figure 38: Effects of Small and Large Look Ahead Distances [24] 34
Figure 39: Left and Right Torque Computations from Wheel Speeds 35
Figure 40: Main Applied Simulink Model for Automated Functionalities on the P3-DX Robot 35
Figure 41: Sensing Block Module of the Main Algorithm 36
Figure 42: Coordinate Transformation inside the Extract Robot Pose Block 37
Figure 43: Control and Obstacle Avoidance Functional Block of the Main Simulink Model 37
Figure 44: Actuating Functional Block. 40
Figure 45: Plant Model for the P3-DX Robot. 41
Figure 46: Available Simulink Blocks for Gazebo in the Robotics System Toolbox 43
Figure 47: Testing Scenario Created within Gazebo used the P3-DX Robot 44
Figure 48: Gazebo Left/Right Wheel Torque Command 45
Figure 49: Coupling of Simulink and Gazebo for Algorithm Testing in Virtual Environment 46
Figure 50: Actual Path Achieved by the Robot in Gazebo (left) Vs Predefined Desired Path (right) 47
Figure 51: MobileSim (left) with Aria Demo Interface (right) 49
Figure 52: Running Aria Demo on the VM operating on Linux on the Same Windows OS-PC 51
Figure 53: Process Workflow Diagram for Algorithm Deployment and Implementation .. 53
Figure 54: ROS 1 Structure [29] 54
Figure 55: Algorithm Split into Several ROS Nodes [29] 55
Figure 56: Playing Back a ROS Bag File in Ubuntu Terminal 57
Figure 57: Some of the Available Topics from the Recorded Intel RealSense ROS Bag File 57
Figure 58: Playing Back IMU data from the Selected Topic from the Intel RealSense Bag File 57
Figure 59: ROS Graph Visualized in rqt 58
Figure 60: rqt tool Interface 58

Figure 61: Rviz Interface with Intel RealSense and P3-DX Robot's Topics Visualization 59
Figure 62: RosAria Available Topics List for the Pioneer Robot 60
Figure 63: Data from /RosAria/Sonar 61
Figure 64: Data from /RosAria/sonar_pointcloud2 Topic 61
Figure 65: Data from /RosAria/pose Topic 61
Figure 66: Simulink Blocks for ROS Applications 63
Figure 67: Simulink APPS Toolbar 63
Figure 68: Connecting to ROS Device from Simulink 63
Figure 69: Simulink Model for Sending Velocity Commands over ROS for the P3-DX Robot 64
Figure 70: ROS Publish Block Parameters 64
Figure 71: Bus Assignment Block Parameters 65
Figure 72: ROS Bag Block Parameter 66
Figure 73: Available Topics for Selection 66
Figure 74: Simulink Model for Displaying P3-DX Sonar Readings from a ROS Bag file 67
Figure 75: Simulink Model Designed for Displaying RGB Video from Intel RealSense ROS Bag 68
Figure 76: List of Available Topics from Intel RealSense ROS Bag 69
Figure 77: Gera Robot's Test-Drive Video Stream in a Dedicated Matlab Scope 69
Figure 78: Sobel Edge Detection Algorithm Applied on an Image Extracted From ROS Message 70
Figure 79: Original Image1 from Gera Test 70
Figure 80: Processed Image1 from Gera Test 70
Figure 81: Original Image 2 from Gera Test 71
Figure 82: Processed Image 2 from Gera Test 71
Figure 83: Simulink Model for Extracting Robot's Pose from a ROS Bag File 72
Figure 84: Bus Selector Block Parameters 72
Figure 85: Exporting Pose Data [X Y θ] into Matlab Workspace 73
Figure 86: Simulink Model Deployed on the Robot as ROS Hardware for a Simple Automated Drive 74
Figure 87: Configuration Parameters Window in Simulink 76
Figure 88: Desired Real Path Plot from Waypoints Coordinates (Y-axis versus X-axis) in Meters 76
Figure 89: Achieved Path by the Robot from Lab A to Lab B using Pure Pursuit Controller 77

Figure 90: Desired Go-Return Path between Lab A & Lab B Constructed by Waypoints w/ Drift Effects 78
Figure 91: Actual Go-Return Path between Lab A & Lab B Achieved by the Robot by Mean of the Algorithm Based on the Waypoints w/ Drift Effects 79
Figure 92: Caption from an Automated Test Drive inside the University Laboratories 80

LITERATURE

[1] Intel: What are Autonomous Mobile Robots? [online]. URL: https://www.intel.com/content/www/us/en/robotics/autonomous-mobile-robots/overview.html#:~:text=An%20autonomous%20mobile%20robot%20is,and%20often%20require%20operator%20oversight.

[2] Fierce Electronics: Warehousing and Logistics Robot Shipments Will Reach 620,000 Units Annually by 2021 [online], 03.03.2017. URL: https://www.fierceelectronics.com/components/warehousing-and-logistics-robot-shipments-will-reach-620-000-units-annually-by-2021

[3] Li, K; Jeffs, J.: Mobile Robotics in Logistics, Warehousing and Delivery 2022-2042 [online], *Jan 2022*. URL: https://www.idtechex.com/en/research-report/mobile-robotics-in-logistics-warehousing-and-delivery-2022-2042/855

[4] Goodwin, D.: The Evolution of Autonomous Mobile Robots [online], *09.09.2020*. In: *Control Automation Technical Article*, URL: https://control.com/technical-articles/the-evolution-of-autonomous-mobile-robots/#:~:text=In%20industry%2C%20most%20of%20the,to%20use%20in%20the%201950s.

[5] Cross: The Difference Between AGVs and Mobile Robots [online]. URL: https://www.crossco.com/resources/articles/the-difference-between-agvs-and-mobile-robots/

[6] Cobot Trends Staff: Locus Robotics, High-Jump Partner on Mobile Robot installations [online], 14.01.2020. In: *Collaborative Robotics Trends*, URL: https://www.cobottrends.com/locus-robotics-highjump-enhance-amr-integration/

[7] Leichsenring, S.: Honda AWV : Autonomes Elektrofahrzeug für Transportaufgaben [online], *16.11.2021*. In: *InsideEVs*, URL: https://insideevs.de/news/548166/honda-awv-autonomes-elektrofahrzeug-baustelle/

[8] MathWorks: Developing Autonomous Mobile Robots Using MATLAB and Simulink [online]. In: *MathWorks Ebook*, URL: https://www.mathworks.com/campaigns/offers/next/autonomous-mobile-robots.html

[9] Voßwinkel, R; Gerwien, M; Jungmann, A; and Schrödel, F.: Intelligent Decision Making and Motion Planning for Automated Vehicles in Urban Environments. In: *Autonomous Driving and Advanced Driver-Assistance Systems [ADAS]*. CRC Press, 2021.

[10] Omron Adept Mobile Robots: Pioneer 3 Operations Manual, 2017.

[11] Stiller, J.: ZF Tech Insights for STEM Students [online webinar], 29.04.2021.

[12] NVIDIA: NVIDIA Announces Jetson Nano: $99 Tiny, Yet Mighty NVIDIA CUDA-X AI Computer That Runs All AI Models [online]. In: *NVIDIA Newsroom Press Release, 18.03.2019*, URL: https://nvidianews.nvidia.com/news/nvidia-announces-jetson-nano-99-tiny-yet-mighty-nvidia-cuda-x-ai-computer-that-runs-all-ai-models

[13] Intel RealSense Technology: Intel® RealSenseTM Product Family D400 Series Datasheet, *Revision 009, June 2020.*

[14] Waveshare: 11.6" HDMI Touch Display [online]. In: *Products Catalogue*, URL: https://www.waveshare.com/11.6inch-hdmi-lcd-h-with-case.htm

[15] Sebring, C.: Rapoo E2700 Wireless Multimedia Touchpad Keyboard Review [online], 03/11/2020, URL: https://www.tweaktown.com/reviews/6332/rapoo-e2700-wireless-multimedia-touchpad-keyboard-review/index.html

[16] QSKJ Power Module Experts: DC-DC Power Modules Products [online], URL: https://www.qskj.cc/shop/product/1935262-dc-dc-6v-32v-to-0-8-28v-150w-15a-buck-boost-voltage-regulator-qs-1212cba-150w-8349

[17] Amazon: Digitus USB to Serial Adapter USB 1.1 [online], URL: https://www.amazon.de/DIGITUS-Converter-USB1-1-Seriell-Verlaengerungskabel/dp/B0038SYJ4Y?th=1

[18] Amazon: HB Digital Network Cable LAN Cable Raw Cable RJ45 Connector [online], URL: https://www.amazon.de/-/en/Connector-Professional-Halogen-Free-Compliant-4x2xAWG26/dp/B07GXKVXCJ?th=1

[19] MathWorks: Differential Drive Kinematics [online]. In: *MathWorks Help Center Documentation*,URL:https://www.mathworks.com/help/robotics/ref/differentialdrivekinematics.html

[20] Egerstedt, M.: 2.2 Differential Drive Robots [online]. In: *Control of Mobile Robots, 14.02.2013*, URL: https://www.youtube.com/watch?v=aE7RQNhwnPQ

[21] MathWorks: Unicycle Kinematics [online]. In: *MathWorks Help Center Documentation*,URL:https://www.mathworks.com/help/robotics/ref/unicyclekinematics.html

[22] Egerstedt, M.: 2.3 Odometry [online]. In: *Control of Mobile Robots, 14.02.2013*, URL: https://www.youtube.com/watch?v=XbXhA4k7Ur8

[23] MathWorks: Control a Differential Drive Robot in Gazebo with Simulink [online]. In: *MathWorks Help Center Documentation*, URL: https://www.mathworks.com/help/robotics/ug/control-a-differential-drive-robot-in-simulink-and-gazebo.html

[24] MathWorks: Pure Pursuit Controller [online]. In: *MathWorks Help Center Documentation*, URL: https://www.mathworks.com/help/robotics/ug/pure-pursuit-controller.html#:~:text=The%20look%20ahead%20distance%20is,compute%20the%20angular%20velocity%20commands

[25] MathWorks: ROS2 Dashing and Gazebo [online]. In: *MathWorks Support Help Center/ Virtual Machine Installation Instructions*, URL: https://www.mathworks.com/support/product/robotics/ros2-vm-installation-instructions-v4.html

[26] Mission MINT_Hochschule Schmalkalden Youtube Channel: Close Coupling of Simulink and Gazebo [online]. 31.*08.2021*, URL: https://www.youtube.com/watch?v=hYWuGKhvp_s

[27] GitHub: reedhedges/AriaCoda [online]. URL: https://github.com/reedhedges/AriaCoda

[28] ROS Wiki: Ubuntu install of ROS Melodic [online]. URL: http://wiki.ros.org/melodic/Installation/Ubuntu

[29] Alferez, I.: Getting Started with ROS: Integrating with MATLAB/Simulink [online]. *14.07.2020*, URL: https://www.youtube.com/watch?v=wgxNt9P_gTE

[30] ROS Wiki: ROS/ Concepts [online]. URL: http://wiki.ros.org/ROS/Concepts

[31] AWS Amazon: AWS RoboMaker Developper Guide [online]. URL: https://docs.aws.amazon.com/robomaker/latest/dg/simulation-tools-rviz.html

[32] GitHub: amor-ros-pkg/rosaria [online]. URL: https://github.com/amor-ros-pkg/rosaria

[33] ROS Wiki: How to use ROSARIA [online]. URL: http://wiki.ros.org/ROSARIA/Tutorials/How%20to%20use%20ROSARIA

[34] GitHub: pengtang/rosaria_client [online]. URL: https://github.com/pengtang/rosaria_client

[35] Mission MINT_Hochschule Schmalkalden Youtube Channel: First Automated Drive of our Pioneer Robot [online]. 28.*12.2021*, URL: https://www.youtube.com/watch?v=MR7OuABx0nw&list=PL7vpyWa9RV_KVP-BoaN_TNHJ9jCqJdHphH&index=5