

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Development of a Half-Size Robot for the Robot at Factory Lite Competition

Joana Daniela Pedrosa Moura



Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Prof. Paulo Gomes da Costa

Second Supervisor: Prof. José Magalhães Lima

July 29, 2026

Resumo

Na era da Indústria 4.0, os Robôs Móveis Autônomos (AMRs) desempenham um papel vital na automação e otimização da logística interna. A competição "Robot at Factory Lite" fornece uma versão acadêmica estruturada destes sistemas, simulando um ambiente de cadeia de abastecimento, onde os robôs móveis são responsáveis pela gestão autônoma do transporte de materiais. Embora já exista uma solução que cumpre os requisitos da competição, as suas dimensões limitam significativamente a agilidade, a otimização do espaço e a escalabilidade multiagente. Para colmatar estas limitações, esta dissertação apresenta o desenvolvimento, a modelação e a validação física de uma plataforma AMR compacta de tração diferencial.

A solução desenvolvida assenta numa filosofia operacional híbrida, combinando a fiabilidade determinística de um Veículo Guiado Automaticamente (AGV) com a flexibilidade de alto nível de um AMR através de comunicação em tempo real via Wi-Fi com um servidor. Mecanicamente, o robô foi estruturalmente miniaturizado para uma dimensão de $151 \times 95 \times 80$ mm, permitindo que múltiplos veículos naveguem simultaneamente e em segurança nas mesmas linhas da arena. A plataforma é controlada por um Raspberry Pi Pico W que executa todo o processamento localmente. O sistema incorpora ainda um sensor de infravermelhos de 5 canais para o seguimento de linha, e odometria diferencial de duas rodas, que permite a monitorização e o controlo da trajetória em tempo real.

Desta forma, o sistema não está rigidamente dependente de caminhos. Ao mapear as suas coordenadas (X, Y, θ) em tempo real, o robô possui a capacidade de abandonar as linhas da arena de forma fluida para executar diversas manobras. Esta capacidade, combinada com as leituras de um sensor de distância frontal, permite ao robô não só contornar obstáculos na pista, mas também colaborar com outros robôs no mesmo espaço da arena.

O resultado é um robô compacto capaz de participar na competição "Robot at Factory Lite", assim como de contribuir para o setor. O produto final servirá de referência para futuros projetos na mesma área e de base para demonstrações académicas e desafios de programação no setor pedagógico.

Abstract

In the era of Industry 4.0, Autonomous Mobile Robots (AMRs) play a vital role in the automation and optimisation of intralogistics. The "Robot at Factory Lite" competition provides a structured academic version of these systems, simulating a supply chain environment where mobile robots are responsible for autonomously managing the transport of materials. Although there is already a solution that meets the competition requirements, its physical footprint significantly restricts manoeuvrability, space optimisation, and multi-agent scalability. To address these limitations, this dissertation presents the design, modelling, and physical validation of a compact differential-drive AMR platform.

The developed solution relies on a hybrid operational philosophy, combining the deterministic reliability of an Automated Guided Vehicle (AGV) with the high-level flexibility of an AMR via real-time server communication over Wi-Fi. Mechanically, the robot is structurally miniaturised to a footprint of $151 \times 95 \times 80$ mm, allowing multiple vehicles to safely share and navigate the exact same physical arena lanes simultaneously. The platform is controlled by a Raspberry Pi Pico W executing all computation locally. The system also incorporates a 5-channel infrared array for line following and dual-wheel differential odometry that allows trajectory tracking.

Consequently, this system transcends rigid track dependencies. By mapping its planar pose coordinates (X, Y, θ) in real time, the robot features the capability to seamlessly depart from the line network to execute flexible, off-track manoeuvres. This capability, combined with a forward-facing Time-of-Flight (ToF) sensor, allows the robot to not only overcome obstacles on the track but also collaborate with other robots in the same arena space.

The end result is a compact robot capable of participating in the competition, as well as of contributing to the sector. The final product will serve as a reference for future projects in the same area and a basis for academic demonstrations and programming challenges in the educational field.

UN Sustainable Development Goals

The United Nations Sustainable Development Goals (SDGs) provide a global framework to achieve a better and more sustainable future for all. It includes 17 goals to address the world's most pressing challenges, including poverty, inequality, climate change, environmental degradation, peace, and justice.

The specific Sustainable Development Goals mentioned have the following names:

SDG 4 Ensure inclusive and equitable quality education and promote lifelong learning opportunities for all.

SDG 9 Build resilient infrastructure, promote inclusive and sustainable industrialisation, and foster innovation.

SDG 12 Ensure sustainable consumption and production patterns.

SDG	Target	Contribution	Performance Indicators and Metrics
4	4.3	The robotic platform developed has a reduced physical size and cost, which lowers the financial barrier for academic institutions, allowing access to quality technical education in robotics.	Estimated cost reduction of the hardware compared to the standard model; Number of academic programs adopting the platform.
	4.4	Developing an open-source educational mobile robot provides students with hands-on experience in Industry 4.0 technologies, improving technical skills for future engineering careers.	Student engagement rates in robotics and automation projects. Participation rates in the Robot@Factory Lite competition.
9	9.5	Advancing applied research in mobile robotics, kinematics, and cascade control architectures contributes to technological upgrades and innovation in factory automation.	Number of future dissertations, research papers and projects building upon the developed platform.
12	12.2	Designing a half-size robot significantly reduces the raw materials, electronic footprint, and overall energy consumption required to build the system.	Percentage reduction in physical volume, weight, and material footprint compared to the standard full-sized robot.

Acknowledgements

I would like to express my gratitude to my supervisor, Professor Paulo Gomes da Costa, and my co-supervisor, Professor José Magalhães Lima. Thank you for your guidance, your availability during our weekly meetings, and for introducing me to the challenging automation domain. Your genuine passion for this field is truly inspiring. I am especially grateful for the opportunity to discover the challenges of RoboCup and to Professor Paulo for welcoming me as a guest in his mobile robotics classes, which proved to be a fundamental pillar for the success of this project.

To FEUP, an institution that gave me lifelong friendships and the most memorable years of my life. Thank you to all my professors for preparing me for the professional world. FEUP did not only educate me, but it also gave me the opportunity to make my dream of studying abroad come true.

To my friends who were there for me during my five years at FEUP. Thank you for being part of this journey, for every word of advice, for every laugh, and for the unforgettable moments that transformed academic stress into good memories.

Finally, my most sincere thanks go to my family, particularly to my parents and my maternal grandparents. I am so grateful for your unconditional love and support and for always believing in my dreams. You made the entire university process feel easier, and every achievement in this work belongs to you just as much as it does to me.

Joana Daniela Pedrosa Moura

"Great things are done by a series of small things brought together."

Vincent Van Gogh

Contents

1	Introduction	1
1.1	Context and Motivation	1
1.2	Objectives	2
1.3	Dissertation Structure	2
2	Background and State-of-the-art	4
2.1	Mobile Robotics in Industry 4.0	4
2.1.1	AGVs vs. AMRs in Intralogistics	4
2.1.2	The Role of Educational Robotics	5
2.2	Existing "Robot at Factory Lite" Solutions	5
2.3	Navigation and Control Algorithms	6
2.3.1	Differential Drive Kinematics	6
2.3.2	Sensor Fusion and Error Estimation	7
2.3.3	PID Control for Line Following	8
2.4	Manufacturing Technologies for Compact Robots	8
2.4.1	PCB-as-Chassis Construction	8
2.4.2	Rapid Prototyping (3D Printing)	9
2.5	Compact Educational Platforms	9
2.5.1	The 3D-Printed Standard: microMVP	9
2.5.2	The Open-Hardware Standard: Mona	10
2.5.3	The Modern ROS Standard: CrowBot BOLT	11
2.6	Competition Rules	12
2.6.1	The Robot	12
2.6.2	The Shop Floor	12
2.6.3	The Parts	14
2.6.4	The Simulator	14
2.6.5	The Competition	14
2.7	Comparison and Conclusions	16
3	Hardware Design and Low-Level Modelling	17
3.1	Components	17
3.2	System Specifications	19
3.3	Structural Miniaturisation	21
3.4	Electronic Schematics	22
3.5	System Block Diagram	23
3.6	Motor Model	25
3.6.1	Electrical Subsystem	25
3.6.2	Mechanical Subsystem	25

3.6.3	Electromechanical Coupling	26
3.6.4	Motor Dead Zone	26
3.6.5	Motor Transfer Function	29
3.7	PI Controller Formulation	31
3.8	Kinematic Transformations	32
3.8.1	Inverse Kinematics	32
3.8.2	Forward Kinematics	33
3.9	Odometry	33
3.10	Infrared Line Sensor	35
3.10.1	Signal Pre-processing	35
3.10.2	Position and Edge Estimation	36
3.10.3	Intersection and Crossroad Detection	39
3.10.4	Reactive Control and Safety Fallbacks	40
3.10.5	Continuous Path-Following Control	41
3.11	Telemetry and Debugging Interface	43
4	Autonomous Navigation and Control Strategies	45
4.1	Track-Based Localization	45
4.1.1	Steady Movement Detection	46
4.1.2	Pose Correction using Homogeneous Transformations	46
4.2	Trajectory Modes	49
4.2.1	Go-To-Point Navigation (gotoxxy)	49
4.2.2	Line Follow Navigation	51
4.2.3	Arc Follow Navigation	53
4.3	Obstacle Avoidance and Overtaking Methodology	55
4.3.1	Finite State Machine (FSM) Architecture	56
4.3.2	Overtake Trajectory Geometry and Parametrisation	56
4.3.3	Algorithmic Phase Execution	58
5	Experimental Results and Validation	64
5.1	Track-Based Localization Performance	64
5.2	Trajectory Tracking Results	67
5.2.1	Go-To-Point Navigation (gotoxxy)	67
5.2.2	Line Follow Navigation	68
5.2.3	Arc Follow Navigation	69
5.3	Obstacle Overtaking and Avoidance Validation	70
5.3.1	Temporal and Cooperative Synchronization Validation	72
6	Final Considerations	74
6.1	Conclusions and System Achievements	74
6.2	Main Contributions	75
6.3	Future Work	75
	References	77
7	Appendix 1	81

List of Figures

2.1	Views of the robot:(a) Front view, (b) Top view, (c) Right side view [12].	5
2.2	Differential drive kinematics model illustrating the distance L [34].	7
2.3	Block diagram of a discrete PID controller feedback loop [1].	8
2.4	A fully assembled vehicle microMVP [40].	10
2.5	(a) Mona robot, (b) an expansion board for the off-the shelf modules: Raspberry Pi Zero, RF transceiver, and XBee [2].	10
2.6	Relevant components of the CrowBot BOLT educational robot [38].	11
2.7	Competition Arena, with warehouses and machines [11].	12
2.8	Dimensions of the machines [11].	13
2.9	Dimensions of the warehouses [11].	13
2.10	Parts [11].	14
2.11	Competition area with infrared emitter and starting position [11].	15
3.1	Physical prototype of the developed line-follower robot showing the structural and electronic configuration.	19
3.2	3D CAD comparison between the previous robot base plate (left) and the new compact design (right).	21
3.3	Electrical diagram.	22
3.4	System block diagram.	24
3.5	Motor model.	25
3.6	Dead zone.	27
3.7	Dead zone compensation scenarios.	28
3.8	Experimental determination of the system's transfer function parameters.	30
3.9	Closed-loop transfer function.	31
3.10	5-channel ITR20001/T infrared line sensor.	36
3.11	Experimental setup diagram alongside the real vs. expected line centre position tracking.	38
3.12	Comparison between real and expected edge tracking measurements during the rotation sweep.	38
3.13	Intersection validation test showing the physical crossroad approach (a) alongside the real-time sensor ratio spike and the resulting step change in the permanent cross counter (b).	40
3.14	Line sensor flowchart.	41
3.15	Non-linear velocity modulation profile, illustrating the parabolic decay of linear speed (v) as a function of angular speed (ω).	42
3.16	The <i>ComRobot</i> user interface showing real-time variables, configuration inputs, and the data plot.	44

4.1	Track-based location reference frame.	46
4.2	Trajectory modes: a) Go-To-Point, b) Line Follow, c) Arc Follow.	49
4.3	Relation between linear velocity and angular error.	50
4.4	Finite State Machine (FSM) transition diagram for the autonomous obstacle avoidance and overtaking sequence.	56
4.5	Geometric parametrisation of the S-shaped overtaking manoeuvre, illustrating the relationship between the splitting angle (α), the radius (R_{arc}), and the targeted lateral offset (h_{offset}).	57
4.6	Finite state machine for the obstacle avoidance manoeuvre with the step counter.	58
4.7	Geometric derivation of the cross-track error (e_{pass_line}), showing the perpendicular relationship between the tracking error and the desired path line.	60
4.8	Linear passing phase metrics, showing the tracking baseline starting at $P_{passing_start}$, the accumulated forward progress distance ($d_{progress}$), and the target threshold (overtake_pass_distance).	60
5.1	Experimental setup.	65
5.2	Estimated robot trajectory for a single lap.	66
5.3	Estimated robot trajectory for four laps.	66
5.4	Experimental 2D trajectory profiles using the Go-To-Point controller.	68
5.5	Experimental 2D tracking trajectories under various line-following boundary conditions.	69
5.6	Experimental 2D tracking trajectories under various arc-following boundary profiles.	69
5.7	Physical experimental setup within the testing arena, demonstrating the head-on initialization vectors for the active overtaking vehicle (left) and the oncoming obstacle platform (right).	70
5.8	Chronological phase breakdown diagram of the cooperative overtaking manoeuvre.	71
5.9	Experimental 2D spatial path traces demonstrating the interaction between the active overtaking vehicle (blue) and the oncoming obstacle vehicle (orange).	72
5.10	Temporal analysis of the cooperative overtaking manoeuvre, demonstrating real-time collision mitigation and sensor-cone clearance.	73

List of Tables

2.1	Comparison of Robotics Platforms regarding the project requirements.	16
3.1	Robot Components.	18
3.2	Comprehensive Mechanical, Electrical, Kinematic, and Operational Specifications of the Mobile Robot Asset.	20
3.3	Dimensional comparison between the previous and the proposed robot.	22
3.4	Quantitative error metrics between experimental tracking data and geometric models.	39
5.1	Control and Localization Parameters.	65
5.2	Localization Error Metrics.	67

List of Acronyms

ADC	Analogue-to-Digital Converter
AGV	Automated Guided Vehicle
AMR	Autonomous Mobile Robots
CPS	Cyber-Physical Systems
CPU	Central Processing Unit
DC	Direct Current
EKF	Extended Kalman Filter
FDM	Fused Deposition Modelling
FSM	Finite State Machine
GP	General-Purpose
HIL	Hardware-in-the-Loop
IoT	Internet of Things
IR	Infrared
LiDAR	Light Detection and Ranging
MCU	Microcontroller Unit
MIT	Massachusetts Institute of Technology
MVP	Multi-Vehicle Platform
PCB	Printed Circuit Board
PI	Proportional-Integral
PID	Proportional-Integral-Derivative
PPR	Pulse-Per-Revolution
PWM	Pulse Width Modulation
RFID	Radio-Frequency Identification
ROS	Robot Operating System
RPM	Revolutions Per Minute
ToF	Time-of-Flight
USB	Universal Serial Bus
2D	Two-dimensional
3D	Three-dimensional

Chapter 1

Introduction

The present chapter aims to illustrate the theme of the dissertation, allowing the reader to understand the main goals behind the project. The first section will present the context and motivation by identifying the engineering area of the project and stating the problem. Subsequently, section 1.2 will present the objectives of the dissertation, which gives the reader a better understanding of what will be done. Finally, the last section contains the dissertation structure.

1.1 Context and Motivation

The automation and robotics domains are in constant growth; current research focuses on advancing automation technologies to make industrial operations safer and more efficient for workers [37]. In education, it could not be different; some researchers believe that robots can be used as a potential tool for transmitting knowledge to students [4]. In this regard, the main goal of this dissertation was to create a robot that contributes to the education of younger generations.

Accordingly, the robot was constrained by size in order to be easily transported. The fact that the machine is portable allows demonstrations in various locations, enabling it to reach more students from different schools. Thus, this dissertation focuses on reducing the physical footprint of the robot without affecting its functional capabilities; the project consists of developing an efficient and compact robotic system to be used in constrained environments.

Additionally, the robot was developed to satisfy the constraints of the "Robot at Factory Lite" competition; this event aims to promote research in autonomous navigation, mobile robotics, and real-time decision-making [12]. During the challenge, the platform should perform tasks inside an arena that simulates an industrial logistics environment, which includes the transport of materials between machines and storage zones [11].

Thereafter, this structural miniaturisation includes two distinct operational advantages that are explicit goals of this design. First, reducing the physical vehicle footprint allows the entire industrial simulation arena to be proportionally downscaled, which reduces the space needed and facilitates the overall system transport. Second, the reduction in the robot's width allows for the

possibility of overtaking and, consequently, multi-agent interaction in the same lane, enabling multiple autonomous assets to simultaneously navigate the exact same arena space.

The real-world viability of the developed architecture was validated at the RoboCup Portugal Open 2026 in Barcelos¹. The hardware of the platform was provided to an academic team that successfully secured 1st place in the "Robot at Factory Lite" category. This achievement validates the physical design of the robot, its manufacturing viability, and its capacity to perform at the competition.

In essence, this project included the design and development of an autonomous robot that complies with the rules and environment of the competition, with a special focus on reducing the platform size to fit in a scaled arena and collaborate simultaneously with other robots in the same limited space.

1.2 Objectives

Concerning the objectives of this dissertation, the primary goal of this project was to develop a half-sized robot. Thus, the first milestone was the design of a compact mechanical platform that respects not only the functional but also the dimensional constraints of the "Robot at Factory Lite" competition.

Subsequently, a core objective involved the development of an embedded system capable of autonomous navigation. The robot was designed to be capable of moving inside an arena to specific locations while avoiding obstacles. This includes not only line following but also odometry-based transformations that allow the robot to be independent from the track, enabling explicit off-track overtaking manoeuvres to bypass obstacles and return to the line. Furthermore, the system must perform industrial tasks, such as the collection, transport, and placement of materials at machines and warehouses using an integrated electromagnet.

Correspondingly, another objective was to integrate localisation and mapping techniques, suitable for the competition environment, into the platform. This was achieved by combining real-time wheel encoder feedback with the signal from the line-following sensor array.

Finally, the last goal was to include interfaces for monitoring and debugging, which ensure real-time communication and control of the robot over Wi-Fi.

1.3 Dissertation Structure

In addition to the introduction, this dissertation contains 5 more chapters.

In Chapter 2, the state of the art is described and related work is presented. The existing solutions were analysed and critically reviewed, as they have an impact on future work. Different authors' proposals were compared and contributed to the creation of the solution presented in this dissertation.

¹Official event website: <https://www.festivalnacionalrobotica.pt/2026/>

Chapter 3 describes the hardware design and low-level electromechanical modelling of the half-sized robot, including the choice of physical components, electronic schematics and strategies for structural miniaturisation. Furthermore, this chapter derives the mathematical transfer functions of the DC motor subsystems, formulates the closed-loop PI velocity controllers, and establishes the forward and inverse kinematic transformations alongside the signal pre-processing routines for the 5-channel line sensor array. The chapter ends with the presentation of the *Com-Robot* interface.

Chapter 4 introduces the high-level autonomous navigation and control strategies implemented to confer operational flexibility to the platform. This section outlines the track-based localisation routines and spatial pose correction frameworks using homogeneous transformations. Additionally, it details the trajectory tracking controllers, including Go-To-Point (`gotoxy`) navigation, line following, and arc following. Besides that, the chapter describes the Finite State Machine (FSM) architecture developed for autonomous obstacle overtaking manoeuvres.

Chapter 5 presents the experimental results and physical validation of the compiled robotic platform. The performance of the track-based localisation and trajectory tracking control loops is rigorously evaluated. The chapter ends with empirical validation of the obstacle-overtaking methodology and confirms its viability within the arena space.

Finally, Chapter 6 presents the final considerations; in other words, it illustrates the main conclusions about the dissertation problem, as well as presents the possible contributions of the work to the field and the future work.

Chapter 2

Background and State-of-the-art

In this chapter, some theoretical aspects and fundamental knowledge relevant to understanding the work developed during the dissertation will be presented. Thus, the competition rules fundamental to the development of the robot will be explained, along with previous works that were used as a basis for the project.

2.1 Mobile Robotics in Industry 4.0

The concept of Industry 4.0 has revolutionised the manufacturing landscape, introducing Cyber-Physical Systems (CPS), where physical machines communicate with digital counterparts in real-time [31]. Within this ecosystem, the transport of materials has shifted from manual labour to autonomous systems [15, 24].

2.1.1 AGVs vs. AMRs in Intralogistics

Fragapane et al. [16] mentioned in their literature review that mobile robotic systems in factories are usually divided into two types: Automated Guided Vehicles (AGVs) and Autonomous Mobile Robots (AMRs).

With reference to AGVs, these robotic platforms rely on physical infrastructures to guide their motion, for example, lines on the floor or magnetic tapes. As indicated by the authors, AGVs typically depend on a central unit for scheduling and routing. This contributes to the reliability of the robot; however, it becomes less flexible [16, 25].

As for the AMRs, these systems are based on decentralised decision-making. As stated in [16], [25], and [14], AMRs are able to "communicate and negotiate independently with other resources," which enables dynamic path planning without fixed structures.

The "Robot at Factory Lite" competition simulates a hybrid scenario, as it combines characteristics of both AGVs and AMRs. The desired solution not only follows lines like a traditional AGV but also adopts the connectivity of an AMR as it receives dynamic orders from a central server. This synergy effectively fills the gap between classic automation and the smart factory [16, 21].

2.1.2 The Role of Educational Robotics

Educational robotic platforms are useful tools for preparing engineers for industrial reality. Krause et al. [20] wrote that an effective educational robot must balance software complexity with mechanical transparency. This perspective is equally supported by Huang [18] in a recent review of classroom applications. The article confirms the use of robotics has become a trending research domain, since it offers promising learning designs capable of boosting students' performance in technical disciplines. This idea is also expressed by Lopes et al. in a conference paper [23].

Furthermore, Varela-Aldas and Palacios-Navarro [38] believe that modern educational robots must be capable of not only simple movement loops but also of integrating IoT (Internet of Things) capabilities, such as Wi-Fi and ROS, in order to truly simulate the data exchange required in a smart factory.

2.2 Existing "Robot at Factory Lite" Solutions

The "Robot at Factory Lite" competition is an event based on the original Robot@Factory, adding some hardware and mechanical simplification [11]. The challenge has been happening for several years with the goal of improving the existing prototype [12].

The official reference robot for the competition was described by Luiz et al. [12] and Lima et al. [21], and corresponds to the baseline for electronic and mechanical performance.

Regarding the mechanical and kinematic architecture, the presented platform is a differential-drive mobile robot, as it has two independently powered traction wheels connected by a single axle and also a caster wheel; this combination allows not only precise movements like spinning but also great manoeuvrability [12, 28].

Furthermore, the author mentioned that the robot's dimensions have an impact on the time the robot spends moving inside the arena. In this case, it has a height of 80 mm, a width of 138 mm, and a length of 151 mm. Figure 2.1 illustrates the solution presented. Additionally, the Automated Guided Vehicle (AGV) also includes an electromagnet as an end effector, used to lift the parts [12].

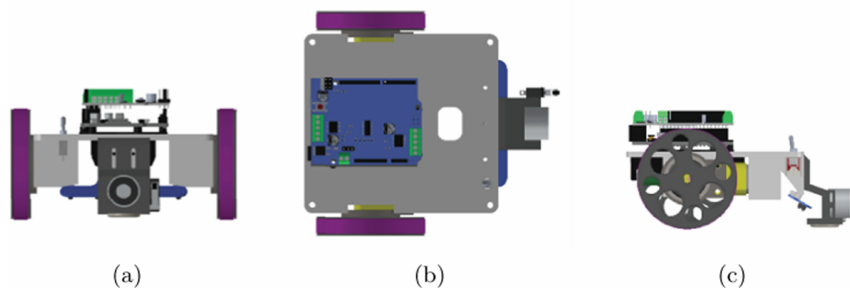


Figure 2.1: Views of the robot:(a) Front view, (b) Top view, (c) Right side view [12].

Concerning control and perception, the robot's motion relies on infrared line-following arrays, and it uses mechanical limit switches, in other words, touch sensors, to detect collisions or materials [12, 21].

Regarding actuation, while earlier versions relied on open-loop time-based navigation, the current platform presents a more reliable solution: it employs DC motors with integrated quadrature encoders. This improvement enhanced the control of speed and minimised path-following errors, as it allowed the implementation of PID control loops. The state machine of the robot was based on time; this indicates that it would have to change when the speed alters. With the encoders, the state machine remains the same, as the transitions are based on the distance travelled and angles turned [21].

Correspondingly, the literature emphasises the role of simulations in the development of robots; in this way, to test the control strategies mentioned, Braun et al. [8] introduced the Hardware-in-the-Loop (HIL) simulation using SimTwo. Thus, the logic presented in timed finite state machines can be tested in a virtual replica of the factory. This validation minimises hardware risk and reduces development time because the virtual environment mimics the physics and sensor noise of the real arena.

With respect to the electronic architecture, recently the state-of-the-art had a significant evolution; there was a shift from local RFID-based task allocation to a centralised Wi-Fi approach, simulating Industry 4.0. Early prototypes replaced the Arduino Uno/Nano architectures with the ESP32 microcontroller (specifically the ESPDUINO-32 form factor). This improvement allows real-time Wi-Fi communication with a factory server (supply chain management system), granting dynamic task scheduling instead of the static behaviour of reading RFID tags attached to boxes [21].

2.3 Navigation and Control Algorithms

The current standard solution, presented in section 2.2, consists of a differential drive architecture. Thus, to control the motion of the robot, the definition of a kinematic model and control strategies is required.

2.3.1 Differential Drive Kinematics

With reference to the kinematics, to determine the velocity of the robot based on the speed of its wheels, a kinematic model is required. According to Siegwart et al. [34], the system is conditioned by a non-holonomic constraint; in other words, the robot can only move in the direction of its drive wheels, not in a transversal direction. Using a car as an example, it has a velocity constraint; that is, it cannot move sideways. However, it can still reach every position through manoeuvres.

In the case of a robot with a wheel radius r and a distance between the wheels' contact points L , represented in Figure 2.2, the relation between the linear velocity v and angular velocity ω , with respect to the individual wheel speeds (v_R, v_L), is given by:

$$v = \frac{v_R + v_L}{2} \quad (2.1)$$

$$\omega = \frac{v_R - v_L}{L} \quad (2.2)$$

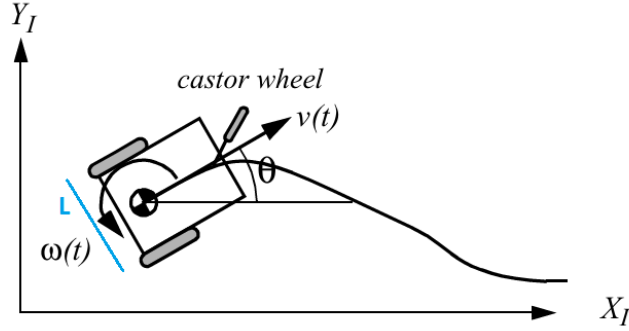


Figure 2.2: Differential drive kinematics model illustrating the distance L [34].

The previous model highlights the influence of the geometry of the robotic platform on its stability. As mentioned in the objectives of the dissertation (section 1.2), the goal is to minimise the robot's dimensions; however, a reduction of the parameter L will significantly increase the rotational sensitivity (ω) to small differences in motor speed, as shown in equation 2.2. Thus, a higher frequency control loop will be needed to maintain stability [34].

2.3.2 Sensor Fusion and Error Estimation

Before the control logic can be applied, there must be a conversion from the discrete data read by the infrared sensors into a continuous position estimate. The technique reviewed in recent line-follower meta-analyses is the weighted average method [9]. Instead of simply calculating the position based on which sensors are on or off, this methodology considers the intensity read by the sensor, allowing for the calculation of the centre of mass of the line.

With S_i representing the intensity read by sensor i , and X_i being the physical position of that sensor relative to the robot's centre, the estimated line position P_{line} is given by:

$$P_{line} = \frac{\sum_{i=1}^N S_i \cdot X_i}{\sum_{i=1}^N S_i} \quad (2.3)$$

The error term, needed in the control system, is calculated by the difference between the reference position, $P_{reference}$, corresponding to the position we want to achieve, in other words, the ideal centre, usually 0, and the estimated line position:

$$e(t) = P_{reference} - P_{line} \quad (2.4)$$

This approach grants sub-millimetre precision even with low-resolution sensor arrays [9].

2.3.3 PID Control for Line Following

Concerning the control system, to guarantee that the robot's position converges to the desired one by correcting the calculated error, a closed-loop controller is required. Thereby, the proportional-integral-derivative (PID) controller, presented in Figure 2.3, is the industry standard for line follower robots [3]. While usually represented in continuous time, the implementation on a digital microcontroller, like the ESP32, requires a discrete PID formulation [5].

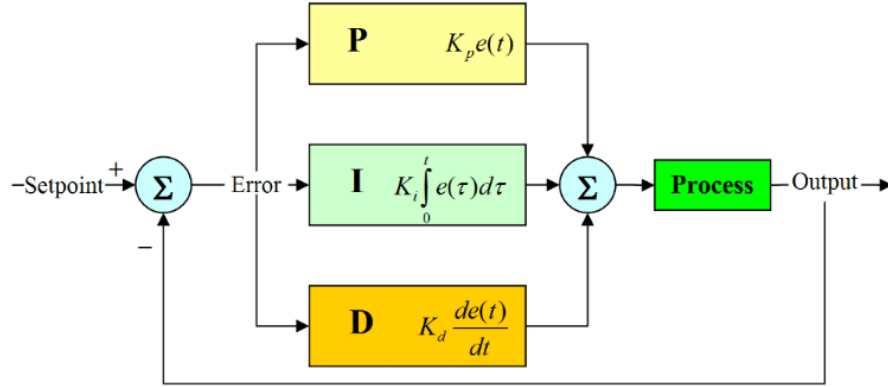


Figure 2.3: Block diagram of a discrete PID controller feedback loop [1].

Approximating the integral term as a summation and the derivative as a finite difference over the sampling time T_s , the control output $u[k]$ at time step k is expressed as:

$$u[k] = K_p e[k] + K_i \sum_{j=0}^k e[j] T_s + K_d \frac{e[k] - e[k-1]}{T_s} \quad (2.5)$$

Where K_p , K_i , and K_d represent the tuning gains. According to a recent review written by Brigido and Oliveira [9], the classic PID control structure remains the most used strategy regarding line-following robots due to its computational efficiency. Using it leaves CPU resources available for higher-level tasks, such as mission planning and Wi-Fi communication.

2.4 Manufacturing Technologies for Compact Robots

Krause et al. [20] included in their guideline to select an educational robot that the manufacturing method significantly impacts the platform modifiability and mechanical robustness. A review of the state-of-the-art platforms highlights two dominant approaches: PCB-as-Chassis Construction and 3D Printing.

2.4.1 PCB-as-Chassis Construction

In the domain of micro-robotics (under 100 mm), the standard approach is the PCB-as-Chassis. Platforms like the Mona robot [2] employ the printed circuit board itself as the structural frame, with motors soldered directly to the substrate.

This methodology, according to Arvin et al. [2], reduces size and eliminates wiring complexity. However, this architecture does not offer mechanical flexibility [27]; changing the original design, for example, to add an end effector would be impossible without manufacturing a new motherboard.

2.4.2 Rapid Prototyping (3D Printing)

On the other hand, larger platforms, including the standard "Robot at Factory Lite" solution [12], rely on fused deposition modelling (FDM) 3D printing. This option is also used in micro-platforms like the microMVP [40].

The primary benefit of this approach is the modularity. Luiz et al. [12] state that 3D printing allows the creation of complex geometries to mount sensors and electromagnets. Similarly, Yu et al. [40] and Walia et al. [39] mentioned that this option permits easily changing the mechanical design without compromising the electronics. The trade-off is structural rigidity. 3D-printed parts generally have lower stiffness, requiring thicker chassis walls to prevent flexing [6, 40].

2.5 Compact Educational Platforms

In the educational and research fields, selecting the most adequate robotic hardware is a critical step for any project [13]. In an article published by Krause et al. (2024) [20], it is emphasised that the hardware decision has a huge impact on the learning outcome and on the success of the implementation. The author even wrote that "hardware limitations" are often the primary bottleneck in educational robotics.

Following Krause et al. [20] guidelines, the market analysis for existing open-source and commercial platforms suitable for the "Robot at Factory Lite" competition allowed us to conclude that, while many robots exist, most of them fail the dimension criterion of this dissertation, since their diameter is typically larger than 150 mm. Conversely, compact robots typically focus on swarm behaviours; that is, random movements [2, 32], rather than the precise path following required for this challenge [11].

Therefore, the following subsections will describe three robots that fit most of the desired characteristics not only in educational robots but also in the competition constraints: the microMVP [40], the Mona robot [2], and the CrowBot BOLT [38].

2.5.1 The 3D-Printed Standard: microMVP

In 2017, Yu et al. [40] introduced the microMVP (micro Multi-Vehicle Platform), presented in Figure 2.4, a project developed at MIT and Rutgers University. Their aim was to create a platform that was affordable and easy to manufacture, with the goal of democratising robotics research.

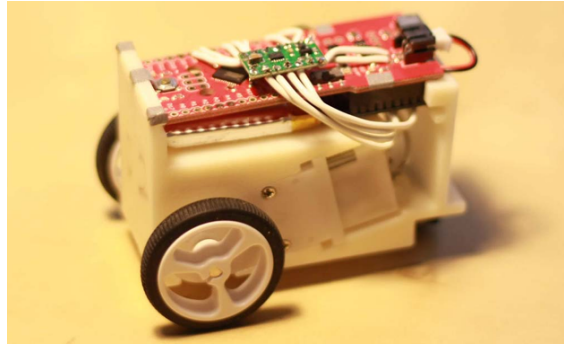


Figure 2.4: A fully assembled vehicle microMVP [40].

The robot created has the dimensions 80 x 50 x 60 mm, which satisfy the half-size goal of this dissertation. Correspondingly, it heavily relies on 3D printing for its structure to reduce construction costs, similar to the existing standard solution for the competition [12, 40].

Nonetheless, the microMPV has a limitation in its navigation strategy regarding the "Robot at Factory Lite" competition: it does not include the high-density onboard line sensors required to navigate the factory floor autonomously. Instead, it relies on external overhead cameras, as it was designed for swarm research. It also does not have an end effector capable of carrying the parts [40].

2.5.2 The Open-Hardware Standard: Mona

Addressing the need for better onboard sensing, Arvin et al. [2] developed the Mona robot, visible in Figure 2.5, in 2019 at the University of Manchester. Mona is an open-source platform specifically designed for swarm robotics education.

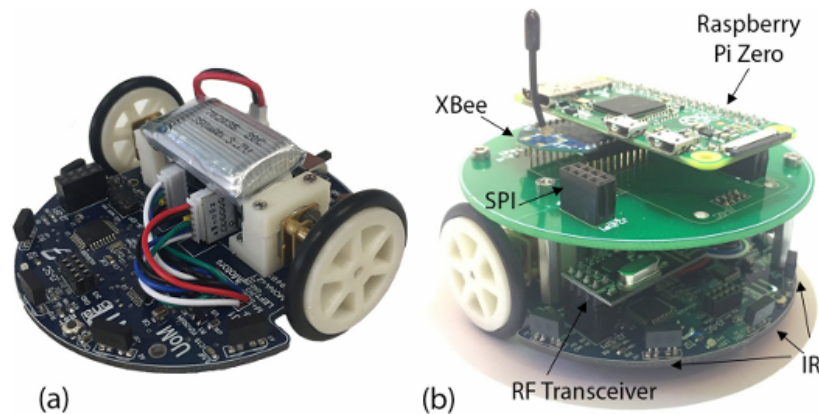


Figure 2.5: (a) Mona robot, (b) an expansion board for the off-the shelf modules: Raspberry Pi Zero, RF transceiver, and XBee [2].

The robotic platform features a circular design with an 80 mm diameter and includes basic proximity sensors to avoid collisions with obstacles. This solution, unlike the microMVP (section 2.5.1), includes a decision-making system that does not require external computers [2, 40].

Despite being closer to the desired requirements, the robot still lacks the vertical sensors essential for line following. Although it has sensors arranged horizontally for avoiding walls, reconfiguring the Mona would require redesigning its entire motherboard [2].

Similarly, the robot does not have the structural stability needed to mount the servo mechanism and electromagnet required to lift the competition boxes because the platform is designed to be low-power and lightweight [2, 11].

2.5.3 The Modern ROS Standard: CrowBot BOLT

Most recently, in 2024, Varela-Aldás and Palacios-Navarro [38] presented a framework using the CrowBot BOLT, illustrated in Figure 2.6. Unlike the previous mentioned robots, this solution includes downward-facing line sensors, making it capable of following tracks.

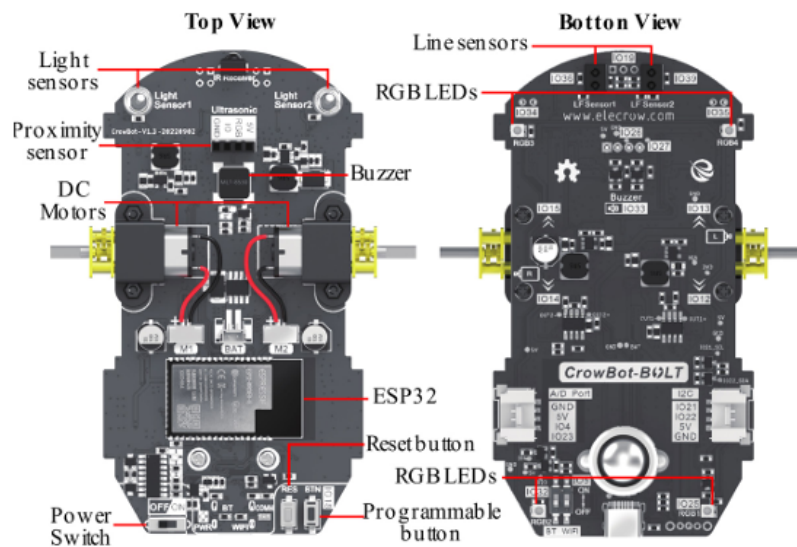


Figure 2.6: Relevant components of the CrowBot BOLT educational robot [38].

This platform represents the modern Industry 4.0 approach to education since, different from robots developed previously, it is based on the ESP32 microcontroller's Wi-Fi capabilities, the same chip used in the standard version of the competition [21]. Besides that, the authors confirmed that the robot could be controlled using ROS (Robot Operating System) over Wi-Fi, integrating the platform into a connected IoT network, proving that small robots can handle complex communication tasks similar to those in a smart factory [38].

Conversely, as a commercial off-the-shelf kit, it presents a physical integration problem: the chassis is a fixed, injection-moulded plastic shell. There is no physical space or mounting points to add the end effector needed for the competition. Modifying the commercial shell to fit an electromagnet would compromise the robot's structural integrity.

2.6 Competition Rules

Although the main goal of this dissertation is to contribute to the educational field, the robotic platform was also developed to participate in the "Robot at Factory Lite" competition. Accordingly, it is essential to understand the rules of this challenge to create a robot that complies with them.

Regarding the competition, the objective is to encourage the participants to create a robot capable of collecting, positioning, and transporting materials between machines and warehouses, as well as self-localising and navigating while avoiding collisions with walls [11].

2.6.1 The Robot

Concerning the robot, the maximum dimensions allowed are 30 x 30 x 30 cm [11]. However, the aim of this dissertation is to develop an even smaller robot, combining the advantages of half-sized robots with the functional capabilities of standard ones.

Correspondingly, the platform must be completely autonomous and cannot communicate with any external system that is not provided by the competition's organisation [11].

Additionally, the robot can be equipped with an electromagnet to drag the materials instead of picking them up with a forklift [11].

2.6.2 The Shop Floor

With respect to the competition area, it aims to represent a shop floor and will be created with a two-sheet A0 print containing black lines on a white background, which contributes to the robot localisation, as illustrated in Figure 2.7 [11]. The maximum dimensions of the arena are 1.7 x 1.2 m [11].

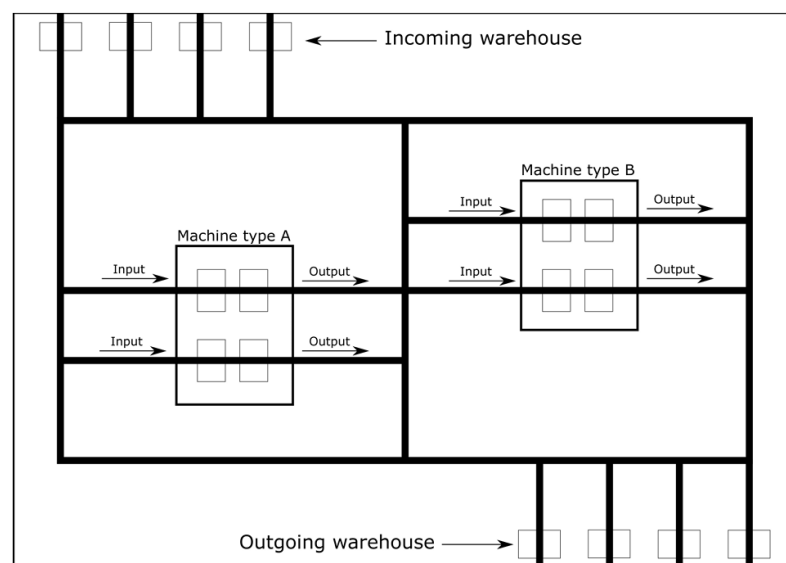


Figure 2.7: Competition Arena, with warehouses and machines [11].

Moreover, the shop floor contains two warehouses and four machines; the first warehouse (incoming warehouse) stores parts to be produced, while the outgoing warehouse contains the finished parts. On the other hand, the machines can be divided into two types: A and B. The arena is composed of two machines of type A and two machines of type B.

With regard to the machines, the robots are responsible for loading the parts on the left side of the machine (input) and, after the material is processed, unloading the part on the right side (output) [11].

Furthermore, the dimensions of the machines and the warehouses are presented, respectively, in Figures 2.8 and 2.9 [11].

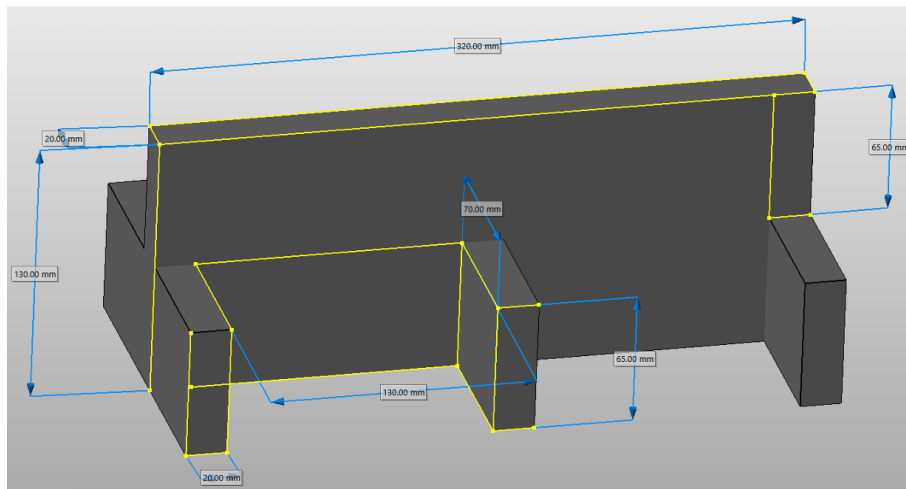


Figure 2.8: Dimensions of the machines [11].

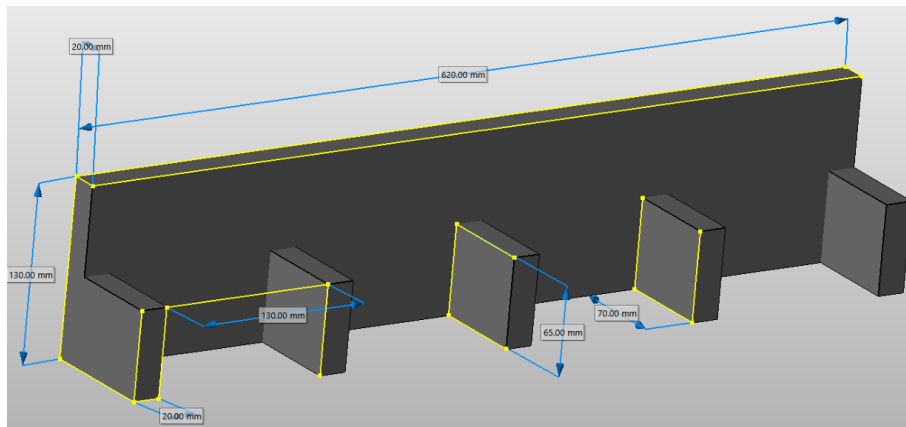


Figure 2.9: Dimensions of the warehouses [11].

Concerning the floor markers, the arena has black lines on the floor, with a width of 24 mm, that can be used to guide the robotic platforms between machines and warehouses. In conjunction with these, the teams are allowed to use other localisation mechanisms. Near the corners of the competition area, there will be squares of 10 x 10 cm, where the participants can place their own markers to help the robots in their navigation [11].

2.6.3 The Parts

With reference to the parts, they have standard dimensions of 90 x 60 mm for the width and length, and their height must be equal to or greater than 65 mm. Each material has a metal plate measuring 20 x 80 mm on their front wall [11].

Besides that, the boxes will have different colours to identify their type: blue represents a final part, green indicates an intermediate, and red is classified as raw material [11].

Figure 2.10 illustrates an example of the parts to be transported by the robot [11].

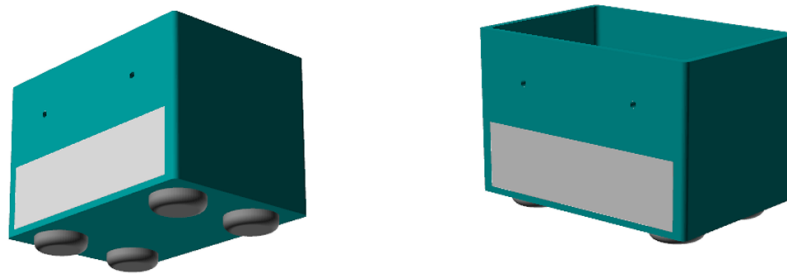


Figure 2.10: Parts [11].

2.6.4 The Simulator

Moreover, the organisation provides a simulator of the full competition environment that allows not only testing and improvement without the necessity of hardware but also control using a hardware-in-the-loop technique through a serial port from an Arduino device [11].

In the simulation, each team can model their robot platform by editing an XML file, since all the tools and source code are provided by the competition [11].

2.6.5 The Competition

Firstly, to start the robot, an infrared message sent by an emitter placed in the arena will be used, as shown in Figure 2.11. Thus, the robot should have an infrared receiver using a carrier of 38 kHz; for example, the VS1838. The signal should be connected to the input pin (Rx) of a serial port with a baud rate of 2400 bps [11].

Subsequently, the received message is interpreted by the robot. If it is a byte with the message "W", it means the starting condition. Conversely, a "U" is broadcast every 500 ms and acts like a test for the connection. Following this, four bytes are sent and indicate the orders, being "w" a red part, "u" a green part, and "o" a blue part [11].

Regarding the competition, the robotic platform will have to face three rounds, held preferably on consecutive days. Each round starts with a testing time of ten minutes. This is followed by the trial in which each team can attempt as much as they can in the ten minutes reserved for this part, knowing that only the best round will be considered [11].

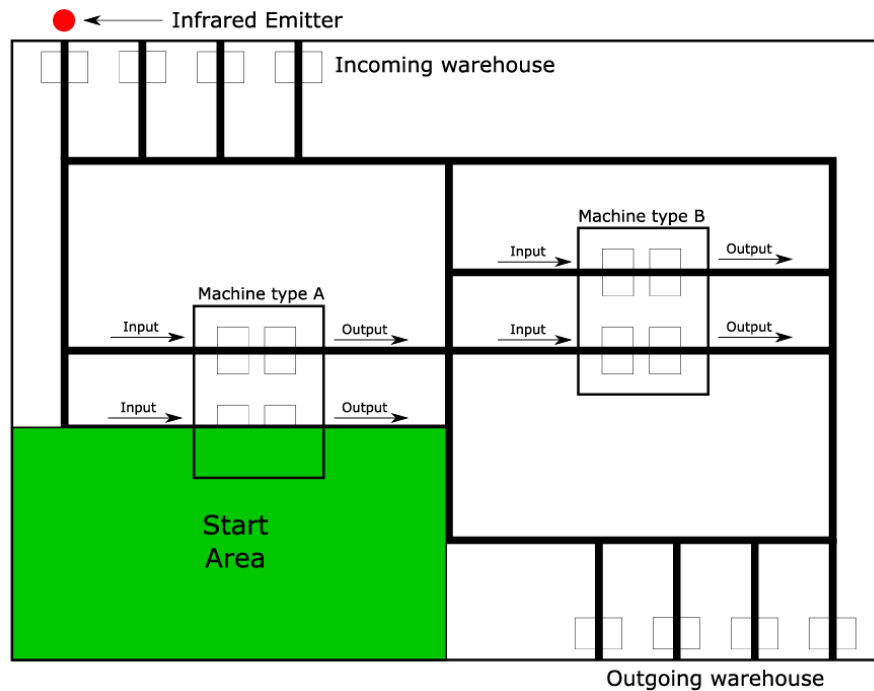


Figure 2.11: Competition area with infrared emitter and starting position [11].

To classify the attempts, the first criterion is the number of parts correctly placed in the outgoing warehouse. The time spent, considering penalties, is used as a second criterion [11].

It is important to highlight that during the trial, nobody can touch the robots, and if a robot abandons the arena, it cannot return for the trial [11].

Respecting the first round, the challenge is to move four materials from the incoming warehouse to the outgoing warehouse as quickly as possible. The second round includes the processing of the green parts before placing them in the outgoing warehouse. Additionally, in the third round, one or more boxes will have to be processed by more than one machine, depending on the parts' colour: the blue parts should be placed in the outgoing warehouse, the green boxes in machine type B, and the red materials in machine type A [11].

For the second and third rounds, it is only possible to start a new attempt one minute after the beginning of the previous one; this limits the number of tries to ten [11].

In addition, for the third round, the teams can receive extra points by successfully placing a part on a machine or in a warehouse, or by correctly returning to the docking station at the end of the attempt [11].

On the other hand, if the robot is in a situation from which it cannot recover by itself, the team may ask to access the robot. During the intervention, the trial stops, but the timer keeps running. After that, the contestants must position the robot and ask permission from the referee; when it is granted, they can prepare the robot for the automatic start. A random element will be placed in the arena [11].

The competition has a jury responsible for guaranteeing justice in the application of regulations and rules, and a referee who ensures compliance with the rules during the competition [11].

2.7 Comparison and Conclusions

The review of the state of the art reveals a clear technological gap. As summarised in Table 2.1, there is currently no platform that combines the compact size of the micro-robots with the manipulation capabilities of standard factory robots.

Table 2.1: Comparison of Robotics Platforms regarding the project requirements.

Feature	Standard R@FL (Reference)	microMVP (2017)	Mona (2019)	CrowBot (2024)	Proposed Robot (This Work)
Dimensions	151 × 138 mm	80 × 50 mm	80 mm (Dia)	~90 mm (Dia)	151 × 95 mm
Nav. Method	Line Sensors	Ext. Camera	Proximity	Line Sensors	Line Sensors
Control	Embedded	Centralized	Swarm	Embedded (Wi-Fi)	Embedded (Wi-Fi)
Lifting Arm	Electromagnet	No	No	No	Electromagnet

The standard robot (Section 2.2) is functional; however, the goal is to find a solution that keeps the capabilities of this version while minimising its dimensions. On the other hand, the compact robots (microMVP, Mona, and CrowBot BOLT) are the right size but lack the specific actuators (magnets) and sensor configurations (line arrays) required for the task [2, 38, 40].

Therefore, this dissertation proposes the design of a hybrid solution: a custom robot that takes the Wi-Fi architecture of the CrowBot, the open-hardware philosophy of the Mona, and the 3D-printed manufacturability of the microMVP, while integrating them into a specialised chassis capable of participating in the "Robot at Factory Lite" competition [2, 38, 40].

Regarding the manufacturing technologies, while the PCB-as-Chassis approach used in Mona [2] is superior for minimising size, it lacks the adaptability required for the specific tasks of this competition. The success of the standard solution [12] proves that 3D printing is the correct approach based on the complexity and the required sensors and electromagnets. Therefore, this project applies the 3D-printing philosophy with the goal of miniaturising it to the scale of the microMVP [40].

Chapter 3

Hardware Design and Low-Level Modelling

This chapter presents how the compact, differential-drive robot was designed, built, and modelled. While the high level code defines the target position of the robot, the chassis and motors determine how the robot moves.

The first part of this chapter is dedicated to the physical and electrical layout, starting with the components, the system specifications, the space-efficient 3D-printed parts, the electrical diagram, and the system layout. Then, it focuses on the behaviour of the motors, the motor dead zone, and the way the PI controller ensures the wheel speeds are accurate. The last part is about the kinematics and the sensors, explaining how wheel rotation corresponds to the actual movement, how wheel odometry tracks the position, and how the infrared line sensor detects tracks, edges, and crossroads. The chapter ends with the advantages of the telemetry and debugging *ComRobot* interface. Having this reliable hardware and control base allows us to build the foundations for the advanced tracking, localisation, and overtaking systems in the next chapters.

3.1 Components

The development of the line-follower robot was based on the existing solution [12], presented in section 2.1. In this case, the aim is to reduce the size of the robot; thus, some adaptations were performed relative to the previous robot.

Table 3.1 presents a list of the components that make up the robot, as well as their quantities and descriptions.

Regarding the mechanical structure, to achieve the half-size requirement, 3D printing was selected. This solution allows for customisation of the design, which is essential for implementing the electromagnet and the board without compromising the equilibrium of the robot. Given the small dimensions of the robot, the centre of gravity is a critical factor in order to maintain balance and performance.

Table 3.1: Robot Components.

Component	Quantity	Description
Robot structure	1	3D printed piece
Motor with encoder and Gearbox	2	Direct Current (DC) Motor 6 V 160 RPM 120:1
Wheel	2	3D printed piece
Elastic band	2	Material: rubber
Electromagnet	1	5V
Electromagnet support	1	3D printed piece
Line sensor	1	5-ch ITR20001/T infrared detector
Distance sensor	1	VL53L0X Time-of-Flight
Battery	2	INR18650P 3.7 V lithium ion battery
Battery holder	1	Holds two 18650 lithium-ion cells
Board holder	1	3D printed piece
Microcontroller	1	Raspberry Pi Pico W (RP2040)
Connection board and motor power drive	1	Pico4drive
Glide	1	Teflon stick

In addition, the robot has all its wiring visible to facilitate debugging and component replacement.

Moreover, the Teflon stick glide was chosen instead of a caster wheel. This option reduces mechanical complexity and saves space; nevertheless, it performs worse on rough surfaces, which does not have an impact in this case.

Additionally, the robot is equipped with a 5 V electromagnet that acts as an end effector to interface with the metallic plates of the competition parts.

Considering the locomotion, both DC motors have a 120:1 gearbox¹. This configuration provides high torque at low speeds, allowing for slow and precise manoeuvres, which are essential when the robot is close to the machines in the arena.

Besides that, the encoders are crucial components for odometry. Even though the main technique of localisation is the line, odometry is an essential complement and will be the source of information when the robot moves backward.

Similarly, a distance sensor² was implemented to guarantee the safety of the robot, allowing for the detection of obstacles in the arena and the overtaking of other robots.

Concerning the electronics, it is centred around the Raspberry Pi Pico W³, which provides dual-modality connectivity through USB and Wi-Fi. While the first is primarily used for software deployment and debugging, the second enables real-time wireless communication, allowing for the exchange of telemetry and command data without the need for physical tethering or recompilation.

¹Product page available at: <https://www.botnroll.com/en/dc-motor/1645-micro-dc-geared-motor-wencoder-sj01-6v-160rpm-1201.html>

²Product page available at: <https://www.botnroll.com/en/infrared/2531-vl53l0x-time-of-flight-distance-sensor.html>

³Product page available at: <https://www.botnroll.com/en/controller-boards/4466-raspberry-pi-pico-w.html>

This wireless autonomy is essential for maintaining the robot's mobility and operational flexibility within the competition arena.

Similarly, the Pico4Drive⁴ simplifies the wiring, which is essential in a reduced size robot, providing the H-Bridge necessary to drive the two DC motors. The complete electrical schematic of the Pico4Drive board can be consulted in Appendix 1 of chapter 7.

The finalised physical prototype resulting from these design selections is shown in Figure 3.1, illustrating the compact assembly from three distinct viewpoints. The top-down view (Figure 3.1a) highlights the dense but accessible wire routing arranged directly above the main microcontroller board to simplify manual debugging. The side profile (Figure 3.1b) demonstrates the custom 3D-printed wheels fitted with red rubber bands for traction and the forward extension designed to hold the end-effector. Finally, the underside view (Figure 3.1c) shows the centralised placement of the two lithium-ion batteries, the infrared line sensor array, the protective Teflon glide, and the internal motor gearboxes.

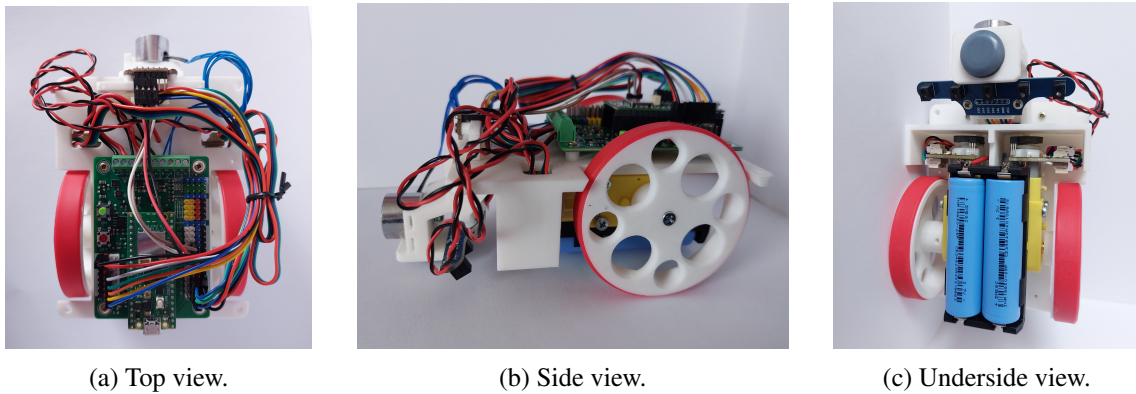


Figure 3.1: Physical prototype of the developed line-follower robot showing the structural and electronic configuration.

3.2 System Specifications

To establish a rigorous baseline for the subsequent electromechanical modelling, kinematic transformations, and localisation algorithms, the physical and operational parameters of the developed mobile platform must be explicitly defined. Table 3.2 consolidates these core properties, categorising the platform's constraints across mechanical, electrical, kinematic, and operational domains. These values represent the definitive constraints of the physical asset and dictate the boundary conditions of the control loops.

The kinematic boundaries represent the theoretical, unloaded maximum capabilities of the propulsion system. Based on the rated gearbox output speed and the nominal wheel radius ($r = 0.034$ m), the maximum linear velocity is derived as:

⁴Product page available at: <https://www.botnroll.com/en/shields-hats/4722-pico4drive-development-board-for-pi-pico.html>

$$v_{max} = 160 \text{ RPM} \times \frac{2\pi}{60} \times 0.034 \text{ m} = 0.57 \text{ m/s} \quad (3.1)$$

Similarly, using the wheel distance of $L = 0.080 \text{ m}$, the maximum theoretical angular velocity during an in-place spin is calculated using Equation 2.2 as follows:

$$\omega_{max} = \frac{0.57 - (-0.57) \text{ m/s}}{0.080 \text{ m}} = 14.25 \text{ rad/s} \quad (3.2)$$

In practice, these targets are programmatically bounded within the firmware control loops to retain torque headroom for PID corrections and to prevent wheel slip on the arena floor.

Table 3.2: Comprehensive Mechanical, Electrical, Kinematic, and Operational Specifications of the Mobile Robot Asset.

Domain	Parameter Description	Value / Designation
Mechanical	Total Operational Mass (m)	0.390 kg
	Chassis Dimensions ($L \times W \times H$)	$151 \times 95 \times 80 \text{ mm}$
	Nominal Wheel Diameter	68 mm
	Nominal Wheel Radius (r)	34 mm
	Distance Between Wheels (L)	80 mm
Electrical	Primary Microcontroller Unit (MCU)	Raspberry Pi Pico W
	Primary Power Source	2S Li-Ion (INR18650P, 7.4 V)
	Primary Actuation System	Micro DC Gearmotors (120 : 1)
	Motor Drive Interface	Pico4Drive Board
	Odometry Feedback System	Integrated Wheel Encoders
	Distance Sensor	VL53L0X Time-of-Flight (ToF)
	Line Sensor	5-Channel IR (ITR20001/T)
Kinematic	Rated Gearbox Output Speed	160 RPM
	Theoretical Max. Linear Velocity (v_{max})	0.57 m/s
	Theoretical Max. Angular Velocity (ω_{max})	14.25 rad/s
	Motor Pulse-Per-Revolution (PPR)	16 PPR (Motor Shaft)
	Effective Wheel Output Resolution	1920 PPR (Gearbox Output)
Operational	Primary Navigation Modalities	Line Following & Trajectory Tracking
	Dead Reckoning Method	Dual-Wheel Differential Odometry
	Obstacle Detection Range	50 mm to 1200 mm (Sensor Bound)
	Operational Avoidance Trigger (d_{detect})	300 mm (Programmatic Bound)

3.3 Structural Miniaturisation

A major goal of this dissertation was to minimise the physical footprint of the mobile platform, to fit into more constrained environments, and to share the "Robot at Factory Lite" arena with other robots. The primary engineering challenge consisted of shrinking the lateral dimension (width) of the robot while keeping the necessary space for internal electronics, power distribution, and actuation.

This structural compaction focused heavily on redesigning the main 3D-printed chassis base plate. A comparison between the base structure of the previous solution [12] and the new "Half-Size" design developed in this dissertation is presented in Figure 3.2.

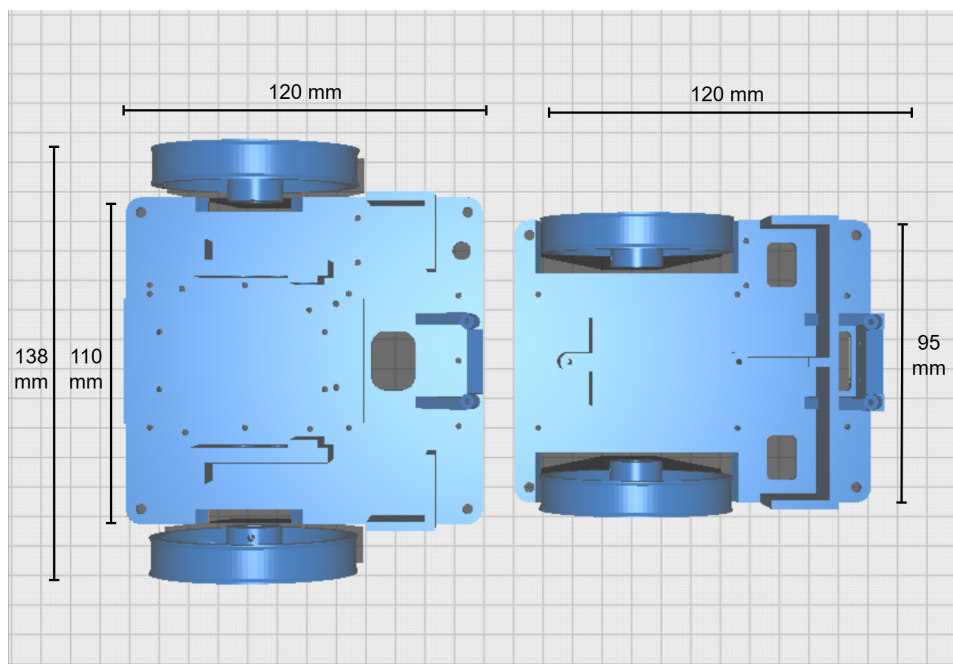


Figure 3.2: 3D CAD comparison between the previous robot base plate (left) and the new compact design (right).

The objective was to organise the existing electronic components to integrate them into the 3D part in the most compact way. In the previous version, the drive wheels were mounted completely external to the 3D-printed robot structure, which significantly expanded the robot's total outer footprint. In contrast, the new chassis design introduces a dedicated space for the wheel within the monolithic 3D part itself. This allows the wheels to sit tucked inside the main structural boundary of the robot.

The quantitative physical dimensions of both configurations, comparing the 3D-printed base plates and the fully assembled robots, are summarised in Table 3.3.

As observed from the experimental data, while the longitudinal axis (length) and vertical profile (height) were kept constant at 151 mm and 80 mm, respectively, to maintain space for the components and sensor clearance, the maximum outer width was successfully reduced from 138 mm down to 95 mm. This represents a significant 31.2% reduction in total operational width,

Table 3.3: Dimensional comparison between the previous and the proposed robot.

Configuration Layer	Previous Version (mm)	New Version (mm)	Net Reduction
3D Base Part			
— Length	120	120	0%
— Width	110	95	13.6%
Fully Assembled Robot			
— Length	151	151	0%
— Width	138	95	31.2%
— Height	80	80	0%

lowering the geometric volume of the robot and making it perfectly suited for high-density factory layouts.

3.4 Electronic Schematics

The electronic schematic diagram, shown in Figure 3.3, illustrates the wiring connections between all the electronic components used in the robot. The mechanism must perform its tasks autonomously within the factory arena. In order to achieve that, the platform must be able to read the external environment, process the received data, and, based on the programmed logic, send control signals to the actuators.

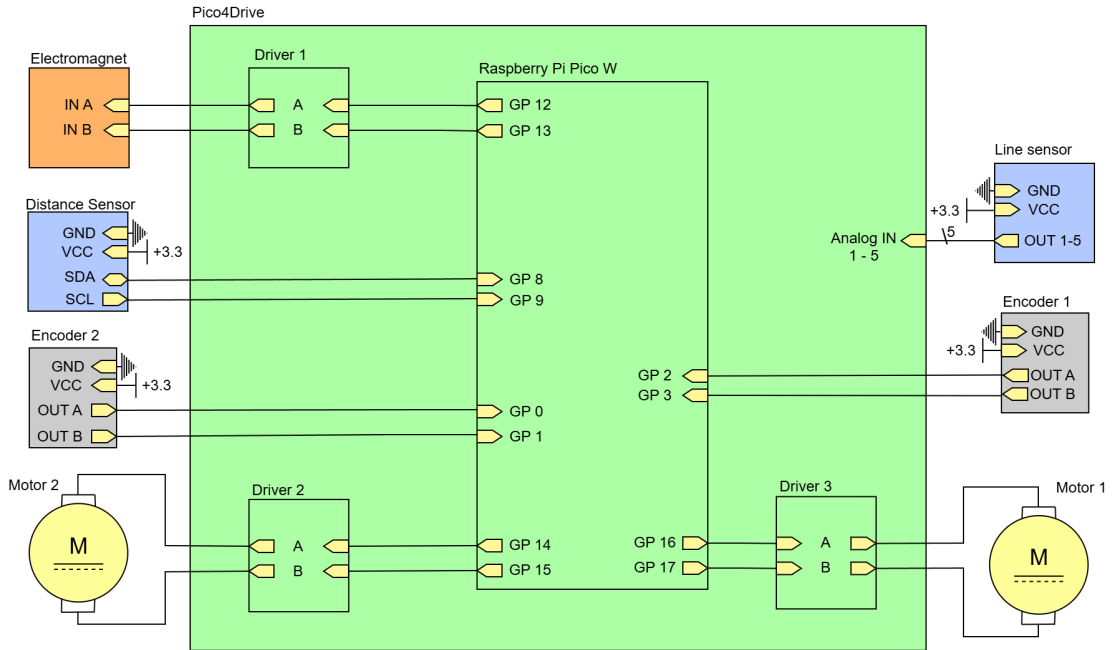


Figure 3.3: Electrical diagram.

Initially, the robot, as a line follower, needs to receive information about the black line in order

to localise itself. This data is acquired by the 5-channel ITR20001/T infrared line sensor⁵. The sensor's analogue signals are read and digitised through the analogue-to-digital converter (ADC) capabilities managed by the Pico4drive expansion board. Thus, it is possible to calculate the position of the robot relative to the centre of the line, allowing the microcontroller to evaluate the necessary trajectory corrections.

In addition, the information from the frontal distance sensor is received through digital pins GP 8 and GP 9, allowing the robot to safely navigate the arena while avoiding obstacles.

Subsequently, driving commands are sent to the DC motors. These commands are transmitted directly from the Raspberry Pi Pico W⁶ to the motor drivers integrated on the Pico4drive board using Pulse Width Modulation (PWM) signals through digital pins GP 14 and GP 15 for motor 2, and GP 16 and GP 17 for motor 1.

Moreover, it is also required for the robot to transport parts within the arena. The loading and unloading process is commanded by the microcontroller using GP 12 and GP 13. The 5 V electromagnet is activated, allowing the robot to magnetically attach to the metallic plates on the competition boxes for transport.

Regarding the location, using the two quadrature encoders allows the robot to estimate the speed of the wheels and perform odometry calculations. The outputs of the first encoder, including both channels A and B, are connected, respectively, to GP 2 and GP 3 digital input pins of the microcontroller. The outputs of the second encoder are connected to GP 0 and GP 1.

Concerning the power supply, the energy source consists of two 3.7 V lithium-ion batteries connected in series, providing a nominal voltage of 7.4 V. This battery pack is connected directly to the Pico4Drive board, which distributes the high-current energy to the DC motors and the electromagnet according to the PWM commands from the microcontroller. Additionally, the expansion board handles voltage regulation, stepping down the battery voltage to securely supply 5 V and 3.3 V to the Raspberry Pi Pico W and the low-power sensors.

3.5 System Block Diagram

Figure 3.4 presents the block diagram of the system; it illustrates that the system is divided into two main data flows: the action stack, which transforms the desired motion of the robot into voltage for the motors, and the sensing stack, which processes feedback from the robot's sensors to estimate its current state [34].

A characteristic of this cascade architecture is the variation in the execution frequencies between the different layers. The low-level ones operate at higher frequencies than the upper-level layers. As detailed in standard digital control theory [17], the PI block must execute continuously

⁵Product page available at: <https://www.botnroll.com/en/infrared/2586-tracker-sensor-infrared-line-tracking.html>

⁶Raspberry Pi Pico W Datasheet: <https://pip-assets.raspberrypi.com/categories/686-raspberrypi-pico-w/documents/RP-008312-DS-1-pico-w-datasheet.pdf>

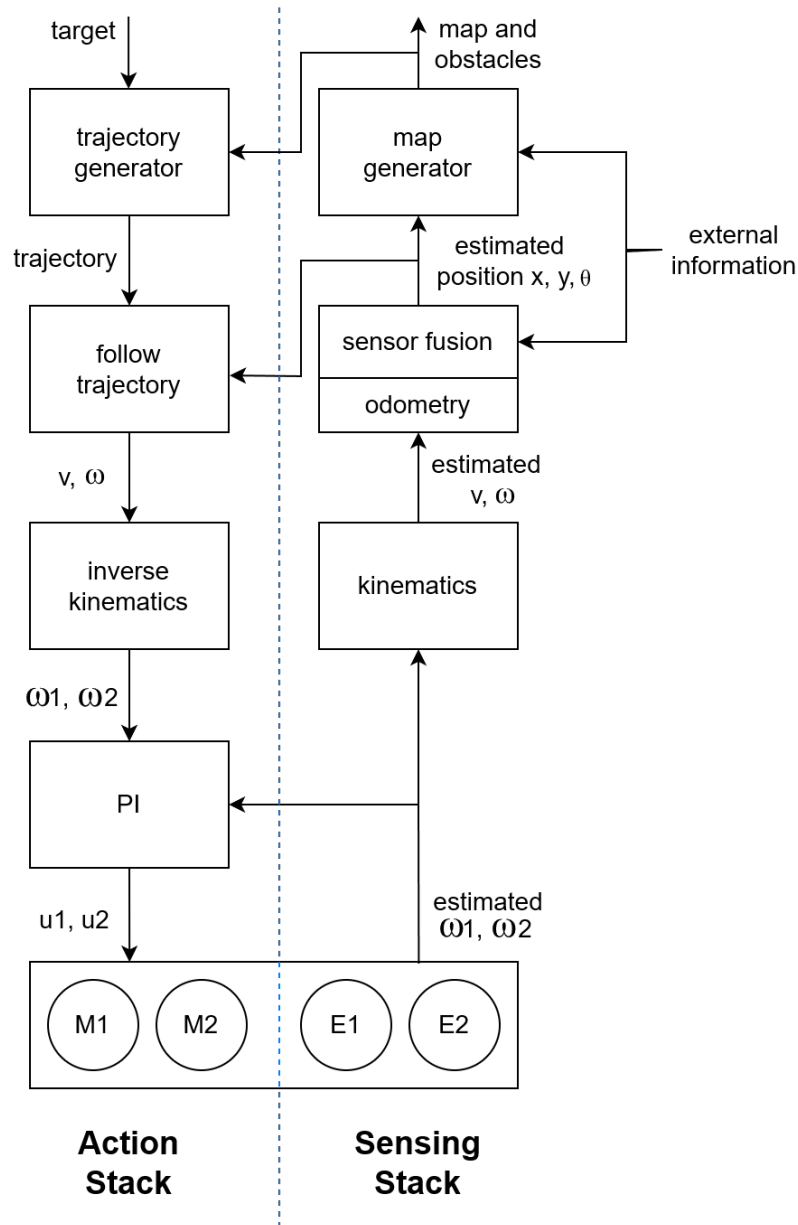


Figure 3.4: System block diagram.

to adjust the energy sent to the motors, reacting instantly to physical disturbances or friction. Conversely, the trajectory generator only needs to recalculate the trajectory periodically and rely on the fast, low-level controllers to handle the immediate, real-time execution of the required velocities.

Additionally, this hierarchical design ensures strict modularity and hardware abstraction, by dividing the system into blocks [34]. For example, if the current DC motors are replaced with a different model, the system's kinematics, odometry, and trajectory generation algorithms remain completely untouched. The only action needed is to retune the PI gains in the PI block. This design makes the system more adaptable and easier to upgrade.

3.6 Motor Model

To control the motion of the robot, it is necessary to analyse the dynamics of the DC motors responsible for driving the wheels. The model of the motor is divided into two coupled subsystems: the electrical circuit and the mechanical dynamics [29], as illustrated in Figure 3.5.

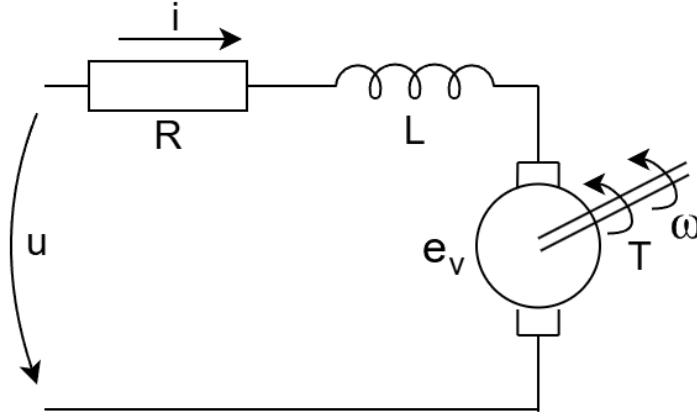


Figure 3.5: Motor model.

3.6.1 Electrical Subsystem

With respect to the electrical subsystem, the circuit is composed of a voltage source, a resistor, an inductor, and a voltage drop caused by the motor's rotation [10]. Thus, we obtain the differential equation 3.3.

$$u(t) = Ri(t) + L \frac{di(t)}{dt} + e_v(t) \quad (3.3)$$

Where:

- $u(t)$ is the applied voltage (V)
- $i(t)$ is the current (A)
- R is the resistance (Ω)
- L is the inductance (H)
- $e_v(t)$ is the back electromotive force (V)

3.6.2 Mechanical Subsystem

Concerning the mechanical dynamics, the differential equation 3.4 describes the rotational motion of the motor shaft [29].

$$\frac{d\omega(t)}{dt} = \frac{1}{J}(T_m(t) - B\omega(t) - T_q) \quad (3.4)$$

Where:

- $\omega(t)$ is the angular velocity of the motor shaft (rad/s)
- J is the equivalent moment of inertia of the rotor and the attached load ($\text{kg}\cdot\text{m}^2$)
- $T_m(t)$ is the mechanical torque generated by the motor (Nm)
- B is the viscous friction coefficient of the motor shaft ($\text{N}\cdot\text{m}\cdot\text{s}$)
- T_q is the static friction torque (Coulomb friction) opposing the motion (Nm)

3.6.3 Electromechanical Coupling

The electrical and mechanical subsystems are not independent, in fact they are coupled through magnetic interactions [10].

First, the mechanical torque produced by the motor (T_m) is directly proportional to the electrical current (i). Their relation can be written as:

$$T_m(t) = Ki(t) \quad (3.5)$$

Similarly, as the motor spins through the magnetic field, it generates a back electromotive force (e_v) that opposes the applied voltage. This force is proportional to the angular velocity of the shaft (ω), as shown in Equation 3.6.

$$e_v(t) = K\omega(t) \quad (3.6)$$

Where both K have the same value, although in Equation 3.5 it represents the motor torque constant ($\text{N}\cdot\text{m}/\text{A}$) and in Equation 3.6 it is the electromotive force constant ($\text{V}\cdot\text{s}/\text{rad}$) [29].

Replacing the parameters in equations 3.3 and 3.4, we obtain a system composed of the two differential equations 3.7 and 3.8.

$$u(t) = Ri(t) + L\frac{di(t)}{dt} + K\omega(t) \quad (3.7)$$

$$\frac{d\omega(t)}{dt} = \frac{1}{J}(Ki(t) - B\omega(t) - T_q) \quad (3.8)$$

3.6.4 Motor Dead Zone

In real-world applications, DC motors exhibit a non-linear behaviour at low input voltages known as the dead zone [19]. This phenomenon is directly caused by the static friction torque (T_q) of the mechanical assembly. To mathematically determine the input voltage threshold that overcomes this dead zone, we analyse the motor's steady-state behaviour [10].

In steady-state, the system is stabilised, meaning both current and angular velocity are constants ($\frac{di(t)}{dt} = 0$ and $\frac{d\omega(t)}{dt} = 0$). Applying this condition to the electromechanical model yields a system of two algebraic equations:

$$u = Ri + K\omega \quad (3.9)$$

$$0 = Ki - B\omega - T_q \quad (3.10)$$

To express the angular velocity as a function of the applied voltage, we isolate the current (i) in Equation 3.10 and substitute it into Equation 3.9:

$$i = \left(\frac{B\omega + T_q}{K} \right) \quad (3.11)$$

$$u = \frac{R}{K}(B\omega + T_q) + K\omega \quad (3.12)$$

By rearranging the terms to solve for ω , we obtain the steady-state angular velocity equation:

$$\omega = -\frac{RT_q}{RB + K^2} + \frac{K}{RB + K^2}u \quad (3.13)$$

The dead zone corresponds to the region where the applied voltage u is insufficient to produce motion, resulting in $\omega = 0$. By setting $\omega = 0$ in Equation 3.13, we can find the dead zone:

$$0 = -\frac{RT_q}{RB + K^2} + \frac{K}{RB + K^2}u \quad (3.14)$$

$$u_{DeadZone} = \frac{RT_q}{K} \quad (3.15)$$

Equation 3.15 demonstrates that the dead zone voltage is directly proportional to the static friction T_q and the resistance R , and inversely proportional to the motor torque constant K . If the applied voltage is $|u| \leq u_{DeadZone}$, the motor remains stationary, as illustrated in Figure 3.6.

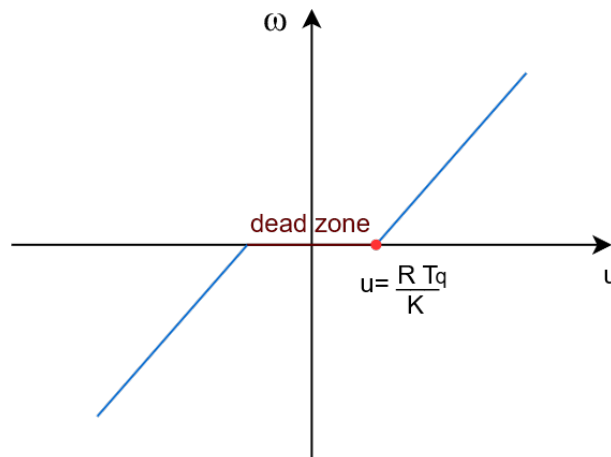


Figure 3.6: Dead zone.

Ideally, we would want to completely eliminate the dead zone; in other words, make the motor's angular velocity proportional to the voltage so that any non-zero control effort immediately results in motor rotation, simplifying the control. However, in practice, trying to eliminate the dead zone could cause a discontinuous relation between angular velocity and voltage, as shown in Figure 3.7. The velocity would instantly jump to a non-zero velocity at the smallest input (e.g., $u = 0.1$ V). This discontinuity introduces an infinite gain at the origin, leading to instability [19], especially when the robot is desired to move at low speeds.

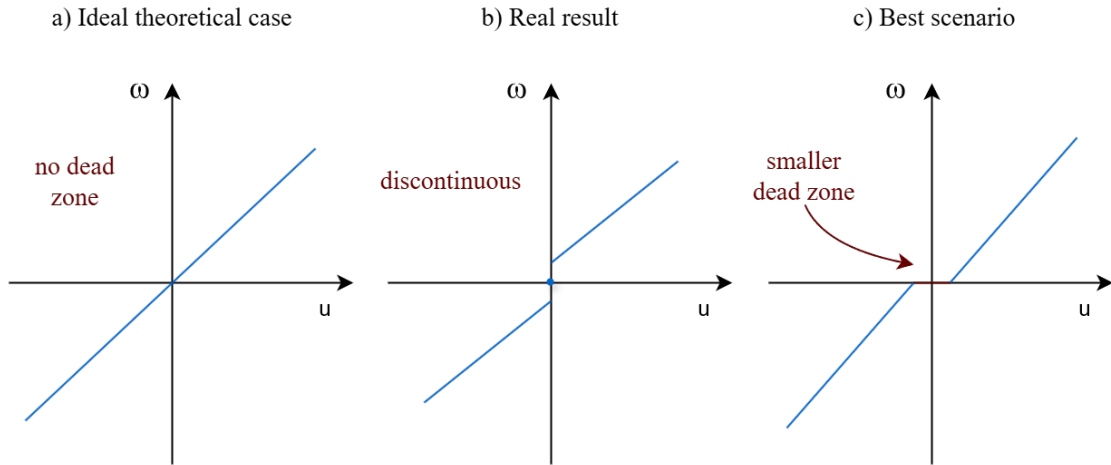


Figure 3.7: Dead zone compensation scenarios.

Instead of trying to completely eliminate the dead zone, a more stable approach is to add an extra voltage, $\Delta u(t)$, to the desired control signal, $u(t)$ [35]. This compensation voltage is designed to counteract the majority of the static friction torque (T_q), effectively reducing the dead zone to a small, linearisable region without causing instability at the origin.

By defining the total applied armature voltage as $u'(t) = u(t) + \Delta u(t)$, we can formulate the modified equations governing the compensated system. At this stage, we assume that the electrical time constant of the motor (L/R) is several orders of magnitude smaller than the mechanical time constant. Because the current reaches its desired value in milliseconds while the motor takes seconds to reach its target velocity, the inductance L can be safely neglected ($L \approx 0$) for the mechanical control analysis [29]. Therefore, the electrical equation becomes:

$$u(t) + \Delta u(t) = Ri(t) + K\omega(t) \quad (3.16)$$

Where:

- $u(t)$ is the nominal control voltage command (V)
- $\Delta u(t)$ is the dead zone compensation voltage applied to offset the static friction (V)

Making:

$$i(t) = \frac{u(t) + \Delta u(t) - K\omega(t)}{R} \quad (3.17)$$

Replacing in 3.8:

$$\frac{d\omega(t)}{dt} = \frac{1}{J} \left(\frac{K}{R} (u(t) + \Delta u(t) - K\omega(t)) - B\omega(t) - T_q \right) \quad (3.18)$$

$$\frac{d\omega(t)}{dt} = \frac{1}{J} \left(\frac{K}{R} u(t) - \left(\frac{K^2}{R} + B \right) \omega(t) - T_q + \frac{K}{R} \Delta u(t) \right) \quad (3.19)$$

As $\Delta u(t)$ was added to compensate the static friction torque (T_q), we can assume that $-T_q + \frac{K}{R} \Delta u(t) = 0$.

$$\frac{d\omega(t)}{dt} = \frac{1}{J} \left(\frac{K}{R} u(t) - \left(\frac{K^2}{R} + B \right) \omega(t) \right) \quad (3.20)$$

With this formulation, the dead zone is significantly reduced, facilitating the control of the system.

3.6.5 Motor Transfer Function

To design a linear controller for the motor's velocity, it is necessary to convert the system from the time domain to the frequency domain using the Laplace transform [30]. Taking the Laplace transform of the linearised mechanical equation (Equation 3.20), and assuming zero initial conditions ($\omega(0) = 0$), we obtain:

$$Js\omega(s) = \frac{K}{R}U(s) - \left(\frac{K^2}{R} + B \right) \omega(s) \quad (3.21)$$

To find the open-loop transfer function, $H(s)$, relating the angular velocity output $\omega(s)$ to the voltage input $U(s)$, we rearrange the terms:

$$\omega(s) \left(Js + \frac{K^2}{R} + B \right) = \frac{K}{R}U(s) \quad (3.22)$$

$$H(s) = \frac{\omega(s)}{U(s)} = \frac{\frac{K}{R}}{\frac{K^2}{R} + B + Js} \quad (3.23)$$

To express this transfer function in the standard first-order form, we can group the physical parameters to define the steady-state DC gain (A) and the mechanical time constant (τ) [30]:

$$H(s) = \frac{A}{\tau s + 1} \quad (3.24)$$

Analytically, these correspond to $A = \frac{K/R}{(K^2/R)+B}$ and $\tau = \frac{J}{(K^2/R)+B}$. However, because exact physical parameters like the moment of inertia (J) and viscous friction (B) are difficult to measure

directly in a fully assembled robot, A and τ were determined experimentally through an open-loop step response test, employing standard system identification techniques [22], as illustrated in Figure 3.8.

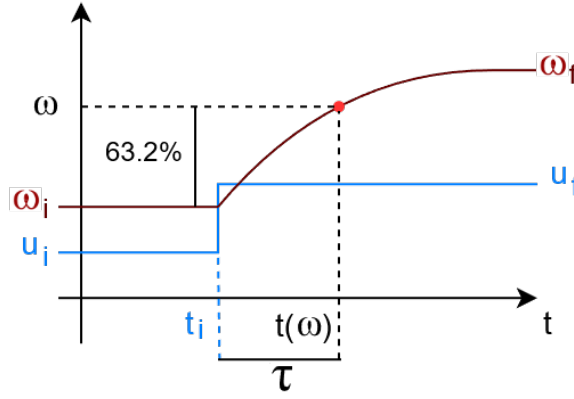


Figure 3.8: Experimental determination of the system's transfer function parameters.

By applying a step change in the input voltage from an initial steady-state u_i to a final voltage u_f , we observe the motor's angular velocity transition from its initial state ω_i to a new steady-state ω_f . The DC gain A is calculated as the ratio of the change in output velocity to the change in input voltage:

$$A = \frac{\omega_f - \omega_i}{u_f - u_i} = \frac{7.912 - 2.272}{3 - 1} = 2.820 \quad (3.25)$$

The mechanical time constant τ represents the time required for the system's response to reach 63.2% of its total steady-state change [30]. First, we determine the target angular velocity at this point, $\omega(\tau)$:

$$\omega(\tau) = 0.632(\omega_f - \omega_i) + \omega_i \quad (3.26)$$

$$\omega(\tau) = 0.632(7.912 - 2.272) + 2.272 = 5.825 \text{ rad/s} \quad (3.27)$$

Then, τ is calculated by finding the time difference between the moment the velocity reaches $\omega(\tau)$ and the moment the step input was applied (t_i):

$$\tau = t(\omega(\tau)) - t_i = 5.82 - 5.64 = 0.18 \text{ s} \quad (3.28)$$

With A and τ identified experimentally, the system's first-order transfer function is fully defined.

3.7 PI Controller Formulation

To accurately control the velocity of the motor and eliminate steady-state error, a Proportional-Integral (PI) controller is applied. [3] In the time domain, the PI control law is expressed as the sum of a proportional term and an integral term:

$$u(t) = K_c \left(e(t) + \frac{1}{T_i} \int_0^t e(\tau) d\tau \right) \quad (3.29)$$

Where:

- K_c is the proportional gain, which provides an immediate control effort proportional to the current error.
- e is the difference between the reference angular velocity and the measured velocity ($e(t) = \omega_r(t) - \omega(t)$).
- T_i is the integral time constant, which eliminates steady-state error by accumulating past errors over time.

To analyse the system's stability and design the controller, we convert this time-domain equation into the frequency domain using the Laplace transform. Assuming zero initial conditions, the Laplace transform of the PI control law is:

$$U(s) = K_c \left(E(s) + \frac{1}{T_i s} E(s) \right) \quad (3.30)$$

The transfer function of the PI controller, $G(s)$, which relates the control effort to the velocity error, can be written as:

$$G(s) = \frac{U(s)}{E(s)} = K_c \left(\frac{T_i s + 1}{T_i s} \right) \quad (3.31)$$

With the controller's transfer function defined, the closed-loop transfer function, represented in Figure 3.9, relating the actual angular velocity $\omega(s)$ to the reference angular velocity $\omega_r(s)$ is given by standard feedback control theory [30]:

$$\frac{\omega(s)}{\omega_r(s)} = \frac{G(s)H(s)}{1 + G(s)H(s)} \quad (3.32)$$

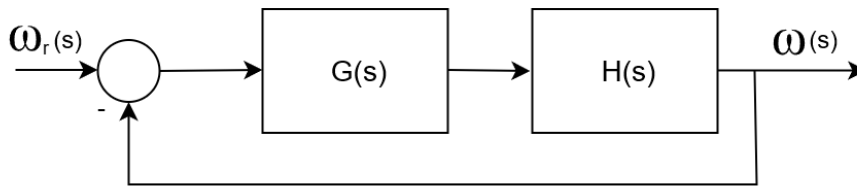


Figure 3.9: Closed-loop transfer function.

Replacing $G(s)$ and $H(s)$ using the equations 3.31 and 3.24, respectively. Applying the pole-zero cancellation method [3]. By setting the controller's integral time constant equal to the motor's mechanical time constant ($T_i = \tau$), the zero of the controller perfectly cancels the pole of the motor's transfer function.

$$\frac{\omega(s)}{\omega_r(s)} = \frac{\left(K_c \frac{\tau s + 1}{\tau s}\right) \left(\frac{A}{\tau s + 1}\right)}{1 + \left(K_c \frac{\tau s + 1}{\tau s}\right) \left(\frac{A}{\tau s + 1}\right)} \quad (3.33)$$

By cancelling the $(\tau s + 1)$ terms, the equation simplifies to:

$$\frac{\omega(s)}{\omega_r(s)} = \frac{\frac{K_c A}{\tau s}}{1 + \frac{K_c A}{\tau s}} = \frac{1}{\left(\frac{\tau}{K_c A}\right) s + 1} \quad (3.34)$$

This represents a first-order closed-loop response with a new tunable closed-loop time constant, τ_c . By defining this desired closed-loop time constant as:

$$\tau_c = \frac{\tau}{K_c A} \quad (3.35)$$

We can rearrange the equation to calculate the required proportional gain, K_c , needed to achieve the desired dynamic response of the robot:

$$K_c = \frac{\tau}{\tau_c A} = \frac{0.18}{0.20 \cdot 2.820} = 0.319 \quad (3.36)$$

The value of τ_c is chosen not to achieve a faster system but rather a more stable one, with higher immunity to disturbances [3].

This allows us to tune the PI controller, where T_i is matched to the measured motor dynamics, and K_c is adjusted based on how aggressively the system is required to track the reference velocity.

3.8 Kinematic Transformations

As previously established in Section 2.3.1, the relationship between the robot's central velocities (v, ω) and its individual wheels is governed by its differential drive geometry [34]. While the theoretical model describes linear wheel speeds, the digital control architecture requires angular velocities to interface with the motor encoders and PID controllers.

3.8.1 Inverse Kinematics

The Inverse Kinematics block acts as the bridge between the high-level trajectory generator and the low-level motor controllers. It receives the target linear velocity (v) and angular velocity (ω) of the robot and computes the necessary angular speeds for the right and left motors, ω_1 and ω_2 , respectively.

Knowing that the linear speed of a wheel is the product of its angular velocity and its radius ($v_{wheel} = r \cdot \omega_{wheel}$), we can rearrange the fundamental kinematic equations from Section 2.3.1 to solve for the individual angular velocities [34]:

$$\omega_1 = \frac{1}{r} \left(v + \frac{\omega L}{2} \right) \quad (3.37)$$

$$\omega_2 = \frac{1}{r} \left(v - \frac{\omega L}{2} \right) \quad (3.38)$$

Where r is the wheel radius and L is the distance between the wheels' contact points, as illustrated in Figure 2.2. These calculated reference values (ω_1, ω_2) are then fed directly into the PI controller.

3.8.2 Forward Kinematics

On the other hand, the Kinematics block transforms the wheels' angular velocities ($\hat{\omega}_1$ and $\hat{\omega}_2$), measured with the encoders, into the estimated linear velocity ($\hat{v}$) and angular velocity ($\hat{\omega}$) of the robot.

To relate these specific wheel rotations to the movement of the entire robotic platform, we used the fundamental kinematic equations from Section 2.3.1:

$$\hat{v} = \frac{r(\hat{\omega}_1 + \hat{\omega}_2)}{2} \quad (3.39)$$

$$\hat{\omega} = \frac{r(\hat{\omega}_1 - \hat{\omega}_2)}{L} \quad (3.40)$$

These estimated velocities ($\hat{v}, \hat{\omega}$) are then passed to the odometry block, where they are integrated over the sampling time to update the robot's estimated position (x, y, θ) [34].

3.9 Odometry

The odometry block is responsible for estimating the position (x, y) and orientation (θ) of the robot in a fixed 2D coordinate system. This estimation is achieved by integrating the estimated linear and angular velocities ($\hat{v}, \hat{\omega}$) provided by the Kinematics block.

In the continuous-time domain, the rate of change of the robot's position and orientation is described by the following differential equations [34]:

$$\frac{dx(t)}{dt} = \hat{v}_x = \hat{v}(t) \cos(\theta(t)) \quad (3.41)$$

$$\frac{dy(t)}{dt} = \hat{v}_y = \hat{v}(t) \sin(\theta(t)) \quad (3.42)$$

$$\frac{d\theta(t)}{dt} = \hat{\omega}(t) \quad (3.43)$$

Because the control system is implemented digitally on a microcontroller, these continuous integrals cannot be computed exactly. Instead, the system operates in discrete time steps, Δt , using the standard forward Euler approximation [36].

$$\frac{\Delta x}{\Delta t} = \hat{v}_k \cos(\theta_k) \quad (3.44)$$

$$\frac{\Delta y}{\Delta t} = \hat{v}_k \sin(\theta_k) \quad (3.45)$$

$$\frac{\Delta \theta}{\Delta t} = \hat{\omega}_k \quad (3.46)$$

Knowing that $\Delta x = x_{k+1} - x_k$, we can obtain the basic discrete-time update equations for the robot's pose at step $k + 1$:

$$x_{k+1} = x_k + \hat{v}_k \Delta t \cos(\theta_k) \quad (3.47)$$

$$y_{k+1} = y_k + \hat{v}_k \Delta t \sin(\theta_k) \quad (3.48)$$

$$\theta_{k+1} = \theta_k + \hat{\omega}_k \Delta t \quad (3.49)$$

While the standard Euler method is computationally simple, it assumes that the robot's heading (θ) remains constant throughout the entire integration step Δt . In reality, the heading changes continuously while the robot moves. This discrepancy introduces a systematic accumulation of integration error over time, reducing the accuracy of the odometry.

To mitigate this error, the software implements the midpoint difference method (a second-order Runge-Kutta approximation) [36]. Instead of using the orientation at the beginning of the step (θ_k), the algorithm calculates the robot's heading at the midpoint of the sample interval ($\theta_{k+\frac{1}{2}}$):

$$\theta_{k+\frac{1}{2}} = \theta_k + \frac{\Delta t}{2} \hat{\omega}_k \quad (3.50)$$

Using this improved heading estimate, the final discrete-time odometry equations executed by the control system become:

$$x_{k+1} = x_k + \hat{v}_k \Delta t \cos(\theta_{k+\frac{1}{2}}) \quad (3.51)$$

$$y_{k+1} = y_k + \hat{v}_k \Delta t \sin(\theta_{k+\frac{1}{2}}) \quad (3.52)$$

$$\theta_{k+1} = \theta_k + \hat{\omega}_k \Delta t \quad (3.53)$$

This modification significantly increases the precision of trajectory tracking without adding significant computational overhead to the microprocessor.

The practical implementation of this logic is presented in Listing 3.1.

```

1 void robot_t::odometry(void) {
2     // Estimate wheels speed using the encoders
3     w1e = enc1 * (TWO_PI / (64.0 * 2.0 * 1920.0));
4     w2e = enc2 * (TWO_PI / (64.0 * 2.0 * 1920.0));
5
6     v1e = w1e * wheel_radius;
7     v2e = w2e * wheel_radius;
8
9     // Estimate robot speed (Forward Kinematics)
10    ve = (v1e + v2e) / 2.0;
11    we = (v1e - v2e) / wheel_dist;
12
13    // Estimate the distance and the turn angle
14    ds = ve * dt;
15    dtheta = we * dt;
16
17    // Save previous value
18    prev_xe = xe;
19    prev_ye = ye;
20    prev_thetae = thetai;
21
22    // Estimate pose
23    xe = xe + ds * cos(thetae + dtheta/2);
24    ye = ye + ds * sin(thetae + dtheta/2);
25    thetai = thetai + dtheta;
26
27    // Relative displacement
28    rel_s += ds;
29    rel_theta += dtheta;
30 }

```

Listing 3.1: Odometry Implementation with Midpoint Integration

3.10 Infrared Line Sensor

To participate in the "Robot at Factory Lite" challenges, the platform must be capable of tracking the black lines on the arena floor. This perception capability is achieved through a downward-facing, 5-channel ITR20001/T infrared line sensor array⁷, shown from both perspectives in Figure 3.10. The sensor emits infrared light and measures the reflected intensity using phototransistors. Since dark surfaces absorb infrared radiation more than reflective white backgrounds, the analogue voltage readings directly correlate with line proximity.

The software architecture programming this sensor is encapsulated in the `IRLine_t` class, which handles ambient noise calibration, multi-modal line positioning, intersection tracking, and adaptive velocity regulation.

3.10.1 Signal Pre-processing

The sensor array consists of $N = 5$ channels, indexed from $i = 0$ to 4, arranged linearly with a fixed physical inter-sensor pitch (d) of 16.0 mm. To ensure robustness against ambient lighting

⁷Product page available at: <https://www.botnroll.com/en/infrared/2586-tracker-sensor-infrared-line-tracking.html>



(a) Top view showing the interface connector.

(b) Bottom view showing the 5 infrared channels.

Figure 3.10: 5-channel ITR20001/T infrared line sensor.

changes and physical surface variances, the raw analogue readings $IR_values[i]$ are filtered using a baseline noise threshold, or water level ($IR_WaterLevel = 50$). The pre-processed intensity for each channel, v_i , is defined as:

$$v_i = \max(0, IR_values[i] - IR_WaterLevel) \quad (3.54)$$

to classify whether a specific channel is actively positioned over the black line, a second threshold ($IR_tresh = 400$) is used:

$$Active(i) = \begin{cases} \text{true} & \text{if } IR_values[i] > IR_tresh \\ \text{false} & \text{otherwise} \end{cases} \quad (3.55)$$

If any sensor is active, the system updates its internal state variable ($seing_line = true$) and continually refreshes a high-resolution safety timer ($last_black_timer$), which marks the exact moment the line was last visible.

3.10.2 Position and Edge Estimation

To convert the discrete channel readings into a continuous lateral error signal to control the angular velocity, the class implements two distinct positioning algorithms: a weighted average and a linear edge interpolation.

The continuous centre estimation calculates the line's centre-of-mass relative to the physical centre of the robot (aligned with the middle sensor, $i = 2$). The total active intensity IR_{total} and the weighted spatial sum sum_{pos} are accumulated across all channels:

$$IR_{total} = \sum_{i=0}^4 v_i \quad (3.56)$$

$$sum_{pos} = \sum_{i=0}^4 v_i \cdot (i - 2) \cdot d \quad (3.57)$$

When the total intensity surpasses the activation barrier ($IR_{total} > IR_tresh$), the continuous centre position pos_{centre} is computed as:

$$pos_{\text{centre}} = \frac{sum_{\text{pos}}}{IR_{\text{total}}} \quad (3.58)$$

This approach provides sub-millimetre tracking accuracy across the discrete array. Alternatively, it is possible to compute the digital centre by treating v_i as a purely binary state (0 or 1) relative to the threshold (IR_{tresh}), calculating the centre as:

$$pos_{\text{centre_digital}} = \left(\frac{\sum_{i=0}^4 \text{Active}(i) \cdot (i+1) \cdot d}{\sum_{i=0}^4 \text{Active}(i)} \right) - 3d \quad (3.59)$$

For specialised manoeuvres or alignment verifications, the boundaries of the line are determined by searching for signal step transitions larger than the threshold IR_{tresh} . The array is scanned sequentially. When a transition from a below-threshold reading ($last_v$) to an above-threshold reading (v) is detected between adjacent sensors, linear interpolation is applied to determine the precise edge coordinate. Using the left side as an example (sensor $i = 3$ as a reference):

$$pos_{\text{left}} = -\frac{w_{\text{line}}}{2} + d \cdot (i-3) + d \cdot \left(\frac{IR_{\text{tresh}} - last_v}{v - last_v} \right) \quad (3.60)$$

where w_{line} is the nominal line width. A symmetric backward-scanning loop is used to compute the continuous right edge coordinate (pos_{right}).

3.10.2.1 Experimental Validation

To validate the accuracy and physical consistency of the positioning algorithms, an experiment was conducted in which the robot was rotated in place over a straight line, as illustrated in Figure 3.11a. During this rotation, the orientation angle θ (in radians) and the real estimated positions (pos_{centre} , pos_{left} , pos_{right}) were recorded simultaneously.

The expected paths are based on basic geometry. As the robot rotates, a sensor mounted at the front sweeps across the straight line. By using the distance ($L = 70\text{mm}$) from the robot's centre to the sensor, we can use trigonometry to predict where the line should appear on the sensor array. Because the sensor measures sideways distances at a right angle to the robot's body, the centre of the line follows a tangent curve. By adding the actual width of the line ($W_{\text{line}} = 24\text{mm}$), we obtain the full set of formulas to calculate the expected positions:

$$pos_{\text{centre, expected}} = L \cdot \tan(\theta) \quad (3.61)$$

$$pos_{\text{left, expected}} = L \cdot \tan(\theta) - \frac{W_{\text{line}}}{2 \cdot \cos(\theta)} \quad (3.62)$$

$$pos_{\text{right, expected}} = L \cdot \tan(\theta) + \frac{W_{\text{line}}}{2 \cdot \cos(\theta)} \quad (3.63)$$

The second term in the edge equations uses a secant modifier ($\frac{1}{\cos \Delta \theta}$) to account for the apparent broadening of the line's width when sliced diagonally by the sensor array at sharp angles.

The collected experimental data were plotted against these theoretical equations across the rotation sweep. The results for the centre, left edge, and right edge estimations are illustrated in Figures 3.11b, 3.12a, and 3.12b, and the quantitative error metrics are compiled in Table 3.4.

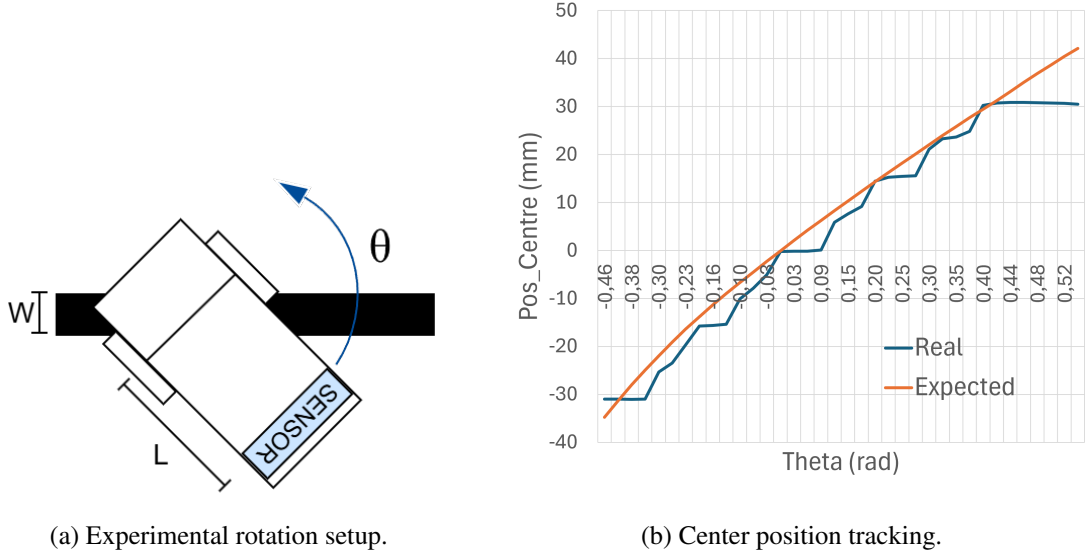


Figure 3.11: Experimental setup diagram alongside the real vs. expected line centre position tracking.

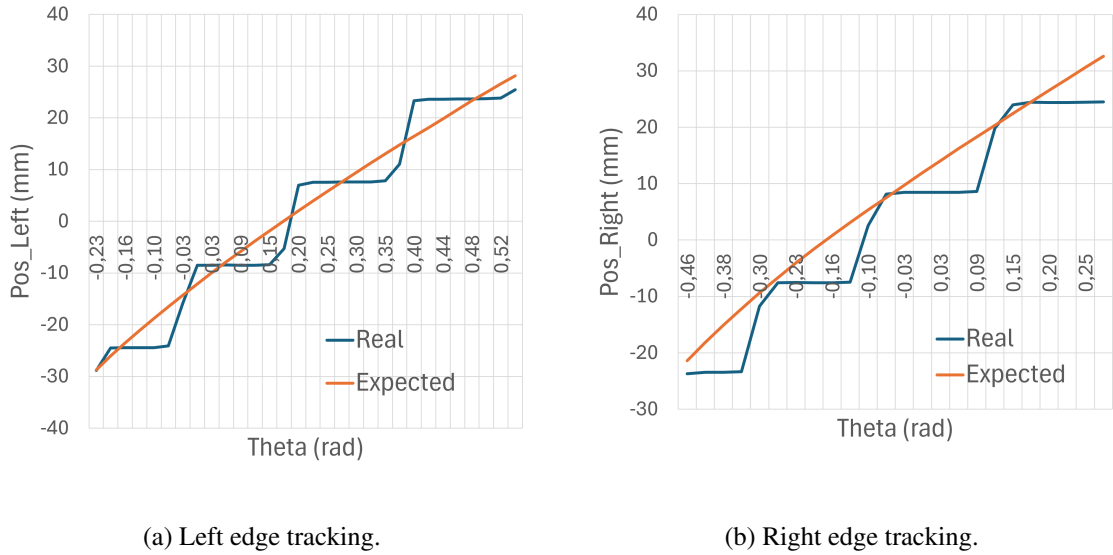


Figure 3.12: Comparison between real and expected edge tracking measurements during the rotation sweep.

The real sensor data closely track the general curves, slopes, and zero-crossing points of our mathematical formulas, validating the accuracy of the geometric model. The small differences shown in Table 3.4 are simply caused by two physical limitations of the robot's hardware.

First, the sensor array has a limited physical width and cannot detect positions beyond roughly $\pm 32 \text{ mm}$ ($\frac{16 \text{ mm} \times 4}{2}$). When the robot rotates to steeper angles, the black line moves entirely past the

Table 3.4: Quantitative error metrics between experimental tracking data and geometric models.

Metric (mm)	Centre	Left Edge	Right Edge
Average Error	3.488	3.191	4.698
Max Error	11.624	7.602	11.182

outermost sensors. While the theoretical model assumes the array can track the line indefinitely, the real hardware caps its reading at a maximum limit of approximately ± 32 mm. This bounding effect creates the flat horizontal sections at the edges of the graphs and results in a peak maximum error of 11.624 mm at the centre.

Second, the measured tracking data feature a wavy, step-like pattern instead of a perfectly straight line. This is caused by sensor spacing; the physical infrared array consists of separate channels placed 16.0 mm apart. As the line slides from one sensor to the next, the interpolation algorithm creates minor, temporary plateaus. Because the ideal geometric model passes smoothly through these discrete steps, it leaves an average error between 3.191 mm and 4.698 mm.

3.10.3 Intersection and Crossroad Detection

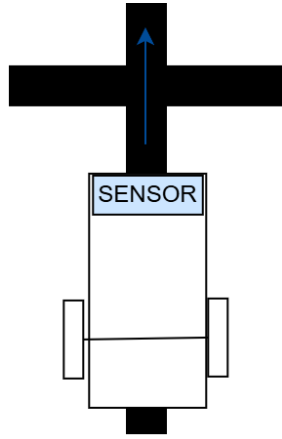
The layout of the "Robot at Factory Lite" arena presents perpendicular grid intersections (crosses) where multiple lines meet. When the robot faces an intersection, several sensors simultaneously report maximum dark intensities, making a standard centre-of-mass tracking loop unstable. To detect these scenarios, we can evaluate the ratio between the cumulative intensity and the maximum single channel intensity:

$$\text{blacks} = \frac{IR_{\text{total}}}{IR_{\text{max}}} \quad (3.64)$$

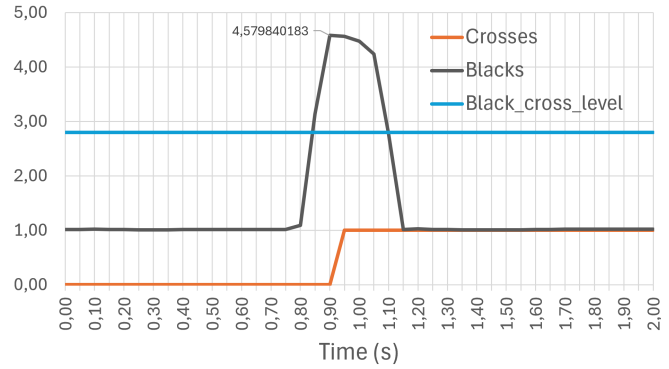
Under standard line-following conditions, only one or two sensors are highly active, keeping the ratio low. However, when a cross section is reached, IR_{total} increases dramatically while IR_{max} saturates at its peak hardware limit. If this ratio exceeds a defined value (`black_cross_level`), a counter (`crosses`) increments. To prevent false positives caused by surface noise or brief spikes, an intersection event is considered a cross only after the state persists for a predefined threshold (`cross_tresh = 3`) of consecutive cycles.

To verify that the intersection detection works correctly, an experiment was conducted by driving the robot straight over a perpendicular crossroad. The physical layout of this test is shown in Figure 3.13(a). During the run, internal state variables were logged continuously to observe how the intensity ratio behaves when encountering the intersection.

The resulting experimental timeline is plotted in Figure 3.13(b). While following a standard line, only a single sensor detects the line, keeping the calculated blacks ratio around 1.0. The moment the sensor array reaches the intersection, the ratio spikes sharply to a peak of approximately 4.58, easily clearing the activation threshold (`black_cross_level = 2.8`). After staying above



(a) Experimental setup diagram.



(b) Experimental sensor telemetry.

Figure 3.13: Intersection validation test showing the physical crossroad approach (a) alongside the real-time sensor ratio spike and the resulting step change in the permanent cross counter (b).

this line for the required 3 consecutive loop cycles to filter out background noise, the system successfully confirms the landmark. This triggers the counter (`crosses`) to step up from 0 to 1, where it remains even after the robot leaves the intersection and returns to standard line-following.

3.10.4 Reactive Control and Safety Fallbacks

To regulate the robot's target linear velocity (v_{req}) and angular velocity (w_{req}), based on the input received from the line sensor, the sequence in Figure 3.14 can be followed.

To minimise tracking overshoot and prevent losing the track at sharp corners, the reference linear velocity decays based on which channels see the line. The base velocity (V_{centre}) occurs when the robot is in the centre of the line. If the line drifts to the intermediate channels ($i = 1$ or 3), it is defined as a slower velocity (V_{side}). If the tracking error degrades further and triggers the edge channels ($i = 0$ or 4), the speed drops to an emergency manoeuvring rate (V_{edge}):

$$v_{\text{req}} = \begin{cases} \min(V_{\text{centre}}, V_{\text{edge}}) & \text{if } \max(\text{IR_values}[0], \text{IR_values}[4]) > \text{IR_tresh} \\ \min(V_{\text{centre}}, V_{\text{side}}) & \text{if } \max(\text{IR_values}[1], \text{IR_values}[3]) > \text{IR_tresh} \\ & \text{and edge sensors inactive} \\ V_{\text{centre}} & \text{otherwise} \end{cases} \quad (3.65)$$

The requested angular velocity w_{req} is derived using a multi-channel proportional gain loop. Thus, the controller multiplies each individual channel intensity directly by a unique gain factor ($\text{IR_W_gain}[i]$). This maps the exact cross-track distribution to an immediate rotational correction request:

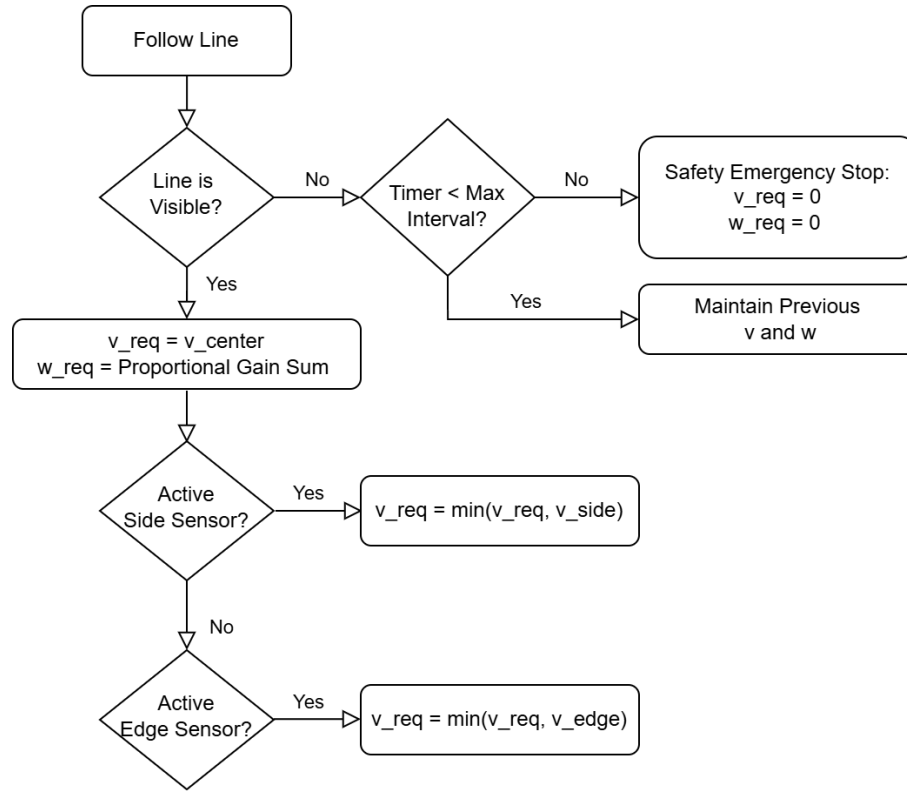


Figure 3.14: Line sensor flowchart.

$$w_{\text{req}} = 10^{-3} \cdot \sum_{i=0}^4 IR_W_gain[i] \cdot IR_values[i] \quad (3.66)$$

If the robot completely loses visual contact with the line (`seeing_line = false`), an immediate stop would be inappropriate during localised gap crossings or brief reflections. Thus, the controller executes a time-delayed safety strategy.

If the time that has passed since the last line is strictly less than a safety parameter (`no_line_interval`), the robot maintains its previous linear and angular velocities to pass smoothly through the blind spot. On the other hand, if the elapsed duration exceeds `no_line_interval` without finding the line tracking path, the microcontroller forces $v_{\text{req}} = 0$ and $w_{\text{req}} = 0$.

Finally, the computed v_{req} and w_{req} values are transmitted to the inverse kinematics block, which dictates the wheel motor commands.

The following track algorithm presented is a simple and discrete solution that can be used in educational robotics platforms, as students can easily define the desired linear velocity behaviours and understand basic logic branching without having to program complex implementations.

3.10.5 Continuous Path-Following Control

To achieve smooth tracking and eliminate the step oscillations of discrete threshold controllers, a continuous control strategy was implemented. This approach is based on the centre-of-mass esti-

mate (`IRLine.pos_center`), presented in section 3.10.2, which measures the lateral distance in millimetres from the track axis to the robot's geometric centre.

When the line path is successfully detected (`found_center = true`), the required angular velocity ω_{req} is defined by a proportional control loop. Its value is proportional to the lateral displacement error through a tracking gain factor K_{track} :

$$\omega_{\text{req}} = K_{\text{track}} \cdot pos_{\text{center}} \quad (3.67)$$

To maintain structural stability, the requested angular velocity is symmetrically bounded by an operational limit of ± 8.0 rad/s.

Instead of using fixed speed steps for different sensor zones, the robot's linear velocity changes smoothly depending on how fast it needs to turn. This is done using an inverted parabolic curve, as shown in Figure 3.15. As the required turning rate increases, this curve automatically slows down the robot's forward movement, which prevents it from overshooting and losing the track on sharp corners. Mathematically, this profile is calculated as follows:

$$v_{\text{req}} = -\frac{v_{\text{nom}}}{\omega_0^2} (\omega_{\text{req}} - \omega_0)(\omega_{\text{req}} + \omega_0) = v_{\text{nom}} \left(1 - \frac{\omega_{\text{req}}^2}{\omega_0^2} \right) \quad (3.68)$$

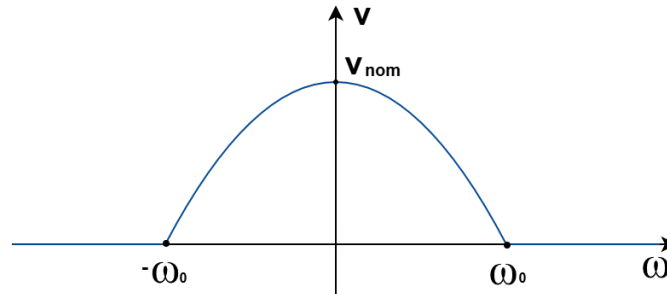


Figure 3.15: Non-linear velocity modulation profile, illustrating the parabolic decay of linear speed (v) as a function of angular speed (ω).

Where v_{nom} is the target forward speed when the robot is going perfectly straight, and ω_0 is the maximum turning speed limit at which the forward velocity would drop to zero. Besides that, the calculated speed is kept within a safe physical range between 0.0 and 0.4 m/s.

If the robot completely loses track of the line (`found_center = false`), a safety backup routine starts immediately. To stop the robot from driving blindly, its forward speed is immediately cut to zero ($v_{\text{req}} = 0.0$ m/s), and the robot begins to spin in place at a steady search speed ω_z (limited to ± 8.0 rad/s) to scan its surroundings and find the black line again.

The practical implementation of this control strategy inside the robot's firmware is managed by the execution function shown in Listing 3.2.

```

1 void action_t::follow_track(void)
2 {
3     if (robot.IRLine.found_center) {
4
5         // Angular velocity calculation and saturation bounding
6         float w_calc = ktrack * robot.IRLine.pos_center;
7
8         if (w_calc > 8.0) w_calc = 8.0;
9         if (w_calc < -8.0) w_calc = -8.0;
10
11         robot.w_req = w_calc;
12
13         // Parabolic linear velocity scaling
14         float v_calc = (-v_nom / (w0 * w0)) * (w_calc - w0) * (w_calc + w0);
15
16         if (v_calc < 0.0) v_calc = 0.0;
17         if (v_calc > 0.4) v_calc = 0.4;
18
19         robot.v_req = v_calc;
20
21     } else {
22         // Safety backup handler triggered when line contact is lost
23         robot.v_req = 0.0;
24
25         float w_search = wz;
26         if (w_search > 8.0) w_search = 8.0;
27         if (w_search < -8.0) w_search = -8.0;
28
29         robot.w_req = w_search;
30     }
31 }

```

Listing 3.2: C++ implementation of the continuous line-tracking and safety search routines within the firmware framework.

3.11 Telemetry and Debugging Interface

To test the performance and debug the robot, we used the *ComRobot* interface. This application can communicate both via USB and wirelessly, allowing the operator to monitor data and adjust settings while the robot is moving on the factory floor.

The *ComRobot* grants the user the possibility of reading internal values, such as encoder counts and sensor readings in real time. Similarly, the user can change variable values, including the position (x , y , and θ) and steering gain (K_θ), by writing directly in the robot's memory while it runs. This helps save time during experiments as it eliminates the need to recompile and re-upload the code when making changes.

Besides that, the application has a graphing tool that plots data on the screen as it is collected, which improves the visualisation of the robot's performance. The application can also save this data to a .txt file for later analysis.

The interface of *ComRobot* is shown in Figure 3.16, displaying the variables, input spaces, and the plotting environment.

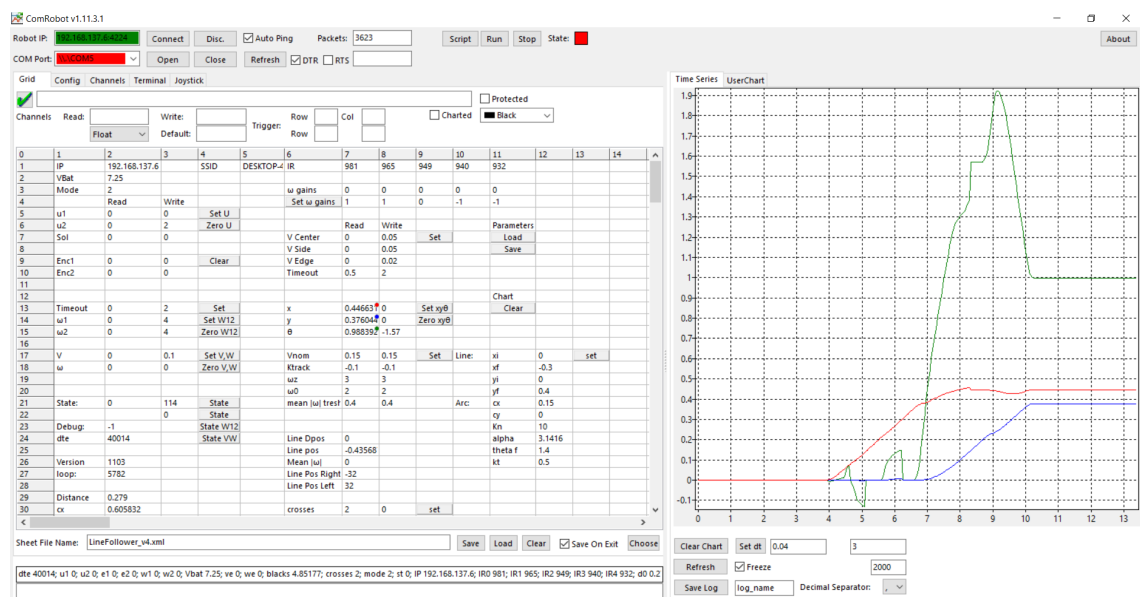


Figure 3.16: The *ComRobot* user interface showing real-time variables, configuration inputs, and the data plot.

Chapter 4

Autonomous Navigation and Control Strategies

This chapter presents the autonomous navigation algorithms and control strategies that allow the robot to move intelligently around the factory floor. Besides standard line following, the robot uses go-to-point and circular steering algorithms to move autonomously off the track, which is vital since its small size allows multiple robots to share the arena and safely overtake each other.

Thus, this chapter begins by introducing the track-based localisation system, explaining how the robot detects steady movement and uses mathematical transformations to correct its position estimates whenever it crosses known lines on the floor. Thereafter, the chapter details the trajectory tracking techniques used for basic movement, which include go-to-point, line following, and arc navigation. Finally, it presents the obstacle avoidance and overtaking methodology. This section outlines the state machine logic used to detect a blocked path, the algorithm used to generate a takeover, and the search routine used to safely re-acquire the track. Together, these control strategies provide the robot with the autonomy and flexibility needed to complete its tasks efficiently.

4.1 Track-Based Localization

While wheel odometry provides a continuous estimation of the position of the robot, integration errors accumulate over time, causing the calculated position to drift significantly from the real robot's coordinates [7, 36]. To compensate for this dead-reckoning drift without adding expensive external sensors, we used the physical lines of the track as absolute reference landmarks to periodically reset and correct the position of the robot.

This section describes the mathematical and algorithmic framework developed to handle these corrections. Starting by explaining the steady movement detection system, which ensures that pose corrections are only performed under stable driving conditions, and then describing the core correction algorithm, which uses homogeneous transformations to realign the odometry frame with the true track geometry.

4.1.1 Steady Movement Detection

Before applying any pose correction, it is necessary to identify whether the robot is stably travelling along a straight segment. A reliable metric to detect steady movement is calculating the average absolute angular speed over the n previous samples. This moving average, $ma_\omega(k)$, is mathematically defined as:

$$ma_\omega(k) = \frac{1}{n} \sum_{i=k-n+1}^k |\omega(i)| \quad (4.1)$$

If this value is below a predefined threshold, it indicates that the robot is travelling with a stable orientation. Based on the applied path-following controller, this stable state only occurs when the robot is aligned with a straight track segment. In the software implementation, this condition is reinforced by verifying that the difference between the robot's estimated heading (θ_e) and the segment's known angle (θ_s) is within a small margin.

4.1.2 Pose Correction using Homogeneous Transformations

To take advantage of all straight lines, whether horizontal, vertical, or inclined, a generalised mathematical approach is employed using 2D homogeneous transformations [34]. We define two coordinate frames: the global world frame $\{W\}$ and a local segment frame $\{S\}$. The frame $\{S\}$ originates at the starting point of the line segment (x_i, y_i) and is rotated by the segment's angle θ_s , such that its local x-axis points exactly along the line, as illustrated in Figure 4.1.

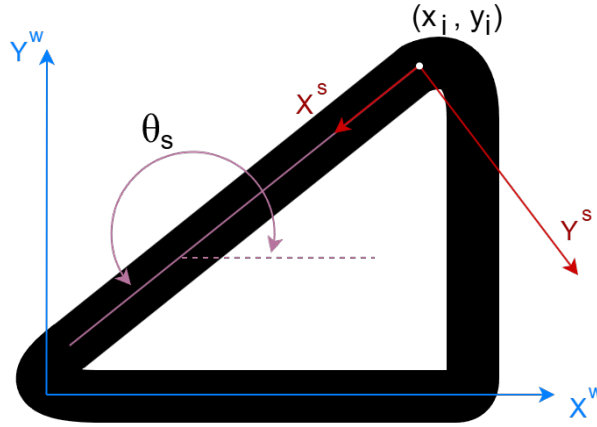


Figure 4.1: Track-based location reference frame.

The homogeneous transformation matrix $\mathbf{H}_{ws}$ that maps coordinates from the segment frame to the world frame is defined as:

$$\mathbf{H}_{ws} = \begin{bmatrix} \cos(\theta_s) & -\sin(\theta_s) & x_i \\ \sin(\theta_s) & \cos(\theta_s) & y_i \\ 0 & 0 & 1 \end{bmatrix} \quad (4.2)$$

When the steady movement conditions are met, the robot is known to be travelling on the line, meaning its lateral position relative to the line should be zero. The algorithm corrects the position of the robot through three matrix operations.

First, the world to segment transformation, when the robot's current estimated position in the world frame, $\mathbf{P}_w = [x_e, y_e, 1]^T$, is transformed into the local segment frame using the inverse matrix:

$$\mathbf{P}_s = \mathbf{H}_{ws}^{-1} \mathbf{P}_w = \begin{bmatrix} x_s \\ y_s \\ 1 \end{bmatrix} \quad (4.3)$$

Thereafter, we correct the lateral error. In the local frame $\{S\}$, the y-axis represents the lateral cross-track error. Because the robot is tracking the line, this error is forced to zero:

$$\mathbf{P}'_s = \begin{bmatrix} x_s \\ 0 \\ 1 \end{bmatrix} \quad (4.4)$$

Furthermore, the corrected local coordinates are projected back into the global world frame to obtain the new, corrected (x, y) position:

$$\mathbf{P}'_w = \mathbf{H}_{ws} \mathbf{P}'_s = \begin{bmatrix} x'_e \\ y'_e \\ 1 \end{bmatrix} \quad (4.5)$$

Finally, the robot's estimated heading is directly updated to perfectly match the segment's angle ($\theta'_e = \theta_s$). By implementing this generalised matrix formulation, the system inherently solves the coordinate correction for any line orientation without requiring separate logic states for horizontal or vertical cases.

The C++ implementation of this track localisation logic is presented in Listing 4.1.

```

1 void track_localization(robot_t& robot, float_list_t& w_list) {
2   robot.led = 0; // LED off
3   robot.align_index = -1;
4
5   // If mean |W| is below the threshold and decreasing, robot is
   // travelling along a straight segment
6   if (robot.mean_abs_w < robot.mean_abs_w_tresh && robot.mean_abs_w <
       robot.prev_mean_abs_w) {
7
8     // Iterate through all globally mapped track segments
9     for (int i = 0; i < segment_list.size(); i++) {
10       segment_t seg = segment_list[i];
11
12       // 1. Verify angular alignment with the segment
13       if (abs(dif_angle(normalize_angle(robot.thetae), seg.angle)) <
           robot.align_angle_tresh) {
14
15         // Create the homogenous transformation matrix (World to Segment)
16         htransf_2d_t Hws(seg.angle, seg.Pi.x, seg.Pi.y);

```

```

17
18     // Map current world coordinates into the local segment
referential
19     Vec2f Pw_current(robot.xe, robot.ye);
20     Vec2f Ps = Hws.apply_inv(Pw_current);
21
22     // Calculate physical segment length
23     float seg_length = sqrt(pow(seg.Pf.x - seg.Pi.x, 2) +
pow(seg.Pf.y - seg.Pi.y, 2));
24
25     // 2. Spatial Validation: Ensure robot is physically within
segment bounds
26     if (Ps.x >= -0.05 && Ps.x <= (seg_length + 0.05) && abs(Ps.y) <
0.15) {
27         robot.led = 1;
28         robot.align_index = i;
29         robot.thetae = seg.angle; // Snap heading
30
31         // 3. Pose Correction: Zero-out the cross-track lateral error
32         Ps.y = 0.0;
33
34         // Project the corrected local coordinates back to the world
frame
35         Vec2f Pw = Hws.apply(Ps);
36
37         // Update odometry
38         robot.xe = Pw.x;
39         robot.ye = Pw.y;
40
41         break; // Correction applied, exit loop
42     }
43 }
44 }
45 }
46 }

```

Listing 4.1: C++ implementation of the track-based localization algorithm.

The algorithm starts by evaluating if the robot is travelling along a straight segment (Line 6), ensuring the moving average of the angular velocity is both below the threshold and strictly decreasing, confirming a steady state.

Thereafter, the algorithm iterates through the known map geometry and filters candidates based on heading (line 13). Then, the algorithm uses the inverse homogeneous transformation (line 20) to calculate the position of the robot in the local frame of the candidate segment.

To prevent misclassification of the segment, because multiple segments may share the same geometric angle, the algorithm performs a strict spatial validation (line 26). The longitudinal distance ($P_s.x$) must fall within the physical endpoints of the segment (plus a marginal 5 cm tolerance for odometry lag), and the lateral cross-track error ($P_s.y$) must be within 15 cm. If these spatial constraints are satisfied, the algorithm updates the estimated heading to the angle of the segment, eliminates the lateral error, and projects the corrected coordinates back into the global odometry frame.

4.2 Trajectory Modes

Once the robot can accurately determine its position on the factory floor, it must be able to execute specific paths to reach its destinations. While the previous section focused on position estimation, this section shifts the focus to motion control. Although line following is the main logic used to guide the robot, introducing trajectory tracking algorithms [26] would allow the robot to temporarily leave the line, improving its efficiency; for example, when it encounters an obstacle.

The tracking subsystem is responsible for computing the precise linear (v) and angular (ω) velocities required to follow a desired path [26]. Depending on the navigation task, the robot can rely on three distinct trajectory modes, as illustrated in Figure 4.2: Go-To-Point, Line Following, and Arc Following [34]. The mathematical formulation and control law designed for each mode are described in the following sections.

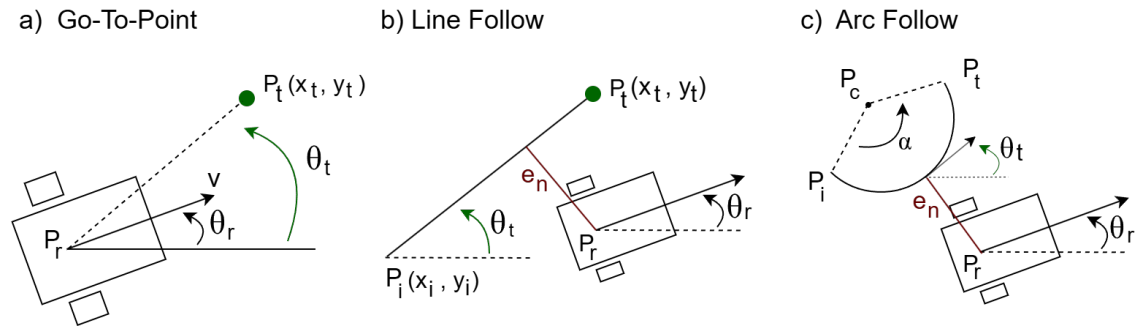


Figure 4.2: Trajectory modes: a) Go-To-Point, b) Line Follow, c) Arc Follow.

4.2.1 Go-To-Point Navigation (*gotoxy*)

The simplest form of navigation commands the robot to move from its current estimated position (x_r, y_r) to a target coordinate (x_t, y_t) . First, the target heading angle (θ_t) is calculated using the four-quadrant inverse tangent function:

$$\theta_t = \text{atan2}(y_t - y_r, x_t - x_r) \quad (4.6)$$

The angular error (e_θ) is the difference between this target angle and the robot's current orientation (θ_r). A proportional controller generates the angular velocity to correct this error:

$$e_\theta = \theta_t - \theta_r \quad (4.7)$$

$$\omega = K_\theta \cdot e_\theta \quad (4.8)$$

For linear velocity, maintaining a constant nominal speed (v_{nom}) often causes the robot to accumulate transversal errors because it moves forward before correcting its heading. Conversely, forcing the robot to stop, rotate, and then move forward creates inefficient motion with constant stops. To solve this, an intermediate velocity profile is applied. The linear velocity is defined as an inverted parabola dependent on the angular error, illustrated in Figure 4.3:

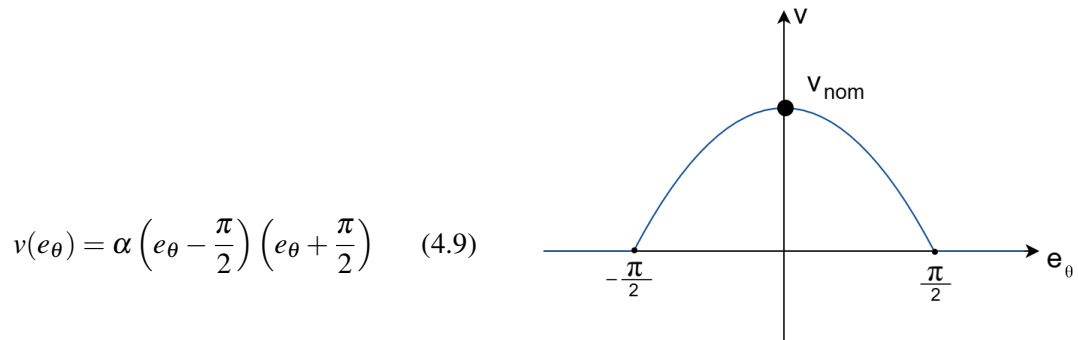


Figure 4.3: Relation between linear velocity and angular error.

Where α is a constant designed to ensure the robot reaches its maximum nominal velocity (v_{nom}) when the error is zero:

$$v(0) = \alpha \left(-\frac{\pi}{2} \right) \left(\frac{\pi}{2} \right) = v_{nom} \quad (4.10)$$

$$\alpha = \frac{-v_{nom}}{\left(\frac{\pi}{2} \right)^2} \quad (4.11)$$

If the angular error magnitude exceeds $\frac{\pi}{2}$ radians (90°), the linear velocity is set to zero ($v = 0$), and the robot rotates in place. As the robot aligns with the target, it smoothly accelerates forward.

While effective for reaching a destination, the limitation of this method is that it does not constrain the robot to a specific path, nor does it control the final arrival orientation.

Listing 4.2 presents the software implementation of this control strategy on the robot's embedded microcontroller. Notably, the code incorporates an explicit angle normalisation function to prevent wrapping issues and correctly computes the parabolic decay of the translation speed.

```

1 void action_t::goto_xy(void)
2 {
3     // 1. Calculate Euclidean distance to the target
4     e_xy = dist(Pf.x, Pf.y, robot.xe, robot.ye);
5
6     // 2. Calculate the target heading angle (desired theta)
7     thetaf = atan2(Pf.y - robot.ye, Pf.x - robot.xe);
8
9     // 3. Compute normalized heading error to find the shortest angular path
10    e_theta = normalize_angle(thetaf) - normalize_angle(robot.thetae);
11    e_theta = normalize_angle(e_theta);
12
13    // 4. Proportional angular velocity control law
14    robot.w_req = ktheta * e_theta;
15
16    // 5. Linear velocity modulation using the parabolic profile
17    if (abs(e_theta) > (PI / 2.0)) {
18        // If the error exceeds 90 degrees, execute a pure rotation-in-place
19        robot.v_req = 0;
20    } else {
21        // Apply inverted parabola profile bounded by the nominal velocity

```

```

22     alpha = -v_nom / pow(PI / 2.0, 2);
23     robot.v_req = alpha * (e_theta - PI / 2.0) * (e_theta + PI / 2.0);
24 }
25
26 // 6. Target arrival check (1 cm tolerance)
27 if (e_xy < 0.01) {
28     robot.v_req = 0;
29     robot.w_req = 0;
30 }
31 }

```

Listing 4.2: C++ implementation of the Go-To-Point navigation algorithm.

4.2.2 Line Follow Navigation

To constrain the robot's movement along a specific straight path, it is necessary to use a Line Follow algorithm. This method maintains the angular correction but introduces a cross-track error (e_n) term to correct lateral deviations from the path:

$$\omega = K_\theta \cdot e_\theta + K_n \cdot e_n \quad (4.12)$$

The cross-track error e_n is defined as the shortest perpendicular distance from the robot's centre $P_r(x_r, y_r)$ to the theoretical line path. Defining the line by its starting point $P_i(x_i, y_i)$ and target point $P_t(x_t, y_t)$, the cross-track error is calculated geometrically as:

$$e_n = \frac{(y_t - y_i)x_r - (x_t - x_i)y_r + x_t y_i - y_t x_i}{\sqrt{(y_t - y_i)^2 + (x_t - x_i)^2}} \quad (4.13)$$

To determine the control gains (K_θ and K_n), we started by analysing the model of the robot when the target θ is zero:

$$\dot{x} = v_{nom} \cos \theta \quad (4.14)$$

$$\dot{y} = v_{nom} \sin \theta \approx v_{nom} \theta \quad (4.15)$$

$$\dot{\theta} = \omega = K_\theta \cdot (-\theta) + K_n \cdot (-y) \quad (4.16)$$

By applying the Laplace transform, we obtained:

$$sY(s) = v_{nom}\theta(s) \quad (4.17)$$

$$s\theta(s) = -K_\theta \cdot \theta(s) - K_n \cdot Y(s) \quad (4.18)$$

Combining the equations, we obtain:

$$(s^2 + K_\theta s + K_n v_{nom})\theta(s) = 0 \quad (4.19)$$

To ensure critical damping without oscillations, we define two identical negative poles located at $s = -P$ (where $P > 0$). Solving the characteristic equation yields the optimal gain relations:

$$K_{\theta} = 2P \quad (4.20)$$

$$K_n = \frac{P^2}{v_{nom}} = \frac{\frac{K_{\theta}^2}{2}}{v_{nom}} \quad (4.21)$$

This mathematical formulation simplifies the tuning process, allowing the tracking system to be tuned using a single parameter (P , or equivalently K_{θ}) based on the nominal velocity [33].

The line-following algorithm is implemented on the embedded microcontroller, as shown in Listing 4.3.

```

1 void action_t::follow_line(void)
2 {
3     // 1. Calculate Euclidean distance to the final target
4     e_xy = dist(Pf.x, Pf.y, robot.xe, robot.ye);
5
6     // 2. Calculate line vector heading angle (path direction Pi -> Pf)
7     thetaf = atan2(Pf.y - Pi.y, Pf.x - Pi.x);
8
9     // 3. Compute normalized heading error
10    e_theta = normalize_angle(thetaf) - normalize_angle(robot.thetae);
11    e_theta = normalize_angle(e_theta);
12
13    // 4. Compute Cross-Track Error (e_n) using perpendicular point-to-line
        formula
14    float num = (Pf.y - Pi.y) * robot.xe - (Pf.x - Pi.x) * robot.ye + Pf.x
        * Pi.y - Pf.y * Pi.x;
15    float den = sqrt(pow(Pf.y - Pi.y, 2) + pow(Pf.x - Pi.x, 2));
16    float e_n = num / den;
17
18    // 5. Linear and angular velocity computation loops
19    if (abs(e_theta) > (PI / 2.0)) {
20        // Suppress forward translation and apply maximum nominal damping for
        pure spin recovery
21        robot.v_req = 0;
22        float ktheta_spin = 2.0 * sqrt(kn * v_nom);
23        robot.w_req = ktheta_spin * e_theta;
24    } else {
25        // Apply parabolic linear velocity profile
26        alpha = -v_nom / pow(PI / 2.0, 2);
27        robot.v_req = alpha * (e_theta - PI / 2.0) * (e_theta + PI / 2.0);
28
29        // Dynamically update K_theta based on current v_req to maintain
        critical damping
30        float v_for_damping;
31        if (robot.v_req > 0.01)
32            v_for_damping = robot.v_req;
33        else
34            v_for_damping = 0.01;
35        ktheta = 2.0 * sqrt(kn * v_for_damping);
36
37        // Combined path-following control law
38        robot.w_req = ktheta * e_theta + kn * e_n;
39    }
40
41    // 6. Vector projection math for the "Over-the-line" boundary check
42    float path_x = Pf.x - Pi.x;

```

```

43 float path_y = Pf.y - Pi.y;
44 float rob_to_target_x = Pf.x - robot.xe;
45 float rob_to_target_y = Pf.y - robot.ye;
46
47 // Dot product: flags if the robot has cross-projected past the target
48 // plane
49 float dot_product = (path_x * rob_to_target_x)
50 + (path_y * rob_to_target_y);
51
52 // 7. Robust target arrival termination check
53 if (e_xy < 0.01 || dot_product < 0) {
54     robot.v_req = 0;
55     robot.w_req = 0;
56 }

```

Listing 4.3: C++ implementation of the Line-Following navigation algorithm with dynamic damping adjustment.

4.2.3 Arc Follow Navigation

For curved trajectories, the robot follows a circular path defined by a centre point (x_c, y_c) , an initial point (x_i, y_i) , and an angle α . The control logic is similar to the Line Follow method, but the error definitions and the control laws are adapted for rotational geometry.

The cross-track error e_n is defined as the difference between the desired radius (R) and the robot's actual distance to the centre:

$$e_n = R - \sqrt{(x_r - x_c)^2 + (y_r - y_c)^2} \quad (4.22)$$

Where:

$$R = \sqrt{(x_i - x_c)^2 + (y_i - y_c)^2} \quad (4.23)$$

The target angle (θ_t) must be tangent to the circular path at the robot's current position. This is calculated using the angle from the centre to the robot, offset by 90 ($\frac{\pi}{2}$). The sign of the offset ($sign(\alpha)$) dictates the direction of travel (clockwise or counter-clockwise):

$$\theta_t = atan2(y_r - y_c, x_r - x_c) + sign(\alpha) \frac{\pi}{2} \quad (4.24)$$

While the linear velocity remains constant at v_{nom} , the angular velocity control law requires an additional term. Because the path is inherently curved, the robot must constantly rotate just to stay on the arc, even when the errors are zero. Thus, the base angular velocity $\frac{v_{nom}}{R}$ is added to the feedback controllers:

$$e_\theta = \theta_t - \theta_r \quad (4.25)$$

$$v = v_{nom} \quad (4.26)$$

$$\omega = K_\theta \cdot e_\theta + K_n \cdot e_n + \frac{v_{nom}}{R} \quad (4.27)$$

This term reduces steady-state tracking errors during sharp manoeuvres, ensuring the robot accurately traces the curve [33].

The software implementation of the arc-following controller is detailed in Listing 4.4. This controller adapts the cross-track and heading error formulations for curved path tracking, where the signs of the target arc angle (α) and the nominal velocity (v_{nom}) support both clockwise and counter-clockwise as well as forward and reverse manoeuvres.

To ensure that the robot completes the path, a hybrid arrival system was implemented to handle the physical limitations of discrete control loops. Initially, using a purely position-based spatial target failed when the requested arc formed a near-complete circle. Because the start and end points were physically close, the robot would immediately assume it had already arrived and refuse to move at all. Conversely, relying solely on the absolute accumulated angle proved unreliable; because the control loop runs at discrete time steps, the robot could easily overshoot the exact target angle between ticks, missing the trigger entirely and entering an infinite circling loop.

The hybrid solution resolves both failure modes by combining directional angular progress with a physical distance guard. The algorithm tracks accumulated progress strictly in the requested direction within a finite angular capture window, which prevents the target from being lost between loop cycles while ignoring wrong-way turns. Additionally, a kinematic condition requires the robot to travel a minimum Euclidean distance of 5 cm away from its starting position before the arrival variable is allowed to become true. This dual-constraint approach effectively filters out initialisation noise and ensures that the robot finishes the desired path.

```

1 void action_t::follow_arc(void)
2 {
3     // 1. Calculate the nominal arc radius from the center (C) to the start
4     // point (Pi)
5     radius = sqrt(pow(Pi.x - C.x, 2) + pow(Pi.y - C.y, 2));
6
7     // 2. Compute Cross-Track Error (e_n) based on radial distance variance
8     e_n = radius - sqrt(pow(robot.xe - C.x, 2) + pow(robot.ye - C.y, 2));
9
10    // 3. Calculate target heading angle tangent to the arc at the current
11    // position
12    thetaf = atan2(robot.ye - C.y, robot.xe - C.x) + sign(alpha) * (PI /
13    2.0);
14
15    // 4. Compute normalized heading error
16    e_theta = normalize_angle(thetaf) - normalize_angle(robot.thetae);
17    e_theta = normalize_angle(e_theta);
18
19    // 5. Enforce lower bound on cross-track gain and compute K_theta to
20    // satisfy critical damping
21    float circle_kn;
22    if (kn > MIN_KN)
23        circle_kn = kn;
24    else
25        circle_kn = MIN_KN;
26    ktheta = 2.0 * sqrt(circle_kn * v_nom);
27
28    // 6. Angular velocity control law incorporating adaptive gains and
29    // directional feedforward

```

```

25  robot.w_req = ktheta * e_theta - sign(alpha) * circle_kn * e_n +
      sign(alpha) * (v_nom / radius);
26
27  // 7. Constant linear velocity execution
28  robot.v_req = v_nom;
29
30  // 8. Hybrid Progress Tracking and Arrival Logic via directional
      angular gating
31  float start_angle = atan2(Pi.y - C.y, Pi.x - C.x);
32  float angle_to_robot = atan2(robot.ye - C.y, robot.xe - C.x);
33
34  // Compute accumulated angular progress while preventing target loss
      between control loops
35  float target_angle = fabs(alpha);
36  float directed_progress;
37  if (alpha > 0.0)
38      directed_progress = angle_to_robot - start_angle;
39  else
40      directed_progress = start_angle - angle_to_robot;
41  directed_progress = fmod(directed_progress, TWO_PI);
42  if (directed_progress < 0.0) directed_progress += TWO_PI;
43
44  // Evaluate Euclidean displacement from the initial pose to reject
      transient start-up noise
45  float dist_from_start = dist(Pi.x, Pi.y, robot.xe, robot.ye);
46
47  // Validate finish line crossing state within a designated capture
      window tolerance
48  const float finish_window = PI / 6.0;
49  bool crossed_finish_line = target_angle > 0.0 &&
50                               directed_progress >= target_angle &&
51                               directed_progress <= target_angle +
52                               finish_window;
53
54  // Kinematic Execution Guard: Restrict termination state triggers to a
      5 cm safe radius (MIN_DIST=0.05)
55  if (crossed_finish_line && dist_from_start > MIN_DIST) {
56      pass_obstacle = true;
57      robot.v_req = 0;
58      robot.w_req = 0;
59  }
60 }

```

Listing 4.4: C++ implementation of the Arc-Following navigation algorithm with directional feedforward compensation and hybrid arrival gating.

4.3 Obstacle Avoidance and Overtaking Methodology

To transform the "Robot at Factory Lite" arena into a decentralised multi-robot environment, the vehicles must share structural tracks. When a robot encounters another vehicle, relying solely on line-following algorithms would result in permanent deadlocks, forcing an inefficient division of the arena lines between specific robots. To overcome this constraint and allow multiple robots to navigate the same space dynamically, the vehicles must be able to leave the track temporarily without losing their spatial localisation. Thus, this section details the dynamic obstacle avoidance

and overtaking framework developed to allow the robot to autonomously leave its tracking line, bypass an obstacle using coordinated open-space geometry, and safely re-acquire the track.

4.3.1 Finite State Machine (FSM) Architecture

The overtaking manoeuvre is orchestrated by an upper-level finite state machine (FSM) embedded within the core control loop. The transition logic relies on a Time-of-Flight (ToF) distance sensor and internal state execution flags, including `avoided_obstacle`, `passed_obstacle`, and `returned_to_line`, as illustrated in Figure 4.4.

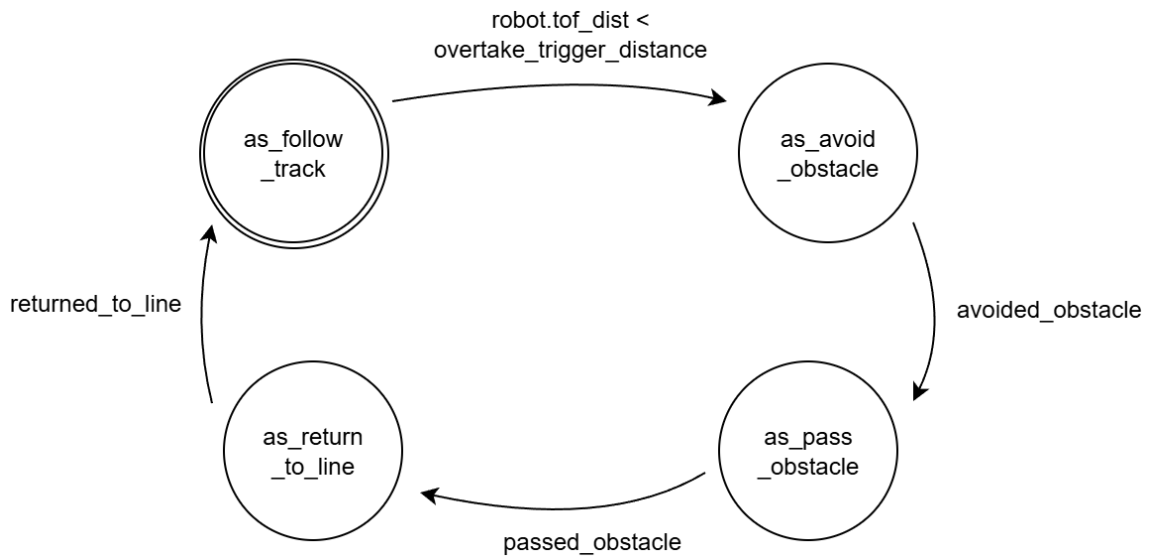


Figure 4.4: Finite State Machine (FSM) transition diagram for the autonomous obstacle avoidance and overtaking sequence.

The default operational state is `as_follow_track`, where the robot tracks the line and updates its local position via track-based localisation. If the ToF sensor detects an object within a predefined threshold (`overtake_trigger_distance`), the internal step counter resets to zero, and the system transitions to `as_avoid_obstacle`. This second state executes the initial lateral displacement to move the robot safely off the line and onto a parallel path. Once completed, it triggers the transition to `as_pass_obstacle`, which drives the robot parallel to the original track for a calculated clearance distance. Upon completing this advance, the system shifts to `as_return_to_line` to guide the robot back toward the track. When the infrared sensor array actively detects the track centreline, the main return flag is raised, safely returning the FSM to the initial track-following mode.

4.3.2 Overtake Trajectory Geometry and Parametrisation

The bypass trajectory relies on sequential, tangent circular arcs calculated dynamically using a target lateral offset (h_{offset}) and an optimal splitting angle (α), as illustrated in Figure 4.5. To

transition from line-following to a parallel offset line, the robot executes an S-shaped layout consisting of two symmetric, opposing circular curves. The radius (R_{arc}) required to hit the target lateral offset exactly at the end of the two arcs is derived from the geometric projection of the centre coordinates and is defined as:

$$R_{arc} = \frac{h_{offset}}{2(1 - \cos(\alpha))} \quad (4.28)$$

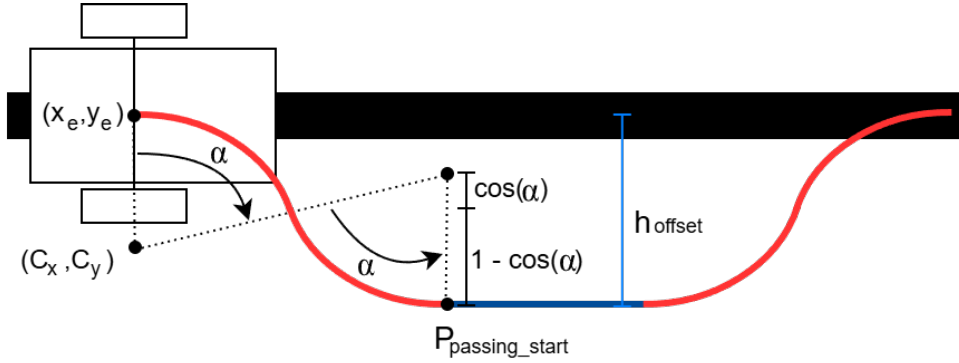


Figure 4.5: Geometric parametrisation of the S-shaped overtaking manoeuvre, illustrating the relationship between the splitting angle (α), the radius (R_{arc}), and the targeted lateral offset (h_{offset}).

When initiating an arc manoeuvre via `start_overtake_arc()`, the algorithm captures the current robot heading (θ_e) and computes the instantaneous coordinates of the first arc's centre (C_x, C_y) relative to the turning direction ($s = \text{sign}(\alpha)$):

$$C_x = x_e + R_{arc} \cos\left(\theta_e + s \frac{\pi}{2}\right) \quad (4.29)$$

$$C_y = y_e + R_{arc} \sin\left(\theta_e + s \frac{\pi}{2}\right) \quad (4.30)$$

These geometry formulations were mapped into the `start_overtake_arc()` function, detailed in Listing 4.5. Where the current state parameters were used in the geometric arc structure, determining the centre coordinates (C_x, C_y) and resetting the path completion flags.

```

1 void action_t::start_overtake_arc(float arc_radius, float arc_angle)
2 {
3     float side = sign(arc_angle);
4     if (side == 0.0) return;
5
6     radius = arc_radius;
7     alpha = arc_angle;
8     Pi.x = robot.xe;
9     Pi.y = robot.ye;
10
11     float center_heading = robot.thetae + side * (PI / 2.0);
12     C.x = robot.xe + radius * cos(center_heading);

```

```

13     C.y = robot.ye + radius * sin(center_heading);
14
15     passed_obstacle = false;
16 }

```

Listing 4.5: Implementation of the arc parameter initialization function within the action framework.

4.3.3 Algorithmic Phase Execution

The physical execution of the software implementation is split into three low-level action routines operating sequentially through an internal step counter (`avoid_step`), as represented in the state machine of Figure 4.6.

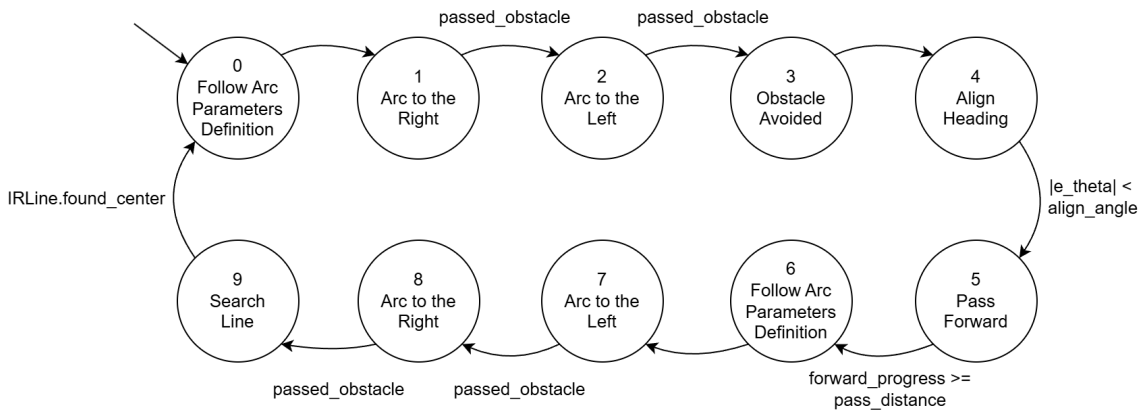


Figure 4.6: Finite state machine for the obstacle avoidance manoeuvre with the step counter.

The variable `passed_obstacle` acts as a state transition condition. It is set to `true` when the robot finishes following the current arc in the `follow_arc()` function and is reset to `false` whenever new arc-following parameters are initialised using the `start_overtake_arc` function.

4.3.3.1 Phase 1: Lateral Avoidance (`avoid_obstacle_values`)

Upon entering step 0, the algorithm backs up the nominal velocity and records the track's heading orientation (θ_{track}), as well as calculates the parameters for an arc that turns the robot away from the line. In step 1, the robot invokes `follow_arc()` to trace this clockwise arc. Once the target angular displacement is met, step 2 initialises a counter-clockwise arc of equal radius. This second arc straightens out the robot's heading, placing it on a parallel path exactly at the offset distance outside the track line, raising the dynamic avoidance status flag.

The algorithmic implementation of this dual-arc lateral displacement sequence is presented in Listing 4.6. The routine enforces programmatic saturation constraints on both the minimum lateral clearance and the operational turning angles to guarantee kinematic feasibility before computing the geometric radius.

```

1 void action_t::avoid_obstacle_values(void) {
2     returned_to_line = false; // Reset flags
3     avoided_obstacle = false;
4
5     //safe lateral space from the obstacle
6     float lateral_offset = overtake_offset;
7     if (lateral_offset < MIN_LATERAL_OFFSET)
8         lateral_offset = MIN_LATERAL_OFFSET;
9
10    //guarantee geometric feasibility
11    float arc_angle = fabs(overtake_arc_angle);
12    if (arc_angle < MIN_ARC_ANGLE) arc_angle = MIN_ARC_ANGLE;
13    if (arc_angle > MAX_ARC_ANGLE) arc_angle = MAX_ARC_ANGLE;
14
15    float arc_radius = lateral_offset / (2.0 * (1.0 - cos(arc_angle)));
16
17    if (avoid_step == 0) { //define follow arc parameters
18        backup_v_nom = v_nom;
19        track_heading = robot.theta_e;
20        start_overtake_arc(arc_radius, -arc_angle);
21        avoid_step = 1;
22    }
23
24    if (avoid_step == 1) { //curve to the right
25        follow_arc();
26        if (passed_obstacle) {
27            start_overtake_arc(arc_radius, arc_angle);
28            avoid_step = 2;
29        }
30
31    } else if (avoid_step == 2) { //curve to the left
32        follow_arc();
33        if (passed_obstacle) {
34            passed_obstacle = false;
35            avoided_obstacle = true;
36            avoid_step = 3;
37        }
38    }
39 }

```

Listing 4.6: Implementation of the lateral avoidance phase and parameter calculation routine.

4.3.3.2 Phase 2: Linear Passing (**pass_obstacle**)

Once parallel, the robot switches to a linear tracking control law. Step 4 uses an angular velocity proportional controller to align the robot's heading with the recorded track heading ($\omega_{req} = K_{\theta}e_{\theta}$). As soon as the alignment falls within a tight tolerance window (`overtake_align_angle`), the robot logs its current position as a reference baseline ($P_{passing_start}$) and transitions to step 5.

In step 5, the robot drives forward at its nominal velocity (v_{nom}) using a combined heading and cross-track error control law to regulate its parallel line deviation (e_{pass_line}):

$$\omega_{req} = K_{\theta}e_{\theta} + K_n e_{pass_line} \quad (4.31)$$

The cross-track error (e_{pass_line}) represents the shortest Euclidean distance from the robot's current position ((x_e, y_e)) to the desired parallel path. As illustrated in Figure 4.7, this error is calculated dynamically by projecting the displacement components ($dx = x_e - P_{passing_start,x}$ and $dy = y_e - P_{passing_start,y}$) onto a vector normal to the path, forcing e_{pass_line} to remain strictly perpendicular to the track heading (θ_{track}):

$$e_{pass_line} = \sin(\theta_{track}) \cdot dx - \cos(\theta_{track}) \cdot dy \quad (4.32)$$

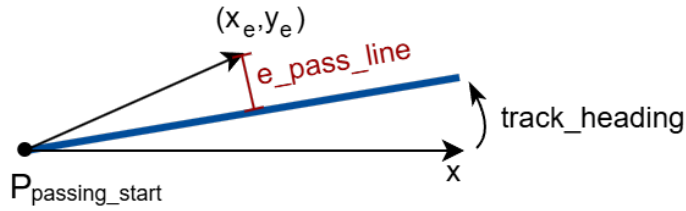


Figure 4.7: Geometric derivation of the cross-track error (e_{pass_line}), showing the perpendicular relationship between the tracking error and the desired path line.

Simultaneously, progress during this straight passing phase (depicted by the blue line segment in Figure 4.8) is quantified continuously by projecting the robot's displacement vector back onto the coordinate frame of the primary track axis. The forward progress distance ($d_{progress}$) relative to the initialisation waypoint ($P_{passing_start}$) is computed as:

$$d_{progress} = (x_e - P_{passing_start,x}) \cos(\theta_{track}) + (y_e - P_{passing_start,y}) \sin(\theta_{track}) \quad (4.33)$$

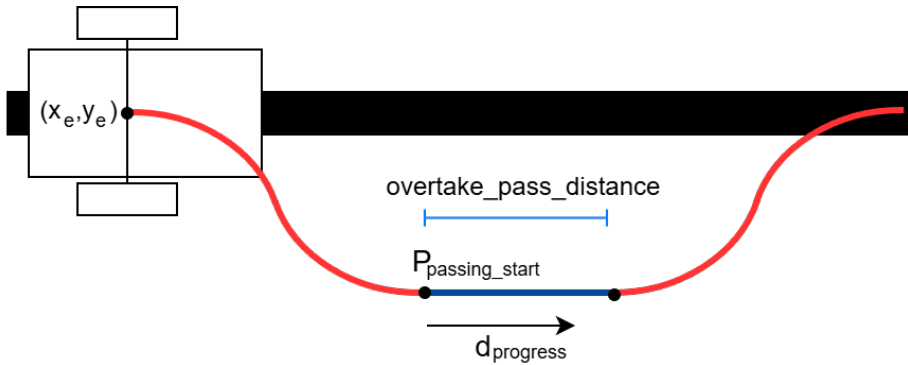


Figure 4.8: Linear passing phase metrics, showing the tracking baseline starting at $P_{passing_start}$, the accumulated forward progress distance ($d_{progress}$), and the target threshold ($overtake_pass_distance$).

The robot remains executing this phase until this calculated forward displacement matches or exceeds the configured safety passing distance ($overtake_pass_distance$), as shown in Figure 4.8. At this point, the internal step increments to trigger the track re-acquisition phase.

The function that maps this sequence is detailed in Listing 4.7. The implementation enforces saturation bounds on controller gains and orientation windows while computing state transitions.

```

1 void action_t::pass_obstacle(void) {
2     avoided_obstacle = false; // Reset flags
3     returned_to_line = false;
4
5     if (avoid_step <= 3) { // If the obstacle was avoided, we can pass it
6         passed_obstacle = false;
7         avoid_step = 4;
8     }
9     float pass_distance = overtake_pass_distance;
10    if (pass_distance < 0.0) pass_distance = 0.0;
11
12    e_theta = normalize_angle(track_heading - robot.thetae); // error
13    float heading_kn;
14    if (kn > MIN_KN)
15        heading_kn = kn;
16    else
17        heading_kn = MIN_KN;
18    ktheta = 2.0 * sqrt(heading_kn * v_nom);
19
20    //Define error tolerance window
21    float align_angle = overtake_align_angle;
22    if (align_angle < MIN_ALIGN_ANGLE) align_angle = MIN_ALIGN_ANGLE;
23    if (align_angle > MAX_ALIGN_ANGLE) align_angle = MAX_ALIGN_ANGLE;
24
25    if (avoid_step == 4) { //Stationary alignment to track heading
26        robot.v_req = 0;
27        robot.w_req = ktheta * e_theta;
28
29        // Transition once alignment error falls inside the tolerance
30        window
31        if (fabs(e_theta) < align_angle) {
32            P_passing_start.x = robot.xe;
33            P_passing_start.y = robot.ye;
34            avoid_step = 5;
35        } else {
36            return;
37        }
38
39        // Calculate perpendicular cross-track error
40        float dx = robot.xe - P_passing_start.x;
41        float dy = robot.ye - P_passing_start.y;
42        float e_pass_line = sin(track_heading) * dx - cos(track_heading) * dy;
43
44        robot.v_req = v_nom; // Compute linear and angular velocities
45        robot.w_req = ktheta * e_theta + heading_kn * e_pass_line;
46
47        if (overtake_forward_progress(P_passing_start) >= pass_distance) { //
48            When distance requirement is met
49            passed_obstacle = true;
50            avoid_step = 6;
51        }
52    }
53 }

```

Listing 4.7: Implementation of the linear passing phase, heading alignment, and cross-track error regulations.

4.3.3.3 Phase 3: Track Re-acquisition (return_to_line)

In step 7, the robot initiates a counter-clockwise arc back toward the track using the geometric radius calculated previously. When this first arc completes, the next step triggers a clockwise arc designed for the robot to smoothly return to the line.

If the discrete tracking loops overshoot or fail to place the robot directly on top of the line, step 9 acts as a safety measure. In this step, the robot drops to a slow forward crawl of 0.1 m/s while applying a slight rotational bias of 0.5 rad/s to sweep the area. As soon as the infrared sensor array registers a valid centreline trigger, the sequence resets the internal step counter back to zero and safely returns to the line-following state.

The firmware architecture mapping this re-acquisition and sensor-search state logic is detailed in Listing 4.8.

```

1 void action_t::return_to_line(void) {
2     avoided_obstacle = false; // Reset flags
3
4     //safe lateral space from the obstacle
5     float lateral_offset = overtake_offset;
6     if (lateral_offset < MIN_LATERAL_OFFSET)
7         lateral_offset = MIN_LATERAL_OFFSET;
8
9     //guarantee geometric feasibility
10    float arc_angle = fabs(overtake_arc_angle);
11    if (arc_angle < MIN_ARC_ANGLE) arc_angle = MIN_ARC_ANGLE;
12    if (arc_angle > MAX_ARC_ANGLE) arc_angle = MAX_ARC_ANGLE;
13
14    float arc_radius = lateral_offset / (2.0 * (1.0 - cos(arc_angle)));
15
16    if (avoid_step <= 6) { //define follow arc parameters
17        returned_to_line = false;
18        passed_obstacle = false;
19        start_overtake_arc(arc_radius, arc_angle);
20        avoid_step = 7;
21    }
22
23    if (avoid_step == 7) { //curve to the left
24        follow_arc();
25        if (passed_obstacle) {
26            start_overtake_arc(arc_radius, -overtake_arc_angle_in);
27            avoid_step = 8;
28        }
29
30    } else if (avoid_step == 8) { //curve to the right
31        follow_arc();
32        if (passed_obstacle) {
33            passed_obstacle = false;
34            avoid_step = 9;
35        }
36
37    } else if (avoid_step == 9) { //searching for the line
38        robot.v_req = 0.1;
39        robot.w_req = 0.5;
40    }
41
42    if (avoid_step > 8 && robot.IRLLine.found_center) { //found the line

```

```
43     returned_to_line = true;  
44     avoid_step = 0;  
45 }  
46 }
```

Listing 4.8: Implementation of the track re-acquisition phase, including the returning S-curve sequence and the infrared line-search routine.

Chapter 5

Experimental Results and Validation

Following the theoretical and algorithmic design established in Chapter 4, this chapter presents the experimental setup, performance metrics, and empirical validation of the developed navigation and control systems. To evaluate the robustness and efficiency of the mobile robot within the "Robot at Factory Lite" arena, a series of structured physical and simulated trials were conducted. These experiments aim to quantify the real-world tracking accuracy, response times, and situational reliability of the control frameworks under nominal and constrained operating conditions.

The chapter begins by detailing the experimental environment, data acquisition, and the validation methodology applied to the track-based localisation framework, analysing its capacity to correct cumulative odometry drifts. Following this, the performance of the core trajectory-tracking techniques, which include go-to-point, line-following, and arc-navigation, is evaluated through experiments with different initial conditions. Finally, we analyse the empirical validation of the obstacle avoidance and overtaking framework. This section shows the reliability of the finite state machine transitions, the geometric accuracy of the parallel passing profile, and the robustness of the return to the track algorithm under physical sensor constraints. Together, these quantitative results validate the system's operational capability and demonstrate its readiness for multi-robot factory applications.

5.1 Track-Based Localization Performance

To validate the performance of the track-based localisation algorithm, physical tests were conducted over both single and multiple laps, comparing the estimated trajectories with and without the positional correction enabled.

The physical track layout, illustrated in Figure 5.1b, was mapped using a set of Cartesian coordinates (P_0 through P_8). These key points define the start and end nodes of the straight track segments, creating the reference list used by the algorithm for spatial correction.

During the experiments, the line-following control parameters, presented in Table 5.1 were tuned to ensure smooth navigation.

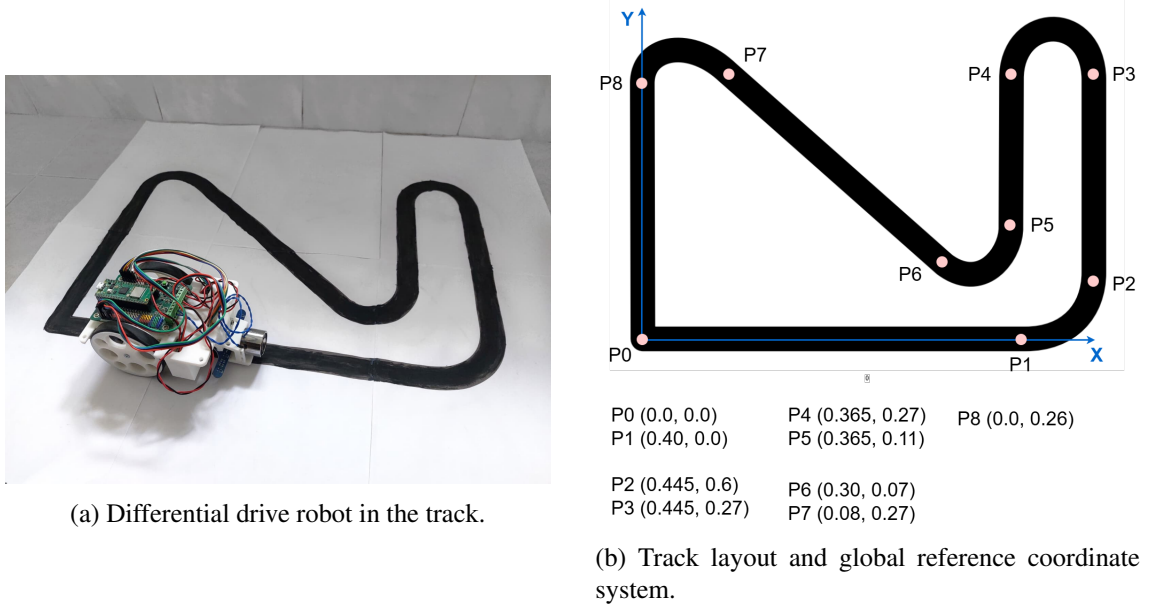


Figure 5.1: Experimental setup.

Table 5.1: Control and Localization Parameters.

Symbol	Description	Value	Unit
v_{nom}	Nominal velocity	0.15	m/s
K_{track}	Track error gain	-0.10	—
ω_0	Non-linear velocity modulation parameter	2.00	rad/s
ω_z	Line-search angular velocity	3.00	rad/s
$\omega_{threshold}$	Steady movement angular velocity threshold	0.40	rad/s

Where:

- v_{nom} represents the baseline forward speed on straight lines;
- K_{track} acts as the proportional controller that maps the line sensor's lateral offset to the target angular velocity;
- ω_0 sets the angular velocity threshold where the forward speed is entirely attenuated to zero;
- ω_z dictates the robot's rotation-in-place recovery behaviour in the event the line is temporarily lost;
- $\omega_{threshold}$ ensures that the odometry correction is only triggered when the robot's rotational movement falls below this limit.

The first set of experiments evaluated the trajectory estimation of the robot over a single lap. Figure 5.2a displays the estimated path, relying strictly on standard wheel odometry, without the

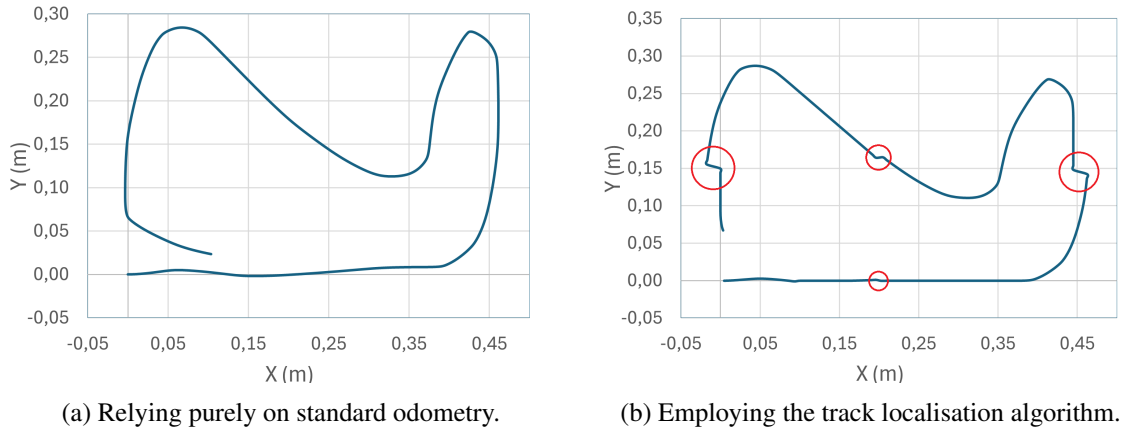


Figure 5.2: Estimated robot trajectory for a single lap.

track localisation algorithm enabled. The base odometry provides a reliable initial estimate that closely resembles the physical track shape.

However, when the track localisation algorithm is enabled (Figure 5.2b), the impact of the correction becomes clearly visible. Upon entering the defined straight line segments, the algorithm identifies the linear navigation phase, and applies the homogeneous transformations, calculating the lateral offset and changing the estimated position of the robot to the true segment lines. These discrete mathematical corrections ensure the estimated trajectory aligns perfectly with the real-world map coordinates during straight lines.

To evaluate the long-term robustness of the system, the experiment was repeated with the robot completing four continuous laps.

Figure 5.3a illustrates that relying solely on standard odometry over long periods of time invariably results in cumulative error. The inevitable errors in wheel slip and heading estimation compound over several turns. The estimated position has diverged significantly from the true track layout by the fourth lap.

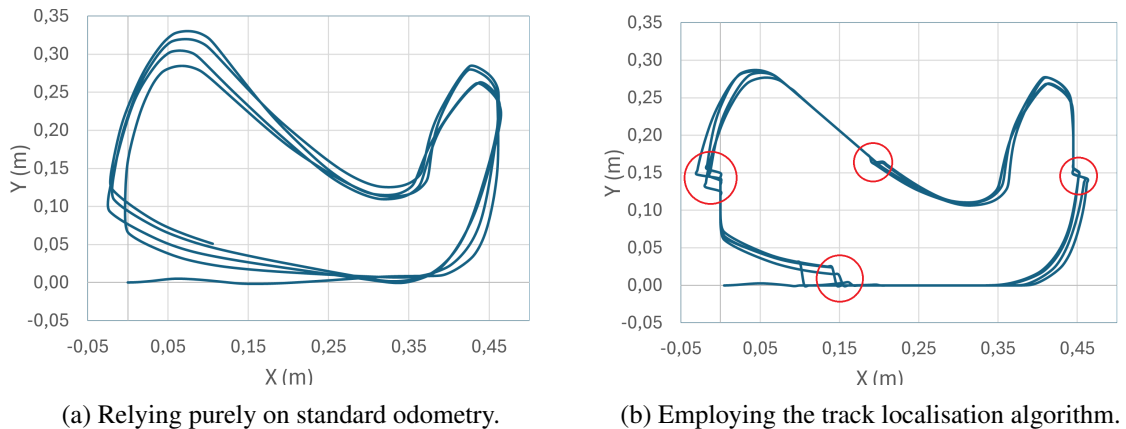


Figure 5.3: Estimated robot trajectory for four laps.

On the other hand, the system's performance during the same four continuous laps with the

track localisation algorithm enabled is shown in Figure 5.3b. The accumulated lateral and heading drift is systematically eliminated by the periodic adjustments made against the known line segments. As a result, the single-lap corrected trajectory and the 4-lap corrected trajectory are almost identical. This demonstrates that, regardless of the operational period, the generalised localisation method effectively prevents long-term odometry divergence and maintains a highly accurate position estimate.

Quantitative results in Table 5.2 show that the proposed algorithm reduced both average and maximum error. To calculate the table values, due to the lack of an absolute positioning system, known geometric constraints of the track were used as a spatial reference (ground truth). The metrics were obtained by calculating the cross-track error, defined as the orthogonal distance between the robot's estimated pose and the centreline of the track segments. To evaluate the improvement of the implemented system, the pure wheel odometry (baseline) was compared against the proposed track-based localisation across single and four lap trials.

Table 5.2: Localization Error Metrics.

Case	Baseline		Proposed		Improvement	
	Avg. (m)	Max (m)	Avg. (m)	Max (m)	Avg. (%)	Max (%)
1 Loop	0.013	0.045	0.009	0.044	29.60	3.46
4 Loops	0.018	0.065	0.011	0.046	38.15	29.12

5.2 Trajectory Tracking Results

To evaluate the mathematical models and control laws introduced in Section 4.2, a series of physical experimental trials were conducted. The goal of these tracking evaluations is to analyse the robustness of the three algorithms: Go-To-Point (`gotoxy`), Line Following, and Arc Following, when facing different initial conditions. For all trials, the real-world positioning data were based on the odometry of the robot and were compared against the targeted reference trajectories.

5.2.1 Go-To-Point Navigation (`gotoxy`)

To validate the performance of the navigation algorithm, experimental trials were conducted by commanding the robot from an initial pose of $(x, y, \theta) = (0, 0, 0)$ toward two distinct quadrants. Figure 5.4 displays the resulting 2D trajectory tracking plots.

The experimental trajectories showed the practical impact of the parabolic velocity modulation. When moving toward the positive target (Figure 5.4a), the initial heading error falls within the $\pm\pi/2$ rad threshold. Consequently, the robot simultaneously translates and rotates, tracing a smooth, continuous path directly to $P_f = (0.5, 0.6)$ m.

In contrast, the negative target configuration (Figure 5.4b) demands an initial orientation change greater than $\pi/2$ rad. In these cases, the robot starts by setting its linear velocity to zero,

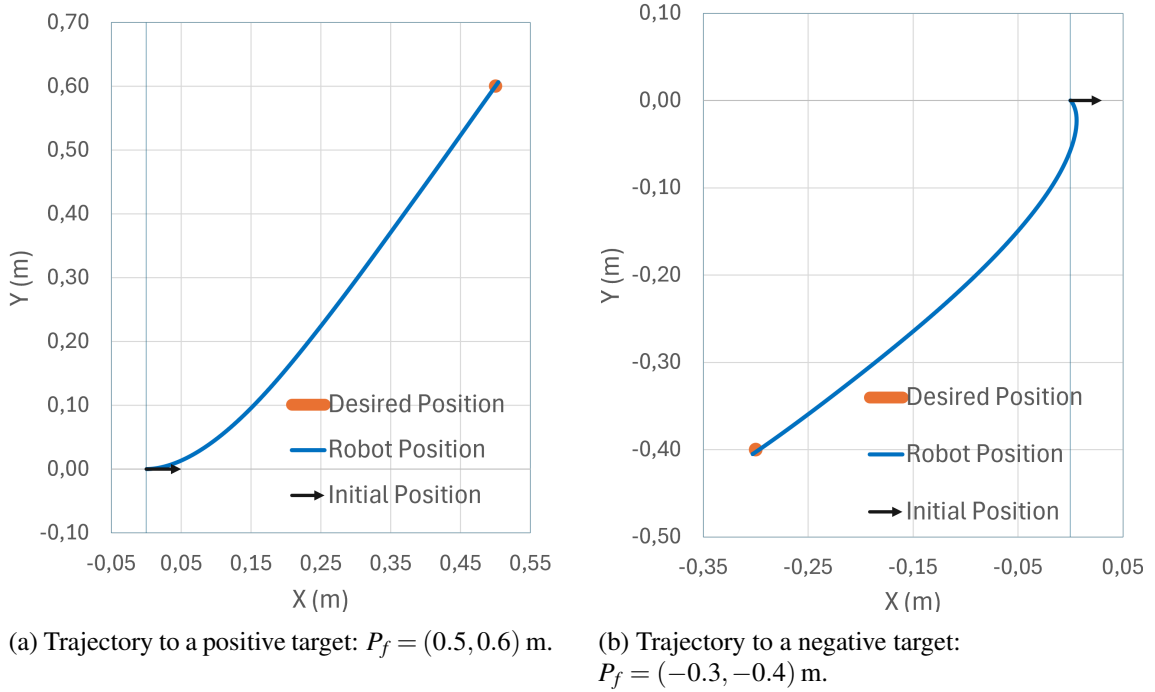


Figure 5.4: Experimental 2D trajectory profiles using the Go-To-Point controller.

executing a pure rotation-in-place at the origin. Once its heading error is less than $\pm\pi/2$ rad, the parabolic profile initiates a smooth linear acceleration toward $P_f = (-0.3, -0.4)$ m.

In both experimental cases, the robot precisely converges to the target coordinates with zero steady-state error, validating the stability of the navigation loop.

5.2.2 Line Follow Navigation

To verify the tracking capabilities of this line-following architecture, experimental scenarios were performed: an initial heading mismatch at the path origin, a pure lateral offset, and a combination of a lateral offset with a heading mismatch exceeding 90° . Figure 5.5 summarizes these experimental 2D trajectories.

The experimental results validate the response and stability of the line-following controller under different experimental scenarios. When the platform starts at the path origin with an incorrect heading (Figure 5.5a), the dynamic tuning loop establishes a smooth convergence to the path vector. Similarly, under a parallel cross-track offset (Figure 5.5b), the e_n feedback term guides a clean entry that merges back onto the reference line. For the most severe test condition combining a lateral displacement with an initial heading error exceeding 90° (Figure 5.5c), the robot executes an initial localised rotation-in-place at its starting point, engaging forward acceleration only after the alignment error falls below $\pi/2$ rad.

Furthermore, all three scenarios highlight the practical utility of the vector dot-product condition. By evaluating the relative orientation of the robot's motion against the target vector, the system automatically detects if physical momentum carries the platform past the perpendicular

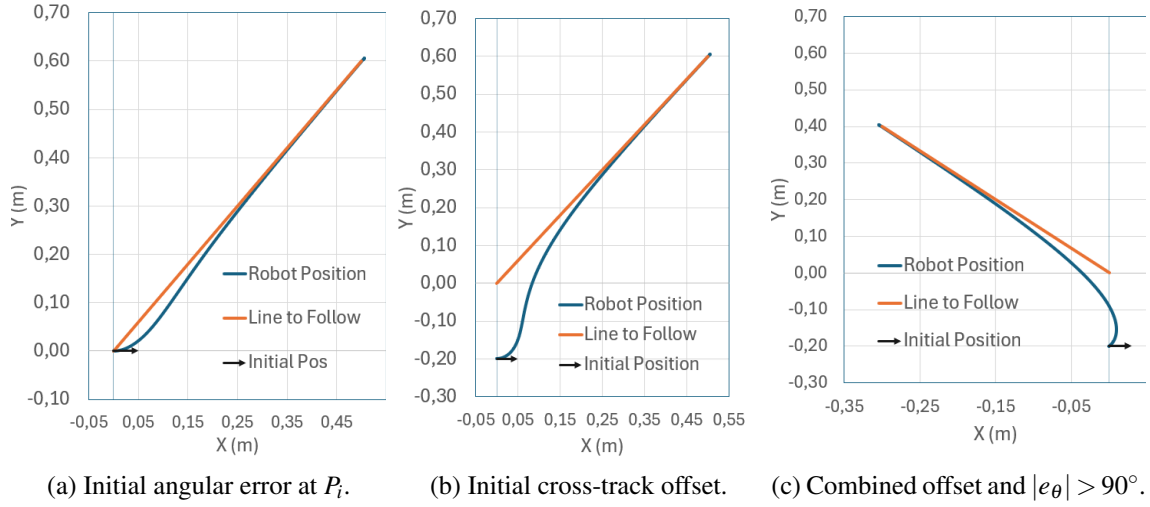


Figure 5.5: Experimental 2D tracking trajectories under various line-following boundary conditions.

goal plane at P_f . The immediate activation of the `dot_product < 0` condition signals that the robot has crossed the target line and stops it, eliminating orbiting behaviours around the terminal coordinate.

5.2.3 Arc Follow Navigation

To validate the tracking accuracy and direction-switching capabilities of the arc-following system, experimental trials were executed under three distinct configurations: a positive angular arc profile ($\alpha > 0$), a negative angular arc profile ($\alpha < 0$), and an initialisation condition in which the robot begins with a severe lateral offset inside the desired boundary radius. Figure 5.6 illustrates these experimental 2D trajectory profiles.

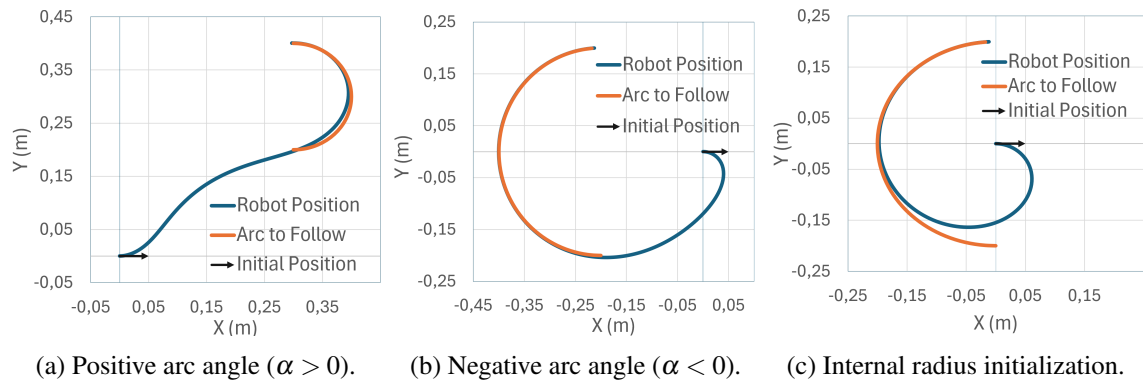


Figure 5.6: Experimental 2D tracking trajectories under various arc-following boundary profiles.

The experimental paths highlight the geometric agility of the arc-following controller and its ability to maintain a constant nominal velocity across diverse profiles. For standard manoeuvres driven by positive (Figure 5.6a) or negative (Figure 5.6b) target parameters, the sign of α sets

the correct rotational orientation. This proper sign configuration scales both the feedback steering commands and the feed-forward velocity ($\frac{v_{nom}}{R}$), allowing the platform to trace the circular arc boundary with negligible steady-state error.

Additionally, the reliability of the radial error control is proven when the robot starts inside the target circle (Figure 5.6c). Even with this large initial positioning error (e_n), the control algorithm guides the robot outward in a smooth curve that merges perfectly with the desired path without overshooting. Finally, stopping the robot based on a hybrid solution, that combines the total angle travelled relative to the starting point with the distance to the target, ensures highly accurate arrival. This method avoids the positional confusion and errors that occur when using simple straight-line distance checks for curved paths.

5.3 Obstacle Overtaking and Avoidance Validation

To validate the integration of the finite state machine (FSM) tracking sequences and the real-time sensor logic detailed in Section 4.3, a physical experiment was conducted on the arena floor. The experiment evaluated a head-on collision encounter involving two mobile platforms initialised on the $Y = 0$ m track line segment, as captured in Figure 5.7.

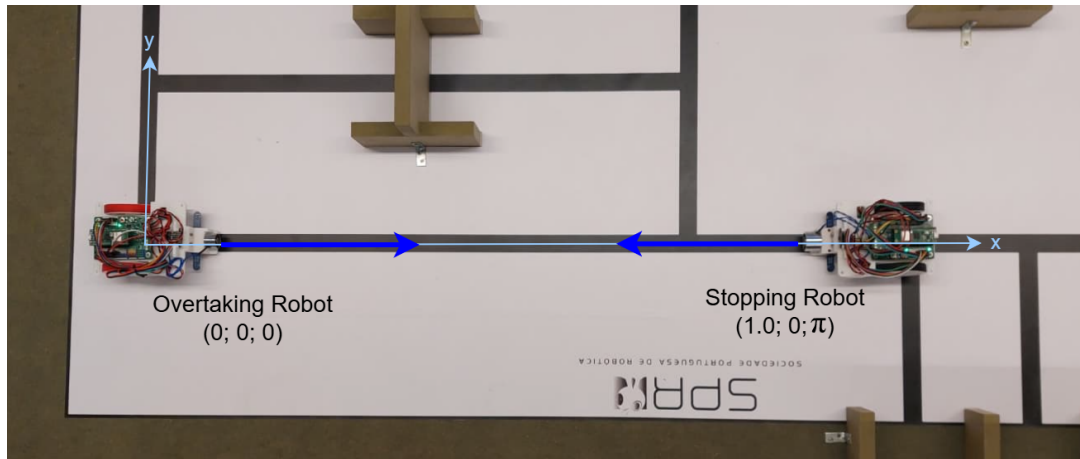


Figure 5.7: Physical experimental setup within the testing arena, demonstrating the head-on initialization vectors for the active overtaking vehicle (left) and the oncoming obstacle platform (right).

The active overtaking vehicle (Robot 1, blue curve in Figure 5.9) was positioned at the origin $P_1 = (0.0, 0.0)$ m with an initial heading of $\theta_1 = 0.0$ rad. The opposing vehicle (Robot 2, orange line in Figure 5.9) was positioned at $P_2 = (1.0, 0.0)$ m facing inward with an initial heading of $\theta_2 = \pi$ rad. Both systems were configured with a Time-of-Flight (ToF) distance threshold (`overtake_trigger_distance`) of 0.30 m; and the overtaking robot had a target lateral offset (h_{offset}) of 0.15 m, an optimal splitting angle (α) of 1.0 rad, and a required safety forward passing distance (`overtake_pass_distance`) of 0.05 m.

To provide a clear, step-by-step visual overview of the physical execution states throughout this manoeuvre, Figure 5.8 breaks down the trial chronologically into eight key operational phases. The video recordings documenting the physical implementation can be accessed online¹.

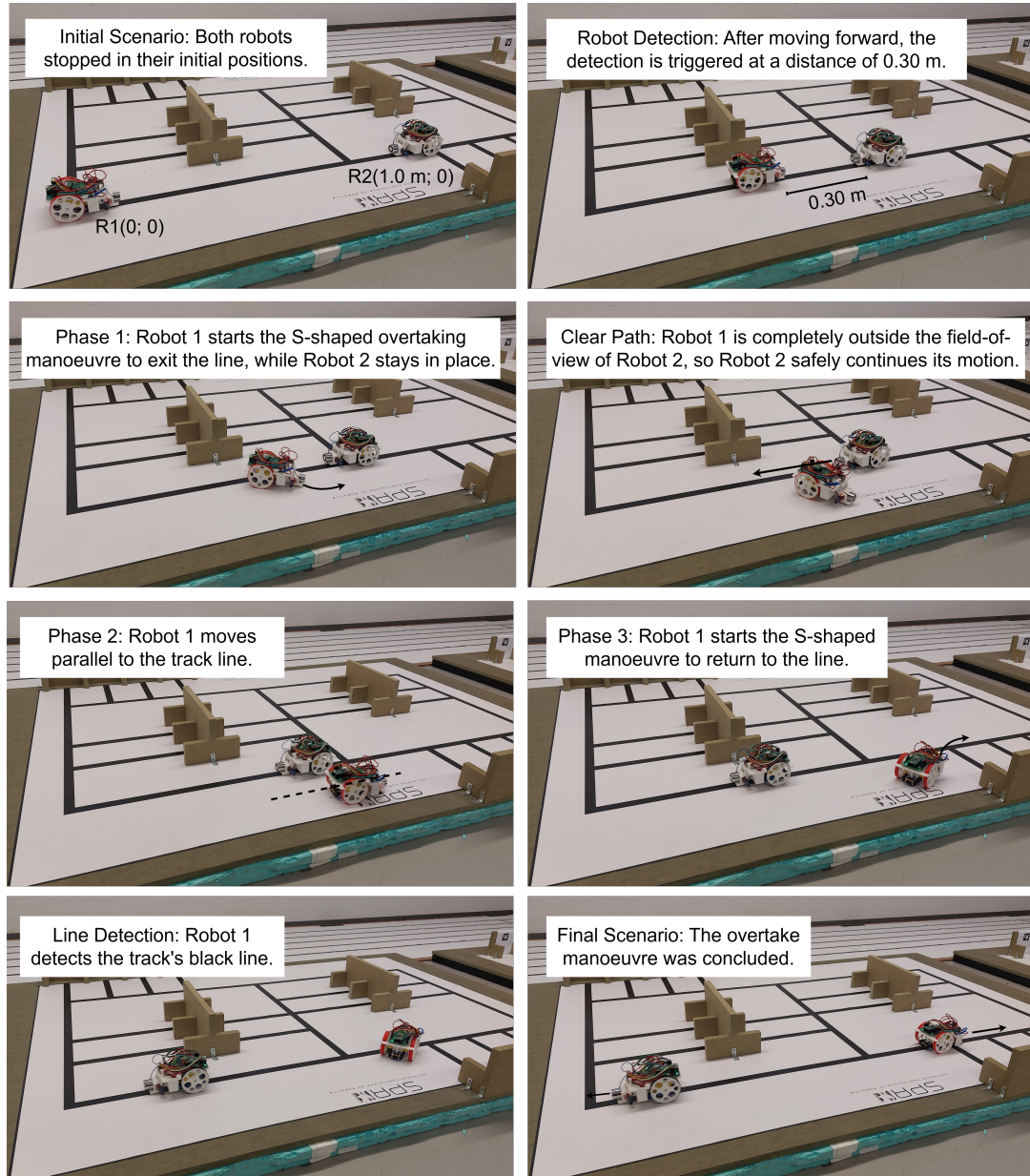


Figure 5.8: Chronological phase breakdown diagram of the cooperative overtaking manoeuvre.

The recorded spatial trajectories of both platforms during the interaction are displayed in Figure 5.9.

The experimental paths validate the physical response and coordination safety of the control loops. As both vehicles move closer than their defined sensor range (0.30 m), Robot 2 successfully

¹Experimental video repository: <https://drive.google.com/drive/folders/1EhOBYDdMOInjEGPVtjRvD6uHqXPMUndI?usp=sharing>

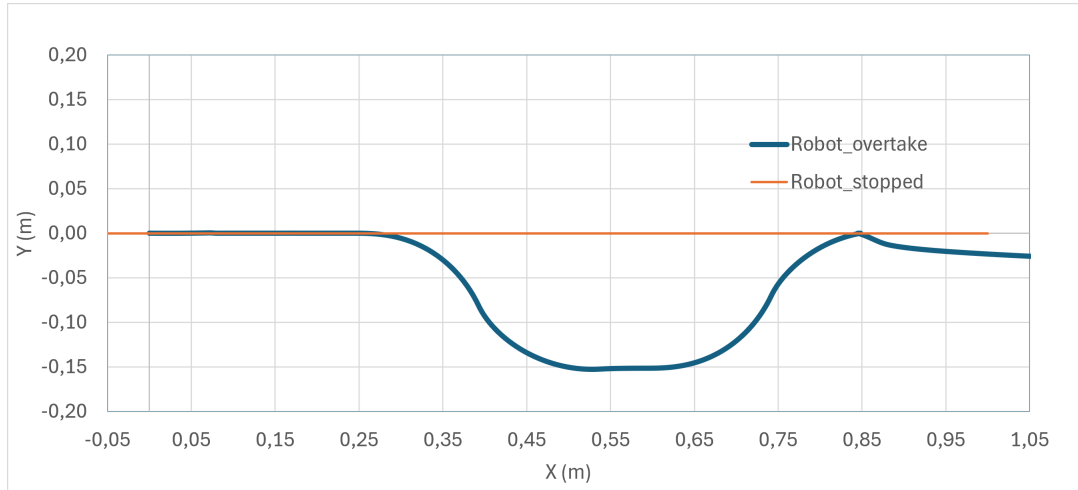


Figure 5.9: Experimental 2D spatial path traces demonstrating the interaction between the active overtaking vehicle (blue) and the oncoming obstacle vehicle (orange).

triggers its obstacle guard condition. The platform immediately drops its linear velocity request to zero ($v_{req} = 0$ m/s), remaining safely stationary on the path at $X \approx 0.569$ m, as shown in the plot of Figure 5.10a.

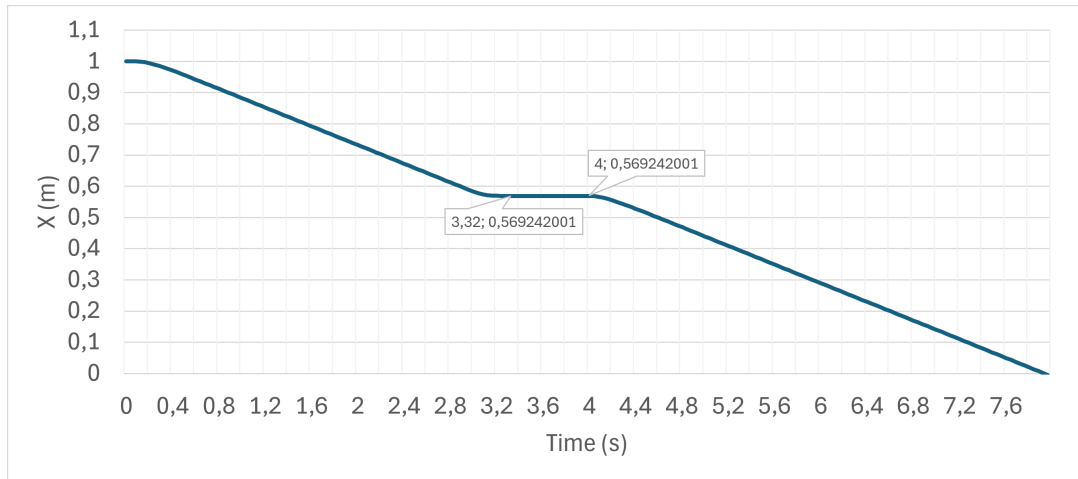
While Robot 2 holds this stationary defensive posture, Robot 1 overtakes it through its lateral arcs. Upon approaching the obstruction zone, Phase 1 initiates near $X = 0.25$ m. The active robot tracks the continuous S-curve smoothly, reaching a stable peak lateral offset of exactly $Y = -0.15$ m near $X = 0.50$ m, leaving a lateral clearance buffer that is completely safe, considering the physical footprint of Robot 2 (9.5 cm in width).

Thereafter, Robot 1 executes the second phase, the parallel tracking loop with minimal lateral ripple. Once the forward progress condition ($d_{progress} \geq 0.05$ m) is satisfied, Phase 3 triggers near $X = 0.60$ m, bringing the platform back toward the primary track. The distinct inflexion point observed at (0.85, 0.0) m marks the transition into step 9, where the robot switches from dead reckoning to an active search to return to the black track axis.

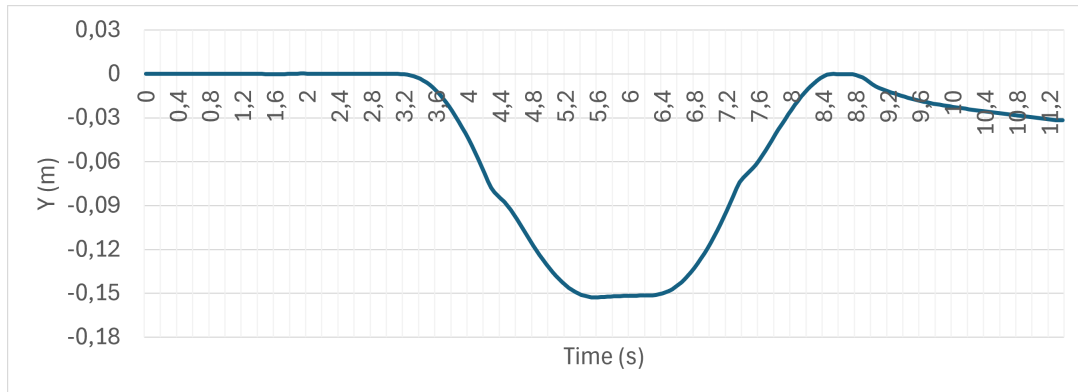
5.3.1 Temporal and Cooperative Synchronization Validation

While the spatial trajectories in Section 5.3 demonstrate geometric success, the temporal dependencies and real-time coordination between the two platforms remain implicit. To validate the state-machine synchronization and timing parameters, the positions of both vehicles were logged simultaneously over the course of the encounter, as illustrated in Figure 5.10.

The data profiles validate the physical behaviour and timing of the cooperative control loops across distinct operational phases. During the initial encounter from $t = 0$ s to $t = 3.32$ s, both vehicles drive directly toward each other on the track line, causing the closure distance to drop rapidly. At $t = 3.32$ s, Robot 2's obstacle guard condition triggers, forcing its linear velocity request to zero, as indicated by the zero slope line in Figure 5.10a at the precise coordinate of $X = 0.569$ m.



(a) Temporal analysis of Robot 2's X position, showing the stop coordinate ($X = 0.569$ m) and the 0.68 second stop duration.



(b) Temporal analysis of Robot 1's Y position, illustrating the active 3-phase lateral avoidance profile.

Figure 5.10: Temporal analysis of the cooperative overtaking manoeuvre, demonstrating real-time collision mitigation and sensor-cone clearance.

Similarly, the distance sensor of Robot 1 registers the obstacle at a distance of less than 0.30 m and triggers Phase 1 of its FSM tracking loop. As shown in Figure 5.10b, Robot 1's lateral coordinate begins its downward deviation away from the track baseline ($Y = 0$ m) at that moment.

Robot 2 remains completely stationary for a total duration of 0.68 s. At $t = 4.00$ s, its safety guard is lifted, and Robot 2 safely continues its motion, which indicates that Robot 1 is completely outside the active field-of-view cone of Robot 2's forward-facing ToF sensor. On the other hand, Robot 1 smoothly transitions to its peak offset of $Y = -0.15$ m before returning to the track line at approximately $t = 8.40$ s.

Chapter 6

Final Considerations

This chapter starts by summarising the engineering problem solved during this dissertation, then refers to the conclusions drawn from the design, implementation, and deployment phases. It outlines the structural and algorithmic achievements of the developed platform, highlighting its contributions to both the academic and educational fields. Finally, it concludes with recommendations for future work.

6.1 Conclusions and System Achievements

The primary objective of this dissertation, the development of a compact and functional Autonomous Mobile Robot (AMR) for the "Robot at Factory Lite" competition, was successfully achieved. By optimising the mechanical and electronic integration, the final platform achieved a structural footprint of $151 \times 95 \times 80$ mm. This represents a significant reduction in width compared to the standard 138 mm baseline solution [12], fulfilling the core goal of the project.

This structural miniaturisation was accomplished without compromising the functional capabilities or structural integrity of the original platform. The customised 3D-printed chassis effectively stabilised the vehicle's centre of mass, allowing coexistence between the electromagnet, sensors, and the power system within a confined volume.

Regarding processing and control, while previous solutions considered alternative microcontrollers, the robot's physical architecture was standardised around the Raspberry Pi Pico W, executing all computational loops locally. This choice provided the ideal balance of low-level hardware control, raw processing power for real-time kinematics, and wireless connectivity.

The embedded software transitioned the platform from a simple line follower to a fully localised AMR, as it used dual-wheel differential odometry to map the position (X, Y, θ) of the platform, transcending rigid track dependencies. This mathematical framework enabled the vehicle to execute autonomous, off-track overtaking trajectories using the frontal ToF sensor to completely bypass obstacles on the track before actively re-acquiring the path line.

To mitigate the unbounded drift and cumulative errors characteristic of pure dead reckoning, the control architecture relies on the physical track network as a deterministic localisation method.

Whenever a straight track segment is detected, the navigation system executes real-time pose corrections using homogeneous transformations. By fusing the continuous wheel encoder data with discrete, track-based geometric references, the robot dynamically resets its coordinate errors, ensuring positioning accuracy throughout extended operational runs.

The physical validation of this architecture took place at the RoboCup Portugal Open 2026 in Barcelos¹. The physical hardware design and low-level firmware frameworks were provided to an academic team competing in the "Robot at Factory Lite" league, where they secured first place. This milestone validates the reproducibility, physical design integrity, and competitive viability of the developed open-hardware architecture.

Furthermore, the scientific impact of this dissertation is reflected in two accepted conference papers. The paper "*Track-Based Position Estimation and Localization for Differential Drive Line-Following Robots*" was accepted at the CONTROLO 2026 conference, and the paper "*Educational Half-Size Robot for the 'Robot at Factory Lite' Competition*" was accepted at the TEEM 2026 conference. These publications attest to both the algorithmic innovation and the educational value of the developed platform.

6.2 Main Contributions

The outcome of this work contributes to the academic, industrial, and educational fields. The dissertation delivers a fully validated open-source hardware and software system to the mobile robotics research community. The comprehensive electromechanical modelling, transfer functions, and finite state machine control strategies are thoroughly documented, providing a reproducible reference architecture for future internal logistics research.

Additionally, this project actively contributes to the educational domain. By reducing the volume of the robot, the overall spatial requirements for the arena can be proportionally downscaled. This lowers the barriers to transporting the entire system, making it an excellent platform for mobile academic demonstrations, multi-robot classroom interactions, and decentralised programming challenges across diverse educational institutions.

6.3 Future Work

While the developed AMR platform demonstrated reliability in the competition, several opportunities remain for future technical extensions, particularly in maximising the system's performance through its off-track navigation capabilities.

Currently, the robot relies on a rigid mathematical reset whenever it encounters a track intersection or a straight segment to correct its odometry. To improve this, the implementation of an Extended Kalman Filter (EKF) could enhance the estimated position by providing a robust probabilistic framework for fusing raw wheel odometry with periodic track-based corrections. Instead

¹Official event website: <https://www.festivalnacionalrobotica.pt/2026/>

of abruptly overwriting coordinates, the EKF would statistically smooth the transitions, ensuring steady positioning accuracy during complex manoeuvres.

Furthermore, the capability to leave the track opens up new operational strategies to optimise the robot's timing and efficiency within the "Robot at Factory Lite" competition. The robot can be programmed to calculate optimal diagonal shortcuts across open arena spaces, cutting travel time between distant machine stations rather than blindly following the perimeter tracks. To further reinforce this line-free navigation, a lightweight LiDAR sensor could be integrated into the platform. This addition would provide the robot with a 360° environmental scan, allowing it to dynamically map obstacles, identify distant warehouse zones, and execute smooth, collision-free path planning entirely independent of any physical lines.

Finally, an improvement would be to connect multiple robots together using a central computer over Wi-Fi. This computer would act like a traffic controller, booking safe driving zones on the open floor for each robot so they can take shortcuts and overtake each other without crashing. In the current implementation, the overtaking logic is binary: one robot is explicitly designated to stop while the other executes the manoeuvre. To improve this, the system can transition to a multi-robot communication network capable of handling complex scenarios with more than two platforms. Rather than forcing a vehicle to come to a complete standstill, the server can resolve spatial conflicts in real time, calculating decentralised velocity profiles that allow a robot to merely slow down while the overtaking asset sweeps past.

References

- [1] Nizar Abbas and Ahmed Sami. “Tuning of PID Controllers for Quadcopter System using Cultural Exchange Imperialist Competitive Algorithm”. In: *Engineering Journal* 24 (Feb. 2018), pp. 80–99. DOI: [10.31026/j.eng.2018.02.06](https://doi.org/10.31026/j.eng.2018.02.06).
- [2] Farshad Arvin et al. “Mona: an Affordable Open-Source Mobile Robot for Education and Research”. In: *Journal of Intelligent and Robotic Systems: Theory and Applications* 94.3-4 (2019), pp. 761–775. DOI: [10.1007/s10846-018-0866-9](https://doi.org/10.1007/s10846-018-0866-9). URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85047239237&doi=10.1007%2fs10846-018-0866-9&partnerID=40&md5=b4da0ec639a668b405cd210ac622d910>.
- [3] Karl Johan Åström and Tore Hägglund. *PID Controllers: Theory, Design, and Tuning*. 2nd. Research Triangle Park, NC: Instrument Society of America, 1995.
- [4] Tony Belpaeme et al. “Social robots for education: A review”. In: *Science Robotics* 3.21 (Aug. 2018). DOI: [10.1126/scirobotics.aat5954](https://doi.org/10.1126/scirobotics.aat5954). URL: <https://www.science.org/doi/10.1126/scirobotics.aat5954>.
- [5] Pujan Bhandari and P. Z. Csursia. “Digital implementation of the PID controller”. In: *Softw. Impacts* 13 (2022), p. 100306. DOI: [10.1016/j.simpa.2022.100306](https://doi.org/10.1016/j.simpa.2022.100306).
- [6] Ouriel Bliah et al. “Fabrication of Soft Robotics by Additive Manufacturing: From Materials to Applications”. In: *Chemical Reviews* 125 (2025), pp. 7275–7320. DOI: [10.1021/acs.chemrev.4c00749](https://doi.org/10.1021/acs.chemrev.4c00749).
- [7] Johann Borenstein and Liqiang Feng. “Measurement and correction of systematic odometry errors in mobile robots”. In: *IEEE Transactions on Robotics and Automation* 12.6 (1996), pp. 869–880. DOI: [10.1109/70.544770](https://doi.org/10.1109/70.544770). URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-0030414418&doi=10.1109%2f70.544770&partnerID=40&md5=5fafel109f3c59cb11aeeada6e9294ad1>.
- [8] João Braun et al. “Robot@Factory Lite: An Educational Approach for the Competition with Simulated and Real Environment”. In: *Advances in Intelligent Systems and Computing* 1092 AISC (2020), pp. 478–489. DOI: [10.1007/978-3-030-35990-4_39](https://doi.org/10.1007/978-3-030-35990-4_39). URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85082101860&doi=10.1007%2f978-3-030-35990-4_39&partnerID=40&md5=9c420afc9f794d25cca3d143a11171c0.
- [9] Williamson Johnny Hatzinakis Brigido and Jose M. Parente de Oliveira. “The line follower robot: a meta-analytic approach”. In: *PeerJ Computer Science* 11 (2025). DOI: [10.7717/PEERJ-CS.2744](https://doi.org/10.7717/PEERJ-CS.2744). URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-105001564248&doi=10.7717%2fPEERJ-CS.2744&partnerID=40&md5=1c257754ab7aacfd8981311f8e01bd4e>.

- [10] Stephen J. Chapman. *Electric Machinery Fundamentals*. 5th. McGraw-Hill Education, 2011. ISBN: 978-0073529547.
- [11] Paulo Costa and José Lima. *Rules for Robot@Factory Lite 2026*. Tech. rep. Competition rules document. Porto, Portugal: Festival Nacional de Robótica / Portuguese Robotics Open, Nov. 2025. URL: <https://github.com/P33a/RobotAtFactoryLite>.
- [12] Luiz E. Luiz et al. “Robot at Factory Lite - A Step-by-Step Educational Approach to the Robot Assembly”. In: *Lecture Notes in Networks and Systems* 589 LNNS (2023), pp. 550–561. DOI: 10.1007/978-3-031-21065-5_45. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85145039529&doi=10.1007%2f978-3-031-21065-5_45&partnerID=40&md5=27d196a22bcb87440a5e9a8d1ab0fd62.
- [13] Salomi Evripidou et al. “Selecting a Robotic Platform for Education”. In: *2022 IEEE International Conference on Consumer Electronics (ICCE)*. 2022, pp. 1–6. DOI: 10.1109/icce53296.2022.9730568.
- [14] Giuseppe Fragapane, René de Koster, and Fabio Sgarbossa. *Autonomous Mobile Robots for Material Handling in Intralogistics*. 2024, pp. 251–274. DOI: 10.1007/978-3-031-50273-6_10. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-105004534026&doi=10.1007%2f978-3-031-50273-6_10&partnerID=40&md5=1e24b66839a64e57ccab32b7f96d82f8.
- [15] Giuseppe Fragapane et al. “Increasing flexibility and productivity in Industry 4.0 production networks with autonomous mobile robots and smart intralogistics”. In: *Annals of Operations Research* 308.1-2 (2022), pp. 125–143. DOI: 10.1007/s10479-020-03526-7. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85079229930&doi=10.1007%2fs10479-020-03526-7&partnerID=40&md5=31c3ef6a2da53f11f0cdf9fe5ce504a6>.
- [16] Giuseppe Fragapane et al. “Planning and control of autonomous mobile robots for intralogistics: Literature review and research agenda”. In: *European Journal of Operational Research* 294.2 (2021), pp. 405–426. DOI: 10.1016/j.ejor.2021.01.019. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85100547833&doi=10.1016%2fj.ejor.2021.01.019&partnerID=40&md5=5327ae392b42c50e983fccfce9cb3de3>.
- [17] Gene F. Franklin, J. David Powell, and Michael L. Workman. *Digital Control of Dynamic Systems*. 3rd. Addison-Wesley, 1998. ISBN: 978-0201820546.
- [18] Xixian Huang. “The rise of educational robots: A review of classroom applications”. In: *Applied and Computational Engineering* 38.1 (Feb. 2024), pp. 57–65. DOI: 10.54254/2755-2721/38/20230529. URL: <https://doi.org/10.54254/2755-2721/38/20230529>.
- [19] Hassan K. Khalil. *Nonlinear Systems*. 3rd. Pearson, 2002. ISBN: 978-0130673893.
- [20] Stefanie Krause, Anna-Lena Henk, and Frieder Stolzenburg. “Selecting an Educational Robot: A Comprehensive Guideline”. In: *Lecture Notes in Networks and Systems* 1084 LNNS (2024), pp. 263–274. DOI: 10.1007/978-3-031-67059-6_23. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85206088461&doi=10.1007%2f978-3-031-67059-6_23&partnerID=40&md5=6b6390c00dd80e9636090f7dcd00169c.

- [21] Jose Lima et al. “An Industry 4.0 Approach for the Robot@Factory Lite Competition”. In: *2020 IEEE International Conference on Autonomous Robot Systems and Competitions, ICARSC 2020*. 2020, pp. 239–244. DOI: [10.1109/ICARSC49921.2020.9096164](https://doi.org/10.1109/ICARSC49921.2020.9096164). URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85085916880&doi=10.1109%2fICARSC49921.2020.9096164&partnerID=40&md5=bf344dd0c82166759998ba98ae0857f>.
- [22] Lennart Ljung. *System Identification: Theory for the User*. 2nd. Prentice Hall, 1999. ISBN: 978-0136566953.
- [23] Maria S. Lopes et al. “From Competition to Classroom: A Hands-on Approach to Robotics Learning”. In: *IEEE International Conference on Autonomous Robot Systems and Competitions, ICARSC 2025 (2025)*, pp. 170–176. DOI: [10.1109/ICARSC65809.2025.10970153](https://doi.org/10.1109/ICARSC65809.2025.10970153). URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-105005714510&doi=10.1109%2fICARSC65809.2025.10970153&partnerID=40&md5=4a04ea8baf25aaad507fe984fa4e79bd>.
- [24] Kashif Mahmood, K. Karjust, and T. Raamets. “Production Intralogistics Automation Based on 3D Simulation Analysis”. In: *Journal of Machine Engineering* 21 (2021), pp. 101–115. DOI: [10.36897/jme/137081](https://doi.org/10.36897/jme/137081).
- [25] Sharul Fitry Abdul Majid et al. “Optimized Global Path Planning with SLAM for Efficient Warehouse Autonomous Mobile Robot”. In: *2025 IEEE 6th International Conference in Robotics and Manufacturing Automation (ROMA)*. 2025, pp. 377–381. DOI: [10.1109/roma66616.2025.11155527](https://doi.org/10.1109/roma66616.2025.11155527).
- [26] Felipe N. Martins et al. “An adaptive dynamic controller for autonomous mobile robot trajectory tracking”. In: *Control Engineering Practice* 16.11 (2008), pp. 1354–1363. DOI: [10.1016/j.conengprac.2008.03.004](https://doi.org/10.1016/j.conengprac.2008.03.004). URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-47549093920&doi=10.1016%2fj.conengprac.2008.03.004&partnerID=40&md5=4032857b89b41c9f2d2677f37f281077>.
- [27] Eva Mötshärg et al. “Robotont 3—an accessible 3D-printable ROS-supported open-source mobile robot for education and research”. In: *Frontiers in Robotics and AI* 11 (2024). DOI: [10.3389/frobt.2024.1406645](https://doi.org/10.3389/frobt.2024.1406645).
- [28] D.K. Nhan and T.T.K. Ly. “Feedback dynamic control design of differential drive wheeled mobile robot”. In: *International Conference on Material, Machines and Methods for Sustainable Development* (2025). URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-105036848615&partnerID=40&md5=cc3c14c41ddc79e15553ef9024a88705>.
- [29] Norman S. Nise. *Control Systems Engineering*. 8th. John Wiley & Sons, 2019. ISBN: 978-1119590132.
- [30] Katsuhiko Ogata. *Modern Control Engineering*. 5th. Prentice Hall, 2010. ISBN: 978-0136156734.
- [31] S. J. Oks et al. “Cyber-Physical Systems in the Context of Industry 4.0: A Review, Categorization and Outlook”. In: *Information Systems Frontiers* 26 (2022), pp. 1731–1772. DOI: [10.1007/s10796-022-10252-x](https://doi.org/10.1007/s10796-022-10252-x).
- [32] Paulo A. F. Rezeck et al. “HeRo 2.0: a low-cost robot for swarm robotics research”. In: *Autonomous Robots* 47 (2022), pp. 879–903. DOI: [10.1007/s10514-023-10100-0](https://doi.org/10.1007/s10514-023-10100-0).
- [33] Bruno Siciliano and Oussama Khatib, eds. *Springer Handbook of Robotics*. 2nd. Springer, 2016. ISBN: 978-3319325507.

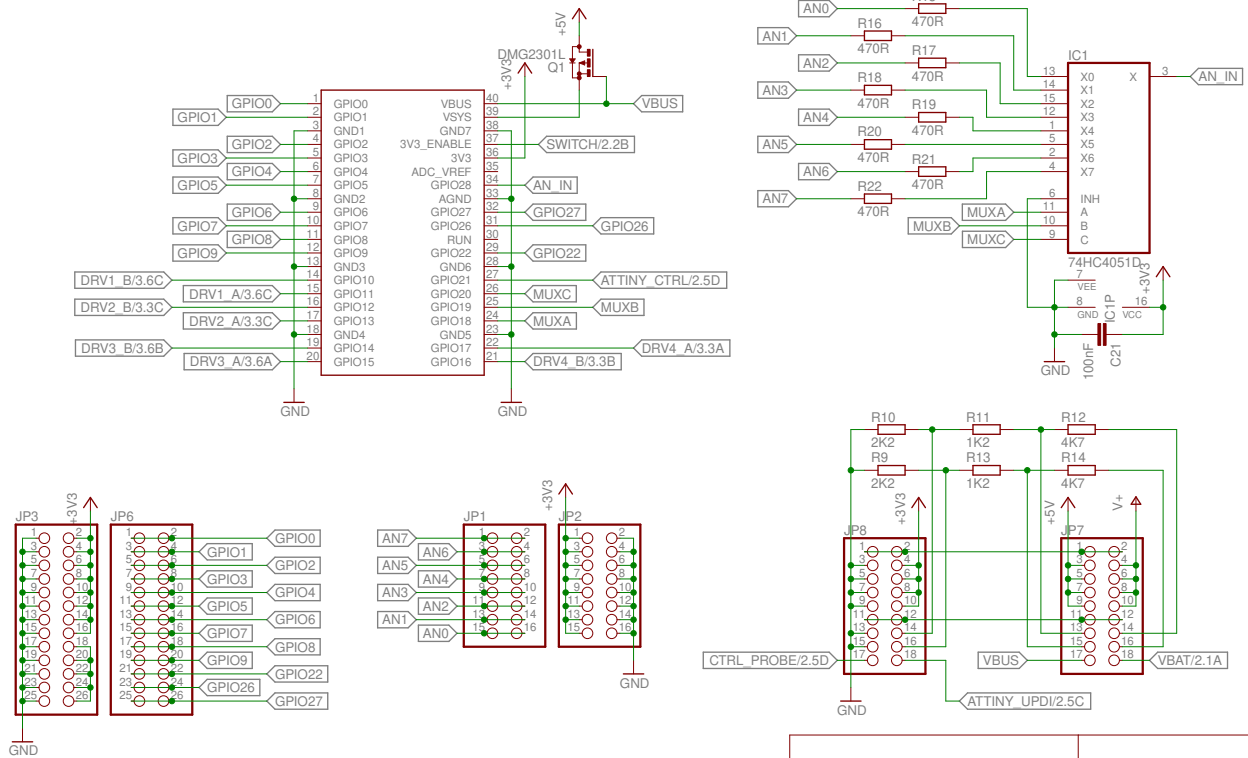
- [34] Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza. *Introduction to Autonomous Mobile Robots*. 2nd. Cambridge, MA: MIT Press, 2011.
- [35] Mark W. Spong, Seth Hutchinson, and M. Vidyasagar. *Robot Modeling and Control*. 2nd. John Wiley & Sons, 2020. ISBN: 978-1119523994.
- [36] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, 2005. ISBN: 978-0262201629.
- [37] Claudio Urrea and J. Kern. “Recent Advances and Challenges in Industrial Robotics: A Systematic Review of Technological Trends and Emerging Applications”. In: *Processes* (2025). DOI: [10.3390/pr13030832](https://doi.org/10.3390/pr13030832).
- [38] Jose Varela-Aldás and Guillermo Palacios-Navarro. “A ROS-Based Open Tool for Controlling an Educational Mobile Robot”. In: *International journal of online and biomedical engineering* 20.1 (2024), pp. 23–39. DOI: [10.3991/ijoe.v20i01.46573](https://doi.org/10.3991/ijoe.v20i01.46573). URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85184247058&doi=10.3991%2fijoe.v20i01.46573&partnerID=40&md5=1cfc437b74bb00306c55e838281ea332>.
- [39] K. Walia, Ahmed Khan, and P. Breedon. “Polymer-Based Additive Manufacturing: Process Optimisation for Low-Cost Industrial Robotics Manufacture”. In: *Polymers* 13 (2021). DOI: [10.3390/polym13162809](https://doi.org/10.3390/polym13162809).
- [40] Jingjin Yu et al. “A portable, 3D-printing enabled multi-vehicle platform for robotics research and education”. In: *Proceedings - IEEE International Conference on Robotics and Automation*. 2017, pp. 1475–1480. DOI: [10.1109/ICRA.2017.7989176](https://doi.org/10.1109/ICRA.2017.7989176). URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85027980435&doi=10.1109%2fICRA.2017.7989176&partnerID=40&md5=d4e4ee4157ce936f19e46a921bc1807c>.

Chapter 7

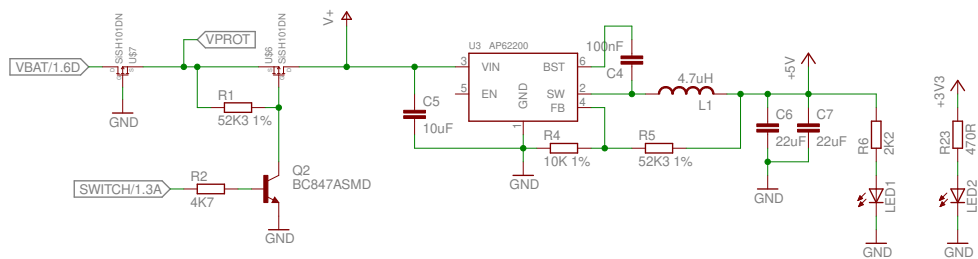
Appendix 1

This appendix presents the electrical schematics of the connection board Pico4Drive.

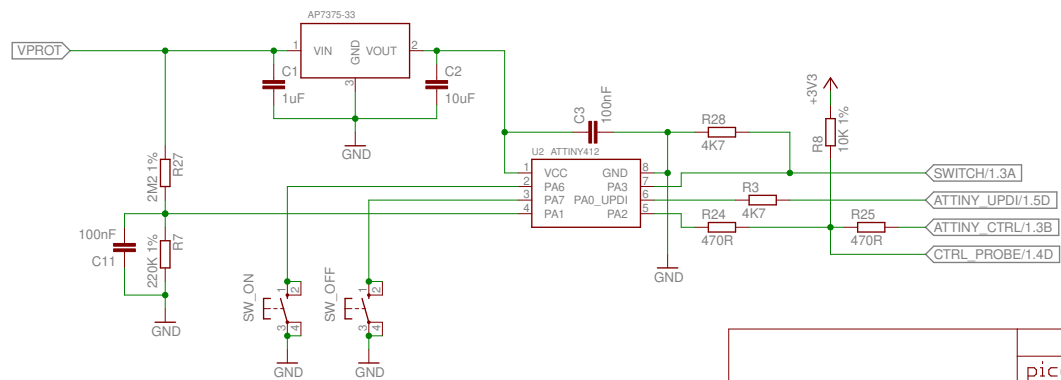
HEADERS AND ANALOG MUX



POWER SUPPLY



POWER CONTROLLER



FULL BRIDGE DRIVERS

