



**Western Norway
University of
Applied Sciences**

BACHELOR'S THESIS

BO24EB-09

Navigation of a Mobile Robot


*Henrik Eikefet
Daniel Sortland*

Bachelor's Thesis in Electrical Engineering at
Department of Computer Science, Electrical Engineering, and
Mathematical Sciences
Western Norway University of Applied Sciences

May 20, 2024

Document Control

Report title: Navigation of Mobile Robot	
Authors: Henrik Eikefet Daniel Sortland	Date/Version May 20, 2024/0.1
Supervisor at HVL: Geir Omar Berland	Report number: B024EB-09
Comments: We allow publishing of the report	Course: Automation and Robotics
Security classification: Open	Number of pages with appendixes: 96
Contracting entity: Fagskolen Vestland	Contracting entity's reference:
Contact at contracting entity, including contact information: Guttorm Lyngær, Phone: 40611630, E-mail: Guttorm.Lyngver@vlfk.no	


Henrik Eikefet


Daniel Sortland

Preface

This report has been prepared as part of our bachelor's thesis at Western Norway University of Applied Sciences (HVL), Bergen Campus, undertaken in the spring of 2024. Our project, led by Guttorm Lyngvær at Fagskolen Vestland, challenged us to develop a set of laboratory exercises demonstrating the navigation capabilities of the Festo Robotino 3 mobile robot.

We are thankful to Guttorm Lyngvær for providing and entrusting us with this exciting task, as well as Geir Omar Berland who provided support and guidance throughout the project.

Summary

In this bachelor's thesis, we have developed a set of laboratory exercises that demonstrates the navigation of a mobile robot, Robotino 3 from Festo. The lab exercises covers key aspects of mobile robotics and blend theoretical concepts from Peter Corke's "Robotics, Vision, and Control" with practical demonstrations of robot navigation and interaction. Additionally, we have created a kinematic model to simulate the robot's movement using its omnidrive system. This model is supported by software simulations to help with understanding.

The assessment of the project's results shows that we achieved most of our objectives. Both the lab exercises and software simulations provide practical experience and theoretical understanding, while tests on the Robotino give valuable insights into its sensor accuracy, which could be useful for future projects.

Weaknesses in the system were identified during the project, but they do not critically affect task execution. Areas for improvement have been identified as well, and are detailed in Chapter 7.1.

Sammenheng

I denne bacheloroppgaven har vi utviklet et sett med laboratorieøvelser som demonstrerer navigasjon av en mobilrobot, Robotino 3 fra Festo. Labøvelsene dekker viktige aspekter ved mobil robotikk og blander teoretiske konsepter fra Peter Corkes "Robotics, Vision, and Control" med praktiske demonstrasjoner av robotnavigasjon og interaksjon. Vi har også opprettet en kinematisk modell for å simulere robotens bevegelse ved hjelp av dens omnidrive-system, støttet av programvaresimuleringer for å hjelpe mer med forståelsen.

Prosjektets resultater tyder på at de fleste målene har blitt oppnådd. Labøvelsene og programvaresimuleringene gir praktisk erfaring og teoretisk forståelse, mens tester utført på Robotino gir verdifulle innsikter i sensorens nøyaktighet, noe som kan være nyttig for fremtidige prosjekter.

Noen svakheter i systemet ble identifisert underveis i prosjektet, men de er ikke kritiske for oppgaveutførelsen. Muligheter for forbedringer er detaljert i Kapittel 7.1.

Table of Contents

Preface	3
Summary	4
Sammendrag	4
1 Introduction	11
1.1 Contracting Entity	11
1.2 Problem Description	11
1.3 Objectives	12
2 Theory in Mobile Robotics	13
2.1 Peter Corke's Robotics, Vision and Control	13
2.1.1 Chapter 4 - Mobile Robot Vehicles	13
2.1.2 Chapter 5 - Navigation	15
2.1.3 Chapter 6 - Localization	17
3 Technical Overview	19
3.1 Hardware	20
3.1.1 Drive system	21
3.1.2 Modules	22
3.1.3 Sensors	23
3.1.4 Interfaces	25
3.1.5 Power Supply	26
3.2 Software	27
3.2.1 Web Interface	27
3.2.2 Programming	27
3.2.3 Simulations	29
3.3 Control Systems and Strategies	30
3.3.1 PID Controller	30
3.3.2 Closed-Loop Controller	31
3.4 TCP/IP	32
3.5 3D-printing	33
4 Design and Implementation	34
4.1 Kinematic Model	34
4.2 Sensor Utilization	36
4.3 Software Development	43
4.4 Simulations	45
4.5 3D Printing	47

5	Testing	48
5.1	Distance Sensor Accuracy Test	48
5.2	Odometry Accuracy Test	48
5.3	Omnidrive Function Block Test	49
5.4	Camera Accuracy Test	50
5.5	Robot Repeatability in Lab Exercise 4	51
6	Results	52
6.1	Lab Exercises	52
6.2	Identified Issues	54
6.3	Discussion	55
7	Conclusion	56
7.1	Room for Improvements	57
	Bibliography	61
A	Kinematics	62
B	Lab Exercise 1	70
C	Lab Exercise 2	73
D	Lab Exercise 3	76
E	Lab Exercise 4	80
F	Template for Report	87
G	Guide to Working With Robotino 3	88
H	Videos Demonstrating Robotino Movement	94
I	External Resources and Documentation	95

List of Figures

1.1	Overview of the Robotino 3 Robotic Platform	11
3.1	Robotino shown with Tower, Segment Base, Gripper, and Signaling Lights	19
3.2	Specifications for Robotino’s Embedded PC	20
3.3	Illustrating Robotino’s Omnidirectional Wheels	21
3.4	Illustrating Omnidirectional Wheels Movement Capabilities	21
3.5	The Robotino’s Electric Gripper with Integrated Light Barrier	22
3.6	Robotino’s Infrared Sensor Array for Navigation and Obstacle Detection .	23
3.7	Graph Illustrating Sensor Output with Distance to Objects	23
3.8	The Robotino’s Safety Bumper	24
3.9	The Onboard Camera Module for the Robotino	24
3.10	The I/O Interface of the Robotino Displaying the Connections	25
3.11	The Robotino View Software Interface	27
3.12	Snapshot of the Robotino SIM Simulation Environment	29
3.13	Example of a Closed Loop Controller	31
3.14	The Four Layers of TCP/IP	32
3.15	Makerbot Replicator Z18	33
3.16	Example of a 3D-Printed Object That Could Be Utilized for Our Project.	33
4.1	The Kinematic Model of Robotino’s Drive System	35
4.2	Sensor Layout for Robot Obstacle Approach	36
4.3	Robotino 3 Distance Sensor Characteristic	37
4.4	Voltage Response of Sensor to Varying Distances from an Aluminum Tape	39
4.5	Optical Sensor Reaction to Robotino’s Position Relative to the Marked Line	40
4.6	Generate Markers	41
4.7	Marker Detection Block	41
4.8	Color Range Finder Function Block	41
4.9	Visual Representation of the Kinematics	44
4.10	Visualization of Robotino in RViz	45
4.11	Simulation of Robotino in a Virtual Environment Using Gazebo	45
4.12	Twist Controller for Robot Operation in Gazebo	46
4.13	A Simple Geometric Model of the Robot with Animated Movement from Kinematics. Created in Python	47
6.1	Overview of Robotino Front Panel with Connections	54
7.1	Example of Lidar We Could Use: Hokuyo URG-04LX-UG01	57
7.2	Showcasing Robotino with a Hokuyo Lidar Mounted	57
7.3	Festo Distribution/Belt Station	58

A.1	Illustrating the Coordinate Frames and Angular Relationships	62
A.2	Representation of Wheel Position and Orientation in the Robot Body Frame	63
A.3	Kinematics of a Robot with Mecanum Compared to Omni-Wheels	64
A.4	Transformation Steps and Equations for Robot Wheel Kinematics	65
A.5	Kinematics with Angle Calculations and Wheel Coordinates	66
A.6	Calculation of Kinematics for Wheel 1	67
A.7	Calculation of Kinematics for Wheel 2	67
A.8	Calculation of Kinematics for Wheel 3	68
A.9	Matrix Representation and Vector Transformation for Wheel Velocities . .	69
B.1	Move forwards for 6 seconds	71
B.2	Move backwards for 6 seconds	71
B.3	Omnidrive Function Block	71
B.4	Drive System Function Block	71
B.5	Drive System with Control Panel	72
B.6	Bumper Function Block	72
C.1	Approach a Wall Subprogram	74
C.2	Move Along Wall Subprogram	75
C.3	Move Around Corner Subprogram	75
D.1	Teach Color Subprogram	77
D.2	Search for the Object Subprogram	78
D.3	Approach the Object Subprogram	78
D.4	Search for the Marker Subprogram	79
D.5	Approach the Marker Subprogram	79
E.1	Find Marker Subprogram	82
E.2	Mark Follow Subprogram	82
E.3	Center Subprogram	83
E.4	Pickup Subprogram	84
E.5	Drive Back Subprogram	85
E.6	Move Odometry Subprogram	85
E.7	Release Gripper Subprogram	86
E.8	Home Pose Subprogram	86
G.1	Robotino shown with Tower, Segment Base, Gripper, and Signaling Lights	88
G.2	User interface of Robotino View, Highlighting the Main Program Structure	90
G.3	The programming Blocks Within Robotino View	90
G.4	Snapshot of the Robotino SIM Simulation Environment	91
G.5	Displaying the Control Function on Robotino Web Interface	92

List of Tables

3.1	The I/O Table of the Robotino Displaying the Connections	25
4.1	Robotino 3 Distance Sensor Measured Values	37
5.1	Detection Results for Various Objects at Different Distances	48
5.2	Omnidrive Test of 1 Meter With a Speed of 100 mm/sec	49
5.3	Omnidrive Test of 1 Meter With a Speed of 400 mm/sec	49
5.4	Camera Accuracy Test with Various Objects and Distances	50
5.5	Performance Test With Line Detector and Uncalibrated Starting Position	51
5.6	Performance Test With AR-Marker and Calibrated Starting Position . . .	51

List of Abbreviations

HVL Western Norway University of Applied Sciences

ROS Robot Operating System

GPS Global Position System

PRM Probabilistic Roadmaps

PCB Printed Circuit Board

PC Personal Computer

MPS Modular Production System

IR Infrared

WLAN Wireless Local Area Network

I/O Interface Input/Output Interface

USB Universal Serial Bus

PCI Peripheral Component Interconnect

VGA Video Graphics Array

DC Direct Current

SLAM Simultaneous Localization and Mapping

TCP/IP Transmission Control Protocol/Internet Protocol

RPM Revolutions Per Minute

Chapter 1

Introduction

1.1 Contracting Entity

Fagskolen Vestland is one of Norway's largest public vocational schools, with over 1,900 students. The school offers a wide range of programs in areas like maritime, technical, petroleum, environmental, and health sciences [4]. It has campuses in several locations including Bergen, Førde, Stord, Ulvik, Måløy, Voss, and Austevoll. Our project is in collaboration with Fagskolen in Bergen.

1.2 Problem Description

Develop laboratory exercises that demonstrate the navigation of a mobile robot platform. The robot we will use is the Robotino 3 from Festo, which is an omnidrive robot, meaning it can easily move in any direction and rotate on the spot [14]. The task provider wishes that the exercises reflect the theory presented in chapters 4, 5, and 6 of Peter Corke's "Robotics, Vision and Control" [1]. It may also be relevant to use the robot as an explanatory model for 2D rotation matrices. For the simulation of the robot, we will use Python or Matlab, as this is requested by the task provider, with the option to utilize Robot Operating System (ROS).



Figure 1.1: Overview of the Robotino 3 Robotic Platform

1.3 Objectives

This project aims to develop lab exercises for the Robotino Mobile robot, focusing on integrating new functionalities not yet mastered by the task provider. We will utilize Peter Corke's robotics methods through Python programming [1], as well as software like Robotino View and Robotino SIM, developed by Festo [16]. These exercises will cover both physical experiments and software simulations, demonstrating robot navigation, object interaction, and theoretical concepts from Peter Corke's "Robotics, Vision, and Control".

The lab exercises, found in Appendix B to E, are designed to start with simple tasks like basic motion control and sensor usage before gradually moving on to more complex activities. Each exercise builds on the previous one, helping to better understand Robotino 3. We have also included a basic guide to working with Robotino 3, which you will find in Appendix G. The guide details how the robot and its software should function, helping students perform the lab exercises. Furthermore, proposed solutions and video demonstrations for all exercises are provided, offering students a reference point to compare and understand their results.

Upon completing the lab exercise, students are required to submit a report on their experiment. This report should include a brief introduction to the exercise, the equipment used, the steps followed to complete the exercise, and the results. A Template for this report can be found in appendix F

In summary, this project aims to connect theory with practice in robotics, using the Robotino mobile robot. By combining programming, simulations, and hands-on activities, we are preparing students to better understand and navigate the field of robotics. The provided guides, lab exercises, and solutions will be designed to give students the skills and knowledge they need to tackle robotics challenges.

Chapter 2

Theory in Mobile Robotics

2.1 Peter Corke's Robotics, Vision and Control

"Robotics, Vision, and Control: Fundamental Algorithms in Python" by Peter Corke is a comprehensive textbook that focuses on the field of robotics and how it intersects with vision and control systems. The book is well-regarded for its clarity in explaining complex concepts in robotics, including kinematics, dynamics, control, and vision. Our project includes chapter 4, 5, and 6.

2.1.1 Chapter 4 - Mobile Robot Vehicles

In this chapter, the focus is on mobile robots, which are robots capable of moving around in their environment. It covers the principles of locomotion, the different types of mobile robots (e.g., wheeled, legged, aerial, and aquatic robots), and the kinematics associated with mobile robot movement. The chapter discusses how mobile robots perceive their environment, interact with it, and the basics of how they are controlled. This foundational knowledge is crucial for understanding more complex topics in mobile robotics, including navigation and localization. Our focus for this chapter has been on developing a kinematic model for the Robotino 3 robot and to gain a better understanding of omnidirectional drive systems. This has deepened our understanding of the robot's operational mechanism.

Kinematic Model

A kinematic model describes the motion of a system without considering the forces causing that motion. It focuses on the positions, velocities, and accelerations of the system's components, providing a mathematical representation of how they move relative to each other. The kinematic model for this robot can be found in Appendix A.

Omnidirectional Drive and Holonomic Motion

A 3-wheeled omnidirectional drive system, often referred to as an omnidrive, allows a robot to move in any direction without changing the orientation of the robot itself [19]. This capability is called holonomic motion, and is achieved through the use of special wheels, such as omni wheels or mecanum wheels, which can roll freely in multiple directions. In a 3-wheeled omnidrive system, the wheels are typically arranged in a triangular configuration, as shown in figure B.3. Omnidirectional drive systems offer several advantages and disadvantages:

Advantages:

- **Holonomic Motion:** The ability to move in any direction without changing the orientation of the robot makes it ideal for navigating tight spaces or complex environments.
- **Efficiency:** Omnidirectional drive systems can achieve smooth and efficient motion since they can move directly toward a target without the need for complex maneuvers or turning.
- **Versatility:** These systems are versatile and can be used in various applications such as robotics competitions, warehouse automation, and indoor navigation systems.
- **Precision:** With precise control over movement in all directions, omnidirectional drive systems can perform intricate tasks with accuracy, such as precise positioning or manipulation of objects.
- **Redundancy:** With three wheels, there is built-in redundancy, meaning that even if one wheel fails or encounters an obstacle, the robot can still continue to move and operate.

Disadvantages:

- **Complexity:** Implementing and controlling omnidirectional drive systems can be more complex compared to traditional drive systems, requiring advanced control algorithms and sensor integration.
- **Cost:** Omnidirectional drive systems, especially those using specialized wheels like omni wheels or mecanum wheels, can be more expensive than conventional drive systems, which may limit their adoption in cost-sensitive applications.
- **Mechanical Design:** The mechanical design of omnidirectional drive systems can be more intricate, requiring careful consideration of wheel placement, alignment, and stability to ensure optimal performance.
- **Terrain Limitations:** While omnidirectional drive systems excel in controlled environments, they may face challenges on rough or uneven terrain, where traditional wheeled systems with larger wheels or tracks may have better traction and stability.
- **Maintenance:** The increased complexity and number of moving parts in omnidirectional drive systems can potentially lead to higher maintenance requirements compared to simpler drive systems, requiring regular inspection and upkeep to ensure reliability and longevity.

2.1.2 Chapter 5 - Navigation

Robot navigation involves guiding a robot to reach a specific goal. This chapter explores the algorithms and strategies used in robot navigation, such as path planning, map representation, and the use of sensors for detecting obstacles. While humans typically rely on maps and signs for navigation, it's not always necessary for robots to do the same. Reactive navigation offers an alternative approach, where robots respond directly to their environment without a predefined map. For instance, they might follow a light, track a line, navigate a maze by tracing a wall, or clean a room by moving randomly. In these situations, the robot responds to signals such as changes in light levels or contact with obstacles. More advanced robots often employ map-based navigation, also known as motion planning.

Map-Based Planning

Map-based planning includes a range of algorithms designed to navigate robots using pre-built maps of their environment. These algorithms utilize the map and knowledge of the robot's location to generate optimal or near-optimal paths towards a goal. Some commonly used map-based planning algorithms include the Distance Transform, D*, Probabilistic Roadmaps (PRM) and Lattice Planner. Distance Transform calculates safe distances from obstacles for path planning. D* adjusts paths as the environment changes. PRM randomly sample points to plan paths in complex but stable environments, and Lattice Planner use set grids to guide vehicles according to their specific needs and surroundings. Each algorithm helps robots navigate different types of environments.

Although these algorithms offer advanced navigation capabilities, they require extensive computational resources and accurate environmental mapping. In other words, these map-based approaches all require the availability of a map and accurate knowledge of the robot's location, which we will explore in chapter 2.1.3.

Reactive Navigation

Reactive navigation, also known as behavior-based navigation, is a navigation approach where a robot responds directly to sensory input from its environment without relying on a pre-built map. Instead of planning a route beforehand, the robot continuously evaluates its surroundings and makes instantaneous decisions on how to move based on the information it receives in real-time.

For the Robotino 3, which lacks map-based navigation capabilities but possesses sensors like optical and inductive sensors, a Logitech C920 camera, infrared (IR) sensors for distance, a gyroscope, and a bumper for collision detection, reactive navigation could involve the following:

1. **Sensor Fusion:** Robotino 3 can integrate data from its various sensors to perceive its environment more accurately. For example, it could use its optical sensors to detect lines on the ground, its IR sensors to measure distances to obstacles, and its camera to recognize visual landmarks.
2. **Obstacle Avoidance:** The robot could employ reactive behaviors to avoid obstacles in its path. When an obstacle is detected by its IR sensors or bumper, it could immediately adjust its trajectory to navigate around it.
3. **Line Following:** Using its optical sensors, Robotino 3 could follow lines on the ground, allowing it to navigate along predefined paths or tracks.
4. **Light Seeking or Avoidance:** By reacting to light intensity captured by its camera or IR sensors, the robot could navigate towards or away from light sources, depending on the desired behavior.
5. **Reactive Path Planning:** Rather than planning a route in advance, Robotino 3 could dynamically adjust its path based on changing environmental conditions. For example, it could follow a wall using its IR sensors, adjusting its distance from the wall as it navigates.

Given our robot's limitations without map-based navigation capabilities, we would mostly depend on reactive navigation methods. We have utilized Robotino's existing sensors to enable it to navigate through various environments. This enables the robot to navigate independently without predetermined maps, making it well-suited in our case. For more information on how we implemented these sensors for our robot, see chapter 4.2.

2.1.3 Chapter 6 - Localization

Localization is the process of determining a robot's position and orientation within its environment. It involves estimating the robot's location relative to a known reference frame or map. Accurate localization is crucial for robots operating in dynamic or unknown environments to navigate safely and effectively.

Dead Reckoning

Dead reckoning is a method used to estimate a robot's current position based on its previous positions and known movements. With Robotino 3, dead reckoning typically relies on wheel encoders and a gyroscope. Here's how it works:

- **Wheel Encoders:** These sensors measure the rotation of the wheels, allowing us to track how far the robot has traveled. By integrating these measurements over time, we can estimate the robot's displacement.
- **Gyroscope:** A gyroscope measures the robot's orientation changes. By integrating these measurements over time, we can estimate the robot's heading (orientation).

However, dead reckoning suffers from cumulative errors. Small errors in measuring distance and orientation accumulate over time, leading to drift in the estimated position. This drift can be corrected using additional localization techniques.

Landmark-Based Localization

Landmark-based localization is a technique used to estimate a robot's position and orientation by recognizing and triangulating specific features or landmarks in its environment. These landmarks could be distinct visual features, such as corners of walls, unique objects, or beacons with known positions. The robot's sensors, such as cameras or range finders, are used to detect and identify these landmarks. Once detected, the robot compares the observed features with a map of the environment or a database of known landmarks to estimate its position relative to them. Landmark-based localization is robust in environments with distinguishable landmarks but may suffer from challenges such as occlusions or changes in lighting conditions

Modeling Vehicle Kinematics

Modeling the robots's movement is crucial for accurate localization. Vehicle kinematics describes how the robot's position and orientation change over time in response to control inputs. This modeling allows us to predict how the robot will move based on its current state and control commands.

For our robot, which features an omnidrive and holonomic drive system, we typically use kinematic models tailored to its motion capabilities. These models consider factors such as wheel speeds, wheel base, wheel configuration and placement to accurately predict the robot's motion.

Estimating Pose

Pose estimation refers to determining the robot's position (x, y) and orientation (θ) within its environment. Dead reckoning provides an initial estimate of the pose, based on wheel encoder and gyroscope data. However, this estimate becomes less accurate over time due to drift. We discuss these errors more in chapter 5.2. To improve pose estimation accuracy, we can integrate other sensor data, such as visual odometry from the camera or feature-based localization using IR sensors. These techniques help correct errors in dead reckoning and provide a more accurate estimate of the robot's pose.

Dead reckoning, vehicle modeling, and pose estimation are fundamental components of localization for a robot. By combining these techniques with other sensor data and probabilistic methods, we can achieve robust and accurate localization, enabling the robot to navigate autonomously in various environments.

Limitations and Considerations

While many aspects of localization can be covered with Robotino 3's sensors, some advanced techniques may be out of reach without additional hardware. Factors such as sensor accuracy, environmental conditions, and the complexity of the robot's surroundings can impact localization performance.

In summary, Localization is essential for enabling a robot to navigate autonomously and effectively in various environments. Leveraging the robot's sensors and applying localization techniques enhances its navigation capabilities. For further details, refer to chapter 4.2.

Chapter 3

Technical Overview

This chapter presents an overview of the essential hardware and software components used with our Robotino project. We will examine each component's significance and role, giving a basic understanding before going into more detail in the following chapters.

Robotino

Robotino 3 is an advanced mobile robotics platform mainly used for educational and research applications. Manufactured by Festo Didactic, it is recognized for its omnidirectional drive system. Equipped with a range of sensors, Robotino 3 provides a comprehensive set of tools for students and researchers to develop skills in areas such as navigation, sensor integration, and robot control.



Figure 3.1: Robotino shown with Tower, Segment Base, Gripper, and Signaling Lights

3.1 Hardware

The Robotino platform is designed to offer robust and flexible hardware capabilities, enabling a wide range of applications in robotics. This chapter provides an overview of the key hardware components, including the control unit, drive system, modules, sensors, interfaces, and power supply. [15].

Control Unit

The control unit in the Robotino includes the Printed Circuit Board (PCB) controller with an embedded PC and a Microcontroller, as well as all associated interfaces. The embedded PC in the Robotino controls the mobile robot system and is mounted directly on the main PCB in the control unit of the Robotino. It is connected to the control unit's interfaces and the microcontroller. The microcontroller monitors the supply voltage, controls the motor, and manages the digital and analog inputs and outputs of the Robotino. The microcontroller is also mounted directly on the main PCB in the control unit of the Robotino.

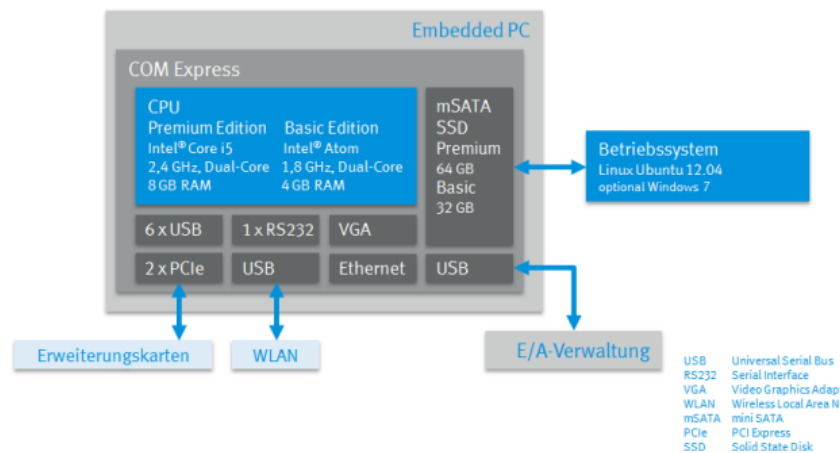


Figure 3.2: Specifications for Robotino's Embedded PC

3.1.1 Drive system

Omnidrive

With its omnidrive system, Robotino can travel in all directions and rotate on the spot. The three independent drive units consist of motors, an incremental encoder, gear unit and wheels, and are integrated into the chassis of the Robotino.

Wheels

The wheels are designed with multiple rotating rollers along their circumference, allowing the wheels to move in the x and y direction simultaneously, which enables holonomic movement, as shown in Figure 3.4



Figure 3.3: Illustrating Robotino's Omnidirectional Wheels



Figure 3.4: Illustrating Omnidirectional Wheels Movement Capabilities

Motors

The Robotino is equipped with three motors which power each of the three omnidirectional wheels independently of each other. An incremental encoder is mounted on each motor and measures its angle of rotation.

Gear Units

Robotino has a 32:1 gear ratio between each motor and wheel, enabling the mobile system to operate at low speeds and with high accuracy.

Incremental Encoder

Each of the motors in the Robotino has an incremental encoder. Based on the values of the incremental encoder, the motor controller can adjust and regulate the real motor speed at the desired speed. In addition, these values can also be used to determine the position of the mobile system.

3.1.2 Modules

Tower

Additional modules can be installed on the tower, allowing for interaction with Modular Production System (MPS) stations, among other functions.

Segment

The segment offers flexible mounting options for sensors and modules on the Robotino platform. This flexibility allows users to integrate additional functionalities such as the gripper, enhancing the robot's capabilities for various applications.

Gripper

The electric gripper integrated into the Robotino platform offers object handling. Utilizing an integrated light barrier, it detects objects between its jaws for accurate gripping. Supported by a slide mechanism, it efficiently picks up objects from surfaces, guided by Robotino to the optimal position. Continuous monitoring of motor current ensures secure gripping.



Figure 3.5: The Robotino's Electric Gripper with Integrated Light Barrier

3.1.3 Sensors

To navigate through an environment and avoid obstacles, Robotino is equipped with different types of sensors

Distance Sensor

The IR distance sensors on Robotino make it possible to determine the distance to objects around it. There are nine infrared sensors positioned around its base at 40° angles from each other. Each sensor measures a voltage level that corresponds to the distance from a reflective object.

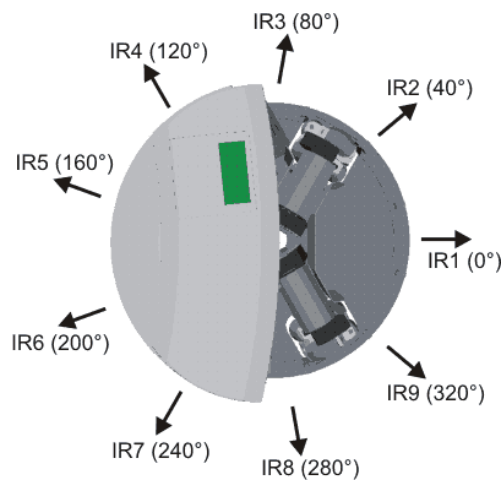


Figure 3.6: Robotino's Infrared Sensor Array for Navigation and Obstacle Detection

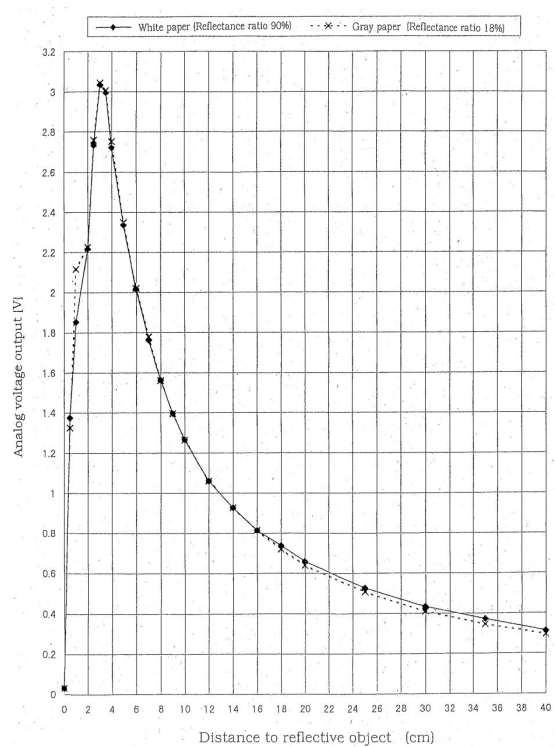


Figure 3.7: Graph Illustrating Sensor Output with Distance to Objects

Inductive Sensor

The inductive sensor is capable of detecting metallic objects both beneath and on the floor. It is utilized, for instance, in path control and precision positioning applications.

Optical Sensor

The two included optoelectronic sensors, also known as diffuse light sensors, can detect various surfaces and colors based on their different reflectance properties. These sensors enable the Robotino to follow a predefined path or to precisely stop at a specified position.

Gyroscope

Robotino is equipped with a gyroscope, which enhances positioning accuracy by detecting changes in its orientation.

Bumper

The bumper on the Robotino ensures that program execution, as well as motion, is stopped in the event of a collision.



Figure 3.8: The Robotino's Safety Bumper

Camera

The camera on the Robotino produces a live image that can be analyzed for navigation purposes, as well as for detecting obstacles and objects.



Figure 3.9: The Onboard Camera Module for the Robotino

3.1.4 Interfaces

Communication between the subsystems for the exchange of information.

WLAN

Robotino can be connected to a smartphone or a PC, or can be integrated into an existing network infrastructure via Wireless Local Area Network (WLAN). The Robotino can be operated and controlled from an external device via the network connection.

I/O Interface

Extensions such as actuators and sensors can be connected via the Input/Output Interface (I/O Interface) on the Robotino. The interface includes digital inputs and outputs, analog inputs and relay outputs, as well as power supply connections.



Figure 3.10: The I/O Interface of the Robotino Displaying the Connections

Connection	Input	Output
Port 11	Gripper	
D1	Acknowledgement pushbutton (NC)	Green
D2		Yellow
D3	Diffuse light sensor right	Red
D4	Diffuse light sensor right	
D5	Diffuse light sensor left	
D6	Diffuse light sensor left	
D7	Optical sensor gripper	
D8	Optical sensor gripper	
A1	Inductive sensor base	
A2		
A3		
A4		
A5		
A6		
A7		
A8		

Table 3.1: The I/O Table of the Robotino Displaying the Connections

Motor/Encoder

Robotino is equipped with a fourth motor output and encoder input for connecting an additional motor and encoder.

USB

The camera, the WLAN-USB adapter, and other components are connected to the control unit on the Robotino via the 6 available USB ports. The USB ports comply with the USB 2.0 specification.

PCI Express

The two PCI express slots on the Robotino make it possible to integrate expansion cards for individual applications.

Ethernet

You can establish a direct connection from your PC to the control computer in the Robotino via the Ethernet port to the embedded PC.

VGA

You can connect a monitor to the VGA output if you would like to be able to directly access the operating system in the Robotino. A keyboard and a mouse can be connected to the USB ports.

3.1.5 Power Supply

Information about power supply and accessories.

Batteries

Robotino is powered by two series-connected, rechargeable 12V batteries, supplying it with 24V DC power.

Power Supply Unit

A power supply unit is used to charge the batteries in the Robotino.

Charging Electronics

Charging electronics are integrated into Robotino to ensure safe and efficient charging of its batteries.

3.2 Software

Robotino is programmable in several languages, including C, C++, Java, .NET, Matlab, LabVIEW, and Microsoft Robotics Developer Studio. There is also support for SmartSoft and ROS [16].

3.2.1 Web Interface

The web interface of the Robotino makes functions available for control, configuration, and maintenance of the robot system. Through the web interface, we can control the robot with our computer and phone, check battery status and change network settings.

3.2.2 Programming

Robotino View

Robotino View is the interactive graphical programming environment for Robotino, enabling the creation and execution of control programs. It simplifies programming by representing hardware components as function blocks. Function blocks are self-contained modules that perform specific functions or tasks. They simplify complex processes by allowing for modular, reusable, and maintainable code. Robotino View also provides tools for image processing (Line Recognizer, Color Range Search, Marker Recognition) and navigation (Position driver, Path driver, Obstacle avoidance). Users can easily download and run programs on Robotino and even create their own function blocks in C++. Figure 3.11 shows an example of a Robotino View program.

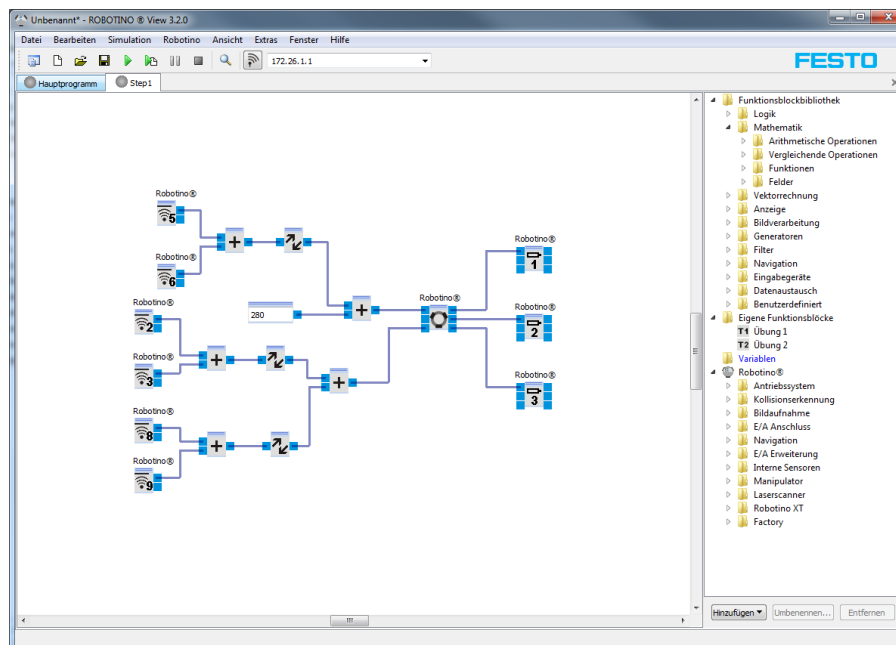


Figure 3.11: The Robotino View Software Interface

Python

Python is a versatile and widely used programming language known for its simplicity, readability, and powerful features. It is essential in various academic and professional areas, including robotics, data science, web development, and more [11]. In the context of our bachelor, Python plays a crucial role, particularly in robotics applications like Peter Corke's toolbox and within the Robot Operating System (ROS).

One of Python's key strengths is its ease of learning and readability, making it accessible to beginners while remaining powerful enough for experienced developers. Its clean syntax and straightforward structure allow for rapid development and prototyping, which is invaluable in academic settings where quick iterations and experimentation are common.

Python's wide range of libraries further enhances its utility. For robotics applications, libraries like Peter Corke's toolbox provide specialized tools for tasks such as robot kinematics, dynamics, and control. Meanwhile, within ROS, Python serves as one of the primary languages for building robot control systems, implementing algorithms, and interfacing with sensors and actuators.

Overall, Python is very useful for our bachelor's studies because of its simplicity, readability, wide range of libraries, and popularity, especially in robotics applications like Peter Corke's toolbox and ROS. Its easy access and powerful features enable students to understand complex concepts, create innovative solutions, and help advance robotics and related fields.

Matlab/Simulink

Matlab/Simulink is also a versatile and widely utilized software platform known for its comprehensive functionality and user-friendly interface. It serves as a foundation in various academic and professional areas, including robotics, control systems, signal processing, and more [13].

Similar to Python, Matlab/Simulink has a wide range of toolboxes and libraries that enhance its utility. For robotics applications, toolboxes like Robotics System Toolbox provide specialized functions for tasks such as robot kinematics, dynamics, and trajectory planning. Additionally, Simulink serves as a powerful environment for modeling and simulating complex dynamic systems, enabling students to develop and test control algorithms efficiently.

Overall, Matlab/Simulink could be useful for our bachelor's studies due to its user-friendly interface, comprehensive features, and broad usage, for example in robotics. Its accessibility and advanced tools enable students to delve into complex control systems and develop creative solutions.

3.2.3 Simulations

Robotino SIM

Robotino SIM is a simulation environment for experimenting with Robotino. As shown in figure 3.12, Robotino SIM provides you with a virtual Robotino in an experimentation environment. It allows you to control the Robotino with Robotino View. This allows us to simulate our Robotino View code, validating and refining our program before deploying it on the physical robot. This approach helps prevent potential damage to the robot and simplifies the process of resetting and experimenting with new functions

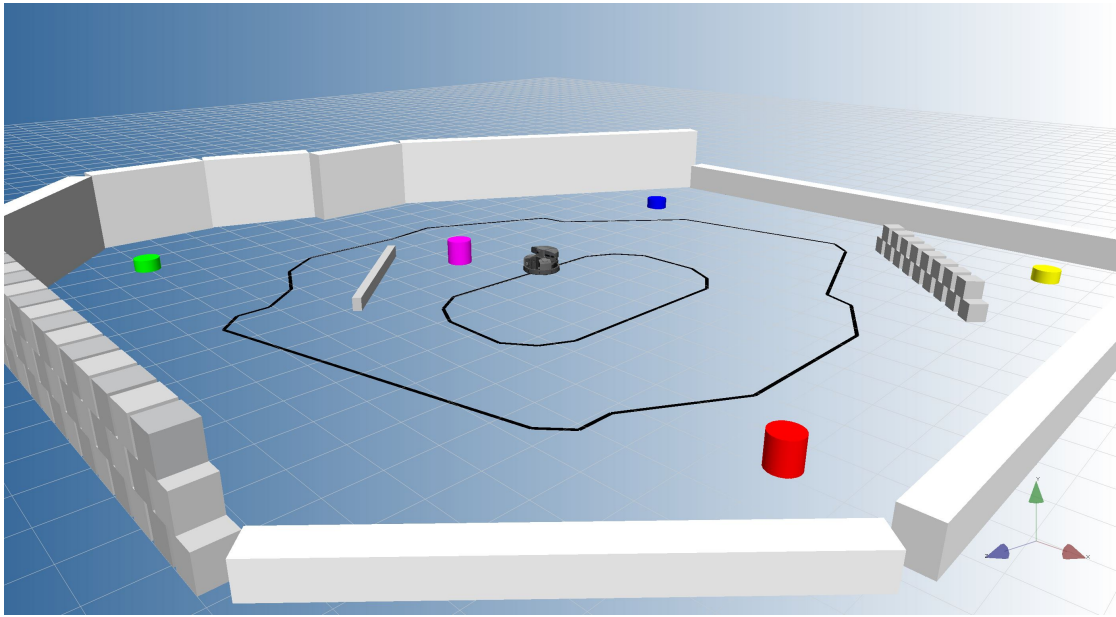


Figure 3.12: Snapshot of the Robotino SIM Simulation Environment

Robot Operating System

ROS is an open-source framework for developing robotics software. It provides tools and libraries to simplify the development of robust and flexible robotic applications. [12]. ROS handles communication between different software components of a robot, manages hardware control, messaging between processes, and package management. It supports various programming languages and is widely used in research, education, and industry to speed up the development of autonomous and interactive robots.

3.3 Control Systems and Strategies

3.3.1 PID Controller

A PID controller is a type of feedback control system used in engineering and automation to regulate processes. PID stands for Proportional, Integral, and Derivative, which are the three main components of the controller [34].

- **Proportional (P):** This component responds to the current error between the desired setpoint and the actual value. It adjusts the control output in proportion to this error.
- **Integral (I):** This component considers the accumulation of past errors over time. It helps to eliminate any steady-state error by continuously adjusting the control output based on the history of errors.
- **Derivative (D):** This component predicts the future behavior of the error based on its rate of change. It helps to anticipate changes in the system and adjusts the control output accordingly to prevent overshoot or oscillations.

By combining these three components, a PID controller can effectively regulate a system's behavior, maintaining it close to the desired setpoint while minimizing overshoot and settling time. It's widely used in various applications such as temperature control, speed control, and robotics. For the Robotino, velocity control of each motor is performed by a PID controller:

$$u(t) = K_p \left(e(t) + \frac{1}{T_N} \int_0^t e(t') dt' \right) + K_d \dot{e}(t)$$

The parameters are:

- K_p
- $K_i = \frac{1}{T_n}$
- K_d

The controller parameters are calculated as:

$$K_p = \frac{k_p}{2}, \quad K_i = \frac{k_i}{1024}, \quad K_d = \frac{k_d}{2}$$

Default values are:

$$k_p = 25, \quad k_i = 25, \quad k_d = 25$$

3.3.2 Closed-Loop Controller

A closed-loop control system autonomously adjusts a system to sustain a desired state or set point without human intervention. It operates through a feedback mechanism or sensor to achieve this regulation. Imagine driving a car and trying to maintain a steady speed. You check the speedometer often and make small adjustments to the gas pedal to keep your speed close to the desired value. This ongoing monitoring and adjustment process is what closed-loop control does. Basically, the system compares the actual output to a desired set-point and makes corrections to minimize any differences. For Robotino 3, closed-loop control means it can navigate, interact with its environment, and perform tasks accurately by adjusting its actions based on feedback from sensors [33].

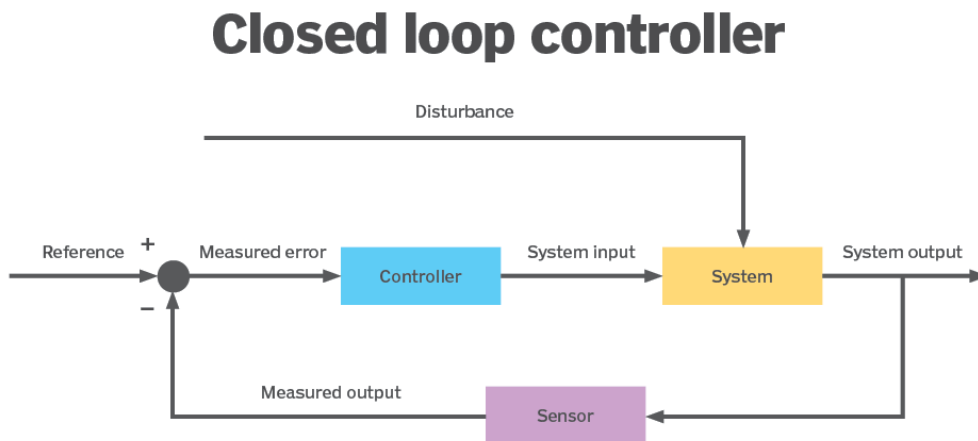


Figure 3.13: Example of a Closed Loop Controller

Advantages of Closed Loop Control Systems

- Ability to control for external factors.
- Provides a more reliable and stable output.
- Resilient to disturbances and changes.
- Utilizes resources more efficiently.

Disadvantages of Closed Loop Control Systems

- Increased complexity.
- Requires tuning or integration processes.
- Susceptible to oscillation or runaway conditions.
- Sensor failure can lead to unwanted system performance.

3.4 TCP/IP

TCP/IP, or Transmission Control Protocol/Internet Protocol, is a data link protocol used to let computers and other devices send and receive data over the internet. It determines how computers transfers data from one device to another. [30].

Transmission Control Protocol (TCP): TCP is responsible for how data is sent and received between devices. It divides data into smaller packets, gives each one a sequence number for tracking purposes, makes sure they're delivered in the right order, and asks for packets to be resent if any are lost or damaged. Think of TCP as a dependable delivery service that ensures your messages are delivered correctly.

Internet Protocol (IP): IP takes care of addressing and routing data packets through various networks. Every device connected to the internet, whether it's a computer or a smartphone, gets a unique IP address. IP makes sure that the data packets find their way from the sender to the correct recipient, navigating through multiple networks if necessary. It functions much like a postal system, directing your data to the right place.

TCP/IP functionality is divided into four layers, each with its own set of protocols [31]:

1. **Application Layer:** Handles communication between applications like web browsers, email clients, and file transfer programs. Protocols include HTTP, FTP, SMTP, and DNS.
2. **Transport Layer:** Manages communication between devices. TCP ensures reliable data delivery, while UDP is faster but less reliable.
3. **Internet Layer:** Routes data packets between different networks using IP. It's like the postal service for the internet.
4. **Network Access Layer:** Connects devices within the same network. Protocols like Ethernet and ARP handle communication between nearby devices.

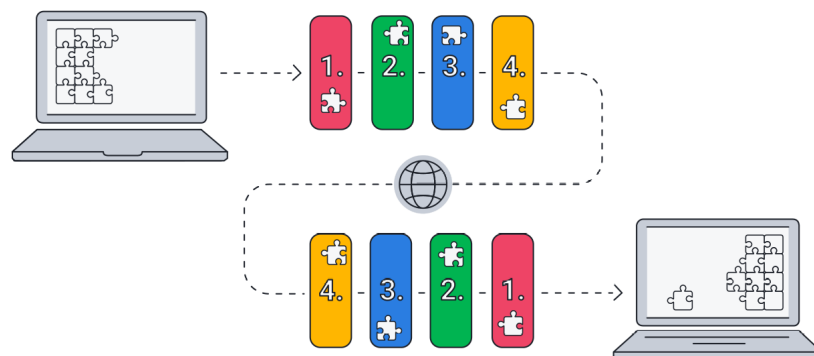


Figure 3.14: The Four Layers of TCP/IP

3.5 3D-printing

3D printing, also known as additive manufacturing, is a process of creating three-dimensional objects from a digital model. It works by laying down successive layers of material—such as plastic, metal, or resin—until the object is fully formed. This method allows for the production of complex shapes and geometries that may be difficult or impossible to achieve with traditional manufacturing methods. It offers advantages such as rapid turnaround times, reduced material waste, and the ability to create highly tailored designs [25].

We have access to a 3D-rinter on campus, specifically the Makerbot Replicator Z18, as shown in figure 3.15. The MakerBot Replicator Z18 is a professional-grade desktop 3D printer known for its large build volume of approximately 12 x 12 x 18 inches. It features dual extruders for multi-material printing and supports various filament types. With user-friendly software and robust construction, it's suitable for rapid prototyping, product development, and small-scale production projects [26].



Figure 3.15: Makerbot Replicator Z18

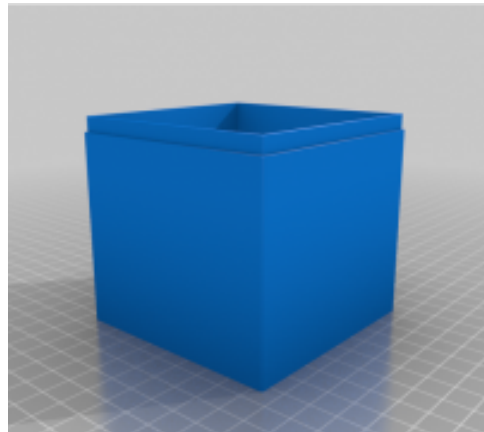


Figure 3.16: Example of a 3D-Printed Object That Could Be Utilized for Our Project.

Tinkercad is used to design precise 3D models. After designing our model in Tinkercad, we can print these models using the 3D printer available on Campus. Tinkercad's easy-to-use interface helps us create accurate designs tailored to the gripper's measurements. This process allows us to efficiently produce custom parts, like the one shown in figure 3.16, that optimize the gripper's performance within our robotic system.

Chapter 4

Design and Implementation

4.1 Kinematic Model

As previously mentioned, a kinematic model is a mathematical way of explaining how a robot moves without worrying about forces. It focuses on the shapes and positions of the robot's parts and how they relate to each other. To develop this kinematic model, we primarily relied on the concepts presented in the Engineering Educator Academy's lecture on the Kinematics of Mobile Robots with Omni Directional Wheels [18] and Chapter 4 in Peter Corke's Robotics, Vision and Control .

Given the desired linear and angular velocities in V_b , the speed of each wheel represented by the U matrix can be calculated through the $H(0)$ matrix.

$$\mathbf{U} = \mathbf{H}(\mathbf{0}) \cdot \mathbf{V}_b$$

where

$$\mathbf{U} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}$$

represents the robot's angular velocity on each wheel needed to achieve the desired motion given by

$$\mathbf{V}_b = \begin{bmatrix} \omega_{bz} \\ v_{bx} \\ v_{by} \end{bmatrix}$$

where ω_{bz} represents the desired angular velocity or rotation of the robot, v_{bx} and v_{by} represents the desired velocity in the x and y direction, respectively.

$$\mathbf{H}(\mathbf{0}) = \frac{1}{r} \begin{bmatrix} L & -\frac{\sqrt{3}}{2} & \frac{1}{2} \\ L & 0 & -1 \\ L & \frac{\sqrt{3}}{2} & \frac{1}{2} \end{bmatrix}$$

$H(0)$ is the Jacobian matrix at the robot's initial configuration, where r is the radius of the wheels and L is the distance from the center of the robot to each wheel. This matrix essentially translates the desired motion given by V_b to angular velocities (U) needed on each wheel to realize that desired motion. For the full calculations and explanations, see appendix A

$$U = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \frac{1}{r} \begin{bmatrix} L & -\frac{\sqrt{3}}{2}L & \frac{1}{2}L \\ L & 0 & -L \\ L & \frac{\sqrt{3}}{2}L & \frac{1}{2}L \end{bmatrix} \begin{bmatrix} \omega_{bz} \\ v_{bx} \\ v_{by} \end{bmatrix}$$

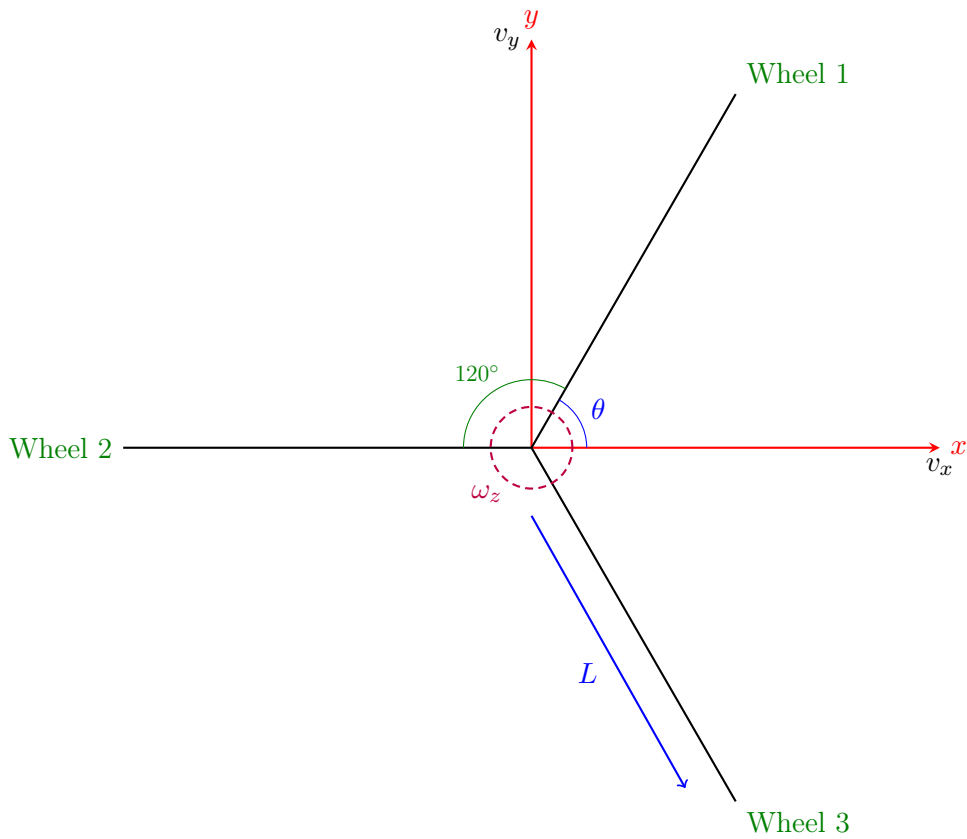


Figure 4.1: The Kinematic Model of Robotino's Drive System

4.2 Sensor Utilization

In this section, we'll look at how Robotino's sensors are used for navigation. Each sensor has a specific role in detecting obstacles, maintaining distance, and ensuring precise movement. By carefully placing these sensors, Robotino can navigate accurately and efficiently. Let's see how each sensor helps in different situations. We've also conducted accuracy tests on the sensors, as detailed in chapter 5.

Distance Sensor

The distance sensors play a crucial role in ensuring Robotino's safe navigation around obstacles. For example, when Robotino needs to approach a wall while maintaining a constant distance, we depend on sensor 1 positioned at 0 degrees. When navigating corners or following walls, we utilize different combinations of sensors. For instance, to execute a precise 90-degree turn along a wall, we employ sensors 2 and 9 positioned at 40 degrees and 320 degrees to ensure accurate maneuvering. Figure 4.2 illustrates sensor 1 positioned at 0 degrees, maintaining a distance of 60mm from an object directly ahead.

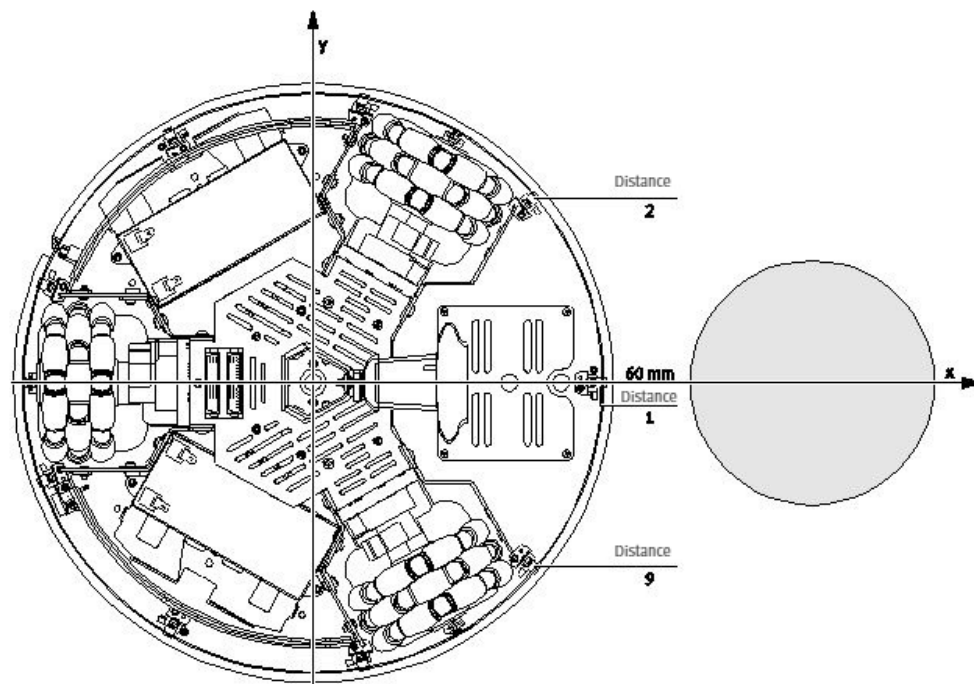


Figure 4.2: Sensor Layout for Robot Obstacle Approach

Distance Sensor Characteristic

Distance (cm)	Voltage (V)	Distance (cm)	Voltage (V)
1	1.95	21	0.59
2	2.55	22	0.59
3	2.70	23	0.55
4	2.32	24	0.49
5	2.00	25	0.46
6	1.76	26	0.44
7	1.71	27	0.42
8	1.46	28	0.40
9	1.23	29	0.38
10	1.14	30	0.36
11	1.03	31	0.34
12	0.97	32	0.32
13	0.90	33	0.31
14	0.83	34	0.30
15	0.77	35	0.29
16	0.72	36	0.28
17	0.69	37	0.26
18	0.65	38	0.25
19	0.62	39	0.24
20	0.59	40	0.24

Table 4.1: Robotino 3 Distance Sensor Measured Values

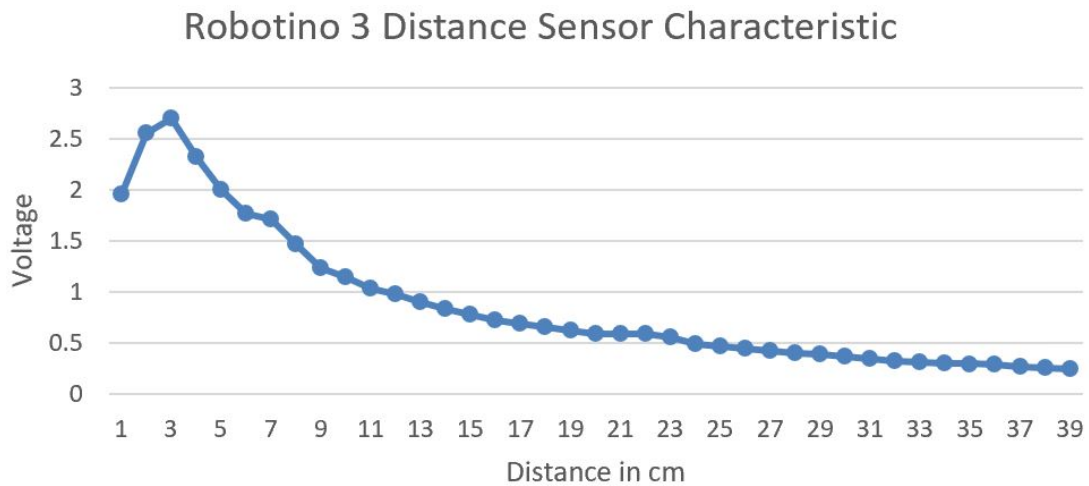


Figure 4.3: Robotino 3 Distance Sensor Characteristic

A useful method for dealing with characteristic curves is to simplify them by making them linear within certain ranges. In our project, we've chosen to focus on the 5 to 10 cm range. This makes the curve easier to work with and understand.

1. General linear equation formula:

$$D = MX + B$$

2. Measured values for the start point and end point:

$$D_1 = 5 \quad X_1 = 2.00 \quad , \quad D_2 = 10 \quad X_2 = 1.14$$

3. Calculate the slope (M) of the linearization lines:

$$M = \frac{(D_2 - D_1)}{X_2 - X_1} = \frac{(10 - 5)}{1.14 - 2.00} = -5.81$$

4. Calculate the zero point offset B of the linearization lines:

$$5 = -5.81 \times 2.00 + B$$

$$5 = -11.62 + B$$

$$B = 16.62$$

5. Formula for calculating the distance X (in cm) from the voltage value D for the range from 5 to 10 cm:

$$D = -5.81 * X + 16.62$$

Inductive Sensor

Inductive Sensors primarily detect metal objects through electromagnetic induction. They are commonly used to detect metallic surfaces like aluminum strips, making them suitable for navigation and object manipulation in environments where such surfaces are present. For example, we could use this sensor to accurately follow paths marked with aluminum tape.

The Inductive Sensor on our robot are analog, giving constant output signals that change based on distance to metal objects. We use the output signals from the Inductive Sensor along with analog input function blocks in Robotino View. These blocks help us understand the continuous signals from the sensor and use them for precise navigation. By integrating the sensor's characteristic curve, which shows how it behaves when moving over the tape, we can calibrate the robot's movements effectively. This calibration ensures that the robot follows paths marked with aluminum tape accurately, adjusting its path based on real-time feedback from the sensor. This approach helps our robot navigate with improved accuracy and reliability towards its goals.

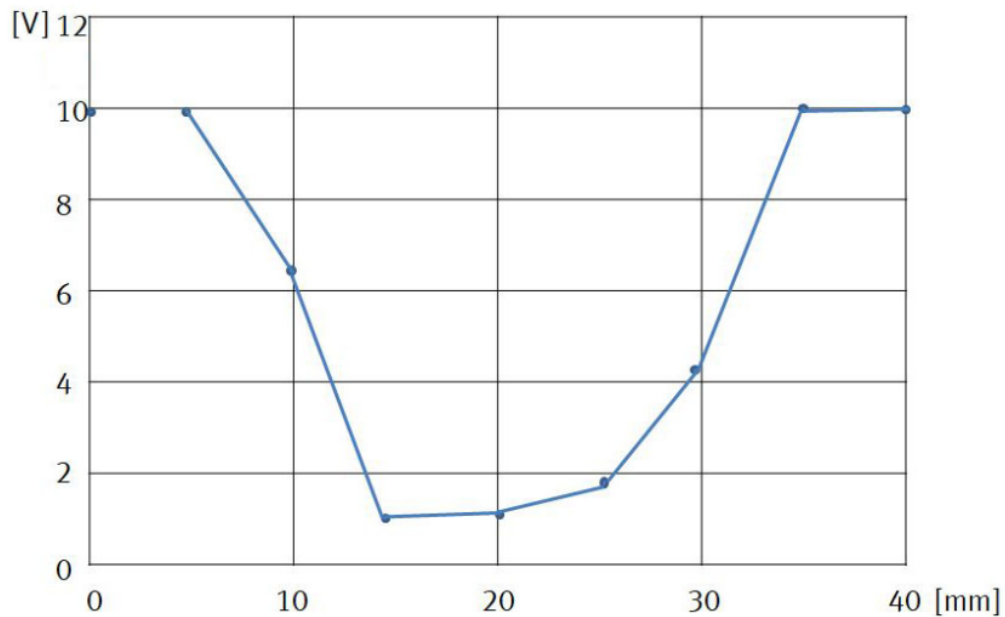


Figure 4.4: Voltage Response of Sensor to Varying Distances from an Aluminum Tape

Optical Sensors

Optical Sensors utilize light to detect objects and surfaces. They can detect a wider range of materials, including non-metallic surfaces and colored markers. They are employed in scenarios where colored markers or non-metallic surfaces are used for guidance or object detection. For example, they could be used to follow colored lines or markers on the ground for navigation or alignment purposes.

The Optical Sensors are digital, meaning they give clear "on" or "off" signals instead of continuous analog signals. Robotino View easily handles these signals with digital input blocks, made for binary data. With straightforward signals, the robot's control system can promptly react to detected objects or surfaces, making navigation and object detection tasks smooth and efficient.

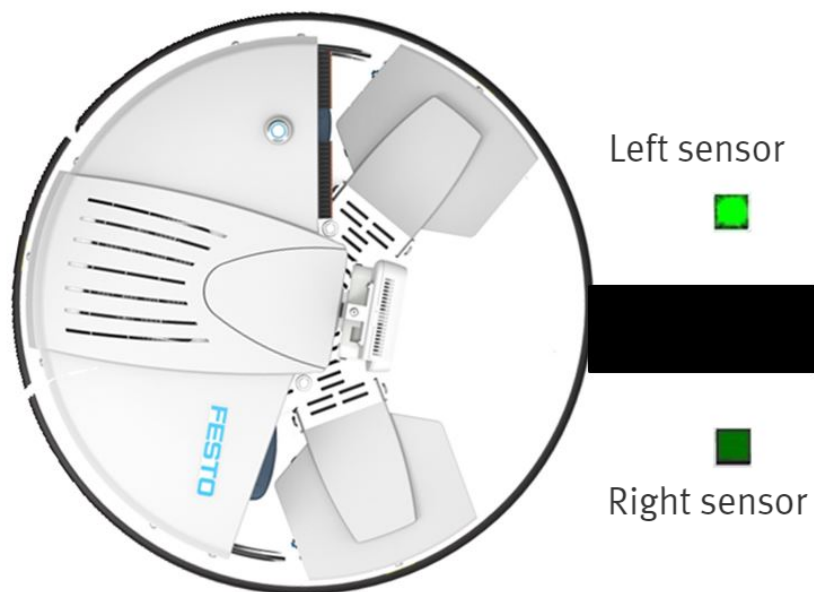


Figure 4.5: Optical Sensor Reaction to Robotino's Position Relative to the Marked Line

Bumper

The Robotino's bumper is a crucial safety feature that detects obstacles in its path. When it senses contact, it signals the control system to stop the robot's movement immediately. This quick response prevents accidents and allows for necessary adjustments by the operator. Integrating the bumper into the robot's programming allows for easy implementation of collision avoidance strategies, making operations safer and more reliable.

Camera

The camera on Robotino 3 is a versatile tool. We mainly use it to find and identify objects in the robot's surroundings. It can detect objects by their color using features like the color range finder, which we configure in Robotino View. After detecting an object, the robot can track it using tools like the segment tracker block. In figure 4.8 we show how the color range finder is used to define the color of a blue plastic box.

The camera on Robotino 3 doesn't just rely on color to detect objects. It also uses other methods like recognizing shapes, analyzing textures, and matching patterns. These techniques help the robot identify objects regardless of their color and differentiate between different surfaces or materials. We also use the camera for marker detection, which can be generated in Robotino View (figure 4.6). We found this approach working better for us, especially in Lab Exercise 4, outperforming color detection.

Marker selection

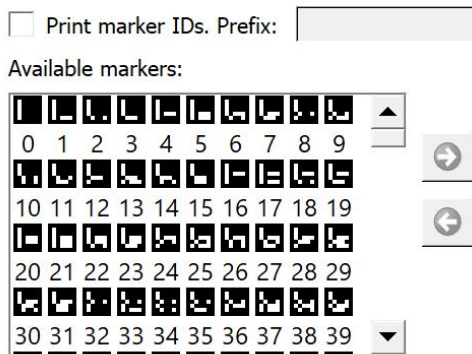


Figure 4.6: Generate Markers

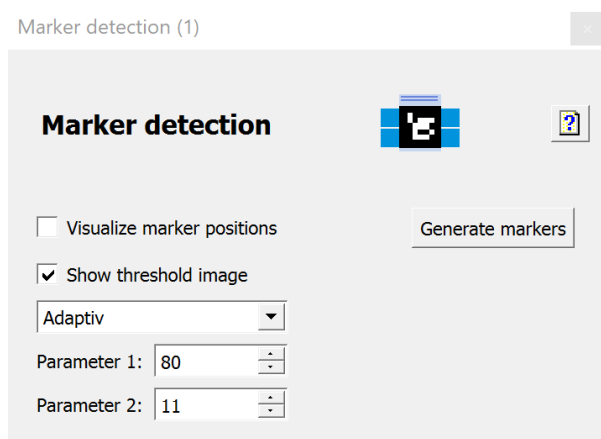


Figure 4.7: Marker Detection Block

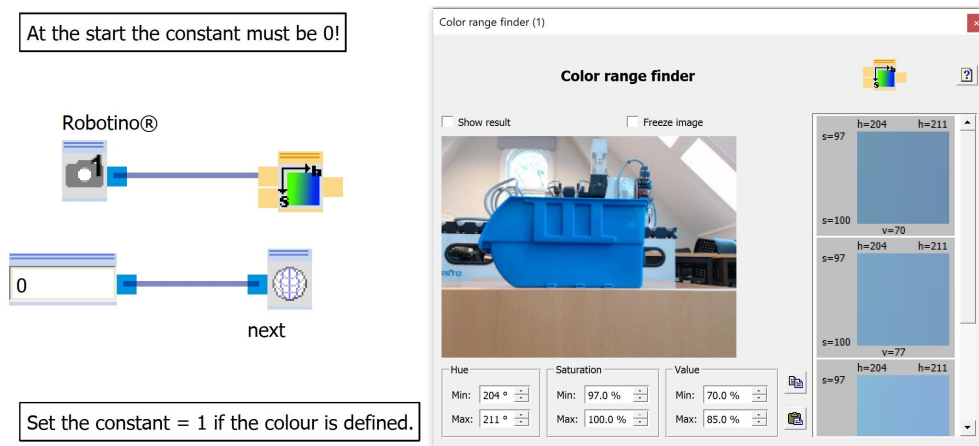


Figure 4.8: Color Range Finder Function Block

Gyroscope

The gyroscope in Robotino 3 is crucial for maintaining balance and orientation during its operations. By measuring rotation rates and providing feedback to the control system, it helps adjust the robot's movements as needed. Integrating gyroscope data into control algorithms ensures precise navigation and stability, especially on uneven terrain or when precise orientation is required.

Using Multiple Sensors for Accurate Task Execution

Combining different sensors is crucial for robots to do tasks accurately. In Lab Exercise 4, see appendix E, the Robotino utilizes its camera to pinpoint the module's location and then navigates towards it. Then, utilizing either an inductive or optical sensor, the robot moves along a marked path on the ground to align perfectly with the module. A distance sensor ensures Robotino 3 maintains the correct distance and alignment from the module, allowing its gripper to grab it accurately. The robot will proceed slowly towards the object until the acknowledge button is pressed, causing the robot to stop moving and the gripper to secure the object. After grabbing the object, the robot will move backwards away from the module. If by any chance the robot's base unexpectedly hits the module, the bumper ensures that the program is stopped to prevent damage to the robot. This combination of sensors ensures Robotino's accuracy and safety in dynamic environments.

4.3 Software Development

Robotino View

To develop the lab exercises, we primarily relied on Robotino View. While there are various methods to control the robot, using this software proved to be the most efficient. Throughout the project, we explored other methods to control the robot, including TCP/IP and ROS, but opted against these due to lack of time and the specified requirements. Function blocks makes it easier to create a program, due to . Additionally, by using Robotino View, we were able to test our program in Robotino SIM to ensure the program functioned as intended. This allowed us to simulate the robot's movement and behavior without risking damage to the robot's hardware.

While we had some background in programming with function blocks, we still had to learn how to use this software. We started by taking an e-learning course titled "Autonomous Mobile Robotics with Robotino", provided by Festo. This course taught us about the functions and features of the Robotino and how to use them. It was a great learning experience that really deepened our understanding of the robot and its software.

All the programs that we have created for this projects can be found in Appendix B to Appendix E. These appendices include the solution for each lab exercise, showing the different function blocks used.

TCP/IP

As we already have established, Robotino uses a TCP/IP protocol as a way of communicating between the robot itself and a computer. Knowing how to properly use this protocol can allow for different ways to communicate with the robot outside of using the web interface or the first party programming tool Robotino View.

We used a program called Wireshark to read the packages being sent between the robot and our computer. These packages can be translated into usable code so that you can write your own program, in for instance Python, to control the robot. Even though this method would provide a new way of controlling the robot, we decided not to use it because it exceeds our understanding of the protocol and the correct way to decrypt the packages.

Visualizing Kinematic Model with Python

When developing the kinematic model for the robot, we aimed to visually represent how the velocity vectors change based on the desired velocity inputs, computed through the Jacobian matrix. As shown in Figure 4.9, the colored dots represent the wheels in relation to the center of the robot body. The arrows extending from each wheel are scaled velocity vectors assigned to each wheel. In this instance, all arrows point in the positive rolling direction of their respective wheels with the same magnitude. This indicates that the desired velocity is purely rotational. The wheel speeds as shown in the upper left corner are converted to Revolutions Per Minute (RPM), and are applied to the motors before the gearbox. The calculated RPM values can be directly utilized in Robotino View, and would give an expected result to the movement of the robot. The full code can be found on Github, appendix I.

Robot Wheels, Velocity Vectors, and Linear Velocity Direction

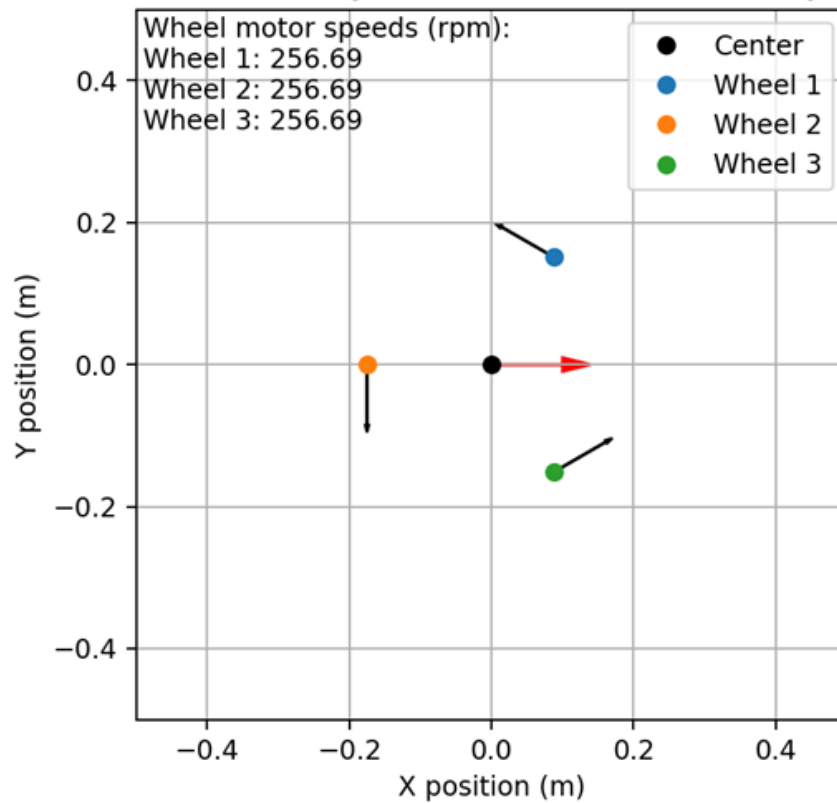


Figure 4.9: Visual Representation of the Kinematics

4.4 Simulations

Robot Operating System

We used a Python script available online to simulate the Robotino in ROS [20]. We made slight modifications, such as removing the Kinect camera, which our robot does not possess. Although the script was intended for a 3-wheeled mecanum robot, we utilized this code even though our model features omniwheels. This was done to demonstrate the movement of a holonomic robot within Gazebo. Since we had previously installed ROS for robot simulations in ELE306 Robotics, we simply downloaded the required files and transferred them to our Virtual Machine for code modification.

To launch Gazebo, spawn the robot, start the twist controller, and open RViz, run this command in the terminal:

```
ros2 launch robotino_bringup robotino_bringup.launch.py
```

As shown in figure 4.10, we demonstrate the capabilities of RViz, the ROS Visualisation, which offers a 3D visualization platform within the Robot Operating System. The image displays a model of a Robotino, showing its design and functions. Additionally, figure 4.11 presents a simulation environment rendered in Gazebo, widely utilized in the field of robotics. While our current robot model does not include Lidar, the Gazebo simulation offers a conceptual visualization of how Lidar technology could be integrated and function within an environmental setting.

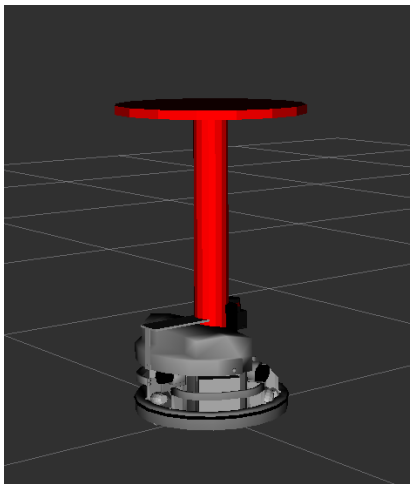


Figure 4.10: Visualization of Robotino in RViz

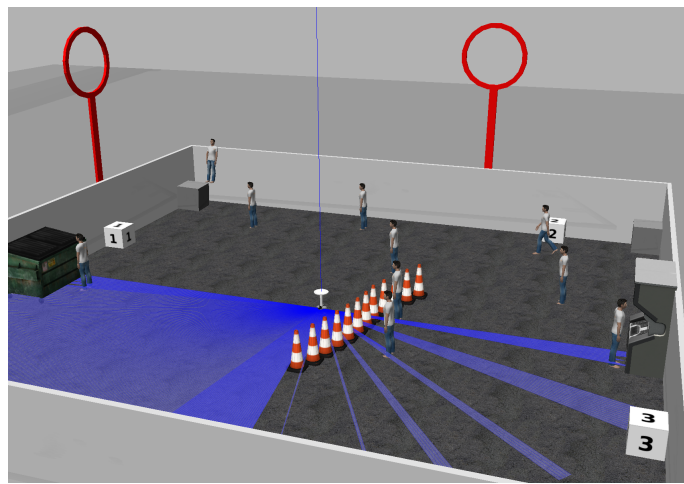
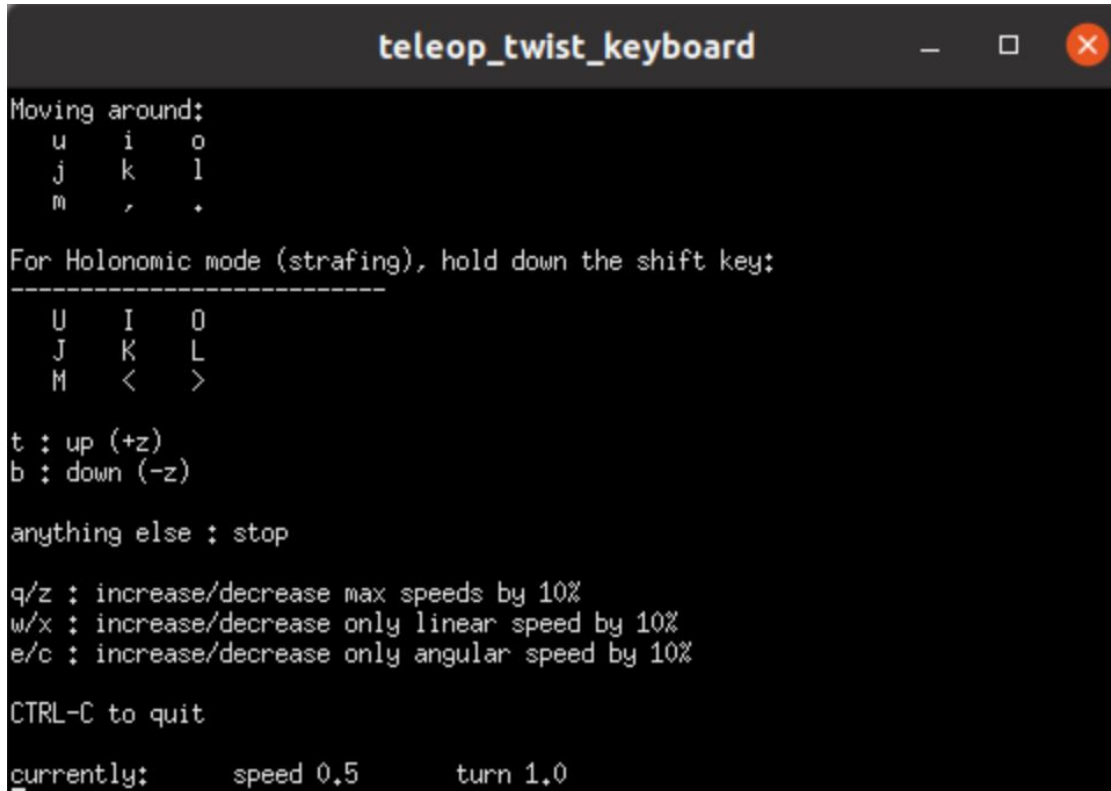


Figure 4.11: Simulation of Robotino in a Virtual Environment Using Gazebo

To control the robot in Gazebo, we utilize a twist controller in ROS. This allows us to manually control the robot in both the conventional mode and holonomic mode, enabling strafing movements. This setup enables us to observe the robot's movement within its environment. You can find a video showing the robot's movements in appendix H.

The image shows a terminal window titled "teleop_twist_keyboard". The text inside the terminal is as follows:

```
Moving around:
  u   i   o
  j   k   l
  m   ,   .

For Holonomic mode (strafing), hold down the shift key:
-----
  U   I   O
  J   K   L
  M   <  >

t : up (+z)
b : down (-z)

anything else : stop

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%

CTRL-C to quit

currently:      speed 0.5      turn 1.0
```

Figure 4.12: Twist Controller for Robot Operation in Gazebo

We haven't extensively utilized ROS beyond this point. Although there's a method of employing ROS packages to control the Robotino, it would require installing software directly onto the Robotino, which the task provider opted against. As utilizing ROS was not a requirement for our thesis, we haven't explored it further.

Python Simulation of Robot Kinematics

As seen in Figure 4.13, the robot is visualized with simple geometric shapes. The blue circles depicting the wheels of the robot and the grey circle representing the robot body. The kinematics are applied to the robot and the corresponding movement is animated. The full code can be found on Github, appendix I.

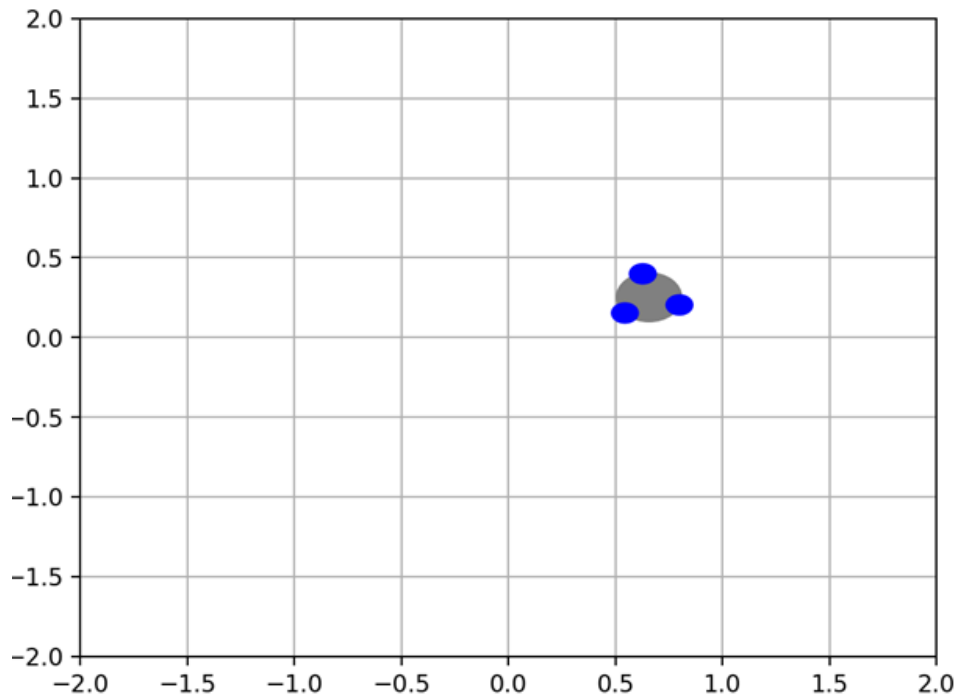


Figure 4.13: A Simple Geometric Model of the Robot with Animated Movement from Kinematics. Created in Python

4.5 3D Printing

In our technical overview, we initially considered using 3D printing to create objects for our tasks. This approach would have allowed us to customize objects to fit specific requirements or to test different designs. However, we discovered that Festo provided accurate objects that were fully compatible with our gripper, eliminating the immediate need for 3D printing.

If we were to use a different type of gripper in the future, we might need to revisit 3D printing to ensure compatibility. Additionally, we could conduct tests to compare the accuracy and performance of Festo-provided objects with those we 3D print ourselves. This comparison would help us determine the best approach for different tasks and gripper configurations.

Chapter 5

Testing

We've conducted a series of tests using the sensors equipped on the Robotino 3 to assess their accuracy and determine the most effective ways to utilize them in our scenarios.

5.1 Distance Sensor Accuracy Test

We wanted to check if the IR sensors could accurately detect different objects from varying distances. We selected various objects that were easily available to us and got these results:

Object	Detection Result					
	5 cm	15 cm	25 cm	35 cm	45 cm	55 cm
Wooden Table	Yes	Yes	Yes	Yes	Yes	Yes
Metallic Surface	Yes	Yes	Yes	Yes	Yes	Yes
Moving Object	Yes	Yes	Yes	Yes	Yes	Yes
Transparent Bottle	Yes	Yes	Yes	Yes	Yes	Yes
Plastic Box	Yes	Yes	Yes	Yes	Yes	Yes

Table 5.1: Detection Results for Various Objects at Different Distances

5.2 Odometry Accuracy Test

We utilized a Robotino View program to compare the performance of odometry between Robotino SIM and the physical robot. In the simulator, the robot navigated flawlessly, consistently stopping at the intended positions as expected in a simulated environment. However, on the actual robot, it encountered difficulties in maintaining its navigation accuracy. The program frequently deviated from its intended path, causing the starting position to shift with each iteration. This deviation worsened over time, making the odometry increasingly unreliable for repeated loops. We later resolved these issues by resetting the odometry at a calibrated position in the real world. This made the deviations small enough to where we could still use the odometry with sufficient accuracy.

There are various reasons that odometry was off on the actual robot compared to simulations. One reason is the difference in the real-world environment compared to the simulated one. Factors such as surface friction, wheel slippage, uneven terrain, or unexpected obstacles can affect the robot's movement and consequently its odometry readings. Additionally, variations in hardware performance or calibration differences between the simulated and physical robot could also play a role.

5.3 Omnidrive Function Block Test

We ran a test to assess the accuracy of the robot's ability to travel a distance of 1 meter using the omnidrive function block within Robotino View. The testing environment was the classroom at Fagskolen, where the floor surface was linoleum. We conducted the test at two different speeds, as shown in table 5.2 and table 5.3

The following applies to the calculation:

$$\begin{aligned}s &= v \cdot t \\ v &= 100 \text{ mm/s} \\ s &= 1000 \text{ mm}\end{aligned}$$

Thus, the time t required is calculated as:

$$t = \frac{s}{v} = \frac{1000}{100} = 10 \text{ s}$$

Distance: $d = 1 \text{ m}$ / velocity: $v = 100 \text{ mm/sec.}$ / time: $t = 10 \text{ s} = 10000 \text{ ms}$

Test	Distance travelled	Deviation from Target
1	0.98	-0.02
2	0.99	-0.01
3	0.99	-0.01
4	0.99	-0.01
5	0.98	-0.02
Mean value:	0.986	-0.014

Table 5.2: Omnidrive Test of 1 Meter With a Speed of 100 mm/sec

Distance: $d = 1 \text{ m}$ / velocity: $v = 400 \text{ mm/sec.}$ / time: $t = 2,5 \text{ s} = 2500 \text{ ms}$

Test	Distance travelled	Deviation from Target
1	0.95	-0.05
2	0.95	-0.05
3	0.96	-0.04
4	0.94	-0.06
5	0.95	-0.05
Mean value:	0.95	-0.05

Table 5.3: Omnidrive Test of 1 Meter With a Speed of 400 mm/sec

5.4 Camera Accuracy Test

Several factors influence a camera's ability to detect specific objects, including lighting conditions, the objects' characteristics, and the size of the markers used. In our tests, we utilized AR markers sized at 7x7 cm and 16x16 cm. Additionally, we used a blue plastic box as a colored object, which was placed under optimal lighting conditions. The goal was to determine the most effective approach for navigation and to identify the optimal size for the AR markers.

As shown in Table 5.4, the camera achieved the highest accuracy when detecting both the AR marker sized at 7x7 cm and the colored objects within a range of 35 cm to 140 cm. With the AR marker sized at 16x16 cm, the highest accuracy was recorded between 190 cm and 270 cm. Therefore, depending on the distance at which you intend to use the camera for object identification, you can use markers of various sizes.

Test	Distance from Object	Color Detection	Marker 7x7 cm	Marker 16x16 cm
1	35 cm	Yes	Yes	No
2	70 cm	Yes	Yes	No
3	105 cm	Yes	Yes	No
4	140 cm	Yes	Yes	No
5	175 cm	No	No	Yes
6	210 cm	No	No	Yes
7	245 cm	No	No	Yes
8	280 cm	No	No	Yes
9	315 cm	No	No	No
10	350 cm	No	No	No
11	385 cm	No	No	No
12	420 cm	No	No	No

Table 5.4: Camera Accuracy Test with Various Objects and Distances

5.5 Robot Repeatability in Lab Exercise 4

We ran tests to assess the robot's repeatability in this exercise. This involved navigating to the module, centering itself, driving towards the object, picking it up, reversing, moving towards another location, dropping off the object, and finally returning to the starting point. These steps are also demonstrated in the videos for Lab Exercise 4, available in Appendix H. Table 5.5 shows the results obtained using the camera line detector and uncalibrated starting position, while Table 5.6 displays the outcomes with marker detection and calibrated starting position. The marker detection method showed much better results and repeatability.

Test	Object Pickup	Object Drop-off	Drive to Start
1	Yes	Yes	Yes
2	Yes	Yes	Yes
3	Yes	No	No
4	No	No	No
5	No	No	No
6	No	No	No
7	No	No	No
8	No	No	No

Table 5.5: Performance Test With Line Detector and Uncalibrated Starting Position

Test	Object Pickup	Object Drop-off	Drive to Start
1	Yes	Yes	Yes
2	Yes	Yes	Yes
3	Yes	Yes	Yes
4	Yes	Yes	Yes
5	Yes	Yes	Yes
6	Yes	Yes	Yes
7	Yes	Yes	Yes
8	Yes	Yes	Yes
9	Yes	Yes	Yes
10	Yes	Yes	Yes
11	Yes	Yes	Yes
12	Yes	Yes	Yes
13	Yes	Yes	Yes
14	Yes	Yes	Yes
15	Yes	Yes	Yes

Table 5.6: Performance Test With AR-Marker and Calibrated Starting Position

Chapter 6

Results

6.1 Lab Exercises

Exercise 1: Introduction to Robotino 3 and Basic Motion

This laboratory exercise aims to install the necessary software, familiarize students with the hardware and software components of the Robotino, and practice basic motion control of the robot. They will make the robot move forward for 6 seconds, then pause for 2 seconds before moving backward for 6 seconds. Next, they will make the robot move linearly in any directions using omnidrive. Finally, they will create an abort function with the bumper in case of a collision.

For detailed instructions and complete information on this exercise, including solutions, please refer to appendix B.

Exercise 2: Distance Sensors and Odometry

This lab exercise is designed to familiarize students with Robotino's distance sensors and their application in navigation tasks. They will program the robot to approach a wall and maintain a distance of 8 cm. Then, they will guide the robot to move alongside the wall while maintaining this distance. Finally, they will navigate the robot along a wall with a 90° corner, using odometry to make the turn.

For detailed instructions and complete information on this exercise, including solutions, please refer to appendix C.

Exercise 3: Image Processing and Object Detection

This lab exercise is designed to familiarize students with image processing and AR-markers using the camera. They will develop a program that instructs the robot to locate a predetermined colored object, approach it, and maintain a distance of 8 cm from it. They will then repeat the process, this time using AR markers instead of colored objects.

For detailed instructions and complete information on this exercise, including solutions, please refer to appendix D.

Exercise 4: Multi-Sensor Navigation and Object Handling

This lab exercise involves guiding the robot towards a module using camera guidance, tracing a line with the diffuse sensors, maintaining proper distance and alignment with the module using distance sensors, and utilizing a gripper to lift an object from the Belt/Distribution Module. After that, the robot will move away from the module, use odometry to reach a specific location, and drop off the object.

For detailed instructions and complete information on this exercise, including solutions, please refer to appendix E.

6.2 Identified Issues

Throughout the project, we found several issues within the system, and we worked on resolving some of them.

- **Unlabeled and Incorrectly Spliced Wiring:**
 - Troubleshooting and maintenance were difficult due to unlabeled and incorrectly spliced wires, causing delays in identifying and fixing issues.
- **Sensor Connectivity Issues:**
 - Wrong wiring made certain sensors unusable at first, which led to the robot not functioning as intended.
- **Camera Mounting:**
 - The current camera mounting setup was less than ideal, potentially impacting the quality of the footage. As a result, we elevated the camera slightly and secured it more effectively.
- **Defective batteries:**
 - Due to defective batteries, the system was constrained to remain plugged in, limiting its mobility. Therefore, we replaced the faulty battery with new ones, which improved the mobility of the robot.
- **Defective gripper:**
 - We had to replace the faulty gripper with a new one and make sure it was connected properly. Turns out, it was originally hooked up wrong, so we corrected that by connecting it to port 11 on the front panel [23], as you can see in figure 6.1

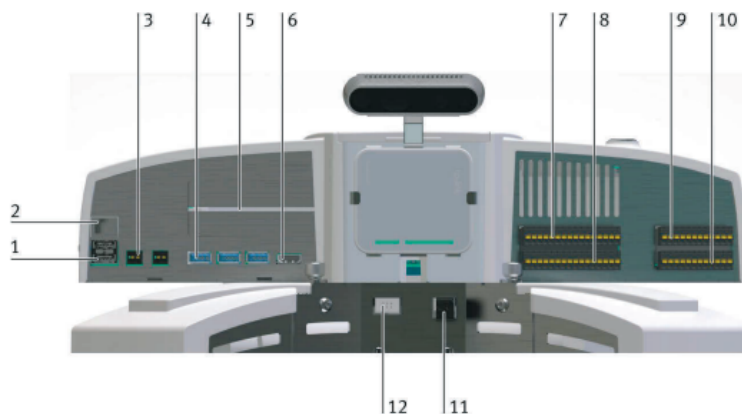


Figure 6.1: Overview of Robotino Front Panel with Connections

6.3 Discussion

We started off the project in early January by putting together a plan based on estimated tasks and their timeframes. This was created with a focus on the documentation requirements and academic aspects specified in the assigned task. To our surprise, we pretty much stuck to this plan as things progressed. The changes we made were minor, like toning down our focus on simulation and robot control with ROS. Apart from that, we managed to complete most of our tasks within their designated timeframes.

As outlined in Chapter 6.2, we encountered some issues that required our attention. These types of challenges are typical in projects like this, so we were somewhat anticipating them. Fortunately, we were prepared to tackle them head-on. We successfully addressed the issues that were hindering our ability to conduct the exercises effectively, particularly focusing on replacing the gripper, batteries, and fixing the camera mounting. These adjustments were crucial for us to proceed with the task, and without them, we would have faced some difficulties in completing our objectives.

Chapter 7

Conclusion

The results indicate that the majority of the objectives were achieved. We have developed a series of laboratory exercises highlighting the navigation capabilities of the Robotino 3 mobile robot. These four lab exercises aim to educate students about Robotino and its functionalities, as well as theoretical concepts from Peter Corke's "Robotics, Vision, and Control"

We have also developed a kinematic model that, along with software simulation, illustrates how the robot moves with its omnidrive system. Additionally, we've conducted several tests on the Robotino to evaluate the accuracy of its sensors, which could be beneficial for future projects.

Throughout the project, we have identified certain weaknesses in the system. While these weaknesses are not critical for task execution, they do suggest areas where improvements could be beneficial, as detailed in Section 7.1.

7.1 Room for Improvements

- **Integrating a Lidar Sensor:**

- Adding a Lidar sensor to the Robotino 3 would significantly enhance its capabilities. With Lidar, the robot could accurately perceive its surroundings in 3D, enabling it to create detailed maps and navigate precisely [22]. Lidar would also support Simultaneous Localization and Mapping (SLAM) algorithms [21], allowing the Robotino 3 to locate itself within the map it creates, even in complex and changing environments. This would enable advanced applications such as autonomous navigation, obstacle avoidance, and collaborative mapping tasks, greatly expanding the range of tasks the Robotino 3 can perform effectively.



Figure 7.1: Example of Lidar We Could Use: Hokuyo URG-04LX-UG01



Figure 7.2: Showcasing Robotino with a Hokuyo Lidar Mounted

- **Upgrading the Gripper:**

- The current gripper on the Robotino 3 is functional, but upgrading to one with a larger range of motion and vertical movement capability would offer significant advantages. A wider range of motion would allow the robot to handle various objects more effectively, regardless of their size or shape. Additionally, enabling vertical movement of the gripper would eliminate the need for precise object placement based on height, simplifying processes. These upgrades would improve the robot's efficiency in tasks such as object handling and pick-and-place operations, expanding its range of potential applications.

- **Integrate a Robot Arm:**

- Integrating a robot arm onto the Robotino 3 would significantly broaden its functionality. With a robot arm, the Robotino 3 could perform various manipulation tasks, such as object handling and assembly, expanding its range of capabilities beyond navigation alone. This enhancement would offer numerous opportunities for the Robotino 3 to tackle diverse real-world challenges with increased versatility and efficiency.

- **Integrating the Festo Distribution/Belt Station with the Robotino:**

- Integrating the Festo distribution/belt station to work alongside the Robotino could be a potential future project. Currently, we use this station solely to adjust the height for object pickup with the gripper. However, there is potential for expansion by integrating this station with the Robotino. This integration could enhance the overall efficiency of the system, allowing for smoother coordination between the Robotino and the distribution/belt station. While this isn't a current focus, it presents an opportunity for further optimization and collaboration between the different components of the system.

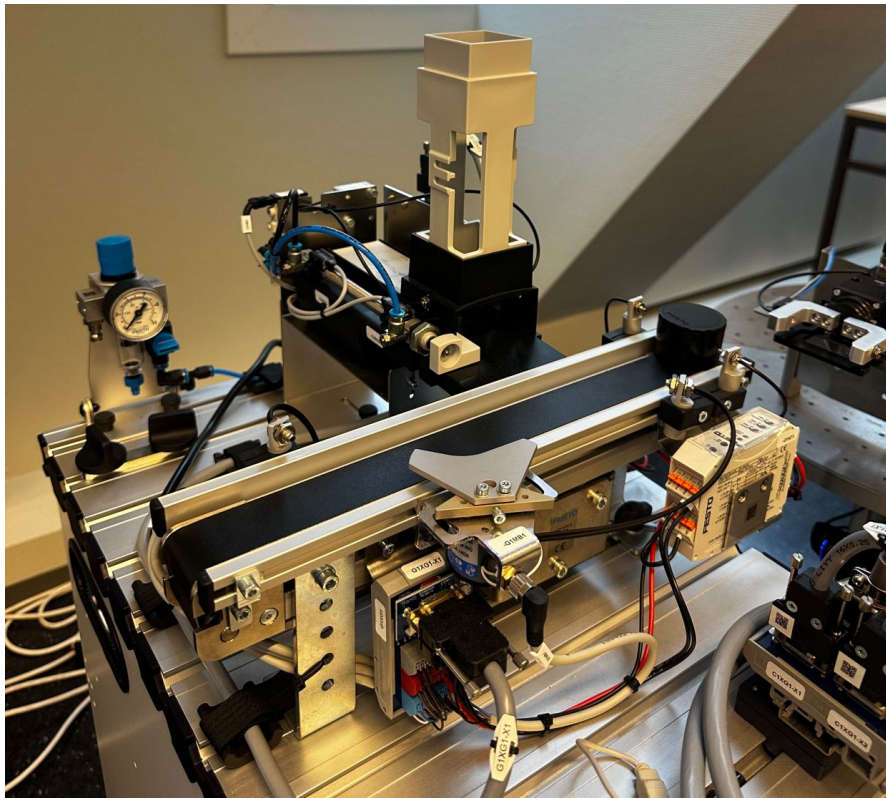


Figure 7.3: Festo Distribution/Belt Station

Bibliography

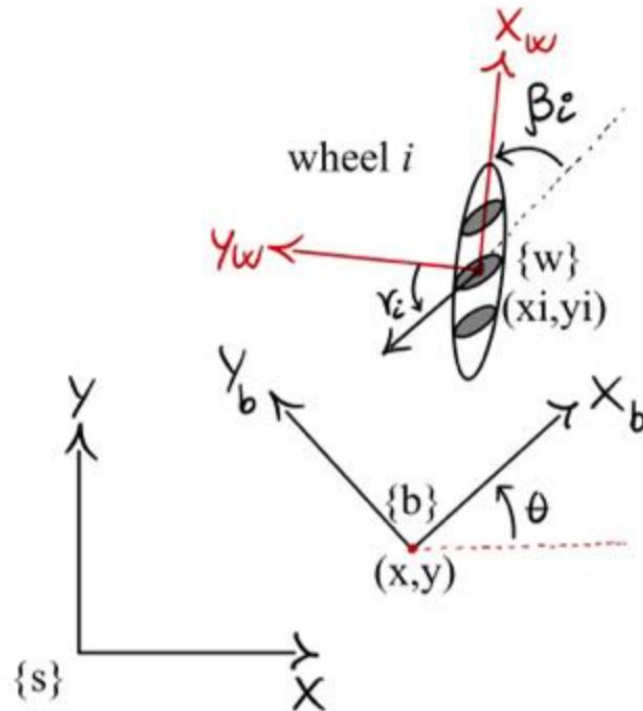
- [1] Corke, P. *Robotics, Vision and Control: Fundamental Algorithms in Python* (3rd ed.). Springer, 2023
- [2] Festo MyLearning. (n.d.). Autonomous Mobile Robotics with Robotino 4. [Online course, restricted access]. Retrieved from [Festo MyLearning] [Accessed: 08.01.2024].
- [3] TeamGantt. (n.d.). TeamGantt: Project Management Software & Gantt Chart Tool. [Internet]. URL: <https://www.teamgantt.com> [Accessed: 15.01.2024]
- [4] Wikipedia. (2023). Fagskolen i Hordaland. [Internet]. URL: https://no.wikipedia.org/wiki/Fagskolen_i_Hordaland [Accessed: 26.01.2024]
- [5] Wikipedia. (2023). Bergen tekniske fagskole. [Internet]. URL: https://no.wikipedia.org/wiki/Bergen_tekniske_fagskole [Accessed: 26.01.2024]
- [6] Fagskolen. (n.d.). Homepage. [Internet]. URL: <https://www.fagskolen.no> [Accessed: 26.01.2024]
- [7] Overleaf. (n.d.). Learn LaTeX in 30 minutes. [Internet]. URL: https://www.overleaf.com/learn/latex/Learn_LaTeX_in_30_minutes [Accessed: 02.02.2024]
- [8] Solberg, T. S., & Djønnne, E. (2023). BO19E-51 FESTO LABORATORIEOPPGAVE MED PLS (Bacheloroppgave, Høgskulen Vestland). URL: https://hvlopen.brage.unit.no/hvlopen-xmlui/bitstream/handle/11250/2602086/Solberg_Djoenne.pdf?sequence=1&isAllowed=y. [Accessed: 28.01.2024]
- [9] Hájková, L. (2023). Task with Robotino 4.0 robot for demonstration purposes (Bachelor of Engineering, Mechanical Engineering, LAB University of Applied Sciences). URL: https://www.theseus.fi/bitstream/handle/10024/793547/Hajkova_Lenka.pdf?sequence=2&isAllowed=y. [Accessed: 28.01.2024]
- [10] Rasel, M. R. I. (2017). Obstacle Detection for Indoor Navigation of Mobile Robots (Master Thesis, Dept. of Computer Science, Chair of Computer Engineering). URL: <https://core.ac.uk/download/153230184.pdf>. [Accessed: 28.01.2024]
- [11] Python Software Foundation. (n.d.). Python Software Foundation. [Internet]. URL: <https://www.python.org/doc/essays/blurb/> [Accessed: 07.02.2024]
- [12] ROS.org. (n.d.). ROS.org. [Internet]. URL: <https://www.ros.org/> [Accessed: 07.02.2024]
- [13] MathWorks. (n.d.). Simulink. [Internet]. URL: <https://www.mathworks.com/products/simulink.html> [Accessed: 07.02.2024]

- [14] Festo Didactic. (n.d). Robotino 3 Overview. [Internet]. URL: <https://ip.festo-didactic.com/InfoPortal/Robotino3/Overview/EN/index.html> [Accessed: 07.02.2024]
- [15] Festo Didactic. (n.d). Robotino 3 Hardware. [Internet]. URL: <https://ip.festo-didactic.com/InfoPortal/Robotino3/Hardware/EN/index.html> [Accessed: 07.02.2024]
- [16] Festo Didactic. (n.d). Robotino 3 Software. [Internet]. URL: <https://ip.festo-didactic.com/InfoPortal/Robotino3/Software/EN/index.html> [Accessed: 07.02.2024]
- [17] IndiaMART. (n.d.). Festo Mobile Robot. [Internet]. URL: <https://m.indiamart.com/proddetail/festo-mobile-robot-15504327830.html> [Accessed: 07.02.2024]
- [18] Engineering Educator Academy (2021). Kinematics of Mobile Robots with Omni Directional Wheels [Video]. YouTube. URL: <https://www.youtube.com/watch?v=-wz18XJopgg> [Accessed: 07.02.2024]
- [19] PAL Robotics. (n.d). Omnidirectional drive robots vs Differential drive robots. [Internet]. URL: <https://blog.pal-robotics.com/omnidirectional-vs-differential-drive-robots/> [Accessed: 10.02.2024]
- [20] NovoG93. (n.d). robotino. [Internet]. GitHub repository. Available: <https://github.com/NovoG93/robotino> [Accessed: 10.02.2024]
- [21] Flyability. (n.d). Simultaneous Localization and Mapping. [Internet]. URL: <https://www.flyability.com/simultaneous-localization-and-mapping> [Accessed: 17.02.2024]
- [22] Synopsys. (n.d). What is LiDAR? [Internet]. URL: <https://www.synopsys.com/glossary/what-is-lidar.html> [Accessed: 18.02.2024]
- [23] Open Robotino Forum. (n.d). The Robotino Gripper Motor Does Not Work. [Internet]. URL: <https://forum.openrobotino.org/forum/main-forum/hardware/robotino3-aa/13102-the-robotino-gripper-motor-does-not-work> [Accessed: 18.02.2024]
- [24] 3D Printing. (n.d). What is 3D Printing? [Internet]. URL: <https://3dprinting.com/what-is-3d-printing/> [Accessed: 18.03.2024]
- [25] Investopedia. (29.11.2023). 3D Printing: What It Is, How It Works, Examples. [Internet]. URL: <https://www.investopedia.com/terms/1/3d-printing.asp> [Accessed: 18.03.2024]
- [26] Makerbot. (n.d). MakerBot Replicator Z18. [Internet]. URL: <https://store.makerbot.com/replicator-z18> [Accessed: 18.03.2024]
- [27] Hokuyo. (n.d). Hokuyo URG Series. [Internet]. URL: <https://www.hokuyo-aut.jp/search/single.php?serial=166> [Accessed: 11.04.2024]

- [28] Wikipedia. (n.d.). Robotino with Hokuyo URG-04LX-UG01. [Internet]. URL: https://en.m.wikipedia.org/wiki/File:Robotino_with_Hokuyo_URG-04LX-UG01.jpg [Accessed: 11.04.2024]
- [29] Servo Magazine. (n.d.). A Look at Holonomic Locomotion. [Internet]. URL: <https://www.servomagazine.com/magazine/article/a-look-at-holonomic-locomotion> [Accessed: 11.04.2024]
- [30] TechTarget. (n.d.). TCP/IP (Transmission Control Protocol/Internet Protocol). [Internet]. URL: <https://www.techtarget.com/searchnetworking/definition/TCP-IP> [Accessed: 11.04.2024]
- [31] AVG. (n.d.). What is TCP/IP? [Internet]. Available: <https://www.avg.com/en/signal/what-is-tcp-ip> [Accessed: 11.04.2024]
- [32] Kyle, L. (n.d.). Networking Theory: Understanding TCP/IP, the Backbone of the Internet. [Internet]. Available: <https://medium.com/@kylelzk/networking-theory-understanding-tcp-ip-the-backbone-of-the-internet-c435f50d7a9a> [Accessed: 11.04.2024]
- [33] TechTarget. (n.d.). Closed-loop control system. [Internet]. Available: <https://www.techtarget.com/whatis/definition/closed-loop-control-system> [Accessed: 16.04.2024]
- [34] Omega Engineering. (n.d.). What is a PID Controllers. [Internet]. Available: <https://www.omega.com/en-us/resources/pid-controllers> [Accessed: 16.04.2024]

Appendix A

Kinematics



$\{s\}$ = world frame

$\{b\}$ = robot body frame

$\{w\}$ = robot wheel frame

(x, y) = robot body center coordinate in relation to the world frame

(x_i, y_i) = wheel center coordinate in relation to the robot body frame

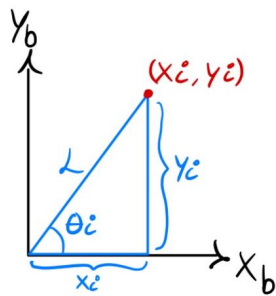
θ = angle of robot body frame in relation to the world frame

γ = angle of free rolling wheel in relation to the wheel's y component

β_i = angle between the wheel's x component (x_w , rolling direction) and the robot body x component (x_b)

i = wheel number 1,2,3,...

Figure A.1: Illustrating the Coordinate Frames and Angular Relationships

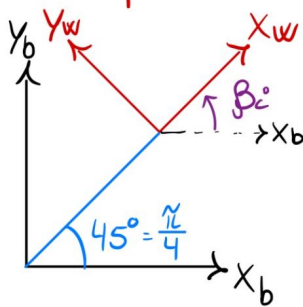


L = distance from robot center to wheel i

$$x_i = L \cos \theta_i$$

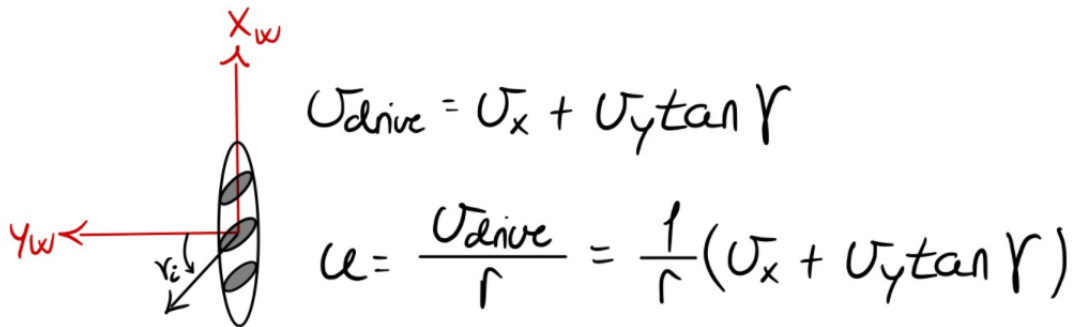
$$y_i = L \sin \theta_i$$

Example

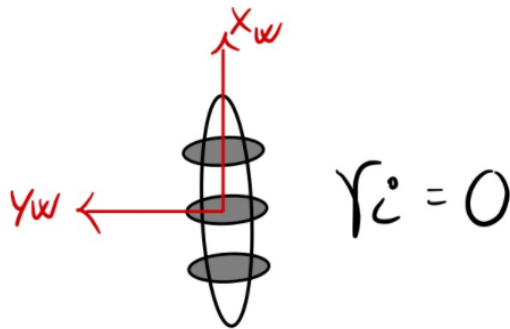


$$\beta_i = \frac{\pi}{4}$$

Figure A.2: Representation of Wheel Position and Orientation in the Robot Body Frame



Mecanum wheel for reference to show that you need to consider the gamma angle in some cases



We use omni-wheels where the free rolling wheel are directly on y_w

Figure A.3: Kinematics of a Robot with Mecanum Compared to Omni-Wheels

$$u_i = h_i(\dot{\theta})\dot{q} = \underbrace{\begin{bmatrix} \frac{1}{r_i} & \frac{\tan \gamma_i}{r_i} \end{bmatrix}}_{\text{D}} \underbrace{\begin{bmatrix} \cos \beta_i & \sin \beta_i \\ -\sin \beta_i & \cos \beta_i \end{bmatrix}}_{\text{C}} \underbrace{\begin{bmatrix} -y_i & 1 & 0 \\ x_i & 0 & 1 \end{bmatrix}}_{\text{B}} \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & \sin \theta \\ 0 & -\sin \theta & \cos \theta \end{bmatrix}}_{\text{A}} \begin{bmatrix} \dot{\theta} \\ \dot{x} \\ \dot{y} \end{bmatrix}$$

$$h_i(\theta) = \frac{1}{r_i \cos(\gamma_i)} \begin{bmatrix} x_i \sin(\beta_i + \gamma_i) - y_i \cos(\beta_i + \gamma_i) \\ \cos(\beta_i + \gamma_i + \theta) \\ \sin(\beta_i + \gamma_i + \theta) \end{bmatrix}^T$$

- A:** Rotation about the z-axis, going from frame {s} to frame {b}
- B:** Translation from frame {b} to frame {w}
- C:** Rotation about the z-axis, going from frame {b} to frame {w}
- D:** Converting to angular velocity with respect to the wheel radius r_i and changing the velocity direction with respect to gamma

$$\theta = 0 \quad \text{and} \quad \gamma = 0$$

$$\mathbf{h}_i(\theta) \Big|_{\theta=0} = \frac{1}{r} \begin{bmatrix} x_i \sin(\beta_i) - y_i \cos(\beta_i) \\ \cos(\beta_i) \\ \sin(\beta_i) \end{bmatrix}^T$$

Setting $\theta = 0$ for the robot's initial state and to make the system independent from the {s} frame, essentially skipping the first rotation matrix A in the $h_i(\theta)$ equation.

This results in a constant matrix without a variable θ

Figure A.4: Transformation Steps and Equations for Robot Wheel Kinematics

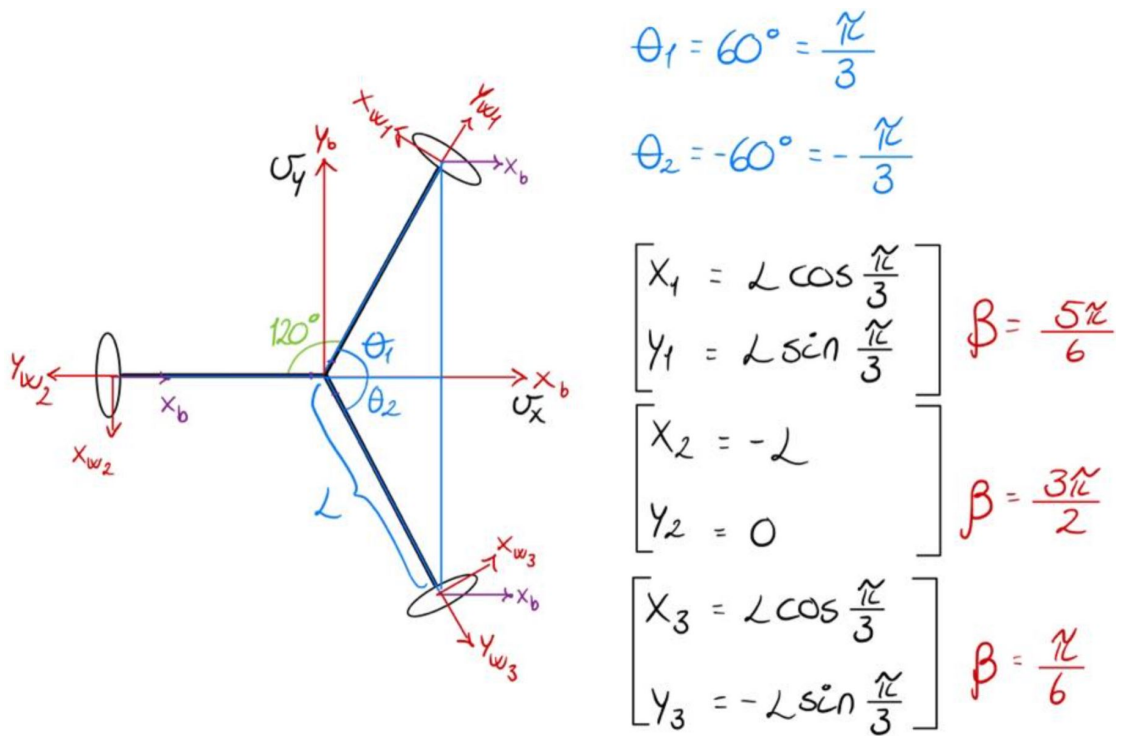


Figure A.5: Kinematics with Angle Calculations and Wheel Coordinates

$$h_1(0) = \frac{1}{r} \begin{bmatrix} L \left(\cos \frac{\pi}{3} \sin \left(\frac{5\pi}{6} \right) - \sin \frac{\pi}{3} \cdot \cos \left(\frac{5\pi}{6} \right) \right) \\ \cos \left(\frac{5\pi}{6} \right) \\ \sin \left(\frac{5\pi}{6} \right) \end{bmatrix}^T$$

$$L \left(\frac{1}{2} \cdot \frac{1}{2} - \frac{\sqrt{3}}{2} \cdot -\frac{\sqrt{3}}{2} \right) = L \left(\frac{1}{4} + \frac{3}{4} \right) = 1$$

$$\cos \left(\frac{5\pi}{6} \right) = -\frac{\sqrt{3}}{2}$$

$$\sin \left(\frac{5\pi}{6} \right) = \frac{1}{2}$$

$$h_1(0) = \left[L, -\frac{\sqrt{3}}{2}, \frac{1}{2} \right]$$

Figure A.6: Calculation of Kinematics for Wheel 1

$$h_2(0) = \frac{1}{r} \begin{bmatrix} -L \cdot \sin \frac{3\pi}{2} \\ \cos \left(\frac{3\pi}{2} \right) \\ \sin \left(\frac{3\pi}{2} \right) \end{bmatrix}^T$$

$$h_2(0) = \left[L, 0, -1 \right]$$

Figure A.7: Calculation of Kinematics for Wheel 2

$$h_3(0) = \frac{1}{r} \begin{bmatrix} L \left(\cos \frac{\pi}{3} \sin\left(\frac{\pi}{6}\right) + \sin \frac{\pi}{3} \cdot \cos\left(\frac{\pi}{6}\right) \right) \\ \cos\left(\frac{\pi}{6}\right) \\ \sin\left(\frac{\pi}{6}\right) \end{bmatrix}^T$$

$$L \left(\frac{1}{2} \cdot \frac{1}{2} + \frac{\sqrt{3}}{2} \cdot \frac{\sqrt{3}}{2} \right) = L$$

$$\cos\left(\frac{\pi}{6}\right) = \frac{\sqrt{3}}{2}$$

$$\sin\left(\frac{\pi}{6}\right) = \frac{1}{2}$$

$$h_3(0) = \left[L, \frac{\sqrt{3}}{2}, \frac{1}{2} \right]$$

Figure A.8: Calculation of Kinematics for Wheel 3

$$\mathbf{H}(0) = \begin{bmatrix} h_1(0) \\ h_2(0) \\ h_3(0) \end{bmatrix} \Rightarrow \mathbf{H}(0) = \frac{1}{r} \begin{bmatrix} L & -\frac{\sqrt{3}}{2} & \frac{1}{2} \\ L & 0 & -1 \\ L & \frac{\sqrt{3}}{2} & \frac{1}{2} \end{bmatrix}$$

$$\mathbf{V}_b = \begin{bmatrix} \omega_{bz} \\ v_{bx} \\ v_{by} \end{bmatrix} \quad \mathbf{U} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}$$

$$\mathbf{U} = \mathbf{H}(0) \cdot \mathbf{V}_b$$

$$\mathbf{U} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \frac{1}{r} \begin{bmatrix} L & -\frac{\sqrt{3}}{2} & \frac{1}{2} \\ L & 0 & -1 \\ L & \frac{\sqrt{3}}{2} & \frac{1}{2} \end{bmatrix} \cdot \begin{bmatrix} \omega_{bz} \\ v_{bx} \\ v_{by} \end{bmatrix}$$

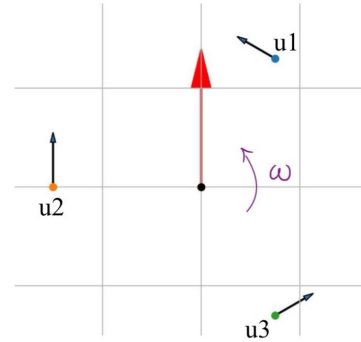


Figure A.9: Matrix Representation and Vector Transformation for Wheel Velocities

Appendix B



Lab Exercise 1

Introduction to Robotino 3 and Basic Motion

Topic

- Introduce Students to Robotino 3
- Observe Robotino's Movements
- Create Abort Function With Bumper

Equipment

- Robotino 3
- Robotino View
- Robotino SIM



Preparation

Install Robotino View and Robotino SIM.

Objective of the Exercise

This laboratory exercise aims to install the necessary software, familiarize oneself with the hardware and software components of the Robotino, and practice basic motion control of the robot. First, you will make the robot to move forward for 6 seconds. Then, have it pause for 2 seconds before moving backward for 6 seconds. Next, you'll make the robot to move linearly in all directions using omnidrive. In the end, create abort function with bumper in case of collision.

Groups

2-3 persons per group

Report

You can access the report template here: [appendix F](#)

Solutions

In the figures below you can find function blocks used to complete these exercises. The actual Robotino View program can be found on GitHub, and the link for this is found in appendix I. You can find video demonstration of the exercise in appendix H.

Move Forward and Backward

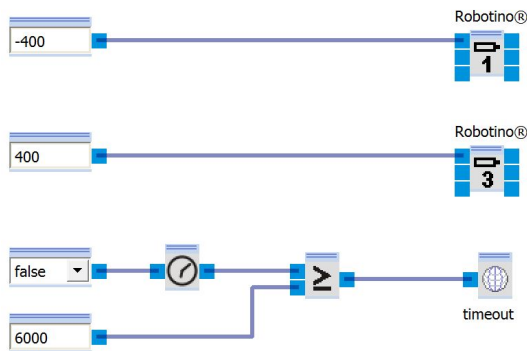


Figure B.1: Move forwards for 6 seconds

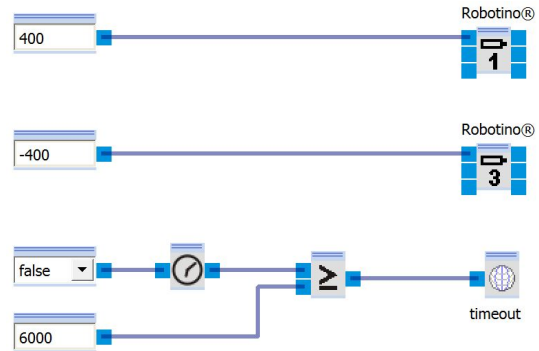


Figure B.2: Move backwards for 6 seconds

Figure B.1 and B.2 Shows the function blocks used to move the robot forwards for 6 seconds, then backwards for 6 seconds.

Move Linearly in All Directions

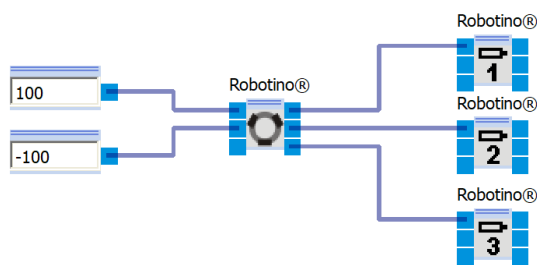


Figure B.3: Omnidrive Function Block

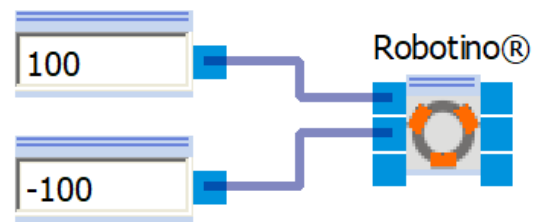


Figure B.4: Drive System Function Block

Figure B.3 and Figure B.4 shows two different ways to make the robot move in a straight line at a 45° angle. You can use the Drive System function block to replace the omnidrive block with connection to the motor blocks, which can make it easier.

Move with Control Panel

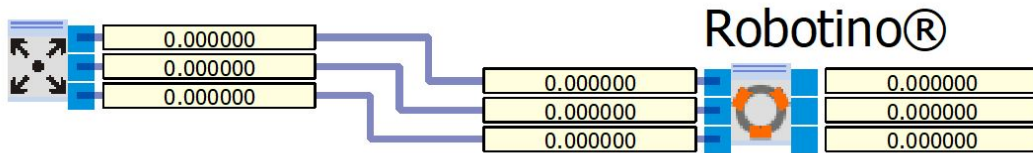


Figure B.5: Drive System with Control Panel

You can also attach a Control Panel to the Drive System, allowing you to manually control the robot in every direction.

Bumper



Figure B.6: Bumper Function Block

Choose the Variable function from the Function block library, then right-click and select "Add" using the mouse. Describe the variable as "timeout". Then modify the primary program for the sequence controller to specify the abort conditions.

Appendix C



Lab Exercise 2

Distance Sensors and Odometry

Topic

- Familiarize with distance sensors and odometry
- Approaching a wall and maintain a defined distance.
- Follow wall and rotate 90°

Equipment

- Robotino 3
- Robotino View
- Two walls that make a 90° corner.



Preparation

Complete Lab Exercise 1

Objective of the Exercise

This lab exercise is designed to familiarize you with Robotino's distance sensors and their application in navigation tasks. You will program the robot to approach a wall and maintain a distance of 8 cm. Next, you will guide the robot to move alongside the wall while maintaining this distance. Finally, you will navigate the robot along a wall with a 90° corner, using odometry to make the turn.

Groups

2-3 persons per group

Report

You can access the report template here: [appendix F](#)

Solutions

In the figures below you can find function blocks used to complete these exercises. The actual Robotino View program can be found on GitHub, and the link for this is found in appendix I. You can also find video demonstration of the exercise in appendix H

In the first subprogram, we use distance sensor 1 with a set point of 1.49V, based on values from Table 4.1. This allows the robot to drive towards the wall and maintain a distance of 8 cm. After that, the main program transitions to the next subprogram, where we use a combination of distance sensor 1 and distance sensor 3 to maintain distance. In this subprogram, the robot moves sideways until it maintains an 8 cm distance from both sensors. When both sensors are within this range, the program advances to the final subprogram, where the robot rotates 90° using odometry and the position driver function block. The set point for this subprogram is 90, hence the 90° rotation.

Approach a Wall Subprogram

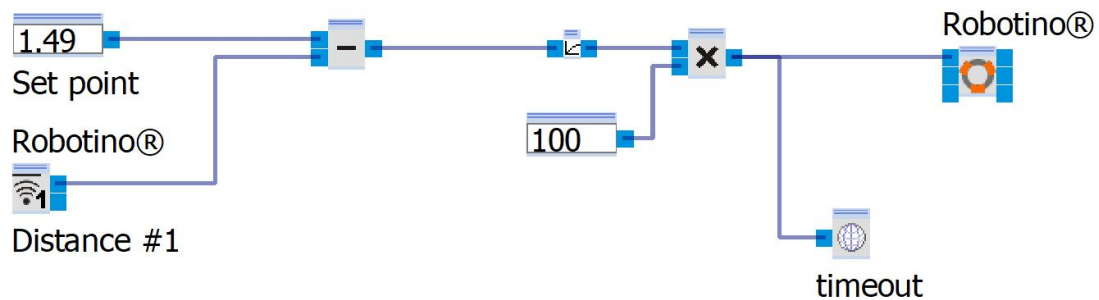


Figure C.1: Approach a Wall Subprogram

Move Along Wall Subprogram

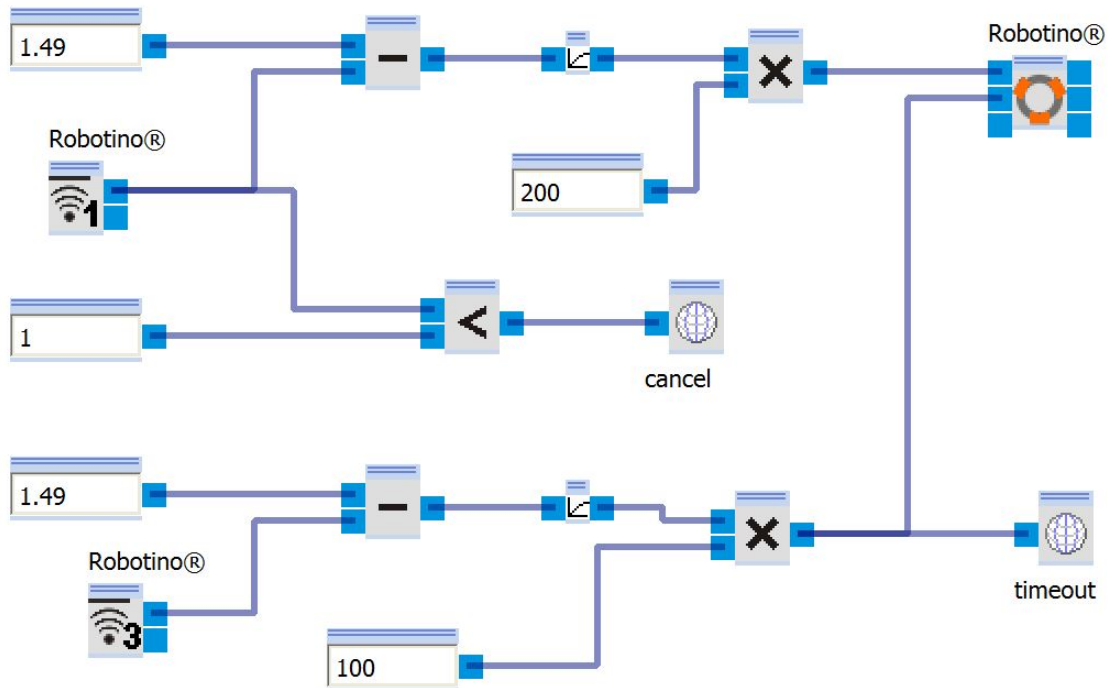


Figure C.2: Move Along Wall Subprogram

Move Around Corner Subprogram

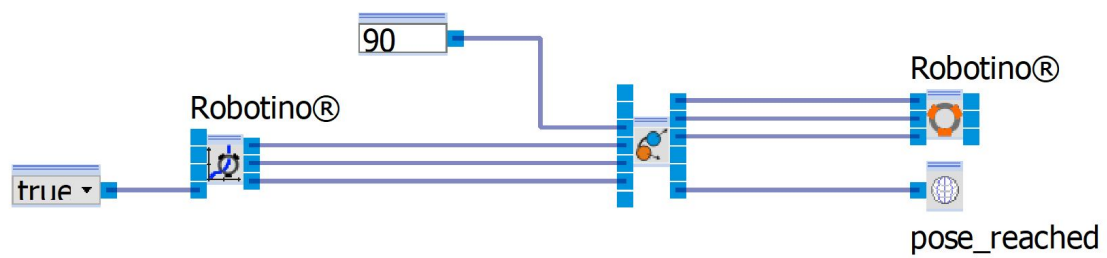


Figure C.3: Move Around Corner Subprogram

Appendix D



Lab Exercise 3

Image Processing and Object Detection

Topic

- Familiarize with Robotino's Camera
- Search for and approach a colored object
- Search for and approach AR-marker

Equipment

- Robotino 3
- Robotino View
- Robotino SIM



Preparation

Complete Lab Exercise 2

Objective of the Exercise

This lab exercise is designed to familiarize you with image processing and AR markers using the camera. You will develop a program that instructs the robot to locate a predetermined colored object, approach it, and maintain a distance of 8 cm from it. You will then repeat the process, this time using AR markers instead of colored objects.

Groups

2-3 persons per group

Report

You can access the report template here: [appendix F](#)

Solutions

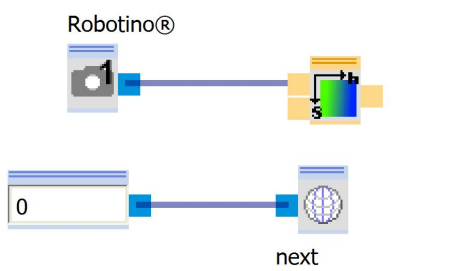
In the figures below you can find function blocks used to complete these exercises. The actual Robotino View program can be found on GitHub, and the link for this is found in appendix I. You can find video demonstration of the exercise in appendix H.

In the first subprogram, we use the color range finder function block to teach the desired colors. You select an area of the intended object and copy values directly from this function block. These values are then placed in the main program under a variable called "color." After this, the robot moves to the next subprogram, where it begins searching for the object. It will rotate on the spot until it sees the object. Note that the camera has the best accuracy within a range of no more than 140 cm. After the object has been detected, the main program moves on to the last subprogram. Here, the robot will move towards the object until it reaches a distance of 8 cm from it. At this point, the program is finished, and the robot will stop.

The concept for using marker detection is very similar. Here, we use the marker detection function block. You can choose your own marker from Robotino View using the marker selection function block. When you start the program, the robot will rotate on the spot until it sees the marker. After this, the main program will proceed to the next subprogram, where the robot will drive towards the marker. As before, it will drive until it reaches a distance of 8 cm before it stops.

Teach Color Subprogram

At the start the constant must be 0!



Set the constant = 1 if the colour is defined.

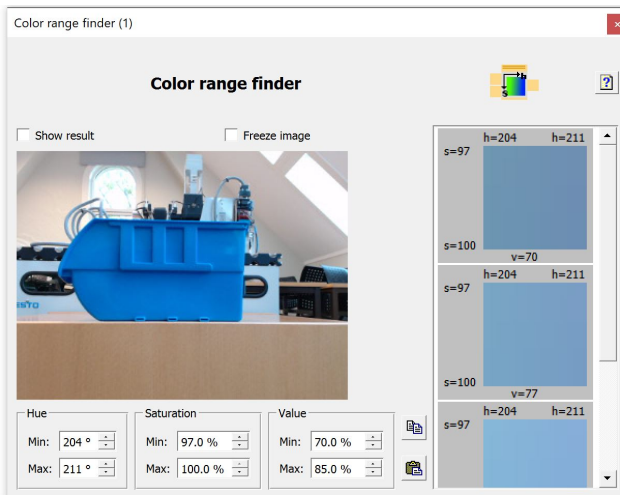


Figure D.1: Teach Color Subprogram

Search for the Object Subprogram

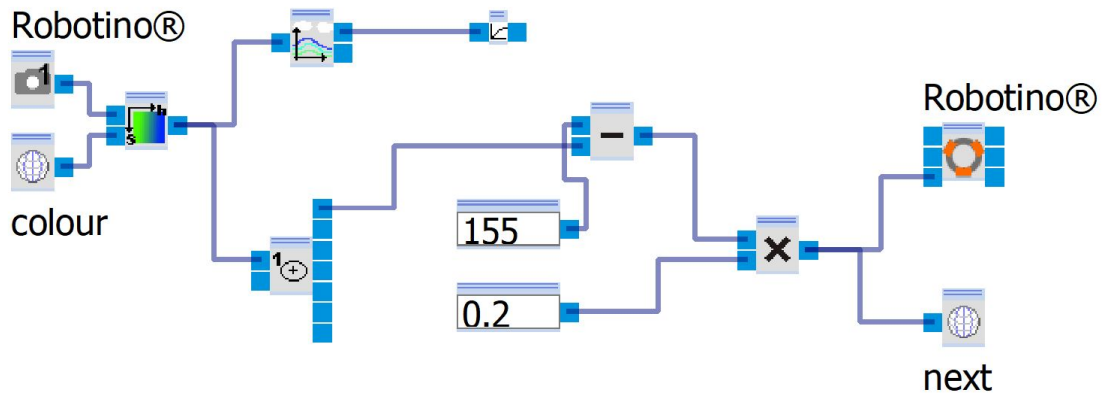


Figure D.2: Search for the Object Subprogram

Approach the Object Subprogram

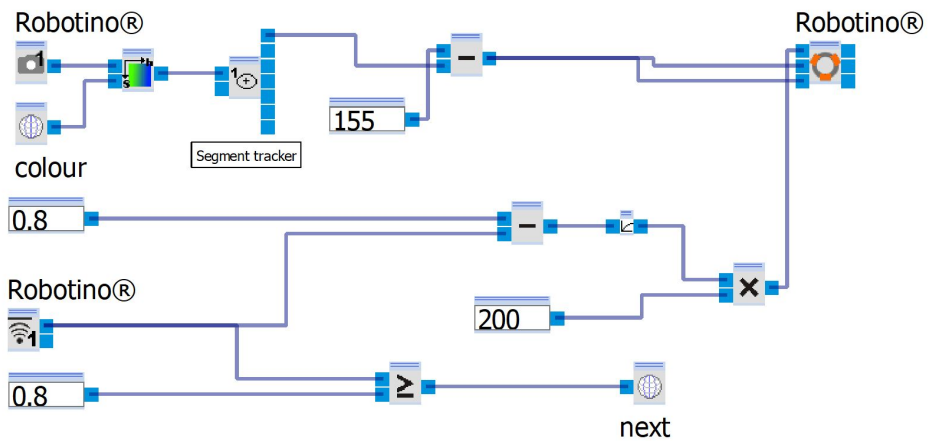


Figure D.3: Approach the Object Subprogram

Search for the Marker Subprogram

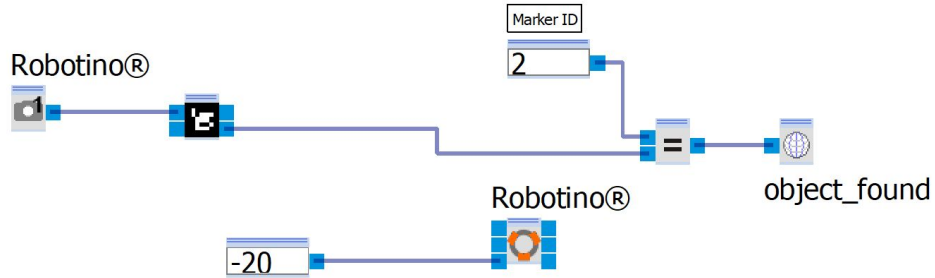


Figure D.4: Search for the Marker Subprogram

Approach the Marker Subprogram

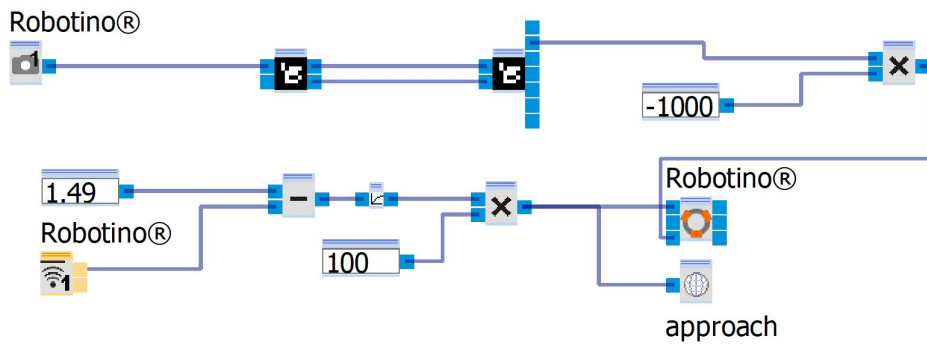


Figure D.5: Approach the Marker Subprogram

Appendix E



Lab Exercise 4

Multi-Sensor Navigation and Object Handling

Topic

- Execute a multi-sensor task
- Use gripper to pick up an object.

Equipment

- Robotino 3
- Robotino View
- Festo Belt/Distribution Module
- A piece of tape



Preparation

Complete Lab Exercise 3

Objective of the Exercise

This lab exercise involves guiding the robot towards a module using camera guidance, tracing a line with the diffuse sensors, maintaining proper distance and alignment with the module using distance sensors, and utilizing a gripper to lift an object from the Belt/Distribution Module. After that, the robot moves away from the module, uses odometry to reach a specific location, and drops off the object.

Groups

2-3 persons per group

Report

You can access the report template here: [appendix F](#)

Solutions

In the figures below you can find function blocks used to complete these exercises. The actual Robotino View program can be found on GitHub, and the link for this is found in appendix I. You can find video demonstration of the exercise in appendix H.

- Find Marker Subprogram: The robot spins around at a constant speed looking for the desired marker using it's imaging system.
- Mark Follow Subprogram: The robot moves forward towards the marker and centers itself in relation to the marker. It stops at a desired length from the wall using distance sensor 1.
- Center Subprogram: The robot centers itself in relation to a strip on the ground using the diffuse sensors. At the same time it adjust the angle so it faces directly towards the platform using distance sensors 2 and 9.
- Pickup Subprogram: The robot moves forward at a constant very low speed until the pressure plate on the gripper is activated. The gripper then closes, picking up the item.
- Drive Back Subprogram: The robot moves backwards at a constant very low speed.
- Move Odometry Subprogram: The odometry is reset and the robot moves holonomically towards a different location given the desired coordinates.
- Release Gripper Subprogram: The gripper is opened and the item is released at the desired location.
- Home Pose Subprogram: The robot moves back holonomically to the "Home Pose", which is the last place the odometry was reset.

Find Marker Subprogram

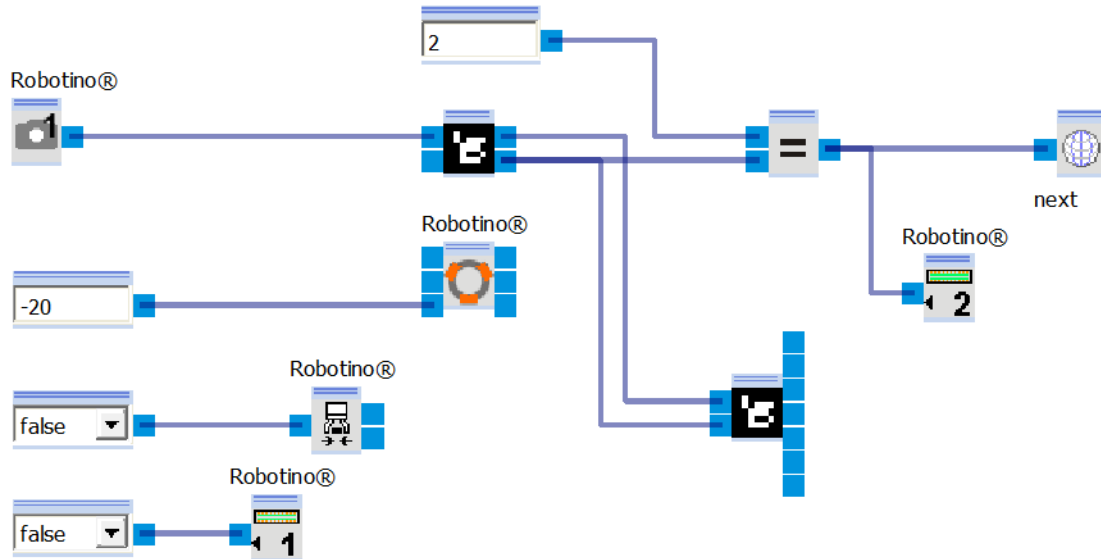


Figure E.1: Find Marker Subprogram

Mark Follow Subprogram

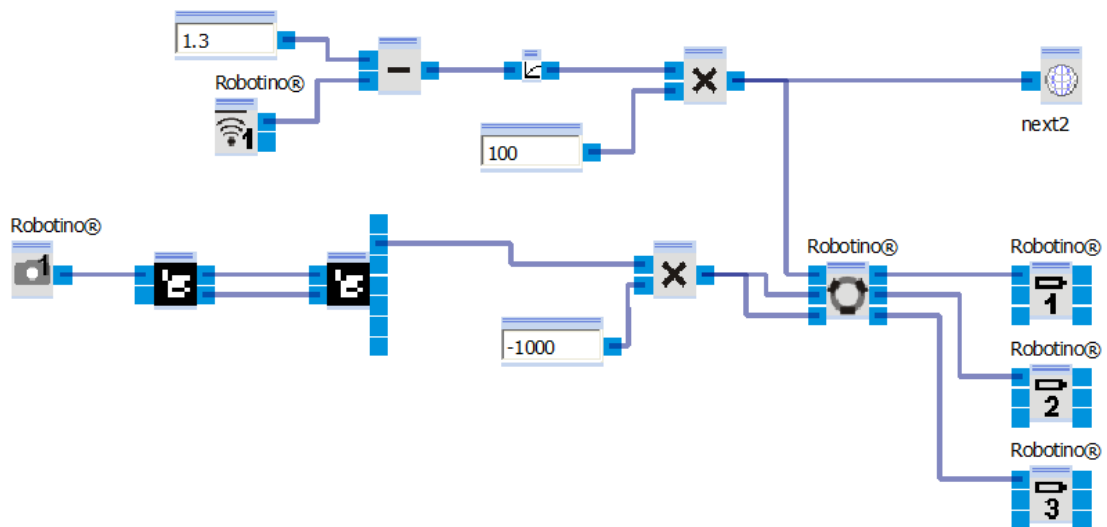


Figure E.2: Mark Follow Subprogram

Center Subprogram

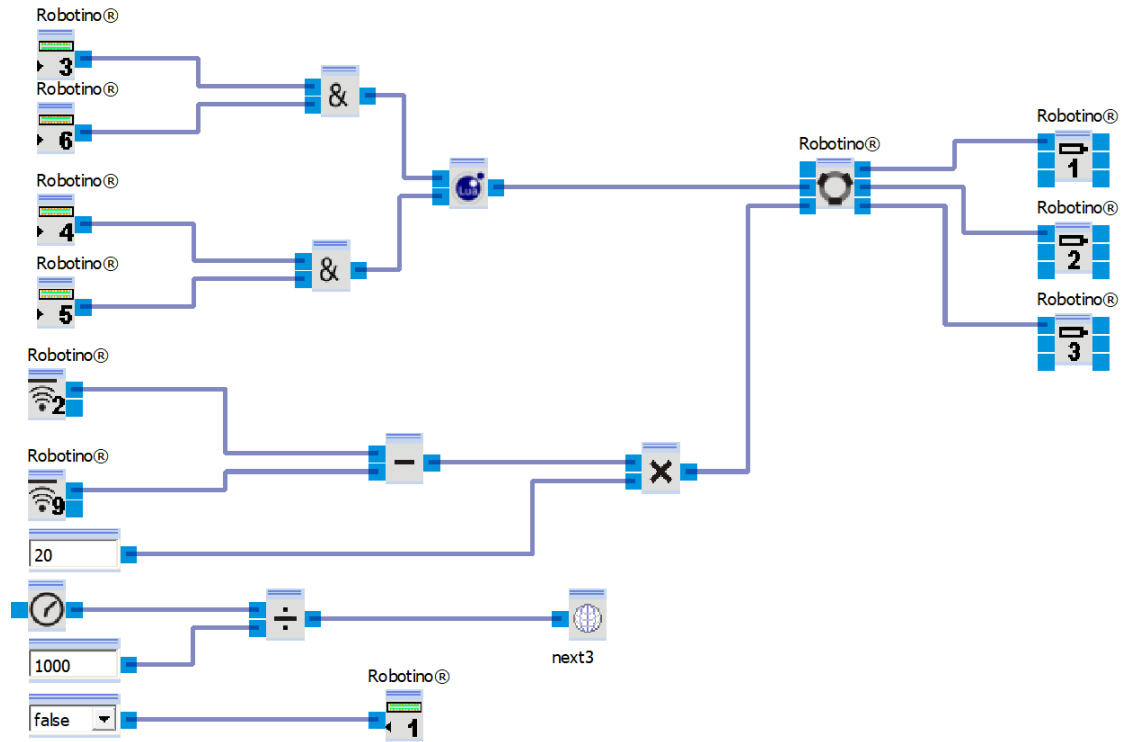


Figure E.3: Center Subprogram

Pickup Subprogram

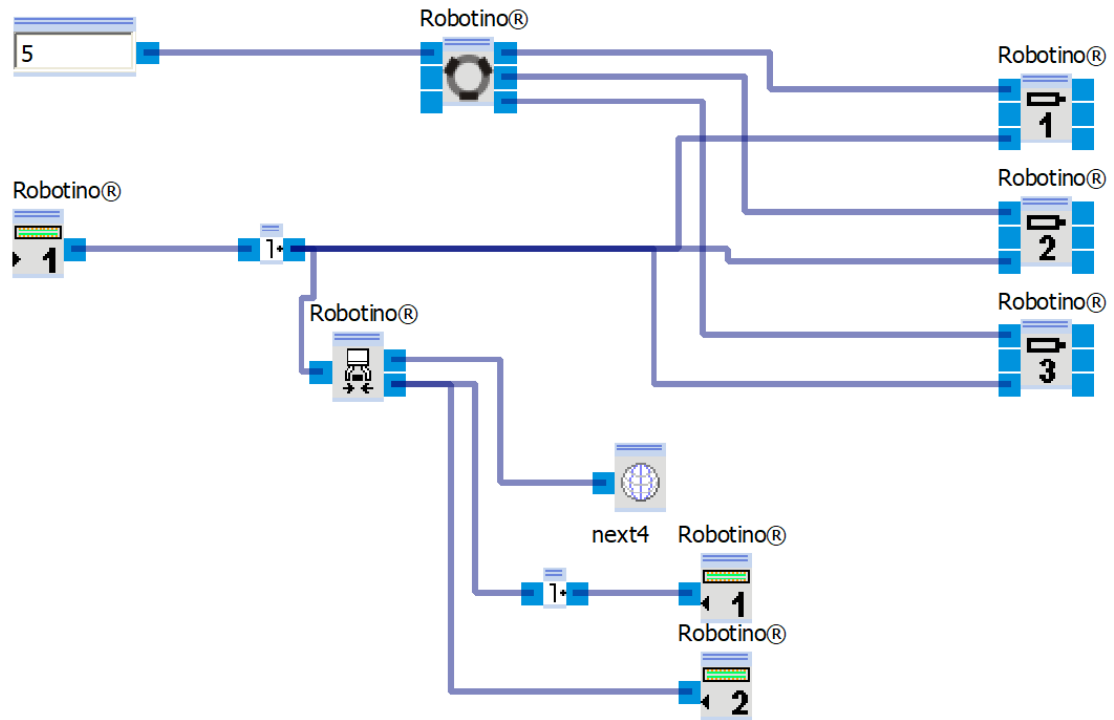


Figure E.4: Pickup Subprogram

Drive Back Subprogram

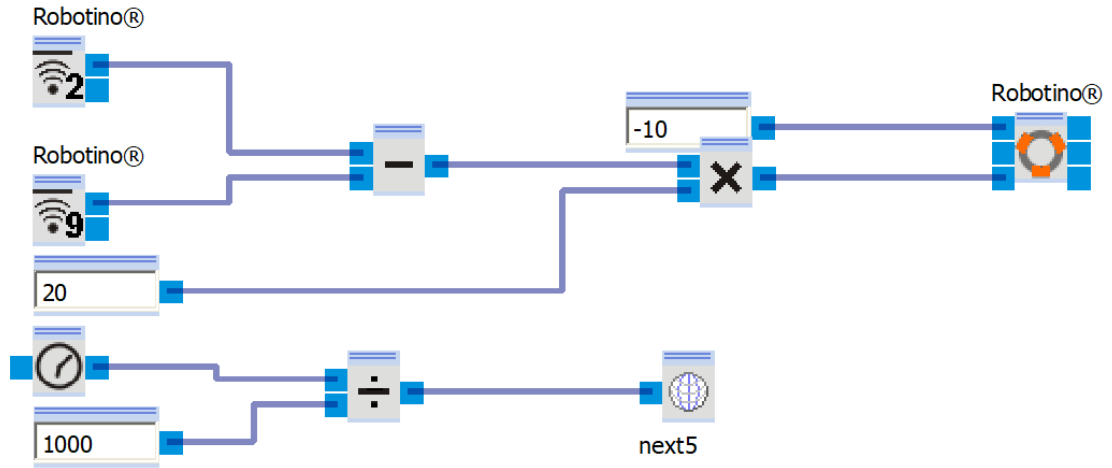


Figure E.5: Drive Back Subprogram

Move Odometry Subprogram

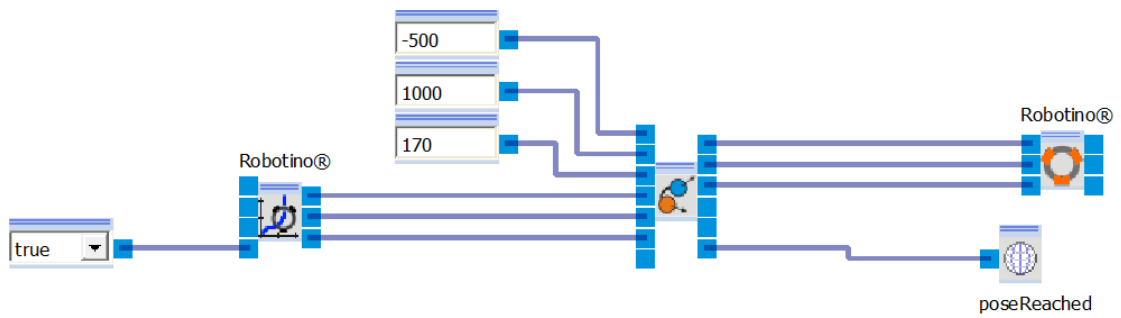


Figure E.6: Move Odometry Subprogram

Release Gripper Subprogram

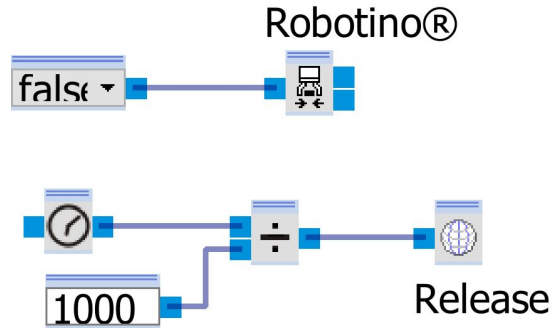


Figure E.7: Release Gripper Subprogram

Home Pose Subprogram

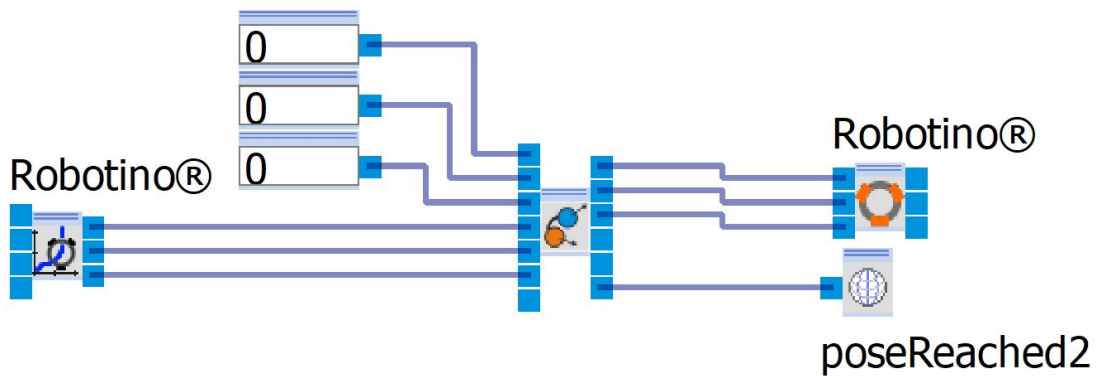


Figure E.8: Home Pose Subprogram

Video

You can find video demonstration of the exercise in appendix H

Appendix F

Template for Report

Student Information

Name: _____

Date: _____

Course: _____

Instructor: _____

Introduction

Provide a brief introduction to the lab exercise, including the objective and any relevant theory.

Equipment Used

List all the equipment and software used to perform the lab exercise.

Procedure

Outline the steps you followed to complete the lab exercise.

Results

Present the results of your lab exercise. Include any figures, tables, or graphs if necessary.

Conclusion

Summarize the main points of the lab, the findings, and their implications.

Appendix G

Guide to Working With Robotino 3

Introduction

Welcome to the Guide to Working with Robotino 3. This guide aims to provide you with a basic understanding of Robotino 3, along with its accompanying software tools, Robotino View, and Robotino SIM. The purpose of this guide is to provide instructions for beginners to effectively utilize Robotino 3 and its associated software.



Figure G.1: Robotino shown with Tower, Segment Base, Gripper, and Signaling Lights

Software Tools

Robotino View: This software provides a user-friendly interface for programming and controlling Robotino 3. With its intuitive design, Robotino View allows users to interact with Robotino 3 in real-time, making it ideal for rapid prototyping and experimentation.

Robotino SIM: Robotino SIM is a simulation environment that enables users to virtually simulate Robotino 3 in various scenarios. By replicating real-world conditions, Robotino SIM allows users to test and validate their algorithms without the need for physical hardware, making it a valuable tool for both learning and development.

Understanding Robotino 3

The Robotino 3

The Robotino 3 is an advanced mobile robotics platform mainly used for educational and research applications. Manufactured by Festo Didactic, it is recognized for its omnidirectional drive system which allows for precise movements in all directions. Equipped with a range of sensors, the Robotino 3 provides a comprehensive set of tools for students and researchers to develop skills in areas such as navigation, sensor integration, and robot control.

Setting up the Robotino 3

The robot has been unboxed and is ready for boot-up. Activate the robot by pressing the button located on top of the Control Unit. If the robot fails to start up, it is likely due to depleted batteries. Connect the power cable and attempt to start it again. To power off the robot, press and hold the power button for approximately 3 seconds.

Getting started with Robotino View

Installing Robotino View

You can find link to install Robotino View in Appendix I, under Software Specifications. Robotino View was primarily developed for Windows operating systems, therefore it is recommended to use it on this platform.

Connecting to Network

To establish communication with the Robotino, the operating computer must be connected to Robotino's Wi-Fi network. Each Robotino comes with its dedicated Wi-Fi router. For this specific Robotino, you'll need to connect to the "robotino" Wi-Fi network. The password for this network is also "robotino".

Connecting to Robotino

By default, the program is connected to the local IP address. To control the robot using Robotino View, you need to connect to the IP address 172.26.1.1. If you wish to control the robot in Robotino SIM, you must use the IP address 127.0.0.1.

Your First Project with Robotino View

Upon launching Robotino View, you will see the user interface, as shown in figure G.2. This interface typically features a main program section and step section, providing a structured layout for initiating program development or execution. The main program consists of sequential steps that follow a set order. Each step defines specific functions in the program, which are activated when certain conditions are fulfilled.

Programming in Robotino View involves using function blocks (as shown in figure G.3), Lua scripts, or Python scripts. While simple programs can be effectively implemented using function blocks, using scripts like Lua or Python can be recommended for improved manageability and flexibility.

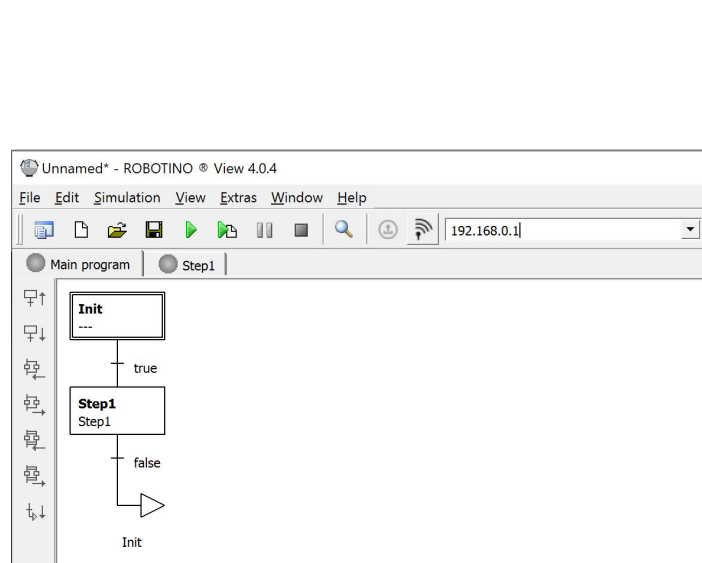


Figure G.2: User interface of Robotino View, Highlighting the Main Program Structure

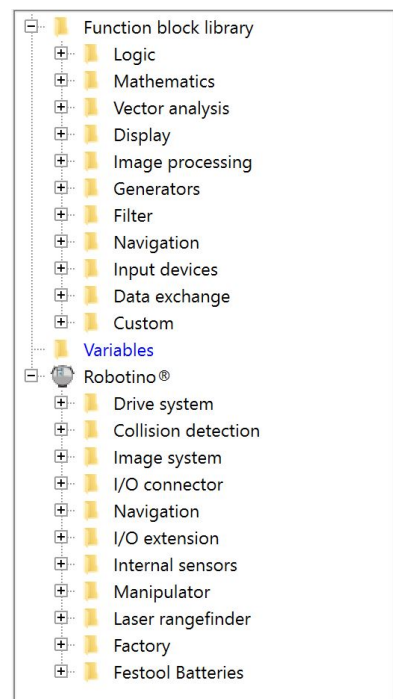


Figure G.3: The programming Blocks Within Robotino View

Control the Program

To initiate the program, click the green play button located at the top next to the save button, as illustrated in Figure G.2. To execute the current subprogram only, use the second play button adjacent to the pause button. This feature allows for testing specific parts of the program without executing the entire cycle. To stop the external control program, press the stop button located on the same row as the play buttons shown. Note that the stop button will appear inactive when not in use.

Getting started with Robotino SIM

Installing Robotino SIM Demo

You can find link to install Robotino SIM in Appendix I, under Software Specifications.

Simulating Your First Task

Upon launching Robotino SIM, you will see the user interface as shown in figure 3.12. You will utilize the same program in Robotino View that you use for simulating the robot in real life. The only difference is that you'll simulate it virtually within a program. It's important to ensure that you change the IP address to 127.0.0.1 so that the Robotino View programs recognize that they should communicate with Robotino SIM.

Robotino SIM includes example simulations such as Color driver, Follow Line (camera), and Follow line (inductive sensor). These examples demonstrate how the robot moves within the simulation environment. However, you have the option to use your own program in Robotino View to simulate the robot in Robotino SIM.

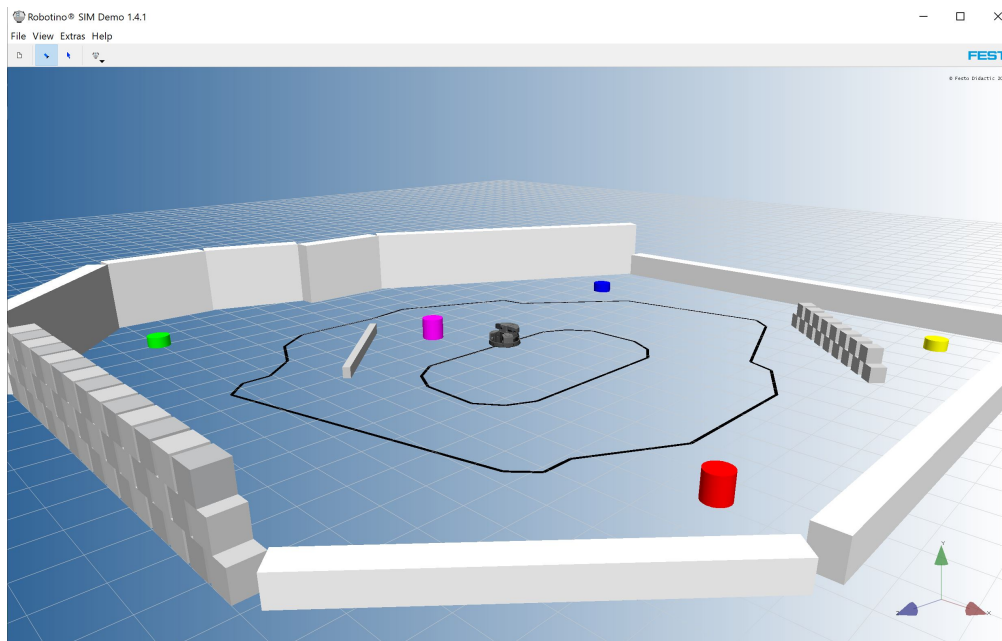


Figure G.4: Snapshot of the Robotino SIM Simulation Environment

Exploring Robotino Web Interface

The web interface of the Robotino makes functions available for control, configuration and maintenance of the robot system. Through web interface we can control the robot with our computer and phone, check battery status, change network settings, and change various parameters for controlling. To access the Robotino web interface, make sure you are connected to the robots Wi-Fi and simply open a web browser and enter the IP address: 172.26.1.1.

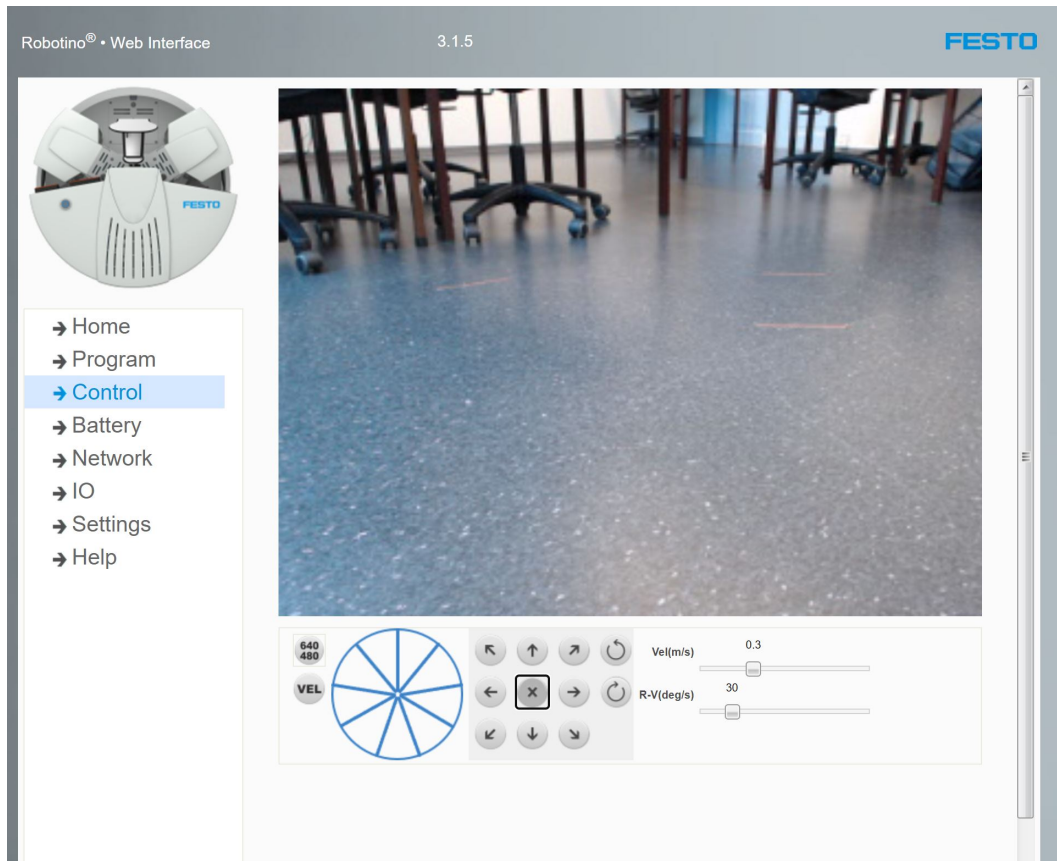


Figure G.5: Displaying the Control Function on Robotino Web Interface

Troubleshooting

Common Issues

- **Connection Loss:** The primary issue we encounter with the Robotino is occasional loss of connection. We're uncertain about the cause of this problem.
- **Battery Capacity:** We also have issues with battery capacity. Although we replaced the batteries once, their capacity decreased again after a few months. We cannot use the robot for long periods without needing a power cable.

Solutions

- **Connection Error:**
 - Reset the Wi-Fi on your computer.
 - If that doesn't work, try restarting both the Robotino and your computer.
- **Battery Capacity:**
 - Replace the batteries with new ones.
 - Keep the robot connected to a power cable.

Further Resources

For more comprehensive details regarding the Robotino 3, Robotino View, Robotino SIM, a step-by-step programming guide, as well as manuals and datasheets, please refer to appendix I

Appendix H

Videos Demonstrating Robotino Movement

Showcasing Manual Control of Robot Movement in ROS

<https://youtu.be/PemLGTzzwVo>

Lab Exercise 1_01

<https://youtube.com/shorts/2z1TwCRW7J0>

Lab Exercise 1_02

<https://youtube.com/shorts/bHUIPFU7Pz4>

Lab Exercise 2_01

<https://youtube.com/shorts/HshCPW52b9U>

Lab Exercise 3_01

<https://youtube.com/shorts/X4NDdDr343I>

Lab Exercise 3_02

<https://youtube.com/shorts/P0PekCZvGq0>

Lab Exercise 4_01

<https://youtu.be/FbCHKwUIH5E>

Lab Exercise 4_02

<https://youtu.be/m1smGWOn7zw>

Appendix I

External Resources and Documentation

Link to Github

<https://github.com/danielsortland/BO24EB-09>

Robotino Manual

https://www.festo.com/net/en-ir_ir/SupportPortal/Files/767492/8029256_8029346_deen_v10_LP8029476_Robotino_Manual.pdf

RobotinoWiki

https://wiki.openrobotino.org/index.php?title=Main_Page

Robotino View Manual

https://doc.openrobotino.org/download/RobotinoView/RobotinoView2_EN.pdf

Electric Gripper

https://www.festo.com/net/en-ir_ir/SupportPortal/Files/767493/8029451_deen_v2.0_LP8030735_Robotino_Electric_gripper_Brief_description.pdf

Robotino Software Specifications

Robotino View

<https://ip.festo-didactic.com/Infoportal/Robotino3/Software/Programming/EN/RobotinoView.html>

Robotino SIM

<https://ip.festo-didactic.com/Infoportal/Robotino3/Software/Simulation/EN/index.html>

Web Interface

<https://ip.festo-didactic.com/Infoportal/Robotino3/Software/Webinterface/EN/index.html>

Robotino Hardware Specifications

Control

<https://ip.festo-didactic.com/Infoportal/Robotino3/Hardware/Controller/EN/index.html>

- Embedded PC Datasheet
- Microcontroller Specifications

Drive Systems

<https://ip.festo-didactic.com/Infoportal/Robotino3/Hardware/DriveSystem/EN/index.html>

- Motor Technical Description
- Incremental Encoder Technical Specs
- Planetary Gearbox Technical Description
- Wheels Product Specification

Sensors

<https://ip.festo-didactic.com/Infoportal/Robotino3/Hardware/Sensors/EN/index.html>

- Incremental Encoder Technical Specs
- Distance Sensors Specifications
- Gyroscope Product Specs
- Camera Product Information
- Opto-electronic Sensors Data Sheet
- Inductive Sensors Data Sheet

Interfaces

<https://ip.festo-didactic.com/Infoportal/Robotino3/Hardware/Interfaces/EN/index.html>

- WLAN Product Datasheet

Supply

<https://ip.festo-didactic.com/Infoportal/Robotino3/Hardware/Supply/EN/index.html>

- Battery Technical Description
- Power Supply Unit Specs
- Charger Specifications