POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

Adaptive Formation Control and Navigation for Multi-Robot Tethering Cooperative Transport

Supervisors

Prof. Marina INDRI
Prof. Pangcheng David
CEN CHENG

Candidate

Andrea ESPOSITO

March 2026

Table of Contents

Abstract

In mobile robotics, cooperative multi-robot systems enable teams of agents to tackle tasks that exceed the capabilities of individual agents. Cooperative transport is highly demanding, as it requires coordinated motion, load handling, and navigation in obstacle-rich environments, while managing the physical coupling between robots. The overall goal is to design coordination strategies that ensure safe and efficient transport under both coupling constraints and environmental limitations.
Therefore, this approach involves some technical challenges: maintaining formation consistency during navigation, coordinating motion through physical connections, managing obstacles, and ensuring operational safety. Addressing these issues requires an integrated approach that combines path planning with real-time formation control.
This thesis addresses cooperative transport in indoor environments with static obstacles by using differential-drive mobile robots equipped with 2D LiDAR sensors and coupled via a flexible tether. The experimental transport setup consists of two robots carrying vertical poles, the tether connects the pole tips, and the transported object is attached at the midpoint of the tether. The flexible tether not only transmits traction forces but also constrains the relative motion of the robots, thus introducing additional restrictions that must be explicitly managed during navigation.
The system is developed in ROS 2 and extended to multi-robot operation. A baseline leader–follower controller is first implemented to establish reproducible formation tracking and safety behaviours. Building on this foundation, the work proposes a centralized "virtual frame" architecture, in which formation objectives are defined at the group level rather than individually for each robot. As a result of this approach, standard navigation components can be used for continuous replanning as the team navigates complex environments.
In order to address the coupling constraint imposed by the tether, a supervisory module monitors obstacle proximity and implements state-based control with hysteresis-based transitions. In the case of obstacles that threaten the tether or payload clearance, the system dynamically adapts the formation geometry by modulating inter-robot distance, therefore ensuring smooth transitions while maintaining coordination.
Experimental validation on Turtlebot3 Burger shows robust navigation in cluttered environments and an effective resolution of critical configurations through coordinated formation adaptation. This thesis describes the complete system implementation, discusses design choices and challenges encountered, and outlines possible future extensions.

Chapter 1

Introduction

Cooperative multi-robot systems represent a significant area of research in mobile robotics, as it addresses scenarios where multiple autonomous agents must work together to accomplish tasks that cannot be efficiently or safely performed by a single robot. Applications range from warehouse automation and environmental monitoring to search and rescue operations, where coordination between multiple platforms not only guarantees improved coverage and robustness but also enhances task execution capabilities.

Among cooperative tasks, coordinated transport is the most challenging. because it requires the simultaneous management of multiple constraints: maintaining desired formation geometry, executing collision-free motion in complex environments, coordinating individual robot actions to achieve team-level objectives, and handling physical interactions when robots are mechanically coupled to shared payloads or to each other. These features become even more demanding in indoor environments, where limited space, static obstacles, narrow passages, and constrained manoeuvring areas impose additional operational restrictions.

Formation control strategies for multi-robot systems have been extensively studied in the literature, with approaches ranging from behaviour-based methods to optimization-based techniques. Many existing solutions rely on the assumption that environments are obstacle-free or weakly constrained, so that purely geometric formation tracking can be effectively maintained through relative positioning alone. In cluttered indoor settings, however, this premise holds up no longer: robots must explicitly account for environmental constraints while coordinating their motion, therefore requiring an integration between formation control and obstacle-aware navigation.

Physical coupling of robots adds a further level of complexity in terms of coordination. It introduces mechanical constraints on relative motion and configuration space, while also requiring careful management of load distribution and clearance to avoid collisions with ground obstacles or environmental structures. As existing navigation frameworks are typically designed for single-robot operation, they must be extended and adapted to support coordinated multi-robot motion under these additional constraints.

This work addresses both the theoretical challenge of coordinating formation geometry with obstacle avoidance and the practical challenge of implementing robust, experimentally validated solutions by using standard mobile robotics platforms and software frameworks.

1.1 Thesis goals

The primary objective of this thesis is to design and implement a multi-robot formation control system capable of maintaining coordinated motion in cluttered indoor environments, while also managing the constraints imposed by the flexible physical coupling between the robots. The work then goes on to investigate whether a centralized reference frame can provide a more effective foundation for coordinating team-level objectives with individual robot navigation capabilities, addressing the limitations of pure relative tracking approaches.

Overall, the key challenge is integrating formation control with obstacle-aware navigation. Standard formation tracking approaches usually generate commands only based on the geometric relationships between the robots, without explicitly accounting for environmental constraints such as walls, narrow passages, or ground-level obstacles. Therefore, this thesis explores how to separate these concerns by defining formation geometry at the group level, while delegating obstacle avoidance and path planning to the individual robot navigation systems. It also explains how to design interfaces between these layers that maintain both coordination and safety.

Furthermore, when robots are physically coupled through flexible connections, additional constraints emerge, which are related to the achievable relative configurations and payload clearance management. This thesis investigates how to monitor environmental conditions and coordinate formation adaptation in response to obstacle proximity, adjusting inter-robot separation to ensure safe payload transport while maintaining operational continuity and avoiding oscillatory behaviours near decision boundaries.

From an implementation perspective, this work aims to develop solutions that are deployable on standard mobile robotics platforms with widely adopted software frameworks. It also focuses on demonstrating that hierarchical coordination strategies can be effectively integrated with existing navigation capabilities, thus producing systems that can be implemented and experimentally validated in real-world indoor scenarios. Ultimately, this thesis seeks to suggest both a concrete architectural solution for cooperative transport under flexible coupling, and broader insights into structuring multi-robot coordination in order to leverage single-robot

autonomy while addressing team-level constraints.

1.2 Thesis structure

This thesis is organized as follows:
Chapter 2: Background provides an overview of control and transport strategies for cooperative multi-robot systems, positioning the proposed approach within the broader context of formation control and discussing the motivations for adopting a centralized virtual frame architecture with flexible physical coupling.

Chapter 3: TurtleBot3 Burger Overview describes the hardware platform used in this work, including the mechanical structure, control and processing system, actuators, and sensors. This chapter defines the operational constraints and capabilities that inform the subsequent design choices.

Chapter 4: ROS 2 Jazzy Framework illustrates the software infrastructure underlying the implementation, covering the distributed architecture, the computational graph structure, communication paradigms (topics, services, actions), the TF2 for reference frame management, and the multi-robot setup required for coordinated operation. This chapter also discusses SLAM for map building, localization on known maps, and the Nav2 navigation stack used for obstacle-aware motion planning.

Chapter 5: Baseline Leader–Follower Control establishes a reference implementation for formation tracking, formalizing the problem statement, defining formation geometry and error variables, and implementing a proportional control law with a finite-state machine for safety management. This baseline provides reproducible behaviour and validates fundamental assumptions before introducing more complex coordination strategies.

Chapter 6: Centralized Control with Virtual Frame introduces the main contribution of this work: a hierarchical architecture based on a virtual reference frame that defines formation objectives at the group level. This chapter describes the virtual centre implementation, dynamic goal tracking through a custom Nav2 Behaviour Tree with online goal updates, and a supervisory state machine for obstacle management and coordinated formation adaptation under coupling constraints.

Chapter 7: Conclusions and Future Works summarizes the topics of the work, discusses experimental validation results, acknowledges current limitations, and outlines directions for future extensions, including larger teams, predictive ob-

stacle management, semantic perception integration, and learning-based parameter optimization.

Chapter 2

Background

Multi-agent robotic systems have become a core paradigm in mobile robotics, enabling cooperation, coordination, and workload distribution among multiple autonomous platforms. By operating as a team, robots can address tasks that exceed the capabilities of a single agent, including the transport and manipulation of bulky, heavy, or kinematically constrained loads.

One of the main challenges of cooperative transport is that the system behaviour is a result of the coupled dynamics between the robots and the carried object. Load motion depends on how interaction forces are generated and distributed, and on how robot movements are synchronized, which makes stability, robustness, and tracking accuracy non-trivial, especially under uncertainty and operational constraints.

Designing cooperative transport systems, therefore, requires an integrated perspective that combines dynamic modeling, control design, motion planning, and inter-robot coordination. For this reason, load-state information availability, sensing and communication capabilities, and the degree of environment structuring are key factors that directly influence design trade-offs in terms of computational burden, centralization, and robot autonomy.

Within this framework, the chosen approach can be organized into both control strategies, which specify the laws used to achieve the transport objectives, and transport strategies, which describe how robots physically interact with the load and the environment. The interplay between these two aspects ultimately determines system performance and the range of manageable operating conditions.

2.1 Control strategies

Control strategies for cooperative multi-robot transport aim to coordinate individual agents so that the load follows the desired configuration or trajectory. Their formulation depends on the control objectives, the robot and actuator characteristics, and the available information about the load and the environment. Furthermore, the way in which sensing, computation, and communication are distributed within the

system is crucial in determining the design choice, and it's on this basis that control architectures are commonly grouped into fully centralized, partially decentralized, and fully decentralized approaches [1].

In fully centralized architectures, state estimation and control computation are performed by a single unit, which may be an external supervisor or a robot with enhanced capabilities. Centralization can yield accurate load regulation and a consistent global behaviour, but it typically limits scalability and introduces sensitivity to failures or degradation of the central node.

Partially decentralized approaches balance global coordination with local autonomy by keeping some functions centralized while distributing others across the team. One of the most well-known examples is the leader–follower scheme, where one or more agents provide reference motion for the load, and the other robots adapt their actions using received or locally inferred information. This structure both reduces coordination complexity and preserves a degree of robustness and operational flexibility.

Finally, fully decentralized strategies do not rely on a central component, as they require each robot to compute its control input based on local measurements and, when available, the limited information it can gather from nearby agents. Such designs are well-suited to large teams and to scenarios with a degree of uncertainty or structural limitations, where scalability and fault tolerance are critical elements. The main issue is ensuring closed-loop stability and high performance even under partial information and with distributed decision-making.

The proposed solution follows a centralized formulation based on a virtual frame that provides a global reference for the formation. The virtual frame serves as the leader, while each robot acts as a follower and adjusts its motion according to the frame's position and speed. This model provides robust coordination, decoupling global motion specification from local command generation and simplifying the management of multi-robot synchronization during transport.

2.2 Transport strategies

Transport strategies in cooperative multi-robot transport describe how robots physically interact with the load and with one another to generate and maintain motion. These strategic choices shape the coupled system dynamics, affect load maneuverability, and determine the effort required to coordinate the agents [2].

First of all, the interaction between robots and objects must be taken into account. In pushing-based transport, for example, robots are not mechanically connected to the load and induce motion through contact forces. This setup is mechanically simple, but it requires accurate coordination to align forces and preserve motion stability, especially with friction variability, uneven terrain, or curved trajectories.

In gripping or hooking strategies, robots physically attach to the load, allowing better control of both translation and rotation. This is especially advantageous when high precision is required, or environmental constraints need to be overcome. The main drawback of these approaches is that they not only increase the complexity of both mechanics and control, but also call for greater consistency in managing contact locations and interaction constraints across the team.

A further category is represented by caging strategies, where robots surround the load and create a configuration that limits its effective degrees of freedom, keeping it within a bounded region during transport. Here, the load motion depends on coordinated group behaviour without a rigid grasp, even though sustained performance requires continuously maintaining the enclosure, in order to prevent a loss of constraint and the consequent degradation of control.

Transport strategies can also be categorized by the geometric organization of the team. Fixed formations maintain the relative arrangement of agents over the transport phase, which simplifies modeling and controller design. Variable formations allow robots to change their relative positions or even the interaction mode, improving adaptability to dynamic environments at the cost of additional coordination and planning complexity.

The proposed solution adopts a strategy based on the flexible physical coupling between robots by means of a rope. This originates a constrained interaction that transmits traction forces while limiting relative configuration, but without enforcing rigid kinematic constraints. The resulting setup reduces mechanical complexity and simplifies control design, while preserving the coordination required for robust cooperative transport, as shown in Figure 2.1.

Figure 2.1: Experimental setup for cooperative transport with rope-based coupling.

Chapter 3

Turtlebot3 Burger Overview

The TurtleBot3 Burger is a differential-drive mobile platform widely adopted for developing and validating mobile robotics algorithms. Its hardware stack combines onboard actuation, sensing, and computing resources with the aim of supporting core functionalities such as state estimation, navigation, and motion control in structured environments.

In this work, the TurtleBot3 Burger is employed to implement and experimentally assess the proposed solutions. The following sections describe the main hardware components of the platform, with emphasis on the elements that most directly influence system performance and operational constraints.

3.1 Architecture

The TurtleBot3 Burger adopts a three-level modular mechanical structure, referred to as the Waffle Plate, which enables compact integration of the main onboard subsystems. The platform uses a differential-drive base with two driven wheels and a passive rear caster, providing a simple and effective solution for indoor locomotion. The Waffle Plates and supporting brackets are distributed by the manufacturer as open-hardware CAD models, allowing mechanical customization and the integration of additional sensors or payloads, when required.

The stacked layout allows a clear physical separation of functionalities: the lower level hosts the locomotion base, while the upper levels accommodate electronics and sensing modules. The robot measures 138 mm $\times$ 178 mm $\times$ 192 mm (L $\times$ W $\times$ H) overall, and has a mass of approximately 1 kg, including the single-board computer, the battery, and sensors, with a declared maximum payload of 15 kg [3]. This form factor is well suited to experiments in confined indoor environments, such as those outlined in the scenarios covered in this work, as illustrated in Figure 3.1.

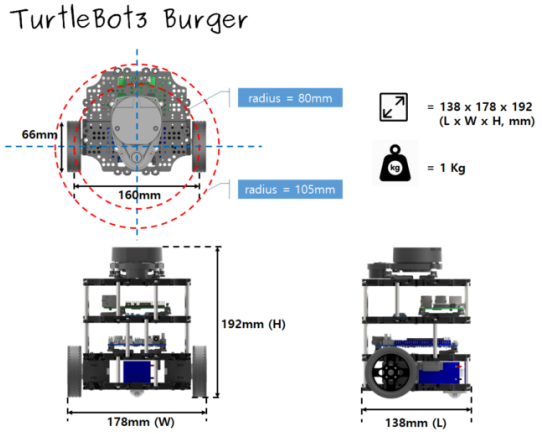


Figure 3.1: TurtleBot3 Burger platform overview [4]

3.2 Control and processing system

The TurtleBot3 Burger separates low-level real-time control from high-level computation by using two main components: an OpenCR 1.0 embedded board and an onboard Single Board Computer (SBC) based on a Raspberry Pi. The OpenCR 1.0 board, built around a 32-bit ARM Cortex-M7 microcontroller, executes time-critical functions such as actuator interfacing, wheel encoder acquisition and odometry computation, onboard sensor sampling (e.g., IMU), and serial communication with the DYNAMIXEL motors [5], as depicted in Figure 3.2.

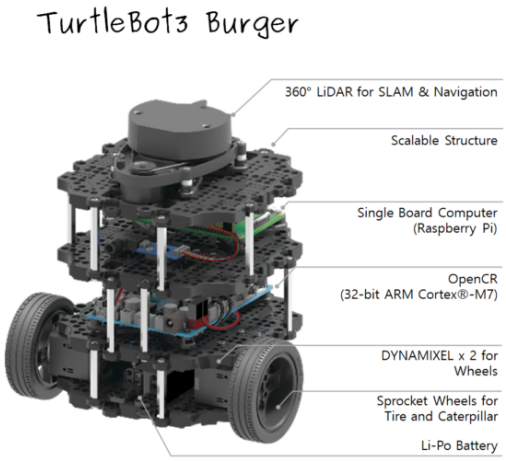
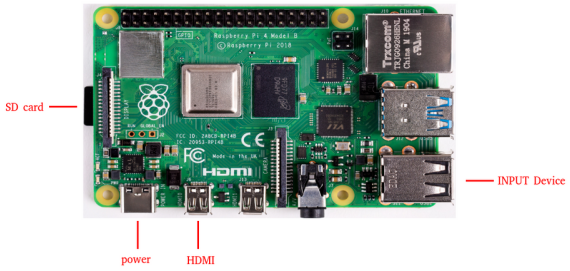
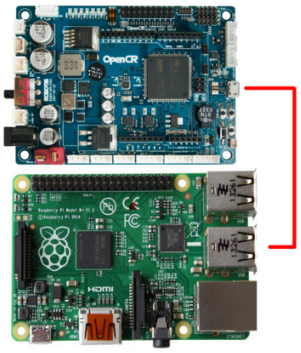


Figure 3.2: TurtleBot3 Burger platform architecture[6]

On the other side, the SBC is responsible for the high-level processing: it runs Linux and the ROS 2 middleware together with the autonomy modules employed in this project (e.g., localization, perception, and planning). The OpenCR and the SBC communicate via a USB link that supports bidirectional exchange of velocity commands, sensor measurements, and diagnostic data. The same interface is also used for firmware deployment and maintenance of the embedded controller. In the experimental setup considered in this thesis, the SBC is a Raspberry Pi 4 [7], as shown in Figure 3.3.



(a) Raspberry Pi 4[7]



(b) OpenCR controller board interfaced with the Raspberry Pi 4[5]

Figure 3.3: Onboard computation and control architecture

3.3 Actuators and locomotion

The TurtleBot3 Burger uses two ROBOTIS DYNAMIXEL XL430-W250-T smart actuators, directly coupled to the left and right drive wheels. Each XL430-W250-T integrates an embedded controller with PID control, a contactless absolute position sensor (12-bit over 360°) with 4096 pulses per revolution, and supports multiple operating modes (velocity, position, extended position, and PWM/voltage control). Communication is provided through a TTL half-duplex multidrop serial bus, with selectable baud rates from 9.6 kbps up to 4.5 Mbps, that enables closed-loop regulation of the wheel motion without external motor drivers [8].

The platform specifications report a maximum translational velocity of 0.22 m/s and a maximum rotational velocity of 2.84 rad/s (162.72 deg/s). The declared climbing threshold is 10 mm or lower, which restricts the robot's ability to climb small steps and cross uneven surfaces typically found in indoor test areas. These

values define the nominal operating envelope used in the motion control and coordination experiments presented in this work [9].

3.4 Sensors

Environmental perception is primarily provided by a rotating 2D LiDAR mounted on the upper plate of the robot: LDS-01 in earlier configurations and LDS-02 in updated versions. The LDS-01 delivers 360° planar scans with a specified range of 120 mm to 3.5 m, 1° angular resolution, and a nominal scan rate of 300 ± 10 rpm; it can be interfaced either through a USB2LDS adapter or via a direct serial connection [10]. The LDS-02 provides 360° coverage with 1° resolution as well, but it has a distance range of 160 mm to 8 m, and a scan frequency of 5 Hz or higher; measurements are transmitted over a one-way serial interface and are typically connected to the SBC through USB2LDS. These scans serve as the main sensing input for mapping, obstacle detection, and localization [11], as shown in Figure 3.4.

In this work, LiDAR measurements are integrated within the ROS 2 SLAM and navigation pipeline (Nav2), which uses them for online map construction, obstacle avoidance, and path planning in indoor environments.

Figure 3.4: TurtleBot3 Burger 2D LiDAR[10]

In addition, the OpenCR 1.0 controller integrates a 9-axis IMU (MPU9250), comprising a three-axis gyroscope, an accelerometer, and a magnetometer. Together with wheel-encoder feedback, inertial measurements contribute to state estimation by improving short-term attitude tracking and compensating for the limitations of wheel odometry, which is particularly relevant during navigation and for multi-robot coordination.

In this work, IMU and encoder signals are fused within the state estimation pipeline to obtain a more reliable pose estimate, especially during rapid rotations or in the presence of wheel slip, where purely encoder-based odometry can degrade.

Chapter 4

ROS2 Jazzy Framework

ROS, which stands for Robot Operating System, has become a widely adopted framework for developing complex robotic applications, as it provides not only a modular software ecosystem, but also standardized interfaces across heterogeneous subsystems. It was ROS 1 in particular that accelerated the spread of reusable tools and packages for visualization, simulation, navigation, and reference-frame management. However, as robotic systems increasingly shifted toward distributed and multi-agent deployments, ROS 1 began to show structural limitations, such as its reliance on centralized coordination, reduced robustness under degraded network conditions, and the lack of built-in security mechanisms. In light of this, the present work adopts ROS 2 as the middleware layer to support multi-robot coordination and distributed communication.

4.1 Architecture

ROS 2 design aimed at addressing the requirements of distributed robotic systems, while preserving the modular design principles established by ROS 1. Its communication layer is built on the Data Distribution Service (DDS), which allows the configuration of communication behaviours through Quality of Service (QoS) profiles, and provides more structured support for security. As a result, ROS 2 is better suited to multi-robot deployments and to scenarios where scalability and communication reliability are critical.

4.1.1 Computational graph and nodes

In ROS 2, the system is modelled as a computational graph in which software entities cooperate through a structured data exchange and service-like interactions [12]. Its basic unit of execution is the node: implemented either as an independent process or as a composable component within a shared process, it fulfils a well-defined function, such as sensor acquisition, state estimation, planning, or command generation. The system's architecture is inherently distributed; there is no central master, and nodes as well as their communication interfaces are discovered dynamically through middleware discovery mechanisms [13].

4.1.2 Topics and messages

Communication in ROS 2 is primarily implemented through topics, i.e., named, unidirectional channels over which strongly typed messages are exchanged. Message types are defined through ROS interface definitions (e.g., .msg) and generate language-specific data structures, thus ensuring consistent serialization and interpretation across heterogeneous nodes. A node, then, publishes data to a topic through a publisher and receives data through a subscriber, following a publish/subscribe paradigm that naturally supports many-to-many communication (multiple publishers and multiple subscribers on the same topic). For such reasons, this model not only is well suited to continuous and asynchronous data streams, such as sensor measurements, state estimates, and velocity commands, but it also promotes a modular system composition by decoupling producers from consumers, therefore allowing each component to evolve independently and to operate at different update rates.

4.1.3 Services and actions

Services in ROS 2 implement synchronous request/response communication between a client and a server and are typically used for short, transactional operations that produce an immediate result, such as parameter queries, mode switching, or routine triggering. The service interface is defined through a typed .srv specification, which enforces a consistent request and response structure across nodes.

In order to support long-running goals, this interaction model is generalized by actions. An action server accepts a goal request from a client, executes it asynchronously, and provides structured feedback during execution, followed by a final result message. In addition, actions support goal cancellation and preemption, which are essential when objectives must be updated online. For these reasons, actions are commonly adopted for navigation tasks (e.g., goal-reaching) and higher-level mission execution.

4.1.4 Parameters, namespaces, and remapping

Parameters in ROS 2 provide a standardized mechanism to configure node behaviour through key-value pairs associated with each node. They can be set at launch time or modified at runtime, allowing for the systematic tuning of algorithmic settings without having to recompile the software. Unlike ROS 1, parameters are not managed by a centralized server, but are exposed through the node interface, which is consistent with the distributed architecture and supports per-node configuration and introspection .

Namespaces and remapping complement this mechanism by controlling how nodes, topics, services, and actions are named within the ROS graph. On one side, namespaces group related entities under a common prefix, while on the other, remapping rules allow names to be overridden at launch without the need to change the source code. This feature proves to be crucial in multi-robot deployments, where the same software stack is instantiated multiple times, but the interfaces of each robot have to be kept isolated and uniquely identifiable.

4.1.5 Middleware layer and QoS

ROS 2 decouples the user-facing client libraries (e.g., rclcpp and rclpy) from the underlying transport by introducing a layered middleware interface. Application nodes are implemented on top of the common client layer (rcl) and communicate through a middleware abstraction (rmw), which allows for different DDS implementations to be selected without changes to the application code. This design improves portability and makes it possible to adapt the communication stack to specific deployment constraints.

By adopting DDS, ROS 2 provides Quality of Service (QoS) policies that explicitly control message delivery and resource management. QoS profiles allow reliability, durability, and history/queue depth to be configured, as well as timing-related properties, such as deadlines, lifetimes, and liveliness. This flexibility is critical when balancing latency, bandwidth, and robustness, particularly in multi-robot systems where network conditions and traffic patterns can vary significantly.

4.1.6 Execution model: callbacks and executors

ROS 2 follows an event-driven execution model in which communication events and timers trigger callbacks, including message receptions, service requests, and action feedback/result handling. Callback execution is managed by executors, which implement the scheduling policy and determine how callbacks are interleaved across one or more threads [14].

By separating callback definition and execution, you can explicitly control concurrency and responsiveness. In particular, single-threaded and multi-threaded executors, together with callback grouping, allow time-critical routines to be isolated, undesired re-entrancy to be avoided, and computational load to be managed. The same mechanisms also support node composition, where multiple components run within a single process to reduce overhead and improve data throughput.

4.1.7 TF2 and reference frames

TF2 provides the infrastructure for managing coordinate frames and time-stamped transformations, which is essential for consistent spatial reasoning in mobile robotics. It does so by maintaining a transformation tree that connects frames such as map, odom, and base_link, thereby allowing measurements and estimates expressed in different frames to be converted into a common representation.

Transformations are associated with timestamps and queries through a buffered interface, which provides interpolation and temporal alignment with the sensor data. This function is central to localization and navigation pipelines, where frame consistency is required to fuse heterogeneous measurements, interpret obstacle positions, and generate motion commands in the appropriate reference frame, as illustrated in Figure 4.1.

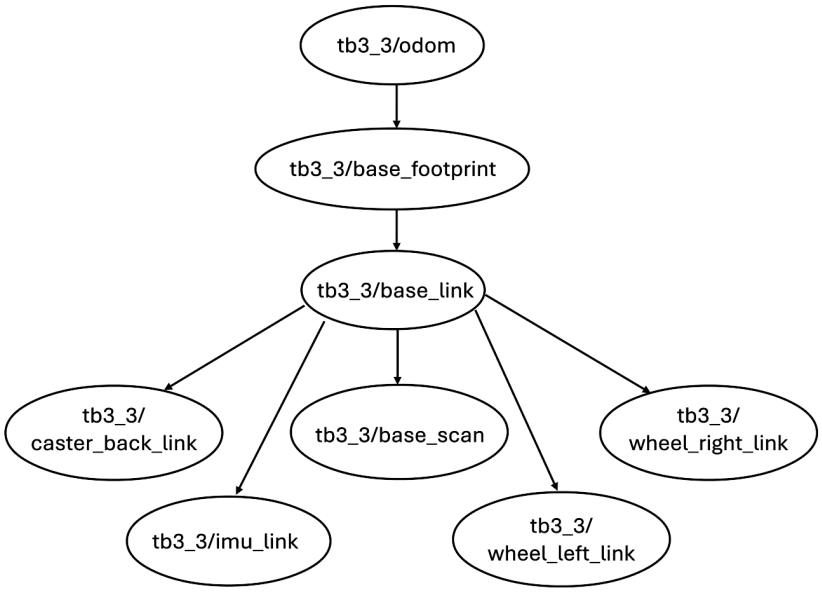


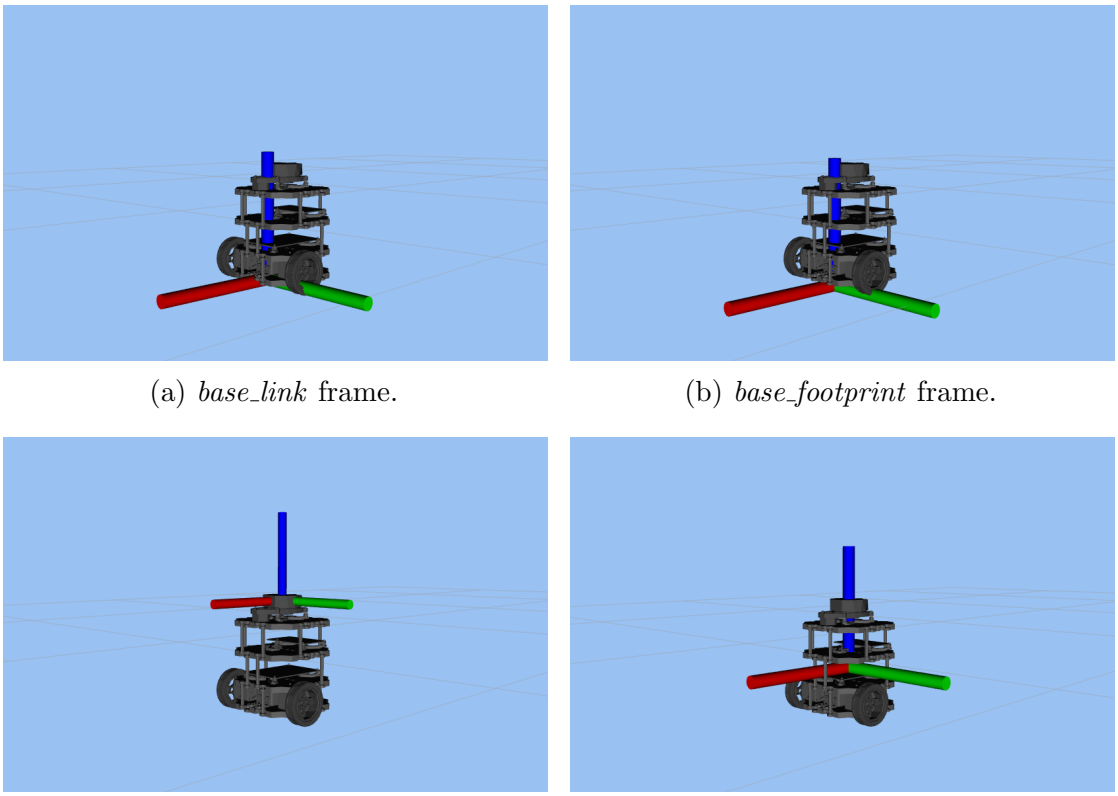
Figure 4.1: TF2 transformation tree for the TurtleBot3 Burger

4.2 Robot description

The robot model is described through the URDF (Unified Robot Description Format), which represents the system as a kinematic tree of rigid bodies (links) connected by joints [15]. In addition to the kinematic structure, the URDF can also specify geometric and physical properties (for instance, visual and collision geometry or inertial parameters). This model is used for visualization (e.g., in RViz)

and, more generally, to define a consistent set of reference frames for the robot, including the rigid mounting transforms of onboard sensors. RViz2 is the standard visualization tool in ROS 2 and is used in this work to inspect the robot model, TF frame transformations, and sensor data streams during SLAM and navigation [16].

For a differential-drive mobile base such as the TurtleBot3 Burger, the description typically includes a primary base frame (e.g., *base_link*) and a ground-projected frame (often *base_footprint*), together with frames for sensors mounted on the platform (e.g., LiDAR and IMU). Sensor frames are commonly defined as fixed joints in relation to the base, so that their poses are expressed through rigid transformations that remain constant over time. Frame names may vary across configurations, but the underlying principle remains unchanged: sensor measurements are interpreted through known, time-invariant extrinsics with respect to the robot base, as visualized in Figure 4.2.



(a) *base_link* frame. (b) *base_footprint* frame.

(c) *base_scan* (LiDAR) frame. (d) *imu_link* frame.

Figure 4.2: Reference frames defined by the TurtleBot3 URDF and visualized in RViz2. Red represents the x-axis, green the y-axis, and blue the z-axis.

At startup, the URDF is loaded into the *robot_description* parameter and is used by the *robot_state_publisher* node to publish the TF transforms of the kinematic model. While these transforms are static for fixed joints, they are updated based on the current joint states when associated with time-varying joints. In the presence of a mobile base, the robot's pose in the environment is not determined by the URDF: transforms such as *odom* → *base_link* are provided by odometry, and *map* → *odom* is produced by the localization system. This distinction separates the robot's geometric description from the dynamic estimation of its pose within the environment.

4.3 SLAM

Simultaneous Localization and Mapping (SLAM) describes a series of methods to estimate a robot's pose while simultaneously building a map of the environment from sensory measurements and motion information. In indoor mobile robotics with 2D LiDAR, SLAM aims to produce a consistent planar metric representation, typically an occupancy-grid map, together with a robust trajectory or pose estimate. By jointly exploiting laser scans and odometry, SLAM mitigates drift and compensates for uncertainty arising from measurement noise, occlusions, and geometric ambiguities in the environment.

4.3.1 Map Representation

In the considered workflow, the SLAM phase produces a 2D map in the form of an Occupancy Grid Map (OGM) [17], which is the standard representation adopted in ROS for planar environments. The map is typically visualized as a grayscale grid [18], where white cells denote free space, black cells represent occupied regions (obstacles), and grey cells correspond to unknown or unobserved areas. This representation can be used directly by costmap-based navigation modules and serves as the global reference for both localization and motion planning. In parallel, SLAM estimates the robot pose in the global frame, ensuring that incoming sensor measurements remain geometrically consistent with the evolving map.

4.3.2 2D LiDAR-based SLAM pipeline

In the 2D LiDAR-based SLAM, the primary input is a sequence of planar range scans acquired at a fixed rate and integrated over time as the robot moves. At a high level, the pipeline combines local motion estimation with scan alignment and global consistency mechanisms [19]. The main functional elements of the pipeline are summarised below:

- **Incremental motion prediction:** wheel-encoder measurements (linear and angular velocity increments, Δv and $\Delta\omega$) are integrated through the platform kinematic model to obtain a relative pose update between consecutive time steps (Δx, Δy, $\Delta\theta$). This prediction provides an initial guess for subsequent scan alignment, but it is affected by drift due to wheel slip, encoder imperfections, and unmodeled disturbances;

- **Scan matching:** the current scan is aligned with a local map or with recent scans to refine the relative motion estimate and improve metric consistency;

- **Loop closure:** when previously visited areas are detected, additional constraints are introduced to reduce accumulated drift and enforce global consistency;

- **Global optimization:** local alignment constraints and loop-closure constraints are combined in a pose-graph formulation, whose optimization yields a globally consistent trajectory estimate and an updated map.

These components are particularly relevant in indoor environments, where odometry drift can become significant and repetitive geometries (e.g., corridors or similar spaces) increase the risk of incorrect data association.

4.3.3 ROS 2 implementation: Cartographer

In this project, the SLAM phase was implemented through Cartographer [20], following the procedure reported in the official TurtleBot3 documentation for launching the SLAM node[21]. Within ROS 2, Cartographer runs as a dedicated node that incrementally estimates the robot pose and builds a 2D map by combining LiDAR measurements with motion information, provided either as odometry and/or as TF transformations.

Its integration within the ROS 2 ecosystem relies on three main elements:

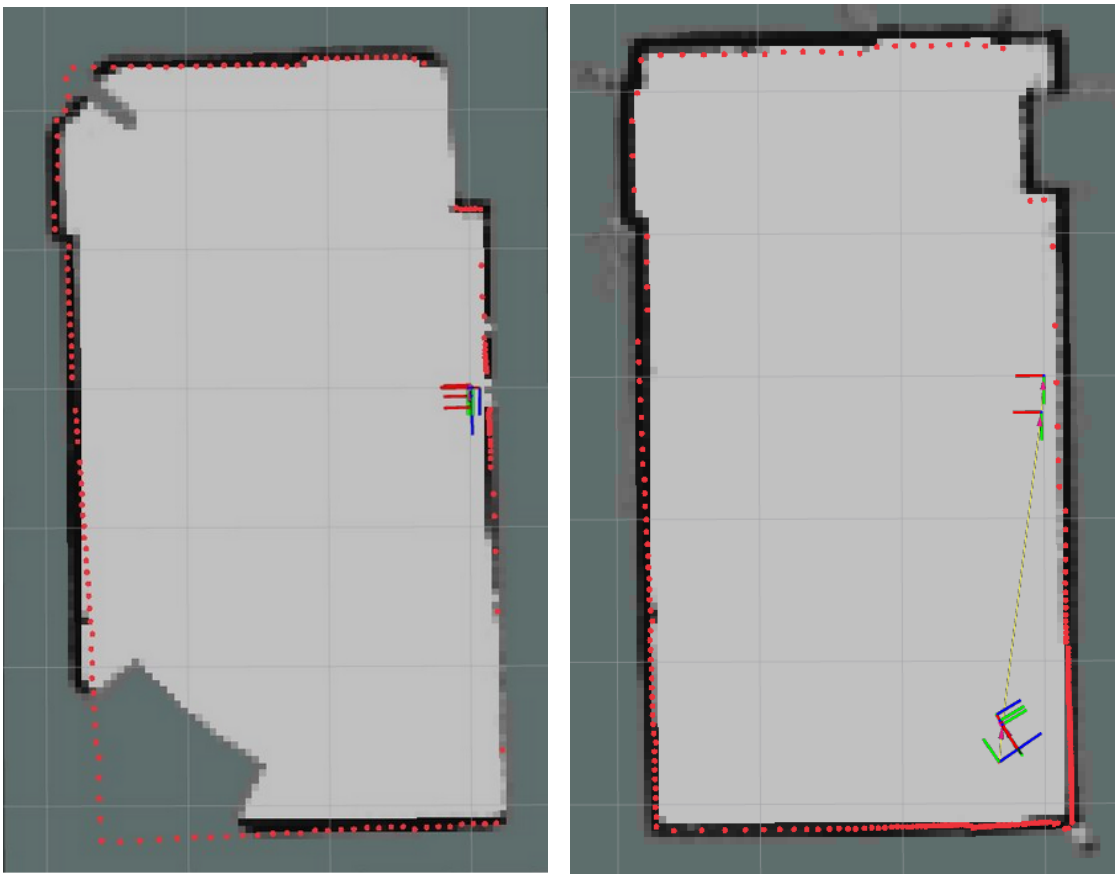
1. **Sensory input:** 2D LiDAR scans provide the primary geometric observations of the environment;

2. **Frame structure (TF):** Cartographer operates within the system's TF graph. Static transforms (e.g., sensor mounting relative to the base) must be available and consistent, while motion-related transforms depend on the adopted odometry configuration;

3. **Map and pose outputs:** the node publishes the 2D map representation and outlines the transformations required to express the robot pose in a global

reference frame. The result can be used directly for map saving, localization, and navigation.

This setup generates a map that is suitable for subsequent navigation tests in indoor environments, as illustrated in Figure 4.3.



(a) Cartographer after startup (b) Map after exploration

Figure 4.3: RViz2 visualization of 2D SLAM with Cartographer

4.3.4 QoS compatibility for SLAM visualization and mapping

In ROS 2, topic endpoints must agree on compatible Quality of Service (QoS) policies. In the considered setup, RViz2 , the SLAM pipeline, and the navigation stack were configured to match the QoS profiles offered by the publishers of the main data streams. In particular, the LiDAR scan topic is published with a best-effort, volatile profile, which is appropriate for high-rate sensor data where occasional

packet loss is preferable to increased latency. Conversely, the occupancy-grid map is configured with a transient-local durability, so that late-joining subscribers (e.g., RViz2 or map-saving tools) can immediately receive the most recent map without waiting for the next publication.
Table 4.1 summarizes the QoS settings adopted for the main topics used during mapping and visualization.

Topic	History	Depth	Reliability	Durability
`/tb3_3/scan`	Keep Last	5	Best Effort	Volatile
`/tb3_3/map`	Keep Last	1	Best Effort	Transient Local

Table 4.1: QoS profiles used for LiDAR scans and map visualization in RViz2

4.3.5 Frame management and integration with TF

A key requirement for 2D LiDAR SLAM in ROS 2 is the consistency of the TF graph, since scan matching and map updates rely on expressing sensor measurements in coherent reference frames [22, 23]. In a typical setup, TF transformations can be grouped into three categories:

- **Static transformations:** rigid transforms derived from the robot description (URDF), connecting the base frame to sensor frames (e.g., LiDAR and IMU). These transforms are time-invariant and are typically published on */tf_static*;
- **Odometric transformations:** motion-related transforms (commonly *odom* → *base_link*) published by the odometry subsystem, providing a continuous local estimate that is smooth in the short term, but subject to drift;
- **Global transformations:** transforms produced by the SLAM module to anchor the robot trajectory to a global reference, typically by estimating *map* → *odom* (or, depending on the configuration, a direct global reference to the robot base).

In this architecture, the robot pose in the global frame is obtained by composing the global and odometric transforms, while sensor measurements are correlated to the base through static extrinsics (e.g., *base_link* → *base_scan*). Therefore, any inconsistency in frame naming, timestamps, or missing links in the TF chain directly degrades scan alignment and can lead to map distortions, unstable localization, or failure of the SLAM process. For this reason, providing a complete and unambiguous TF tree is a prerequisite for reliable mapping and subsequent navigation.

4.3.6 Changes to the Cartographer configuration

The configuration adopted in this work is derived from the official ROBOTIS TurtleBot3 Cartographer setup and was adapted to the experimental context under consideration. In particular, the applied modifications focus on supporting execution in a multi-robot setting and on making the frame conventions used by Cartographer explicit and consistent with the TF structure of the overall system.

Launch file
Namespace support was introduced by parameterizing the launch file, and applying a prefix to the nodes, topics, and resources spawned by the SLAM stack. This allows multiple Cartographer instances to run concurrently on the same network without name collisions, and keeps the communication graphs of different robots separated. Where required, remapping rules were also used, in order to ensure that each instance consumes the correct TF streams associated with its namespace.

Configuration file (.lua)
The Cartographer configuration was modified primarily to align the SLAM node with the TF conventions adopted in the system [24]. In a multi-robot scenario, namespaced frames are used to make sure that measurements, pose estimates, and published transformations are unambiguously associated with the corresponding robot [25]. This prevents conflicts between frames sharing the same nominal name and ensures scan matching and pose estimation to be performed using a consistent TF chain.

Estimation parameters
A subset of scan matching and pose-graph optimization parameters was introduced and fine-tuned to control the trade-off between local estimation stability and global map consistency. In indoor environments, where sensor noise, partial occlusions, and repetitive geometries can degrade data association, conservative thresholds and appropriate weighting help reduce the impact of unreliable constraints and improve the overall map robustness.

4.4 Navigation and localization

4.4.1 Navigation stack architecture (Nav2)

In ROS 2, Navigation2 (Nav2) is the reference framework for autonomous navigation [26]: it provides a set of coordinated nodes that transform a goal pose into motion commands executable by the mobile base. From an implementation perspective, Nav2 is organized around dedicated servers: *planner_server* computes

a global path, *controller_server* tracks the path and generates velocity commands while accounting for obstacles, and *behaviour_server* manages auxiliary and recovery behaviours. A *lifecycle_manager* supervises the node lifecycle transitions (configure, activate, deactivate), enabling startup control and fault handling.

At a functional level, Nav2 consists of three main modules:

1. **Global planning:** the planner computes a collision-free route from the current pose to the target on the global costmap, optimizing a cost criterion derived from traversability;

2. **Local control:** the controller generates linear and angular velocity commands to track the global path, relying on the local costmap to react to obstacles and to enforce the platform kinematic constraints. In this work, path tracking is implemented using the *Regulated Pure Pursuit Controller* (RPP)[27], configured in the *controller_server* as the *FollowPath* plugin. Compared to sampling-based local planners (e.g., DWB), RPP computes a curvature-based command toward a lookahead point on the path. It also modulates the commanded speed through regulation terms that account for curvature and obstacle proximity, thus improving stability and safety in cluttered indoor environments, as shown in Figure 4.4.

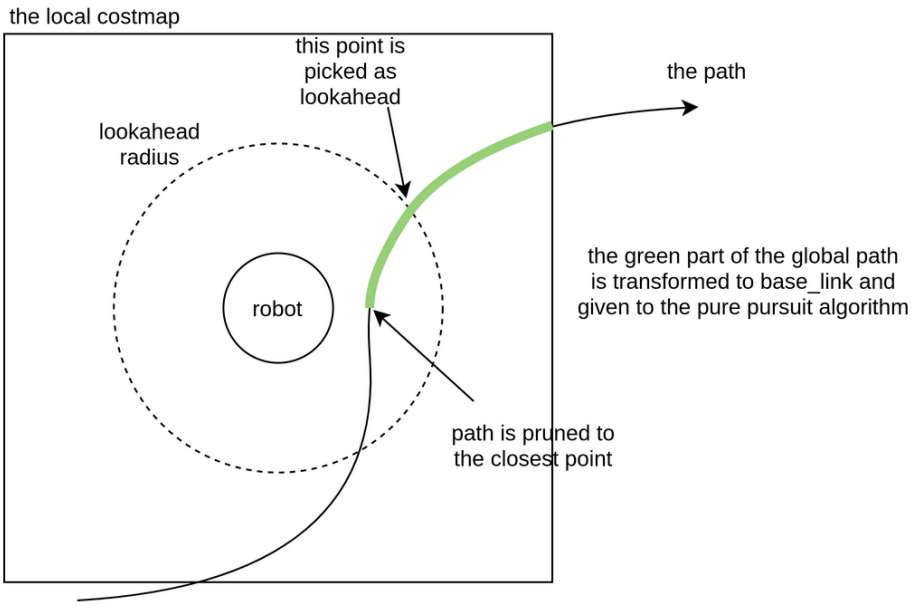


Figure 4.4: Regulated Pure Pursuit (RPP) controller overview[27]

3. **Behaviour management:** execution is coordinated through Behaviour Trees (BT), which orchestrate navigation actions and trigger recovery strategies such as in-place rotations, backtracking, or costmap clearing (when progress is not possible).

Navigation is exposed through the *NavigateToPose* action interface, which allows goals to be issued and monitored through periodic feedback, with support for cancellation or goal updates during execution. Moreover, Nav2 adopts a plugin-based design [28, 29], allowing planners, controllers, and behaviour modules to be replaced without altering the overall system architecture, which is particularly convenient for experimentation on TurtleBot3 Burger.

4.4.2 Localization on a known map

Once a 2D map has been generated and stored, the subsequent step is localization, i.e., estimating the transformation between the global *map* frame and the odometric frame. Operationally, localization compensates for the odometry drift by leveraging sensor observations against a static representation of the environment.

In 2D LiDAR-based systems, it is common to use probabilistic grid localization based on particle filters (e.g., AMCL) [30], where the pose is inferred by matching incoming laser scans to the map and updating the distribution of pose hypotheses. The main output is the $map \rightarrow odom$ transformation, which separates globally consistent positioning from the locally continuous odometric estimate. In this way, the robot pose can be expressed in a stable global reference while also preserving odometry as a high-rate local frame for short-term control.

4.4.3 Frame graph and map roles, odometry and base

Integration of localization and navigation in ROS 2 requires a consistent TF graph, since planning, perception, and control require that measurements and commands are expressed in well-defined reference frames. Conventionally, in a standard mobile robot, there are three frames that play a central role:

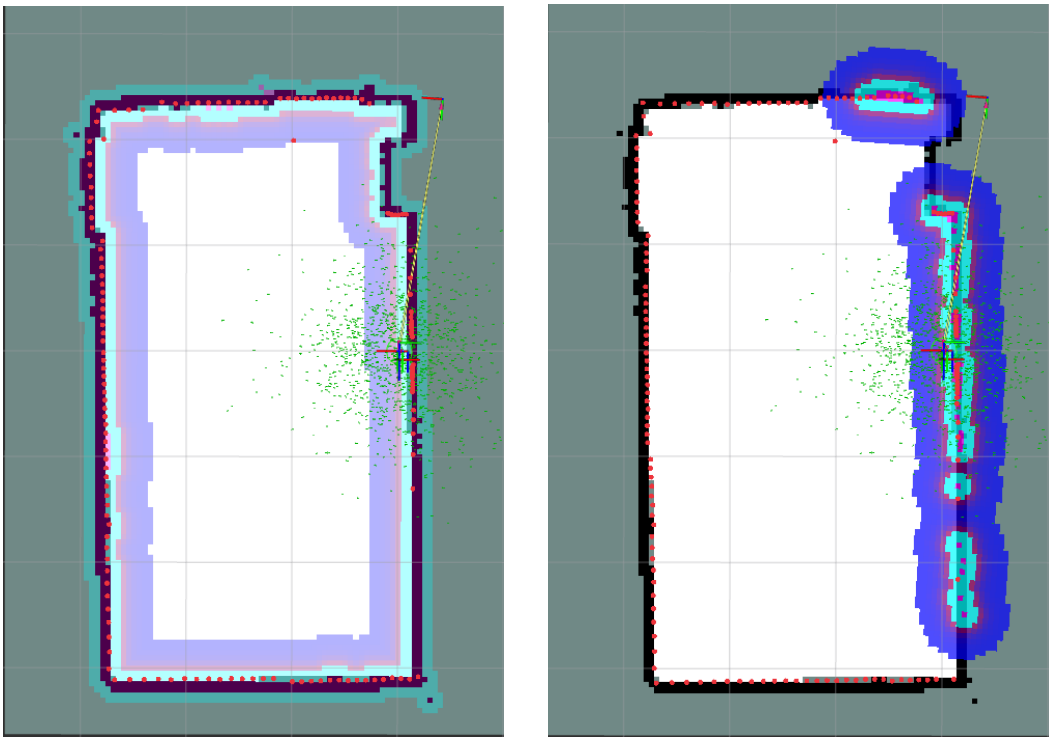
- *base_link*: the robot body frame, used to express onboard sensor poses and velocity commands;
- *odom*: a continuous local frame updated by odometry, smooth in the short term, but affected by drift over time;
- *map*: a global frame anchored to the environment map, designed to remain stable over long periods of time.

Within this structure, odometry publishes the transform *odom* → *base_link*, while localization estimates *map* → *odom*. Their chaininggenerates the robot pose in the *map* frame, which is then used by global planning and high-level decision-making. This separate approach not only preserves the high-rate continuity of odometry for short-term control, but also maintains long-term global consistency through map-based localization.

4.4.4 Costmap

Navigation requires an operational representation of obstacles and traversable space. In Nav2, this role is fulfilled by costmaps [31], i.e., grid-based maps in which each cell encodes a traversal cost that reflects the presence of obstacles, safety margins, and navigation constraints.

Nav2 typically manages two costmaps with complementary roles, as illustrated in Fig. 4.5. The *global costmap* is built from the static map and is used by the global planner to compute a collision-free route toward the goal. The *local costmap* is updated online using sensor data (e.g., 2D LiDAR) and provides a short-range representation of the nearby environment for obstacle avoidance [32, 33] and local control. By integrating real-time measurements, the local costmap enables the robot to react to obstacles that are not present in the static map or that appear temporarily during operation. The effectiveness of this process depends heavily on the consistency of TF transforms and the accuracy of pose estimates, since sensor observations must be correctly projected into the costmap frames in both space and time.



(a) Global costmap (b) Local costmap

Figure 4.5: Nav2 costmaps: (a) global costmap from the static map; (b) local costmap updated from sensor data. White: free space, black: obstacles, blue/purple: inflated cost regions.

4.4.5 Localization-Navigation Correlation

Localization and navigation are tightly coupled: on one side, errors in pose estimation directly reduce the quality of global planning and local control; on the other, poorly conditioned control can degrade localization by introducing discrepancies between predicted motion and sensor observations. For this reason, while preserving a clear interface between modules, the system architecture separates responsibilities: odometry provides a continuous, high-rate local estimate used for short-term control, localization corrects long-term drift by anchoring the pose to the known map, and navigation assumes a consistent pose estimate in the *map* frame to perform planning, obstacle avoidance, and decision-making.

Consistent performance also depends on the selected parameters and update rates across the pipeline (localization, costmap updates, and controller frequency), which must be compatible with both the available computational resources and the platform dynamics, in order to avoid excessive latency and degraded responsiveness.

In a multi-robot deployment, despite maintaining a strict separation of topics, frames, and parameters, the same localization and navigation components must be instantiated per robot. In this context, namespaces and consistent TF naming conventions are essential to prevent collisions and to ensure that each instance only consumes its own sensor data and pose estimates.

4.5 Motion planning

Motion generation is formulated as a multi-stage process that transforms a navigation goal into velocity commands compatible with the kinematic constraints of the mobile platform. In ROS 2, this functionality is implemented through the Nav2 stack, which explicitly separates global path planning from local control. This separation guarantees map-level consistency and allows long-range route selection to be handled independently from short-range obstacle avoidance and constraint-aware command generation.

4.5.1 Planning and control pipeline

Starting from a goal pose expressed in the global *map* frame, the system first computes a global path using the grid-based representation of the environment. This path is then tracked by a local controller, which produces velocity commands on the *cmd_vel* topic while also accounting for both obstacles perceived online and the kinematic and dynamic limits of the platform. Planning and control are executed iteratively: global planning can be recomputed when the current route becomes invalid or suboptimal, whereas local control runs at a higher frequency to preserve responsiveness and robustness under changing environmental conditions.

Global planning operates on the *global costmap*, which encodes traversability costs derived from the static map and, when enabled, additional layers, such as obstacle inflation. At this stage, the planner computes a collision-free route from the current pose to the goal by minimizing a cost functional, while respecting occupied regions in the costmap. The output is a discrete sequence of planar poses that provides the geometric reference to be followed by the subsequent tracking and control stage.

4.5.2 Local control and configuration

The local controller receives the global path and produces linear and angular velocity commands that are consistent with the kinematics of the mobile base. Unlike global planning, local control relies on a *local costmap* that is updated online from onboard sensors, enabling the controller to react to obstacles that are

not represented in the static map and to temporary changes in the environment. At this stage, the controller update rate, the collision checking strategy, and the enforcement of velocity and acceleration limits are critical, since they directly affect tracking accuracy, safety margins, and motion stability.

From an implementation perspective, these planning and control choices are specified through YAML configuration files, which define both the selected plugins and the numerical parameters governing their behaviour. This configuration includes the global planner and local controller settings, together with costmap parameters, such as grid resolution, robot footprint/inflation radius, and the sensor integration mode used for obstacle marking and clearing.In addition, criteria for detecting a lack of progress (*progress checker*) and for determining goal achievement (*goal checker*) are defined to support recovery behaviours when navigation becomes unstable or the robot cannot advance. This organization separates the conceptual pipeline from its experimental instantiation and makes the adopted technical choices explicit and reproducible. The full navigation sequence is illustrated in Figures 4.6, 4.7, and 4.8, showing respectively the goal pose, the computed global path, and the goal reached confirmation.

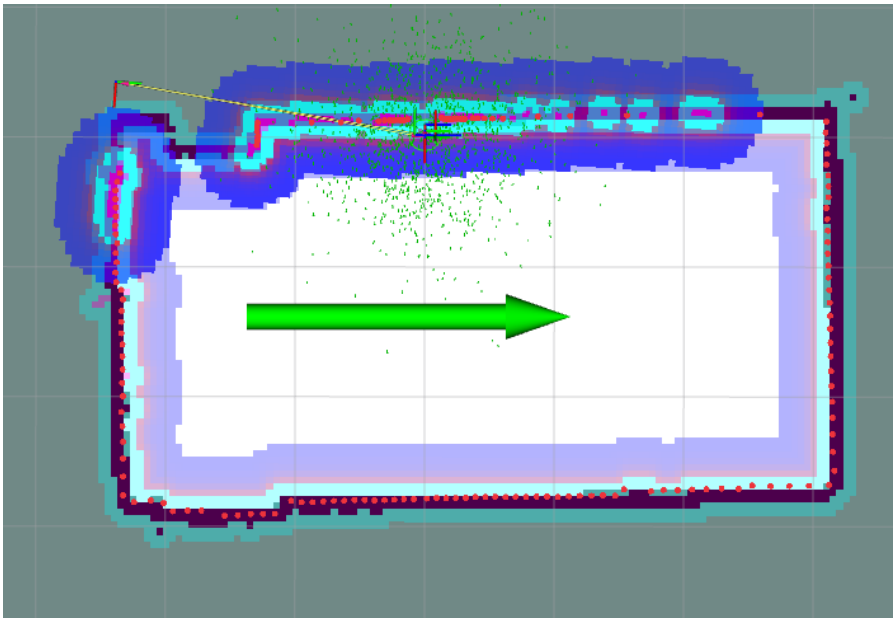


Figure 4.6: Goal pose (green arrow) set in RViz2.

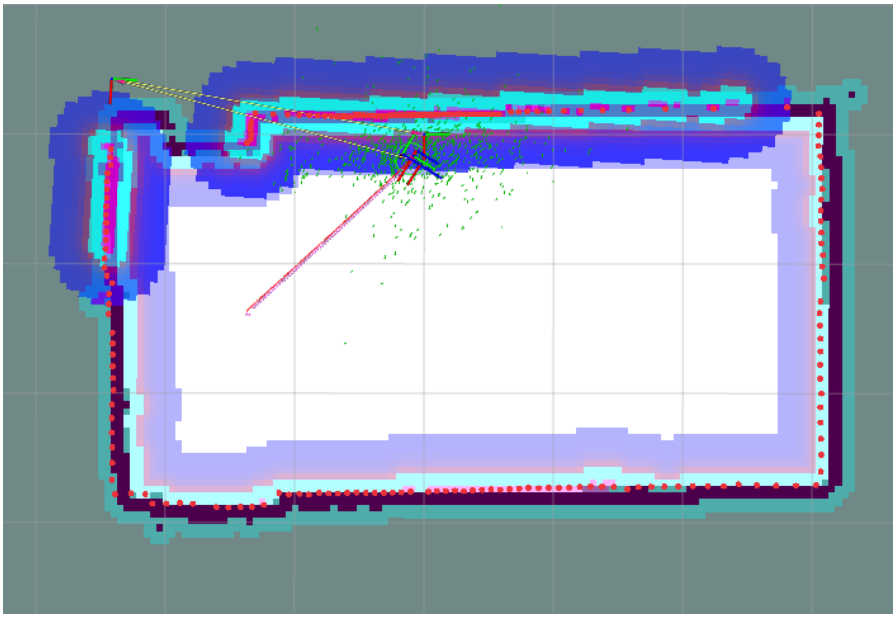


Figure 4.7: Global path (red line) produced by the planner.

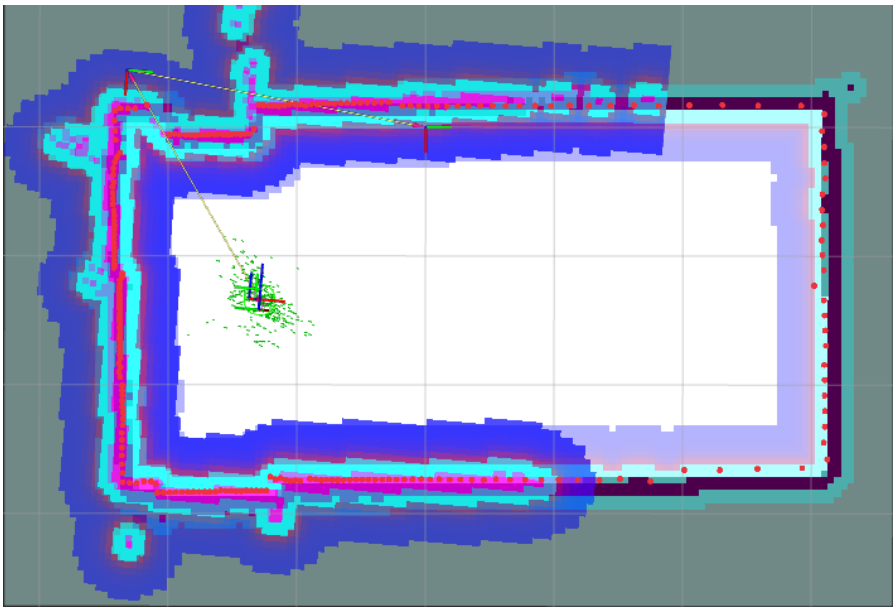


Figure 4.8: Goal reached feedback in the Nav2 panel.

4.6 Multi-robot setup

When extending the system from a single robot to a multi-robot deployment, the primary requirement is to allow multiple instances of the navigation stack to operate

concurrently without mutual interference [34]. In ROS 2, running several robots on the same network can lead to name collisions across nodes, topics, and services, as well as ambiguity in TF transformations when identical frame identifiers are published by different instances. These issues directly affect localization and navigation consistency. For this reason, each robot is assigned a dedicated namespace and a consistent naming convention, supported by parameterized configuration files that make the resulting setup explicit and reproducible. The main configuration changes, introduced to enable multi-robot execution, are outlined below.

TF topic remapping for multi-robot support
In a multi-robot deployment, the management of TF data becomes crucial, since transformations are published on the default global topics */tf* and */tf_static*. When multiple robots share the same ROS 2 network, transformations generated by different instances can be multiplexed on the same channels. This can create ambiguity whenever different robots publish frames with identical identifiers and, more generally, can make the association between sensor measurements, pose estimates, and the corresponding transformation chain non-deterministic for localization and navigation modules.

To prevent these interferences, the robot launch files remap */tf* and */tf_static* into the robot namespace (e.g., *tf* and *tf_static* under `/<robotName>/`), as illustrated in Figs. 4.9 and 4.10. This isolates the transformation streams of each instance and ensures that every robot only consumes its own static transforms (e.g., sensor extrinsics) and dynamic transforms (e.g., odometry and pose estimates), without contamination from other agents. As a consequence, the TF graph remains internally consistent at the per-robot level, which is a prerequisite for transformation-dependent modules, in particular localization, costmap construction, and navigation.

```
GNU nano 7.2                        robot.launch.py

    IncludeLaunchDescription(
        PythonLaunchDescriptionSource([lidar_pkg_dir, LDS_LAUNCH_FILE]),
        launch_arguments={'port': '/dev/ttyUSB0',
                          'frame_id': 'base_scan',
                          'namespace': namespace}.items(),
    ),

    Node(
        package='turtlebot3_node',
        executable='turtlebot3_ros',
        parameters=[
            tb3_param_dir,
            {'namespace': namespace}],
    #       {'tf_prefix': 'tb3_3/'}],
        arguments=['-i', usb_port],
        remappings = [
            ('/tf','tf'),
            ('/tf_static','tf_static')
            ],
        output='screen')
])
```

Figure 4.9: TF remapping in the robot launch file.

```
GNU nano 7.2          turtlebot3_state_publisher.launch.py
  # Reference page: https://github.com/ros2/demos/pull/426

  rsp_params = {'robot_description': robot_desc}

  # print (robot_desc) # Printing urdf information.

  return LaunchDescription([
      DeclareLaunchArgument(
          'use_sim_time',
          default_value='false',
          description='Use simulation (Gazebo) clock if true'),
      Node(
          package='robot_state_publisher',
          executable='robot_state_publisher',
          output='screen',
          parameters=[
                  rsp_params,
                  {'use_sim_time': use_sim_time}],
          remappings =[
              ('/tf','tf'),
              ('/tf_static','tf_static')])
  ])
```

Figure 4.10: TF remapping in the robot state publisher launch.

Once TF trees are isolated through namespaces, a global view may still be required by centralized tools or monitoring components that expect the standard */tf* and */tf_static* topics. For this purpose, an aggregation node was introduced to subscribe to `/<robotName>/tf` and `/<robotName>/tf_static` for all robots and republish the collected transforms onto the conventional global topics.

SLAM configuration in a multi-robot environment
To enable the SLAM phase in a multi-robot setting, both the Cartographer launch file and the node configuration (.lua) were adapted to support concurrent execution and unambiguous frame management.

At the launch level, namespace support was introduced and applied to all nodes and resources instantiated by the SLAM stack, preventing name collisions when multiple instances run on the same ROS 2 network. In addition, key launch arguments were made explicit to improve controllability and to ensure that experimental runs can be reproduced with consistent settings.

The .lua configuration was updated to enforce TF consistency and improve estimation robustness. In particular, the tracking and published frames were replaced with namespaced frames associated with each robot base, in order to ensure a one-to-one association between transformations and the corresponding platform. Furthermore, all parameters related to odometry usage and *odom* frame handling were aligned with the adopted TF structure. Finally, scan matching, motion filtering, and pose-graph optimization settings were introduced or tuned using more conservative thresholds, with the objective of improving map stability in indoor environments affected by noise or characterized by partial occlusions and repetitive geometries.

Multi-robot Nav2: launch and configuration
To extend localization and navigation to a multi-robot deployment, both the launch files and the Nav2 parameter sets were adapted to support concurrent execution without cross-interference. In the Nav2 launch, the namespace is provided as an execution argument, and the entire stack is instantiated within the namespace of the individual robot. This isolates navigation nodes and topics across instances and prevents name collisions when multiple robots operate on the same ROS 2 network. In addition, RViz and the components spawned by the launch wereconfigured with consistent remapping of *tf* and *tf_static*, thus ensuring that each instance only consumes its own transformation streams and is not affected by transforms published by other robots. The launch procedure was then further parameterized through command-line arguments that specify the map file, the YAML parameter file, the execution mode (simulation or real), and the robot namespace, thereby

improving reproducibility across experimental runs.

The Nav2 YAML configuration files were modified to make the separation of frames and topics explicit for localization, costmap generation, and control [35]. While default single-robot setups typically reference non-namespaced entities (e.g., *odom*, *base_link*, *scan*), the configurations adopted in this work specify namespaced frames and sensor topics (e.g., `<robotName>/odom`, `<robotName>/base_link`, and `/<robotName>/scan`). This ensures that each localization and navigation instance only consumes the sensor data and TF transforms of its own robot, preserving consistency between pose estimation, obstacle projection into the costmaps, and velocity command generation. Finally, maintaining a dedicated parameter file per robot makes it possible to use a common configuration structure while varying only instance-dependent elements (frames and topics) in a controlled manner. This approach improves experiment reproducibility and reduces configuration errors.

Chapter 5

Baseline Leader–Follower Control

The baseline reference for the development of cooperative formation control methods was the leader–follower configuration. The objective was to establish a simple, reproducible control setup in which the leader prescribes the formation motion while the follower regulates its own dynamics to maintain a specific geometric relationship with respect to the leader. Such a baseline framework isolates the core elements of the problem addressed in this work, before introducing more advanced coordination and planning components.

This chapter outlines the formation-tracking problem by defining premises, objectives, and variables of interest. It then presents the adopted formation geometry, the tracking control law, and the supervisory logic implemented through a discrete-state machine, with emphasis on safety requirements and the conditions governing the stop and restart behaviour.

5.1 Problem statement

The considered system consists of two TurtleBot3 Burger platforms, denoted as leader L and follower F, operating on a planar workspace. Each robot is represented by its planar pose expressed in a global reference frame and is commanded through linear and angular velocity inputs (v, ω), consistent with a unicycle kinematic model. In the baseline formulation, the leader is treated as an exogenous reference, i.e., its trajectory is not influenced by the follower. The control objective is, therefore, to synthesize the follower commands (v_F, ω_F) in such a way that a prescribed relative configuration is maintained with respect to the leader.

It is assumed that both robots are localized in the *map* frame and that the required transformations are available through TF. In particular, the algorithm acquires the real-time poses of the leader and follower by querying the corresponding *map*→*base_footprint* transforms for each platform. Control commands are computed at a fixed sampling time $T_s = 0.05\,\mathrm{s}$ and published on the follower velocity interface.

The formation objective is specified by prescribing a relative offset of the follower with respect to the leader. In the baseline configuration, the follower is required to maintain a fixed separation distance d from the leader while remaining aligned

with the leader's direction, thus achieving a stable and reproducible tracking behaviour that can serve as a reference for future extensions. The desired relative configuration is formulated by defining a target point (x^*, y^*) as a function of the leader pose and the prescribed distance, and by enforcing the convergence of the follower toward this point within a bounded position error.

The problem formulation involves operational and safety constraints. First, all follower commands must satisfy bounded actuation limits in terms of linear and angular velocity:

$$|v_F| \leq v_{\max}, \qquad |\omega_F| \leq \omega_{\max}. \tag{1}$$

Second, a minimum inter-robot separation is needed, which relies on a supervisory logic that reduces speed and, if necessary, stops the follower when the distance falls below the predefined safety thresholds. To avoid chattering near the switching boundary, these thresholds were implemented with hysteresis. In addition, to prevent unsafe configurations in which the follower approaches the leader from the front, a recovery behaviour was introduced: the follower is commanded to move toward a pre-defined safe waypoint before re-entering the nominal formation. Finally, an exclusion region centred on the leader was defined to discourage overly aggressive approaches during transients, improving robustness throughout formation re-acquisition.

The control aims at defining a command generation law (v_F, ω_F) where:

1. the follower converges to, and remains within, a neighbourhood of the desired relative configuration with respect to the leader, within prescribed position and orientation tolerances;

2. the generated commands satisfy the kinematic model and the saturation constraints;

3. safety requirements on inter-robot separation are enforced, including appropriate recovery behaviour under non-nominal conditions.

5.2 Formation

The leader–follower formation is defined in geometric terms by assigning the follower F a reference pose, which depends on the instantaneous pose of the leader L. All quantities are expressed in the global *map* frame. The leader and follower poses are acquired through TF, which provides a consistent and time-stamped representation

of the relative geometry between frames.

5.2.1 Planar pose and orientation: quaternions and yaw angle

The planar pose of each robot is described by the following equation:

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ \psi \end{bmatrix}, \tag{2}$$

where (x, y) are planar coordinates and ψ is the yaw angle. The leader and follower poses are denoted as follows:

$$\mathbf{p}_L = \begin{bmatrix} x_L & y_L & \psi_L \end{bmatrix}^T, \qquad \mathbf{p}_F = \begin{bmatrix} x_F & y_F & \psi_F \end{bmatrix}^T. \tag{3}$$

In ROS 2, TF represents orientation using a unit quaternion

$$\mathbf{q} = (q_x, q_y, q_z, q_w), \qquad \|\mathbf{q}\| = 1, \tag{4}$$

which provides a non-singular parameterization and supports numerically robust composition of rotations, a key requirement for maintaining consistency within a TF transformation tree. For planar motion, the relevant quantity is the yaw component, i.e., the rotation about the z-axis. Given $\mathbf{q}$, the yaw angle is computed as

$$\psi = \operatorname{atan2}\big(2\,(q_w q_z + q_x q_y),\ 1 - 2\,(q_y^2 + q_z^2)\big)\,. \tag{5}$$

This is the standard procedure for extracting the yaw angle ψ from a unit quaternion. The two arguments of the `atan2` function correspond to the sine- and cosine-like components of the planar rotation, and their joint use preserves the correct quadrant information of the angle. As a result, ψ is recovered without the sign ambiguities that can arise with a single-argument arctangent, yielding a numerically robust conversion from TF quaternion poses to planar state variables.

When using angular differences, it is necessary to manage the intrinsic periodicity of the yaw angle ψ. Let us call α the generic angular difference between the reference angle of the leader or the formation target, and the current angle of the follower:

$$\alpha = \psi_{target} - \psi_F \tag{6}$$

An angular quantity alpha is therefore normalized in the main interval $[-\pi, \pi]$ using the wrapping function. This function operates in iterative cycles: it subtracts 2π

from the angle until it exceeds π and adds 2π until it is less than $-\pi$, ensuring robust numerical convergence even for very large or negative angles. Mathematically, this is equivalent to the following equation:

$$\text{wrap}(\alpha) = \alpha - 2\pi \left\lfloor \frac{\alpha + \pi}{2\pi} \right\rfloor \tag{7}$$

This operation prevents numerical discontinuities, ensuring that the result always remains within the desired range and is comparable to tolerance thresholds.

5.2.2 Leader-attached unit vectors and goal construction

Given the leader yaw angle ψ_L, an orthonormal basis attached to the leader is defined in the plane as

$$\mathbf{t}_L = \begin{bmatrix} \cos\psi_L \\ \sin\psi_L \end{bmatrix}, \qquad \mathbf{n}_L = \begin{bmatrix} -\sin\psi_L \\ \cos\psi_L \end{bmatrix}. \tag{8}$$

The unit vector $\mathbf{t}_L$ represents the forward direction of the leader, while $\mathbf{n}_L$ is the corresponding leftward lateral direction. This construction allows the formation specification to be expressed in a leader-attached frame, so that the desired offset is invariant to the global orientation and is mapped into the global frame through the instantaneous value of ψ_L.

The formation reference point for the follower is defined as

$$\mathbf{p}_G = \begin{bmatrix} x_G \\ y_G \end{bmatrix} = \begin{bmatrix} x_L \\ y_L \end{bmatrix} + a\,\mathbf{t}_L + b\,\mathbf{n}_L, \tag{9}$$

where the scalars a and b represent the desired longitudinal and lateral offsets, respectively, in relation to the leader. This parameterization preserves a fixed mathematical structure, while also allowing different formation geometries to be obtained by modifying the coefficients (a, b) only.

5.2.3 Tested configurations

In order to assess the behaviour of the baseline controller under distinct relative arrangements and to evaluate the robustness of the reference-point construction and of the associated error variables, two formation geometries were taken into consideration. In both cases, the follower reference point is computed from the leader pose as described in the previous subsection.

Although the control architecture is identical in both cases, the controller is instantiated with configuration-dependent parameter values — such as gains, tolerances, and safety thresholds — to account for the different dominant error components in lateral tracking and trailing tracking. On one side, lateral tracking requires tighter heading regulation; on the other, trailing tracking can tolerate larger lateral deviations but is more sensitive to longitudinal closing speed near the leader. All parameters are exposed as ROS 2 node parameters and were tuned experimentally for each configuration.

Behind-the-leader configuration As shown in Fig. 5.1, the follower is required to maintain a fixed separation distance d along the negative longitudinal direction of the leader. The target point is defined as:

$$\mathbf{p}_G = \begin{bmatrix} x_G \\ y_G \end{bmatrix} \tag{10}$$

is defined as

$$\mathbf{p}_G = \begin{bmatrix} x_L \\ y_L \end{bmatrix} - d\,\mathbf{t}_L, \tag{11}$$

i.e.,

$$x_G = x_L - d\cos\psi_L, \tag{12}$$
$$y_G = y_L - d\sin\psi_L. \tag{13}$$

This configuration represents the standard tracking case, in which the dominant error component is aligned with the direction of motion. It is therefore suitable for validating distance regulation and the response to changes in the leader speed and curvature.

Left-lateral configuration In the second configuration, the follower is required to maintain a fixed lateral offset from the leader, defined along the leader's left normal direction (Fig. 5.2). The corresponding target point is given by

$$\mathbf{p}_G = \begin{bmatrix} x_L \\ y_L \end{bmatrix} + d\,\mathbf{n}_L, \tag{14}$$

i.e.,

$$x_G = x_L - d\sin\psi_L, \tag{15}$$
$$y_G = y_L + d\cos\psi_L. \tag{16}$$

Figure 5.1: Behind-the-leader configuration: the follower tracks a goal point at distance d along $-\mathbf{t}_L$.

Compared to the first configuration, this geometry is more sensitive to lateral errors and typically requires tighter heading alignment during turning manoeuvres. As a result, it provides a useful scenario to validate the correctness of the reference construction when the desired offset is not collinear with the direction of travel.

In both configurations, the desired follower orientation is set equal to the leader orientation:

$$\psi_G = \psi_L, \tag{17}$$

so that the formation heading remains consistent with the leader's motion.

5.2.4 Formation errors

Once the target point p_G of the formation has been defined, in both kinds of configurations, a set of error variables is to be introduced in order to evaluate the quality of formation maintenance. These values have two distinct purposes: to quantitatively describe the distance of the follower from the desired reference, and to provide a parameterization of the error that can be easily interpreted from a geometric point of view.

In the global *map* frame, the position error is defined as the difference between the

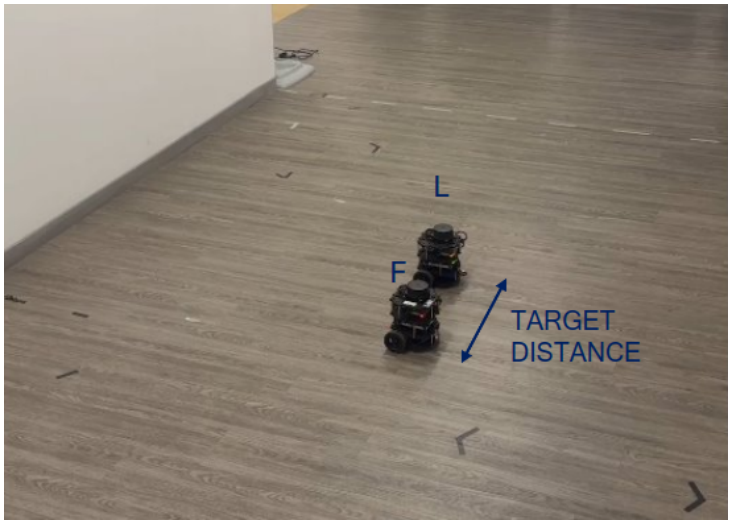


Figure 5.2: Left-lateral configuration: the follower tracks a goal point at distance d along $\mathbf{n}_L$

goal and the follower's position:

$$\Delta x = x_G - x_F, \tag{18}$$
$$\Delta y = y_G - y_F \tag{19}$$

from which the Euclidean distance from the target is obtained:

$$d_G = \sqrt{(\Delta x)^2 + (\Delta y)^2} \tag{20}$$

This information is also associated with the bearing angle towards the goal, computed as follows:

$$\theta_G = \text{atan2}(\Delta y, \Delta x) \tag{21}$$

which represents the direction of the ray connecting the follower to the target point. Orientation consistency is assessed by introducing a desired orientation, which in the adopted configuration matches the one of the leader.

The angular error is therefore defined as

$$e_\psi = \text{wrap}(\psi_G - \psi_F) = \text{wrap}(\psi_L - \psi_F) \tag{22}$$

to normalize the angle in the interval $[-\pi, \pi]$. This normalisation is necessary because orientation is a periodic variable: without wrapping, differences between angles close to $-\pi$ and π would produce errors that are not representative of the actual physical discrepancy.

To obtain an error description that is consistent with the kinematics of the differential base, the position error is also expressed in the frame attached to the follower. By introducing the planar rotation associated with the following equation: ψ_F,

$$R(\psi_F)^T = \begin{bmatrix} \cos\psi_F & \sin\psi_F \\ -\sin\psi_F & \cos\psi_F \end{bmatrix} \tag{23}$$

we obtain:

$$\begin{bmatrix} e_{x,F} \\ e_{y,F} \end{bmatrix} = R(\psi_F)^T \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} \cos\psi_F\,\Delta x + \sin\psi_F\,\Delta y \\ -\sin\psi_F\,\Delta x + \cos\psi_F\,\Delta y \end{bmatrix} \tag{24}$$

The components $e_{x,F}$ and $e_{y,F}$ have a specific meaning: the first measures the error along the longitudinal axis of the follower (target "front/back"), while the second measures the lateral deviation (target "left/right"). This decomposition allows the error component associated with translational motion to be separated naturally from the one linked to direction correction.
In addition to the errors described above, the relative geometry between leader and follower in the leader's frame is also calculated, which is useful for describing the real-time configuration of the pair regardless of the global orientation. By defining

$$\Delta x_{FL} = x_F - x_L, \qquad \Delta y_{FL} = y_F - y_L, \tag{25}$$

the coordinates in the leader frame are obtained as follows:

$$\begin{bmatrix} \Delta x^L_{LF} \\ \Delta y^L_{LF} \end{bmatrix} = R(\psi_L)^T \begin{bmatrix} \Delta x_{FL} \\ \Delta y_{FL} \end{bmatrix} = \begin{bmatrix} \cos\psi_L & \sin\psi_L \\ -\sin\psi_L & \cos\psi_L \end{bmatrix} \begin{bmatrix} x_F - x_L \\ y_F - y_L \end{bmatrix}, \tag{26}$$

which is equivalent to:

$$\Delta x^L_{LF} = \cos\psi_L(x_F - x_L) + \sin\psi_L(y_F - y_L), \tag{27}$$

$$\Delta y^L_{LF} = -\sin\psi_L(x_F - x_L) + \cos\psi_L(y_F - y_L). \tag{28}$$

The component Δx^L_{LF} quantifies the relative position along the direction of travel of the leader, while Δy^L_{LF} describes the lateral deviation.

Finally, to define operationally when the formation can be considered satisfactory within an assigned accuracy, tolerances have to be introduced on position and orientation:

$$d_G < \varepsilon_{pos}, \qquad |e_\psi| < \varepsilon_\psi, \tag{29}$$

where $\varepsilon_{pos} =$ `pos_tolerance` and $\varepsilon_\psi =$ `yaw_tolerance`. These criteria provide a quantitative condition for benchmark convergence, keeping the geometric definition of the formation separate from the control and supervision choices discussed in the following sections.

5.3 Control law

At each iteration of the control cycle, the adopted control law converts the geometric quantities described in the previous section (formation goal, errors $\Delta x, \Delta y$, d_G, $e_{x,F}$, e_ψ, and leader-follower relative terms) into velocity commands for a differential drive base, expressed as (v, ω). The controller is implemented as a ROS 2 node executing a periodic loop at 20 Hz and publishing a `TwistStamped` command to the follower.

At runtime, the control node continuously updates the formation reference from the current leader pose and expresses the relative configuration of the robots through the error quantities defined above. First of all, these errors are used to identify the desired direction of motion within the plane, consistently with the formation objective and the current leader-follower geometry. This planar direction is then translated into a heading mismatch with respect to the follower orientation, yielding a heading error that drives the angular rate command. In parallel, the linear speed command is shaped with the aim of reducing the longitudinal component of the tracking error, while maintaining stable behaviour close to the target. Before publication, the raw commands are further processed through practical layers that ensure feasibility and robustness, including saturation, optional smoothing, and state-dependent constraints designed to avoid abrupt changes and to ensure safe operation.

5.3.1 Direction of motion: attraction towards the goal and repulsion from the leader

The first step of the control law consists of defining the desired direction of motion within the plane. In the absence of constraints, this direction is purely goal-oriented and is naturally identified with the bearing from the follower position (x_F, y_F) to the target point (x_G, y_G). The corresponding attractive heading is therefore computed as follows:

$$\theta_{\text{tgt}} = \operatorname{atan2}(y_G - y_F,\ x_G - x_F)\,. \tag{30}$$

When the leader is present, a second contribution is introduced to discourage trajectories that would enter a circular region of influence around the leader (no-go disc). By assuming that the leader and follower positions are (x_L, y_L) and (x_F, y_F), respectively the relative displacement and the associated distance are calculated as follows:

$$\Delta x_L = x_L - x_F, \qquad \Delta y_L = y_L - y_F, \qquad d_L = \sqrt{\Delta x_L^2 + \Delta y_L^2}\,. \tag{31}$$

A repulsive direction is then defined as the direction opposite to the follower-to-leader vector:

$$\theta_{\text{rep}} = \text{atan2}(-\Delta y_L,\ -\Delta x_L)\,, \tag{32}$$

so that, when the leader is sufficiently close, the desired direction is progressively deflected away from the goal and towards a heading that increases the separation from the leader.

This deflection is activated only within a prescribed avoidance radius r_{avoid}. For $d_L < r_{\text{avoid}}$ (and excluding the numerically degenerate case $d_L \approx 0$), the final planar direction is obtained by blending attraction and repulsion through a scalar coefficient:

$$\lambda = \text{sat}_{[0,1]}\left(\frac{d_L - d_{\text{safety,enter}}}{r_{\text{avoid}} - d_{\text{safety,enter}}}\right)\,, \tag{33}$$

where $d_{\text{safety,enter}}$ denotes the entry threshold of the most conservative region, and $\text{sat}_{[0,1]}$ enforces $\lambda \in [0, 1]$. With this choice, repulsion dominates close to the safety threshold ($\lambda \to 0$), while the attractive behaviour is recovered near the edge of the avoidance region ($\lambda \to 1$).

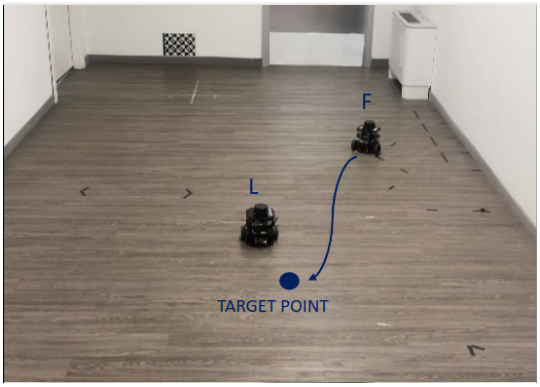
Operationally, the blending is performed in Cartesian form by combining the unit vectors associated with the two headings:

$$\mathbf{u} = \lambda \begin{bmatrix}\cos\theta_{\text{tgt}} \\ \sin\theta_{\text{tgt}}\end{bmatrix} + (1-\lambda)\begin{bmatrix}\cos\theta_{\text{rep}} \\ \sin\theta_{\text{rep}}\end{bmatrix}, \qquad \theta_{\text{field}} = \text{atan2}(u_y,\ u_x)\,. \tag{34}$$

Outside the region of influence, the field reduces to the purely attractive direction,

$$\theta_{\text{field}} = \theta_{\text{tgt}}, \qquad \text{for } d_L \geq r_{\text{avoid}}\,. \tag{35}$$

This construction results in a continuous directional field that transitions smoothly from goal-seeking behaviour to leader-avoidance behaviour, thereby preventing abrupt changes in the commanded direction of motion, as illustrated in Figure 5.3.



(a) Path generated by the controller towards the formation target point

(b) Safety zone (no-go disc) around the leader

Figure 5.3: Controller-generated path and safety-zone definition used for leader avoidance during formation tracking

The resulting circumnavigation behaviour around the safety zone, in which the follower steers away from the leader while maintaining progress toward the formation target, is shown in Figure 5.4.

Figure 5.4: Circumnavigation of the safety zone during navigation

5.3.2 Heading error and choice of angular control variable

After defining the desired direction of motion in the plane through θ_{field}, the controller expresses the mismatch between the commanded direction and the

current follower attitude by introducing a heading error:

$$e_{\text{head}} = \text{wrap}(\theta_{\text{field}} - \psi_F)\,, \tag{36}$$

where ψ_F is the follower yaw angle and $\text{wrap}(\cdot)$ denotes an angle normalisation operator returning values in $[-\pi, \pi]$.
While e_{head} provides an effective steering signal during the approach phase, it becomes less reliable in the immediate vicinity of the goal. When d_G is small, θ_{tgt} (and thus θ_{field}) may vary rapidly even under minor perturbations in $(\Delta x, \Delta y)$, leading to unnecessary oscillations in the angular command. In this regime, the best approach to maintain robustness is prioritising a direct closure of the orientation error e_ψ towards the desired terminal attitude ψ_{target}. For this reason, the implemented control uses a distance-based selector that switches the angular regressor according to the current goal distance:

$$e_\omega = \begin{cases} e_{\text{head}}, & d_G > 0.2 \text{ m}, \\ e_\psi, & d_G \leq 0.2 \text{ m}. \end{cases} \tag{37}$$

This choice determines a direction-tracking behaviour when the target is far, which progressively turns into an attitude-regulation policy when fine alignment is required near the goal.

5.3.3 Nominal control law on a differential basis

The nominal controller maps the translational and rotational error measurements into velocity commands through a proportional structure. The linear component is driven by the longitudinal tracking error expressed in the follower frame ($e_{x,F}$) so that the commanded speed directly acts on the forward/backward displacement in relation to the current follower heading. In parallel, the angular component is driven by the selected angular error e_ω, which represents either a direction-tracking mismatch during approach or an attitude-adjustment mismatch when operating close to the goal.
Using the notation $k_v = \texttt{k\_linear}$ and $k_\omega = \texttt{k\_angular}$ for the proportional gains, the resulting commands are as follows:

$$v = k_v\, e_{x,F}, \qquad \omega = k_\omega\, e_\omega\,. \tag{38}$$

This choice is consistent with the kinematics of a differential-drive platform: v governs motion along the follower forward axis, while ω regulates the evolution of the yaw angle to achieve alignment with the commanded direction of motion or, in the terminal regime, convergence to the desired final orientation.

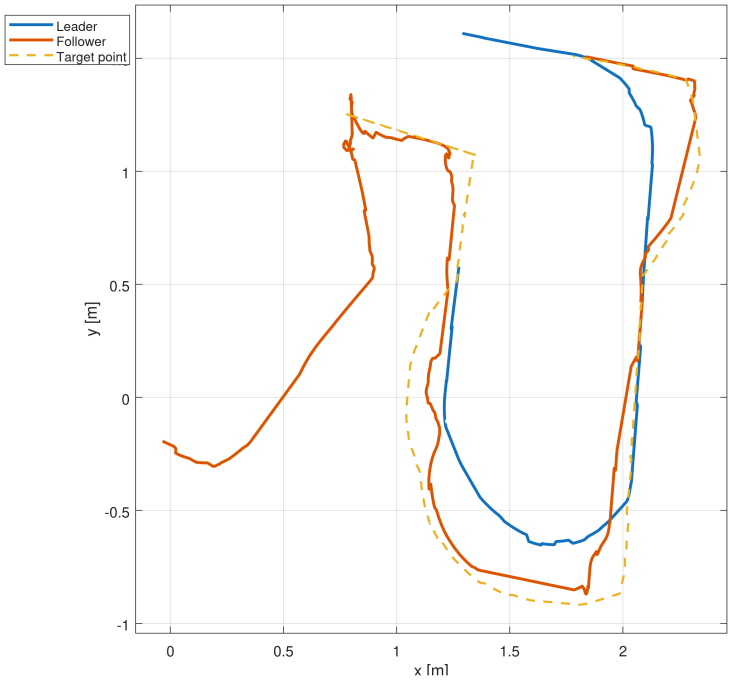


Figure 5.5: Leader and follower trajectories in the map frame during baseline formation tracking.

While (v_0, ω_0) provides the nominal continuous-time tracking action, the commands actually published to the follower are obtained after a supervisory modulation.
A finite-state machine (`NORMAL`, `SLOWDOWN`, `SAFETY`, `REACHED`) scales or overrides the nominal commands as a function of the leader-follower distance and of the relative geometry, in order to enforce inter-robot safety constraints and to stabilize behaviour near critical configurations. The resulting leader-follower motion within the plane is illustrated in Fig. 5.5.
The convergence towards the formation target is quantified in Fig. 5.6.

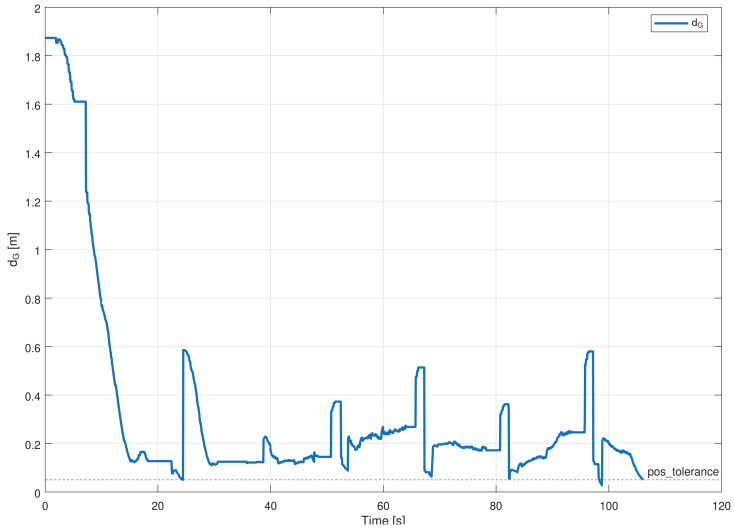


Figure 5.6: Distance to the formation target over time. The horizontal line indicates the position tolerance $\varepsilon_{pos} =$ `pos_tolerance`.

The state-dependent modulation rules are outlined in greater detail in the following section.

5.4 State machine

The supervision of the follower behaviour is implemented through a finite state machine that, at each iteration of the control loop, selects the operating mode and regulates how the continuous control law is applied. This supervisory layer does not replace the continuous controller; rather, it modulates its output by imposing reductions and dedicated constraints whenever the leader-follower geometry approaches critical configurations. In the current implementation, four operating states have been considered (`NORMAL`, `SLOWDOWN`, `SAFETY`, `REACHED`). In addition, an auxiliary flag enables a transitional repositioning phase when the initial leader-follower configuration is unfavourable, so that the follower can enter the nominal regime from a geometrically consistent arrangement.
The mode is updated by a dedicated function depending primarily on the leader-follower distance d_L. Moreover, to avoid rapid switching when d_L fluctuates close to a boundary, the transitions are governed by distinct entry/exit thresholds (hysteresis). This supervisory logic relies on:

- the leader–follower distance d_L;

- hysteresis thresholds for the conservative safety region:

$$d_S^{in} = \texttt{safety\_distance\_enter}, \qquad d_S^{out} = \texttt{safety\_distance\_exit},$$

and for the slowdown region,

$$d_D^{in} = \texttt{slow\_down\_distance\_enter}, \qquad d_D^{out} = \texttt{slow\_down\_distance\_exit};$$

- the convergence tolerances used to detect `REACHED`, already introduced as $\varepsilon_{pos} = \texttt{pos\_tolerance}$ and $\varepsilon_{\psi} = \texttt{yaw\_tolerance}$.

Furthermore, in order to identify those situations in which the leader is effectively ahead of the follower (as opposed to being lateral or behind), a frontal-cone condition is evaluated in the follower frame:

$$\chi_{\text{front}} \iff (x_L^F > 0) \ \wedge \ \left|\text{atan2}(y_L^F, x_L^F)\right| < \theta_{\text{cone}}, \tag{39}$$

with $\theta_{\text{cone}} = \texttt{front\_cone\_angle}$.
At each cycle, the continuous controller first produces nominal commands based on the error variables defined in the previous section:

$$v_0 = k_v \, e_{x,F}, \qquad \omega_0 = k_\omega \, e_\omega, \tag{40}$$

where $e_{x,F}$ is the longitudinal error in the follower frame and e_ω is the selected angular error (heading-based during the approach phase, yaw-based near the goal). The current operating mode, stored in `current_mode`, then determines how (v_0, ω_0) are transformed into the commands that are actually published. Depending on the state, this modulation may preserve the nominal behaviour, apply conservative reductions, or enforce safety-oriented constraints: the aim is to maintain a stable response and to prevent entering critical configurations when the leader-follower distance approaches the hysteresis regions and/or the leader lies within the frontal cone.

NORMAL
The `NORMAL` state corresponds to nominal operation. In this regime, the supervisor does not alter the output of the continuous controller, and the commanded velocities coincide with the nominal ones:

$$v = v_0, \qquad \omega = \omega_0. \tag{41}$$

SLOWDOWN
The `SLOWDOWN` state introduces a conservative reduction of the commands when the leader-follower distance enters an intermediate region. The objective is to temper the pursuit dynamics as the follower approaches the leader, without immediately

resorting to the most restrictive safety behaviour. In practical terms, the translational component responsible for closing the distance is reduced; moreover, when the leader lies within the frontal cone, the angular command is also attenuated to prevent sudden rotations while the platform is already decelerating.
The frontal configuration deserves special attention, because a large forward command tends to reduce d_L quickly along the direction of travel. This could potentially bring the follower close to the leader while a lateral misalignment is still present. To mitigate such an effect, the linear speed is scaled continuously as a function of d_L through the factor:

$$s = \mathrm{clip}\left(\frac{d_L - d_S^{out}}{d_D^{out} - d_S^{out}},\ 0.2,\ 1\right), \tag{42}$$

so that the resulting commands become

$$v = s\, v_0, \qquad \omega = 0.8\, \omega_0. \tag{43}$$

Here, $\mathrm{clip}(\cdot)$ denotes a saturation operator that bounds its argument within a prescribed interval. In general, for $x \in \mathbb{R}$ and $a < b$,

$$\mathrm{clip}(x, a, b) = \begin{cases} a, & x < a, \\ x, & a \le x \le b, \\ b, & x > b. \end{cases} \tag{44}$$

With $\mathrm{clip}(\cdot, 0.2, 1)$, the scaling factor is constrained to $s \in [0.2, 1]$, yielding $s = 0.2$ for $d_L \le d_S^{out}$ and recovering $s = 1$ for $d_L \ge d_D^{out}$. While the lower bound prevents the follower from excessively reducing v and actually being stalled in the slowdown region, the upper bound prevents any amplification of the nominal command. A gradual dependence on d_L, rather than a step reduction, provides a smoother approach and reduces sensitivity to small oscillations of d_L around the thresholds.

The additional attenuation $\omega = 0.8\, \omega_0$ in the frontal case limits overly aggressive angular corrections that could otherwise generate zigzagging trajectories or local tracking instabilities while the platform slows down. The goal is to preserve a smooth and predictable motion by reducing angular reactivity without suppressing the steering action.

When the leader is not within the frontal cone, the risk of a rapid direct closing of the distance is reduced. In this configuration, a constant reduction of the linear command is sufficient, while the angular component can be left unchanged:

$$v = 0.85\, v_0, \qquad \omega = \omega_0. \tag{45}$$

The 0.85 factor provides a compromise between caution near the leader and adequate responsiveness, so that convergence to the formation pose is not excessively degraded.

SAFETY

The `SAFETY` state represents the most conservative operating mode and is entered when the leader-follower distance reaches the innermost, most critical region. The purpose of this regime is to minimise collision risk and to prevent aggressive manoeuvres when the available space is limited. As in the slowdown case, the supervisory logic determines whether the leader lies within the follower's frontal cone, since the probability of an unsafe approach is highest when the leader is ahead along the direction of travel.

When the leader is located within the frontal cone, the controller explicitly avoids a further approach by inhibiting the translational component and imposing a minimum linear command:

$$v = v_{\min}, \tag{46}$$

with $v_{\min} = \texttt{v\_min\_safety} = 0$, i.e., a translational stop. This choice separates the roles of the two control channels: translation, which would directly decrease d_L, is suppressed, while rotation is preserved so that the follower can change its attitude and "unlock" the configuration, allowing a return to a safer geometry.

The angular velocity is then generated through a piecewise law that differentiates between extreme proximity and a less critical, yet still conservative, condition:

$$\omega = \begin{cases} 0.8\,\sigma, & d_L < 0.9\,d_S^{in}, \\ 0.5\,k_\omega\,e_{\text{head}}, & d_L \geq 0.9\,d_S^{in}. \end{cases} \tag{47}$$

The additional internal threshold $0.9\,d_S^{in}$ separates two regimes. The closest one features a decisive rotation with fixed magnitude to rapidly modify the relative configuration; the second one, instead, is characterized by a proportional steering to avoid unnecessarily abrupt rotations, while still maintaining a conservative behaviour.

The sign $\sigma \in \{-1, +1\}$ is selected depending on the lateral position of the leader in the follower frame:

$$\sigma = \begin{cases} +1, & y_L^F < 0, \\ -1, & y_L^F > 0. \end{cases} \tag{48}$$

This rule induces a rotation that tends to move the follower orientation laterally away from the leader: if the leader is positioned on the left of the follower ($y_L^F > 0$),

the commanded sign produces a rotation in the opposite direction, whereas if the leader is positioned on its right ($y_L^F < 0$), the rotation is commanded with the opposite sign. In this extreme proximity regime, the objective is not to optimise target tracking, but to promptly exit the riskiest configuration by achieving a rapid change in orientation.

In the less critical branch, the angular command is expressed directly in terms of e_{head}. Operationally, this choice prioritises alignment with the planar direction defined by the directional field (which already includes the repulsive contribution), rather than focusing on the terminal yaw regulation towards the final formation orientation.

When the leader is not within the frontal cone, the situation tends to be less critical: a direct closing of the distance along the direction of travel is less likely, and the follower has more freedom to correct its pose. In this case, a cautious tracking behaviour is maintained by significantly reducing the linear command and attenuating the angular response:

$$v = 0.3\, v_0, \qquad \omega = 0.7\, k_\omega\, e_{\text{head}}. \tag{49}$$

The 0.3 factor limits forward motion while preserving a non-zero translational component, thus allowing convergence when the geometry is favourable. The use of e_{head} for ω remains consistent with the objective of the mode, that is to favour a safe direction of motion within the plane rather than imposing the final yaw alignment.

Overall, the `SAFETY` state applies a highly conservative modulation of the nominal commands: in the most critical frontal configuration, translation is inhibited and rotation is used to rapidly escape the frontal collision risk with the leader, whereas in less critical configurations the tracking action is maintained, but with strongly reduced authority and with an orientation reference driven by the directional field.

REACHED

The `REACHED` state represents a stable stop condition that is entered once the follower has achieved the formation pose within the prescribed tolerances. In this regime, the control objective is no longer to continuously drive the residual errors to zero, but rather to prevent micro-corrections and limit-cycle oscillations that may arise from sensor noise, discretisation of the control loop, and small fluctuations in the pose estimates. This state, therefore, defines a deadband-like behaviour, in which the formation is deemed as satisfactorily reached and the controller output is stabilised.

The transition to `REACHED` is only evaluated when the repositioning phase is inactive: in that case the tracked reference corresponds to an intermediate waypoint rather than to the final formation pose. The entry condition requires the simultaneous satisfaction of a position and a yaw tolerance:

$$d_G < \varepsilon_{pos}, \qquad |e_\psi| < \varepsilon_\psi, \qquad \texttt{reposition\_active} = \text{False}. \tag{50}$$

These inequalities ensure that the follower is not only close to the target point, but also aligned with the desired terminal orientation, thereby confirming that the intended relative configuration has been achieved rather than approached only in position.
When `REACHED` is active, the supervisor enforces a complete stop by publishing a null command,

$$v = 0, \qquad \omega = 0, \tag{51}$$

and the current control iteration is terminated immediately after publication. This early return prevents any additional control computations from being carried out and applied within the same cycle, thus ensuring that the commanded output is determined solely by the reach condition. Operationally, this provides a clean termination of the tracking behaviour and improves formation holding stability in the presence of measurement noise and small variations of the estimated target pose.

5.5 Transitions on d_L

The transition logic is designed around a hysteresis mechanism on the leader-follower distance d_L, which is implemented by adopting distinct entry and exit thresholds for each region. This prevents rapid mode switching when d_L fluctuates close to a boundary, and generates a more stable supervisory behaviour.
When the system is already in `SAFETY`, the supervisor remains in the conservative regime as long as the distance does not clearly exceed the corresponding exit threshold. Only once the separation increases beyond d_S^{out} does the state transitions to `SLOWDOWN`:

$$d_L \leq d_S^{out} \Rightarrow \text{remains } \texttt{SAFETY}, \tag{52}$$

$$d_L > d_S^{out} \Rightarrow \texttt{SAFETY} \rightarrow \texttt{SLOWDOWN}. \tag{53}$$

When operating in `SLOWDOWN`, the mode acts as an intermediate buffer between the nominal behaviour and the safety regime. If the distance decreases and crosses the safety entry threshold d_S^{in}, the supervisor escalates to `SAFETY`. Conversely, if the distance increases sufficiently and exceeds the slowdown exit threshold d_D^{out}, nominal

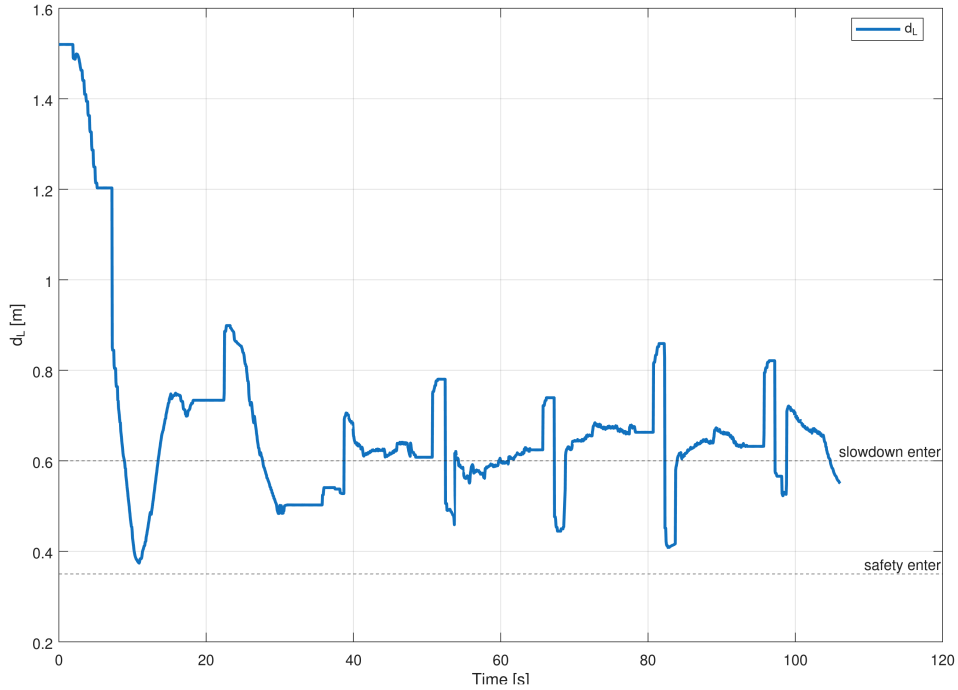


Figure 5.7: Inter-robot distance d_L over time with the configured slowdown and safety thresholds.

operation is restored, and the state transitions to `NORMAL`. If neither condition is met, the system remains in `SLOWDOWN`:

$$d_L < d_S^{in} \Rightarrow \texttt{SLOWDOWN} \rightarrow \texttt{SAFETY}, \tag{54}$$

$$d_L > d_D^{out} \Rightarrow \texttt{SLOWDOWN} \rightarrow \texttt{NORMAL}. \tag{55}$$

Finally, when the controller is in `NORMAL` (and likewise when exiting `REACHED`), the supervisor selects a more conservative mode as soon as d_L enters the corresponding regions. More specifically, entering the innermost band triggers a direct transition to `SAFETY`, whereas entering the intermediate band triggers a transition to `SLOWDOWN`:

$$d_L < d_S^{in} \Rightarrow \texttt{NORMAL} \rightarrow \texttt{SAFETY}, \tag{56}$$

$$d_S^{in} \leq d_L < d_D^{in} \Rightarrow \texttt{NORMAL} \rightarrow \texttt{SLOWDOWN}. \tag{57}$$

Overall, the use of separate $(\cdot)^{in}$ and $(\cdot)^{out}$ thresholds ensures that transitions occur only when d_L changes sufficiently, thereby reducing sensitivity to noise and limiting oscillations of the supervisory mode around the boundaries. The effect of the supervisory thresholds on the inter-robot distance is shown in Fig. 5.7.

5.6 Safety logic

In the leader-follower scheme considered in this work, certain relative geometries can make the pursuit of the nominal formation pose inefficient and, in the most conservative settings, may even lead to quasi-stall behaviour. For instance, this is what happens when the follower is located ahead of the leader along the leader direction of travel: although the desired formation position lies behind the leader, a direct target-tracking strategy may fail to reliably generate the bypass manoeuvre required to regain a favourable arrangement, especially when speed reductions or safety constraints limit the available translational authority.

It is important to note that the waypoint-based repositioning manoeuvre is *not* triggered by the safety modes with distance thresholds. Distance thresholds ($d_S^{in/out}$, $d_D^{in/out}$) only select conservative speed/stop behaviours through the state machine, whereas repositioning is activated by a separate geometric condition indicating an unfavourable configuration (i.e., the follower lies in front of the leader in the leader-attached frame, within a bounded lateral corridor, and at a moderate distance).

With the aim of handling these situations deterministically, the proposed approach introduces a dedicated repositioning manoeuvre based on an intermediate waypoint. The role of this waypoint is to drive the follower back into a geometrically favourable region, behind and slightly to the side of the leader, after which the nominal formation pursuit can be resumed under conditions where the continuous controller provides consistent convergence.

5.6.1 Detection of unfavourable configuration: follower in front of leader

In order to detect an unfavourable configuration, the follower's relative position must be expressed within the leader frame. Denoting the leader pose by (x_L, y_L, ψ_L) and the follower position by (x_F, y_F), the relative displacement in the global frame is defined as

$$\Delta x_{FL} = x_F - x_L, \qquad \Delta y_{FL} = y_F - y_L. \tag{58}$$

By projecting this displacement into the leader frame, the following is obtained:

$$\Delta x_{FL}^L = \cos\psi_L \, \Delta x_{FL} + \sin\psi_L \, \Delta y_{FL}, \tag{59}$$

$$\Delta y_{FL}^L = -\sin\psi_L \, \Delta x_{FL} + \cos\psi_L \, \Delta y_{FL}. \tag{60}$$

Repositioning is activated when the follower is actually ahead of the leader ($\Delta x_{FL}^L > 0$) and the positioning corresponds to a genuine "blocking" geometry, rather than

a wide overtaking configuration that provides sufficient lateral clearance. For this reason, activation is limited to a bounded lateral corridor and to moderate distances:

$$\Delta x_{FL}^{L} > 0, \qquad |\Delta y_{FL}^{L}| < 0.4, \qquad d_L < 1.0, \tag{61}$$

where the leader–follower distance is

$$d_L = \sqrt{(x_L - x_F)^2 + (y_L - y_F)^2}. \tag{62}$$

Together, these criteria restrict the manoeuvre to cases where it is both meaningful (the follower is truly in front of the leader) and practically manageable (not too far from the leader, and not already laterally displaced enough to resolve the configuration without an explicit recovery action).

5.6.2 Choosing the side and constructing the "safe" waypoint

Once an unfavourable configuration is detected, the controller constructs an intermediate waypoint behind and to the side of the leader. The detour side is selected according to whether the follower lies to the left or right of the leader, as indicated by the sign of Δy_{FL}^{L}, so as to produce a consistent bypass direction:

$$\text{side} = \begin{cases} +1, & \Delta y_{FL}^{L} \geq 0, \\ -1, & \Delta y_{FL}^{L} < 0. \end{cases} \tag{63}$$

Two offsets are then defined: a longitudinal offset behind the leader, and a lateral offset away from the leader centreline. In the implementation process, both are expressed in terms of nominal formation distance and additional safety margins:

$$d_{back} = \texttt{distance}+\texttt{safe\_back\_offset}, \qquad d_{side} = \texttt{distance}+\texttt{safe\_side\_offset}. \tag{64}$$

The waypoint (x_{safe}, y_{safe}) in the global frame is obtained by combining the leader longitudinal and lateral directions:

$$x_{safe} = x_L - d_{back} \cos\psi_L - \text{side}\, d_{side} \sin\psi_L, \tag{65}$$

$$y_{safe} = y_L - d_{back} \sin\psi_L + \text{side}\, d_{side} \cos\psi_L. \tag{66}$$

The resulting waypoint, illustrated in Figure 5.8, is placed behind and laterally offset from the leader, ensuring a geometrically consistent entry into the nominal formation regime.

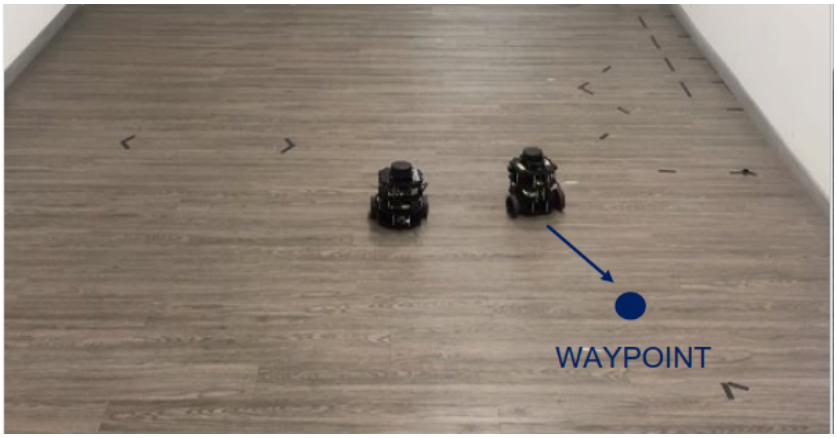


Figure 5.8: Safe waypoint generated behind and laterally offset from the leader.

The target yaw associated with the waypoint is chosen as ψ_L, so that, when the follower exits the manoeuvre, it is already aligned with the leader heading and can transition smoothly to the subsequent formation-maintenance phase. Figure 5.9 shows the follower position after the waypoint has been successfully reached.

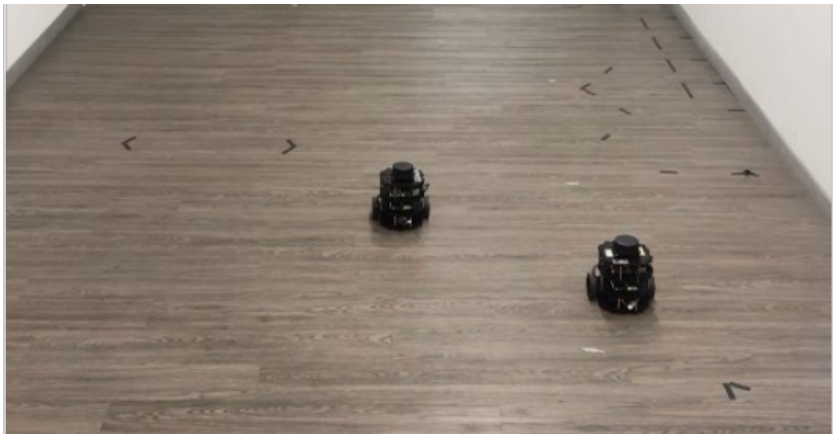


Figure 5.9: Follower position after reaching the safe waypoint and before resuming formation tracking

During repositioning, the commanded target is the waypoint rather than the final formation pose. The manoeuvre is terminated when the follower enters a neighbourhood of the waypoint defined by the position tolerance:

$$\sqrt{(x_{safe} - x_F)^2 + (y_{safe} - y_F)^2} < \varepsilon_{pos}, \tag{67}$$

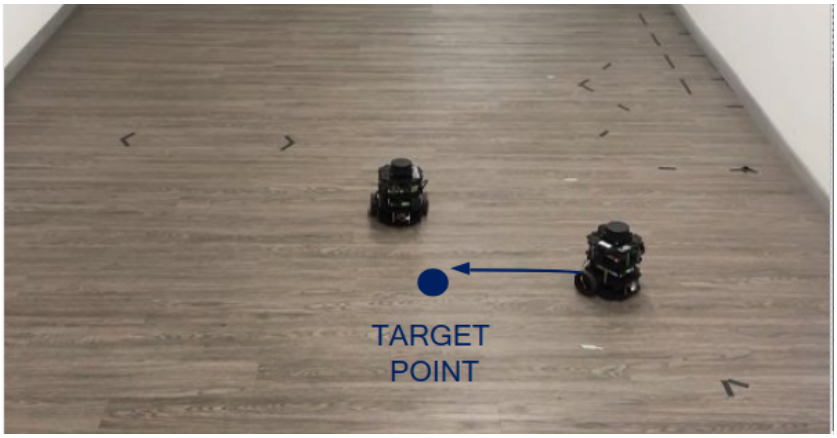


Figure 5.10: Return to the nominal formation target after waypoint-based recovery.

with $\varepsilon_{pos} = \texttt{pos\_tolerance}$. Once this condition holds, the controller returns to tracking the nominal formation target, as illustrated in Figure 5.10.

5.6.3 Why waypoints reduce the risk of stalling

The waypoint strategy replaces a single recovery problem - that may be poorly conditioned in geometries concerning the front of the leader- with a sequence of targets that provides a clearer geometric descent direction.

When the follower is ahead of the leader, but the desired formation point is behind it, direct pursuit typically requires a coordinated rotation-translation manoeuvre that can be strongly degraded by conservative speed constraints and by the coupling between heading regulation and forward motion. By explicitly directing the follower to a point in the back and laterally offset, the repositioning phase restores a configuration in which the nominal formation target can be reached without trajectories that pass close to the leader.
Moreover, the intermediate target reduces the likelihood of accumulating near-null command sequences: even under conservative limits, the waypoint introduces a consistent gradient in the relative geometry that promotes a reliable exit from the unfavourable region.

Overall, the waypoint-based repositioning acts as a geometric recovery procedure: it leaves the formation definition unchanged but introduces an intermediate reference that makes convergence more robust whenever the initial leader-follower arrangement is not nominal.

Chapter 6

Centralized Control with Virtual Frame

In obstacle-free or weakly constrained environments, a leader–follower baseline provides a simple and effective mechanism to maintain the desired formation: the leader defines the motion, and the followers regulate their relative pose based on a direct tracking law. However, this approach is not free of structural limitations, clearly exposed in cluttered indoor settings with static obstacles and narrow passages.

Pure formation tracking does not explicitly account for obstacle geometry and may therefore produce follower commands that would be feasible in a free space, but turn out to be unsafe or inefficient in the vicinity of obstacles. Furthermore, constrained environments amplify transient formation distortions and can lead to degraded configurations, such as excessive stretching of the coupling constraint, local deadlocks, or stop-and-go behaviours, when formation maintenance and collision avoidance need to be simultaneously fulfilled.

In order to address these limitations, this chapter introduces a centralized virtual-frame strategy that decouples group-level coordination from local motion execution. Rather than commanding each robot solely through relative tracking, the formation is specified with respect to a shared virtual reference frame anchored to a virtual centre (or virtual leader) representing the collective motion of the team. Each robot is assigned a desired target pose defined by a fixed offset in this virtual frame, and these targets are updated online as the virtual centre evolves. The experimental setup used to validate the proposed approach is shown in Figure 6.1: two TurtleBot3 robots connected by a flexible tether navigate an indoor environment in the presence of a static obstacle.

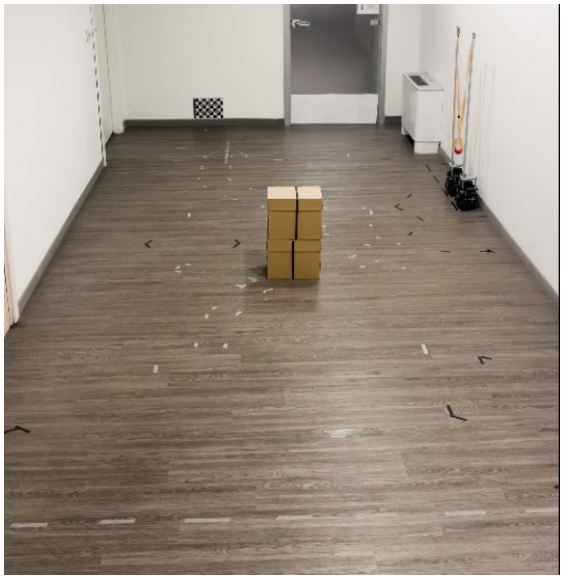


Figure 6.1: Experimental environment used for validation. The two TurtleBot3 robots are equipped with vertical poles connected by a flexible tether carrying a suspended payload. A static obstacle (cardboard boxes) is placed along the navigation path to trigger the obstacle management pipeline.

6.1 Virtual Frame Description

The virtual frame provides a shared reference describing the team motion at the group level. A virtual centre (virtual leader), propagated in the global map frame, is then used to generate formation targets as fixed geometric offsets. This formulation preserves a coherent team objective and allows each robot to execute obstacle-aware motion through the navigation stack.

6.1.1 Implementation

The virtual centre is implemented as a ROS 2 node that propagates a unicycle kinematic model driven by commanded velocities. The node subscribes to a velocity command and publishes the resulting pose of the virtual centre. .

- **Input:** `TwistStamped` on `/formation/cmd_vel`
- **Output:** `PoseStamped` on `/formation/pose`
- **Reference frame:** `map`

Stamped messages provide a consistent time reference for multi-robot operation and support a reliable implementation of the time-dependent logic (e.g., command timeouts).
At a fixed update rate (default 30Hz), the node updates the virtual centre state (x, y, θ) by integrating the commanded linear and angular velocities over Δt:

$$\begin{aligned} x_{k+1} &= x_k + v\cos(\theta_k)\,\Delta t, \\ y_{k+1} &= y_k + v\sin(\theta_k)\,\Delta t, \\ \theta_{k+1} &= \mathrm{wrap}(\theta_k + \omega\,\Delta t)\,. \end{aligned} \tag{68}$$

where v and ω are the commanded linear and angular velocities, and $\mathrm{wrap}(\cdot)$ denotes normalization to $[-\pi, \pi]$. The yaw angle θ is then converted into the quaternion $(0, 0, \sin(\theta/2), \cos(\theta/2))$ to populate the orientation field of the published PoseStamped.

6.1.2 Stability and Robustness Mechanisms

To ensure a stable, teleoperation-friendly virtual reference, three practical mechanisms have been introduced.

Command saturation First, the commanded velocities are bounded by predefined limits $(v_{\max}, \omega_{\max})$, in order to prevent abrupt or unrealistic updates of the virtual reference and maintain compatibility with the physical platform constraints.

Low-pass filtering Second, to mitigate discontinuities introduced by discrete teleoperation inputs, velocity commands are smoothed via exponential filtering:

$$\begin{aligned} v_f^{k+1} &= \alpha_v\, v_{\text{target}}^{k} + (1-\alpha_v)\, v_f^{k}, \\ \omega_f^{k+1} &= \alpha_\omega\, \omega_{\text{target}}^{k} + (1-\alpha_\omega)\, \omega_f^{k}. \end{aligned} \tag{69}$$

with $\alpha_v, \alpha_\omega \in [0, 1]$. Smaller values increase smoothing, whereas larger values improve responsiveness.

Command timeout If no new command is received within a configurable time interval, the target command is set to zero. This prevents the virtual centre from drifting under stale inputs and ensures behaviour predictability in the event of teleoperation interruptions.
Finally, the initial pose of the virtual centre is set on the parameters x_0, y_0, and θ_0, enabling consistent alignment with the global map and the experimental setup.

The virtual centre is driven by the publication of velocity commands to `/formation/cmd_vel`. During experiments, keyboard teleoperation was used to remap the standard teleop output to the virtual-centre command topic:

```
ros2 run teleop_twist_keyboard teleop_twist_keyboard \
  --ros-args -r /cmd_vel:=/formation/cmd_vel
```

Figure 6.2 illustrates the initial SETUP phase, when the robots converge to their assigned formation positions in relation to the virtual centre.

Figure 6.2: SETUP phase: the robots move from their initial positions to the formation configuration defined by the virtual centre. Blue arrows indicate the commanded trajectories; the red arrow represents the virtual centre position and heading.

6.1.3 Visualization and Validation

The published pose of the virtual centre (`/formation/pose`) is visualized in RViz using a `Pose` display. RViz renders an arrow that represents the virtual centre position and heading in the global `map` frame and updates it in real time as commands are applied. This visualization, shown in Figure 6.3, allows efficient

monitoring of the following features: correct integration of the unicycle model, consistency of the published reference frame, the bounded, smooth reference motion under saturation and filtering, and the correct timeout behaviour when commands stop.

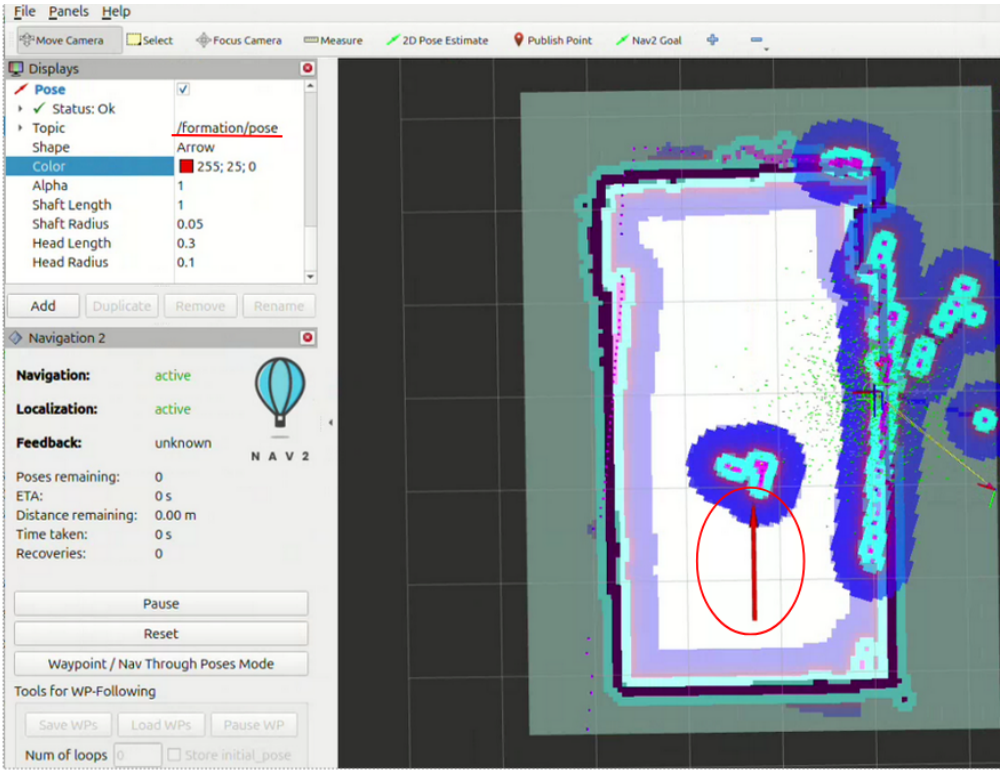


Figure 6.3: RViz visualization of the virtual centre during the experiment. The red arrow (highlighted by the red circle) represents the virtual centre pose published on `/formation/pose` in the global `map` frame. The surrounding costmap shows the obstacle inflation layers used by the navigation stack for collision avoidance.

6.2 Dynamic goal tracking using Nav2

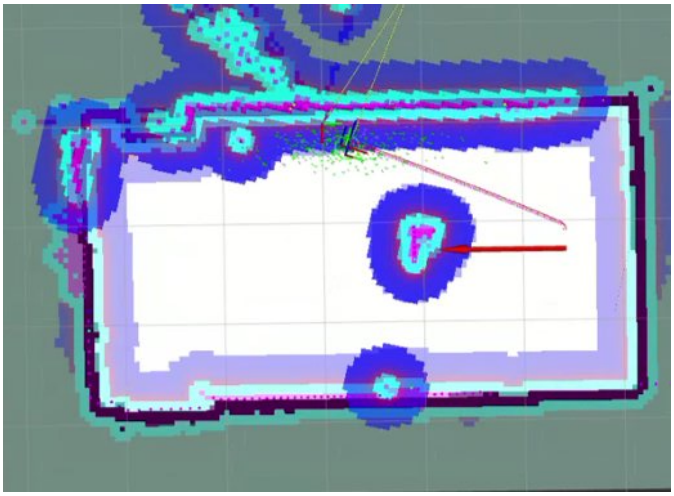
The virtual-centre pose, published on `/formation/pose`, is monitored in RViz with a `Pose` display. RViz renders an arrow in the global `map` frame, whose position and orientation correspond to the virtual centre and are updated in real time as velocity commands are applied. This visualization immediately validates the accuracy of the unicycle model numerical integration, consistency of the published reference frame, smoothness of the bounded reference motion under command saturation and low-pass filtering, and correctness of the timeout behaviour when command

updates stop.

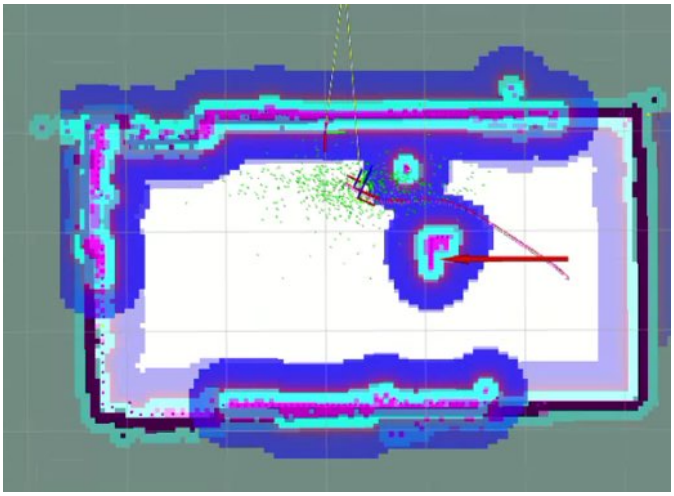
6.2.1 Dynamic goal tracking as a navigation problem

While conventional navigation is based on a fixed goal pose, in this project, instead, each follower must track a goal that evolves over time, because the desired pose is defined as a constant offset from the moving virtual centre. The resulting task is therefore a dynamic goal tracking problem: the navigation stack must continuously adapt its motion while the target changes, maintaining stable behaviour and avoiding stop-and-go effects or oscillations, especially in proximity to obstacles.

A more straightforward approach would involve repeatedly sending new `NavigateToPose` action goals at a high frequency. In practice, however, aggressive goal preemption is often detrimental: it interrupts the ongoing execution, frequently reinitializes parts of the navigation pipeline, and may induce oscillations or trigger unnecessary recovery behaviours. For this reason, the adopted strategy initializes navigation only once and updates the goal online without continuously preempting the action. The resulting behaviour for both robots is illustrated in Figure 6.4, where the global path is continuously recomputed toward the updated formation goal as the virtual centre moves.



(a) Robot 1: global path toward the formation goal.



(b) Robot 2: global path toward the formation goal.

Figure 6.4: RViz visualization of the dynamic goal tracking pipeline for both robots. The red arrow represents the virtual centre pose published on `/formation/pose`; the magenta path shows the global plan computed by Nav2 toward the current formation goal, updated online as the virtual centre moves.

6.2.2 Implementation overview

This design is implemented through two coordinated components:

- a custom Nav2 Behaviour Tree that enables online goal updates and periodic replanning;
- a follower node that computes the desired target pose from the virtual centre and publishes goal updates while ensuring continuity of the Nav2 action.

This preserves a clear division of responsibilities: the formation layer generates consistent team-level targets, whereas Nav2 provides obstacle-aware planning, control, and recovery behaviours at the robot level.

Behavior Tree structure

The custom Behaviour Tree is defined in an XML file and is organised around a `PipelineSequence`, which allows planning and control to overlap in time. This aspect is essential in the described setting, since the robot not only can continue tracking the currently available path, but it can also replan at a controlled rate, rather than waiting for a new plan before resuming motion. As a consequence, this structure mitigates the strict plan-then-move execution pattern and reduces the stop-and-go effects that typically arise under a purely sequential execution.

The resulting BT hierarchy can be summarised as follows:

1. **PipelineSequence**: root node enabling concurrent planning and control execution;
2. **ControllerSelector** and **PlannerSelector**: selection of the active controller and planner plugins, enabling runtime reconfiguration if required;
3. **RateController** (0.5 Hz): enforcement of periodic replanning at a fixed frequency, limiting unnecessary planning adjustments;
4. Within the rate-controlled block:
   - **GoalUpdater**: retrieval of the most recent target pose from the goal-update topic, which is then stored as a blackboard variable;
   - **ComputePathToPose**: computation of a new global path to the updated goal through the selected planner, storing the result in `path`.
5. **FollowPath**: execution of the local controller on the most recent path at its nominal high-frequency loop, typically ≥ 20 Hz.

The replanning rate is a critical design parameter. Since global planning is computationally more demanding than local control, it is neither necessary nor desirable to execute it at the same frequency as goal updates or controller iterations. In this work, a replanning rate of 0.5 Hz was adopted as a compromise: it limits unnecessary replanning actions and reduces path fluctuations, while still allowing the system to react to target motion and obstacle-driven deviations within a short, bounded delay of about 2s.

Blackboard variables and data flow The Behaviour Tree relies on blackboard variables to exchange data between nodes while avoiding tight coupling:

- `goal`: initial goal pose provided by the `NavigateToPose` action.
- `updated_goal`:most recent target pose retrieved by `GoalUpdater` from the formation goal topic.
- `path`: global plan computed by `ComputePathToPose`.
- `selected_planner`, `selected_controller`: identifiers of the active planner and controller plugins selected at runtime.

This blackboard-driven organisation keeps individual BT nodes independent and facilitates the reconfiguration or extension of the tree without requiring changes to the node implementations.

Goal update interface In Nav2, `GoalUpdater` waits for goal updates on a dedicated topic. In this implementation, the input interface is configured in the BT XML through the `input_goal` parameter, which is bound to the blackboard entry `goal` and is automatically namespaced by Nav2 for each robot. As each follower publishes goal updates on `/<robot_ns>/formation_goal`, the Nav2 namespace resolution ensures that each robot consumes only the updates associated with its own namespace. This topic-based interface cleanly separates the formation layer, which generates targets, from the navigation layer, which executes obstacle-aware planning and control.

6.2.3 Follower-side target generation

The `FormationFollower` node implements dynamic target generation and publication for a given robot namespace. Its main loop is designed to compute an up-to-date formation target from the virtual centre, keep the Nav2 action running, and publish regulated goal updates.
The node subscribes to the virtual centre pose on `/formation/pose`. Prior to publishing any target, it verifies that the robot pose is available in the `map` frame via TF (e.g., `/<robot_ns>/base_footprint`). This is an essential step: it prevents goals from being issued before consistency between localization and the TF tree is achieved, a circumstance that would lead to unreliable navigation behaviour or immediate failures. In practical terms, enforcing TF readiness improves start-up robustness and avoids publishing targets while the state estimate is still being finalised.

Target pose computation as a fixed offset

Given the virtual-centre pose (x_c, y_c, θ_c), each follower computes the desired goal pose as a lateral offset with respect to the virtual-centre heading. Such an offset is parameterised by:

- `distance_nominal`: $d_{\text{nom}} = 0.2\,\text{m}$ (compact spacing),
- `distance_open`: $d_{\text{open}} = 1.0\,\text{m}$ (expanded spacing),
- `side_sign`: $s \in [+1, -1]$ (default -1, placing the robot on the right side).

The target pose is defined as

$$x_{\text{goal}} = x_c - s, d_{\text{eff}} \sin(\theta_c),\ y_{\text{goal}} \ = y_c + s, d_{\text{eff}} \cos(\theta_c),\ \theta_{\text{goal}} = \theta_c. \tag{70}$$

This construction determines a perpendicular displacement in relation to the virtual centre heading, thereby preserving a lateral formation geometry regardless of the global orientation.

The effective spacing d_{eff} can be continuously modulated via a scalar $\lambda \in [0, 1]$ when `use_lambda=True` (default). The node subscribes to `/formation/lambda` (`Float32`) and applies

$$d_{\text{eff}} = (1 - \lambda), d_{\text{nom}} + \lambda, d_{\text{open}}. \tag{71}$$

This mechanism deforms the formation spacing without altering the underlying geometric model. The follower treats λ as an external bounded input, clamps it to $[0, 1]$, and thus ensures $d_{\text{eff}} \in [d_{\text{nom}}, d_{\text{open}}]$ even in the presence of noise or intermittent updates.

Starting and maintaining the Nav2 action

Navigation is initiated by sending a single `NavigateToPose` action goal to `/<robot_ns>/navigate_to_pose`. The goal message includes:

- `pose`: the initial target pose computed from the virtual centre,
- `behavior_tree`: the absolute path to the custom BT XML file (`bt_xml_path`).

If the XML file is missing, the node reports an error and does not start navigation, therefore preventing execution under an unintended default BT. If the action server is not available yet (e.g., Nav2 is still launching), the node repeats the attempt at regular intervals until the server is ready.

Action lifecycle and recovery After acceptance, the node tracks the action status via asynchronous callbacks. If the action terminates, either due to success or failure, the node automatically re-sends a new action goal using the current target pose. While this mechanism is rarely triggered under normal operation (since the goal is continuously updated), it provides robustness against edge cases, such as transient localization loss or terminal recovery failures. Overall, maintaining a single active action with an automatic restart mechanism ensures continuous operation without external supervision being necessary.

Continuous goal publication with regulation

Once the action is running, the node publishes both goal updates as `PoseStamped` messages on `/<robot_ns>/formation\_goal` and a debug copy on `/<robot_ns>/formation_goal_debug`. The debug topic enables the RViz inspection without interfering with the goal-update stream consumed by Nav2.
The publication of each small numerical fluctuation of the computed target is undesirable, as it may induce unnecessary replanning and jitter, particularly near obstacles. For this reason, goal updates are regulated by a dedicated routine

(`_filter_and_gate_goal`) combining three complementary mechanisms: low-pass filtering, rate limiting, and thresholding.

Low-pass filtering. The target position (x, y) and yaw θ are smoothed using a first-order exponential filter (`goal_smoothing_alpha` $= 0.35$). For position,

$$x_{\text{filtered}}(t) = \alpha, x_{\text{raw}}(t) + (1 - \alpha), x_{\text{filtered}}(t - 1), \tag{72}$$

with $\alpha = 0.35$ providing a compromise between responsiveness and smoothness. For yaw, filtering is performed on the wrapped angular increment to avoid discontinuities at $\pm\pi$:

$$\Delta\theta = \text{normalize}(\theta_{\text{raw}} - \theta_{\text{filtered}_{\text{prev}}}), \theta_{\text{filtered}} \quad = \text{normalize}(\theta_{\text{filtered}_{\text{prev}}} + \alpha, \Delta\theta). \tag{73}$$

This reduces the impact of discrete teleoperation steps and TF/localization fluctuations.

Rate limiting. Although the internal loop runs at `goal_update_hz` $= 20\,\text{Hz}$, publication is constrained by `goal_update_min_period` $= 0.10\,\text{s}$ (i.e., at most $10\,\text{Hz}$). The higher internal rate guarantees responsiveness of the target computation and frequent evaluation of the gating logic, while the capped publication rate avoids redundant message bursts.

Thresholding A new goal is published only if it differs from the last published goal by at least:

- `goal_update_min_translation` $= 0.03\,\text{m}$ in translation, or
- `goal_update_min_yaw` $= 0.05\,\text{rad}$ in yaw.

These thresholds suppress micro-updates that would not generate meaningful motion: even though comparable to typical TF/localization noise, they could still bring about planning adjustments or oscillations in constrained spaces.

End-to-end workflow

The resulting dynamic-goal tracking pipeline can be summarised as follows:

1. The formation layer publishes the virtual-centre pose on `/formation/pose` in the `map` frame.
2. Each follower verifies the TF availability, computes its target as an offset from the virtual centre (optionally modulated by λ), and publishes regulated goal updates on the namespaced topic.

3. Nav2 is started by sending a single `NavigateToPose` goal specifying the custom Behaviour Tree.

4. During execution, Nav2 periodically replans (0.5Hz) toward the most recent goal retrieved by `GoalUpdater` and continuously tracks the latest path through `FollowPath` at the controller rate ($\geq$ 20Hz).

5. If the action terminates, the follower re-issues the action goal to maintain operation.

This architecture establishes formation objectives through the virtual frame, while also leveraging Nav2 for obstacle-aware planning, control, and recovery. The interface between the two layers remains minimal (one goal topic per robot), supporting modular development and independent tuning, provided that the interface contract is preserved.

6.3 Obstacle management and robot coordination

Within the proposed architecture, each robot relies on the navigation stack to execute collision-free motion, while the formation layer continuously generates target poses that encode the desired group geometry. In cooperative transport, however, local obstacle avoidance alone is not sufficient: the robots are physically coupled, and uncoordinated detours can induce excessive separation, loss of formation coherence, or mechanically feasible, yet operationally unsafe configurations. For this reason, an additional supervisory layer needs to be introduced to manage obstacle interactions at the formation level and to apply coupling constraints in a coordinated manner.

6.3.1 Obstacle Detection and Spatial Awareness

Obstacle awareness depends on the local obstacle representation produced by the navigation pipeline. The supervisor monitors the local costmaps of both robots and analyzes occupancy grid cells within a configurable radius to detect nearby obstacles. Obstacle proximity is then summarised by a scalar metric obtained by searching for the closest relevant obstacle cell within this bounded region. Finally, a costmap threshold of 20 is used to identify inflated regions, which represent potential navigation hazards.

In order to reduce false positives and improve detection reliability, two robustness mechanisms are applied. First, detections are filtered directionally by restricting the search to a forward sector aligned with the current direction of motion, estimated from the robot-to-goal heading. This reduces spurious detections originating from structures behind the robot or laterally offset from it that do not pose an immediate threat to the formation. In the current configuration, the detection cone is typically set to a 120° aperture, providing sufficient forward coverage while rejecting rear and lateral cells.

Second, detections that can be attributed to teammates are rejected by excluding regions around the estimated pose of the other robot. This prevents mutual interference due to inflated footprints appearing in each robot's costmap. By retrieving teammate poses from TF and applying configurable ignore radii, the algorithm focuses exclusively on external obstacles and disregards occupied cells associated with the robots themselves.

Finally, to increase robustness against sensor noise and transient map updates, detections are validated through both count-based and time-based confirmation. An obstacle must be observed consistently across multiple consecutive scans and persist for a minimum duration before a state transition is triggered. This dual confirmation policy mitigates the effects of local mapping inconsistencies, localization jitter, and short-lived sensor artefacts that do not represent actual navigation hazards.

6.3.2 Coupling Constraint and Geometric Model

Coordination is formulated through an inter-robot distance constraint induced by flexible coupling. Let L denote the total length of the tether and h_p the attachment height on each robot. If d is the horizontal separation between the robots and the payload is modelled at the midpoint of the link, a simple geometric relationship links d to a clearance proxy z (payload elevation under the tether model):

$$\left(\frac{L}{2}\right)^2 = \left(\frac{d}{2}\right)^2 + (h_p - z)^2 \quad \Longrightarrow \quad z = h_p - \sqrt{\left(\frac{L}{2}\right)^2 - \left(\frac{d}{2}\right)^2} \tag{74}$$

This model provides a direct mapping based on the desired clearance level and the corresponding target inter-robot distance $d^\star$, while enforcing absolute bounds to avoid overly tight configurations or overstretching of the coupling. For the experimental setup ($L = 1.5\,\mathrm{m}$ and $h_p = 0.95\,\mathrm{m}$), the required horizontal separation can therefore be computed for any target payload elevation, enabling clearance regulation without an explicit vertical actuation. From a practical point of view, the desired clearance target z^* is specified as a parameter, and the corresponding

inter-robot separation d^* is computed analytically from the geometric model. In the experiments presented in this work, two values were tested: $z^* = 0.30\,\mathrm{m}$ and $z^* = 0.60\,\mathrm{m}$, representative of two distinct obstacle heights.

6.3.3 Supervisory state machine and hysteresis

The obstacle management policy is implemented as a finite-state supervisor with hysteresis, in order to avoid oscillatory behaviour when the estimated obstacle proximity fluctuates near decision boundaries. There are four defined operating modes:

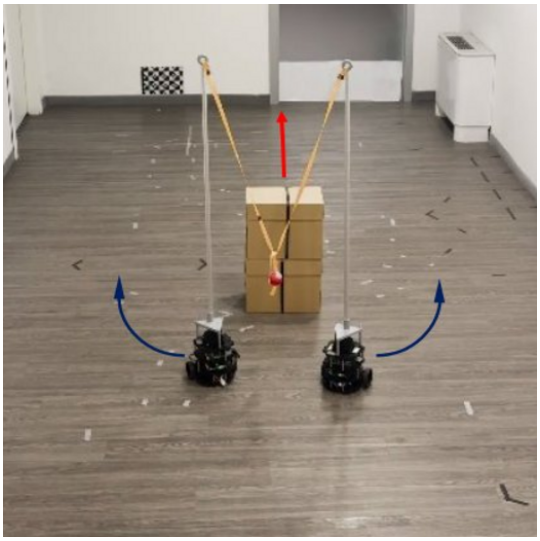
1. **NORMAL**: nominal operation. Formation goals are forwarded to the navigation layer, only subject to hard bounds on admissible separation. No obstacle-driven adaptation is applied, as shown in Figure 6.5a.

2. **APPROACHING**: an obstacle is detected within an intermediate range (0.80 m). The supervisor initiates a controlled transition from the nominal separation toward a more conservative configuration by gradually increasing the inter-robot distance, thereby preparing the system for a potential clearance increase, as illustrated in Figure 6.5b.

3. **LIFTING**: full intervention. When obstacle proximity exceeds the activation threshold (0.50 m), the target separation is set to the value associated with increased clearance, which is maintained as long as there is a critical condition. In this mode, the formation adopts the maximum configured separation to elevate the suspended payload above obstacles, as shown in Figure 6.5c.

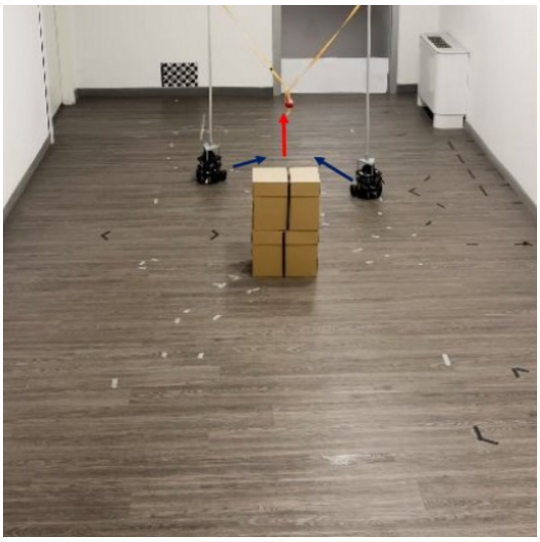
4. **CLEARING**: recovery. Once the obstacle is no longer critical (distance exceeding 0.65 m), the supervisor transitions back to the nominal formation by progressively reducing the inter-robot distance, as illustrated in Figure 6.5d.

(a) NORMAL phase.



(b) APPROACHING phase.

(c) LIFTING phase.



(d) CLEARING phase.

Figure 6.5: The four operating modes of the supervisory state machine. (**a**) NORMAL phase: the robots maintain the nominal compact formation; the red arrow indicates the virtual centre direction of motion. (**b**) APPROACHING phase: the blue curved arrows show the commanded lateral separation increase as the robots diverge around the obstacle; the red arrow indicates the virtual centre position and heading direction. (**c**) LIFTING phase: the red vertical arrow indicates the achieved payload clearance height z^*, while the red horizontal arrow shows the corresponding inter-robot separation d^*. (**d**) CLEARING phase: the blue arrows indicate the robots converging back toward the nominal formation after the obstacle has been cleared.

Hysteresis is enforced through distinct entry and exit thresholds, complemented by persistence checks requiring a minimum duration and/or consecutive detections before a transition is accepted. This asymmetric thresholding produces decisive state changes: preparation is triggered at 0.80,
mathrmm, full lifting at 0.50 m, clearing begins at 0.65 m, and nominal operation is restored at 0.75,
mathrmm. The combined use of hysteresis and confirmation mitigates chattering and improves run-to-run reproducibility when obstacle-distance estimates vary near the switching boundaries. Figure 6.6 shows the measured tether height during the obstacle-clearing manoeuvre for both target clearance values, validating the proposed geometric model and supervisory state machine.

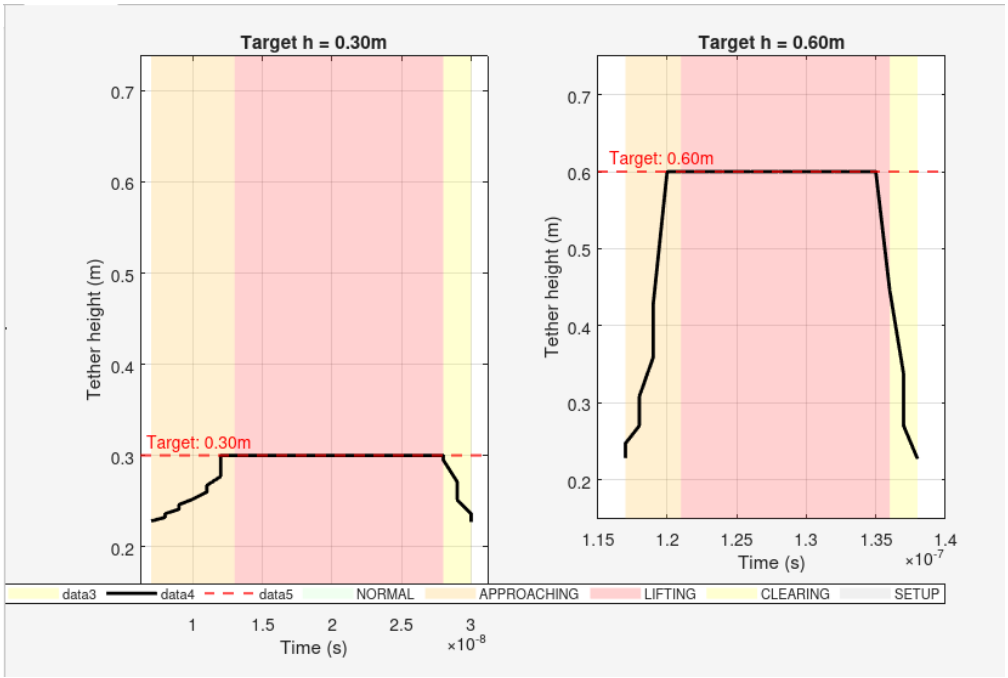


Figure 6.6: Measured tether height during the obstacle-clearing manoeuvre for two target clearance values. Left: $z^* = 0.30\,\mathrm{m}$; Right: $z^* = 0.60\,\mathrm{m}$. The coloured background indicates the active supervisory mode: yellow for APPROACHING, red for LIFTING, and light yellow for CLEARING. The dashed red line indicates the target clearance height z^*, which is reached and maintained during the LIFTING phase, confirming that the geometric model correctly maps the desired clearance to the corresponding inter-robot separation d^*.

6.3.4 Coordinated formation adaptation

When intervention is required, coordination is achieved by modifying the formation targets rather than overriding low-level control. Specifically, the supervisor reshapes the two navigation goals so that the commanded inter-robot separation matches $d^\star$, while also preserving the midpoint of the goals. Midpoint preservation is essential because it maintains consistency with the team-level intent encoded by the virtual frame trajectory: adaptation focuses on scaling the formation along the line connecting the robots, whereas Nav2 remains responsible for obstacle-aware planning, local avoidance, and path tracking.

This hierarchical organisation separates responsibilities: the formation layer determines the configuration to be achieved, computing target poses from obstacle context and coupling constraints; the navigation layer, instead, determines how to reach those targets. Absolute minimum and maximum separation limits are enforced at all times to prevent mechanically unfeasible or operationally unsafe configurations, regardless of the current mode.

6.3.5 Smooth transitions and feedback correction

Transitions between separations are executed smoothly rather than through different steps. In **APPROACHING** and **CLEARING**, the target distance is interpolated over finite transition intervals, selected to balance responsiveness and stability: $1.5\,\mathrm{s}$ for approach transitions, and $2.0\,\mathrm{s}$ for clearing. Temporal interpolation reduces aggressiveness and limits the likelihood of oscillations when the navigation layer replans in proximity to obstacles.

Optionally, the approach interpolation rate can be modulated by obstacle distance, increasing urgency when a threat draws close faster than the nominal ramp could accommodate. This feature is implemented by combining time-based progression with a distance-dependent term that accelerates the transition under a rapidly decreasing obstacle range.

In **LIFTING** mode, the supervisor maintains the conservative separation associated with the desired clearance. If required, a bounded feedback correction can be applied based on the measured inter-robot distance - and the corresponding clearance proxy from the geometric model - to reduce steady state offsets while respecting hard bounds. Such a correction uses proportional gain and saturation limits to ensure stability.

6.3.6 Re-trigger protection and stability mechanisms

A common issue in reactive, state-based policies is the undesirable re-triggering due to sensor noise or residual obstacle detections after the formation has safely overcome a hazard. To stabilise the recovery phase, a cooldown interval is activated immediately after **CLEARING** begins: during this stage, the re-activation of **LIFTING** is temporarily inhibited, which prevents rapid oscillations in case an obstacle briefly reappears due to localization uncertainty or transient mapping artefacts.

In addition, monitoring obstacle-distance trends helps to distinguish approaching hazards from obstacles that were left behind. Re-triggering is only allowed when the estimated obstacle distance decreases by more than a configurable margin (typically 0.05 m). This directional condition suppresses false alarms caused by stationary obstacles behind the robots that remain visible in extended sensor ranges, thus ensuring that interventions are restricted to genuine threats lying ahead.

Overall, obstacle management and coordination are achieved by combining three factors: the local obstacle monitoring with directional filtering and teammate exclusion, a hysteresis-driven supervisory state machine with persistence checks, and the midpoint-preserving reshaping of formation goals to satisfy coupling constraints through smooth transitions. Such an approach leverages standard single-robot navigation for local obstacle avoidance and provides the formation-level adaptability required for cooperative transport under flexible coupling, especially in narrow passages and obstacle-dense indoor environments.

6.4 Advantages of the proposed solution

The architecture presented in this chapter mitigates the structural limitations of pure leader-follower tracking in cluttered indoor environments by combining hierarchical coordination, obstacle-aware navigation, and state-based supervision. This section summarises the main advantages of the proposed solution and relates them to the design choices described above.

6.4.1 Separation of coordination and execution

Introducing a virtual frame as a shared reference decouples team-level coordination from individual motion execution. In a baseline leader-follower scheme, each robot directly tracks a relative pose through kinematic coupling; such an approach, however, does not account explicitly for obstacle geometry and may cause an unsafe or inefficient behaviour near walls and narrow passages. By contrast, the virtual

frame formulation specifies formation objectives at the group level and delegates the local motion execution to the navigation stack.

This separation has three practical benefits. First, formation targets remain coherent and geometrically consistent because they are generated from a single shared reference rather than from the robot states, which may diverge under local interactions with obstacles. Second, each robot can exploit the full potential of the navigation pipeline, including costmap-based avoidance, recovery behaviours, and adaptive replanning, without compromising the formation intent at the team level. Third, the overall system remains modular: formation logic and navigation components can be developed, tested, and tuned individually, provided that the interface contract –a namespaced goal topic per robot – is preserved. In practice, this modularity enables planner and controller parameters to be adjusted without modifying the formation code, while also allowing the coordination policy or formation geometry to evolve without re-engineering the low-level motion control.

6.4.2 Dynamic goal tracking without action preemption

A key challenge when using Nav2 for formation tracking is the need to update goals continuously as the virtual centre moves, while avoiding stop-and-go behaviours and oscillations. The solution proposed in this project addresses this requirement through a custom Behaviour Tree that supports online goal updating and keeps only one navigation action active, rather than repeatedly preempting and restarting execution.

Compared to a naive strategy that sends `NavigateToPose` goals at a high frequency, this approach avoids the counterproductive effects of repeated preemption, such as interruptions of the ongoing execution, repeated reinitialisation of the planning pipeline, and unnecessary recovery behaviours. Furthermore, the use of `PipelineSequence` allows the overlapping of planning and control: as the robot continues to follow the current path, replanning occurs at a controlled rate (0.5 Hz), thereby reducing motion discontinuities associated with a strict plan-then-move pattern.

In addition, the goal-update regulation policy ensures that published updates are actually informative rather than being dominated by noise. By suppressing micro-variations that could be related to localization and TF uncertainty, the system reduces planning deviations and oscillations near obstacles, thus improving smoothness and stability without sacrificing responsiveness to meaningful target motions. This feature is particularly relevant in constrained environments, where excessive

replanning can lead to path switching or chattering near decision boundaries.

6.4.3 Coordinated obstacle management under coupling constraints

Under flexible coupling, local obstacle avoidance alone is insufficient: uncoordinated detours may increase the inter-robot separation, alter the formation, or even result in mechanically feasible but operationally unsafe configurations. The supervisory state machine introduced in this chapter addresses such a limitation by managing obstacle interactions at the formation level and by enforcing coupling constraints in a coordinated manner.

The robustness of this formation-level management relies on several concurring mechanisms. Hysteresis-driven transitions with confirmation mechanisms reduce oscillatory switching when obstacle distance estimates fluctuate near thresholds and improve run-to-run reproducibility. Directional filtering and teammate exclusion suppress false positives by focusing detection on threats ahead and rejecting inflated teammate footprints. Moreover, count-based and time-based persistence further filter transient artefacts, increasing reliability without introducing excessive delay.

Coordination is achieved by reshaping formation targets rather than overriding low-level controllers. As the supervisor preserves the midpoint of the two goals during separation changes, it is able to keep consistency with the group-level intent encoded by the virtual frame trajectory. At the same time, Nav2 continues to handle local collision avoidance and path tracking. Furthermore, smooth transitions between separation modes, implemented through temporal interpolation (optionally modulated by obstacle urgency), limit aggressiveness and reduce oscillations during replanning. Finally, re-trigger protection mechanisms – cooldown intervals and trend-based reactivation conditions – prevent the undesired re-entry into conservative modes once an obstacle has been cleared.

6.4.4 Geometric model for clearance control

The explicit geometric relationship between inter-robot distance and payload clearance provides a controlled mechanism to regulate vertical constraints without a direct vertical actuation. Given the tether length and attachment height, the required horizontal separation for the desired clearance level can be computed deterministically. This model-based mapping generates predictable behaviour and allows to control of clearance only through planar motion primitives. If required, a bounded feedback correction based on the measured inter-robot distance can

compensate for modelling errors and dynamic effects, while saturation and gain tuning prevent aggressive corrections near constraint boundaries.

6.4.5 Robustness and operational continuity

The overall architecture includes multiple robustness mechanisms to support reliable operation in practice. Command saturation, low-pass filtering, and timeout handling in the virtual-centre node prevent abrupt reference motions and ensure predictable behaviour under teleoperation interruptions. TF consistency checks of the follower node prevent goal publication before localization is available, enhancing start-up reliability.

Continuous operation is further supported by the automatic action restart upon termination, which preserves navigation even when Nav2 completes an action due to tolerance-exceeding conditions or terminal failures. The clean and well-defined interface between formation and navigation layers simplifies integration and reduces coupling. Finally, stamped messages provide a consistent time base for multi-robot operation and guarantee a reliable implementation of the time-dependent logic.

Overall, the combination of virtual-frame coordination, regulated dynamic goal tracking, and hysteresis-based obstacle management allows formation objectives to be maintained while leveraging standard navigation components for single-robot obstacle avoidance. The resulting system is modular, maintainable, and robust enough to tackle the challenges of cooperative transport under flexible coupling in cluttered indoor environments.

Chapter 7

Conclusions and future works

This thesis addressed coordinated multi-robot formation control under flexible coupling constraints in cluttered indoor environments. The work was motivated by the limitations of pure leader-follower tracking, which does not explicitly account for obstacle geometry and can cause unsafe or inefficient behaviour in obstacle-dense layouts and narrow passages.

The main contribution of the project is a hierarchical control architecture that separates team-level coordination from local motion execution. The proposed solution adopts a virtual-frame formulation, in which each robot is assigned a target pose defined as a fixed offset from a shared virtual centre. This preserves formation coherence and enables obstacle-aware execution through Nav2. Dynamic goal tracking is performed via a custom Behaviour Tree that maintains a single navigation action active, supports online goal updates, and enforces periodic re-planning. In addition, goal regulation through low-pass filtering, rate limiting, and threshold-based gating both reduces planning churn and improves motion smoothness. All interactions with obstacles are handled by a supervisory finite-state machine with hysteresis, which monitors obstacle proximity and coordinates formation adaptation by reshaping target poses. Moreover, directional filtering and persistence checks improve detection robustness and mitigate oscillatory switching. Finally, a geometric model correlating inter-robot separation to payload clearance allows to regulate vertical constraints through planar motion primitives.

Experimental validation with two TurtleBot3 platforms showed that the system not only maintains formation coherence during teleoperated manoeuvres, but it also guarantees smooth, obstacle-aware motion. The obstacle management supervisor reliably detected critical situations and triggered coordinated lifting behaviours, with hysteresis preventing false activations and chattering. Furthermore, the modular architecture also proved to be robust with respect to localization uncertainty, sensor noise, and transient mapping artefacts.

The current implementation also suggests several directions for future work. First, the system is restricted to two robots: extending it to larger teams would require a generalised target generation and, potentially, distributed or optimisation-based coordination policies. Second, the clearance model does not capture payload dynamics, and the obstacle-detection pipeline lacks semantic information (e.g.,

obstacle type and height). Third, the supervisor operates reactively and does not anticipate upcoming constraints. Future developments could, therefore, include scalable coordination for larger teams, predictive obstacle management, semantic perception for clearance reasoning, learning-based tuning of key parameters, and validation in more realistic missions involving planning and long-term autonomy.

Overall, the results demonstrate that virtual frame coordination, combined with dynamic goal tracking and state-based obstacle management, provides an effective approach to formation control under flexible coupling in cluttered environments. The proposed architecture maintains a clear separation between coordination and execution layers, shows robust behaviour during experiments and provides a foundation for scaling the system to larger teams and more autonomous cooperative capabilities.

Bibliography

[1] H. Farivarnejad and S. Berman, "Multirobot control strategies for collective transport," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 5, no. 1, pp. 205–219, 2022.

[2] E. Tuci, M. H. M. Alkilabi, and O. Akanyeti, "Cooperative object transport in multi-robot systems: A review of the state-of-the-art," *Frontiers in Robotics and AI*, vol. 5, p. 59, 2018.

[3] ROBOTIS, "Turtlebot3 burger datasheet," https://www.mybotshop.de/Datasheet/Datasheet_ROBOTIS_Turtlebot_Burger.pdf, accessed: 2026-01-30.

[4] ROBOTIS. (2026) Turtlebot3 specifications. [Online]. Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/features/#specifications

[5] ROBOTIS, "Opencr 1.0 e-manual," https://emanual.robotis.com/docs/en/platform/turtlebot3/opencr_setup/#opencr-setup, accessed: 2026-01-30.

[6] ROBOTIS, "Turtlebot3 e-manual – features," 2026, accessed: 2026-02-12. [Online]. Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/features/#components

[7] ROBOTIS, "Turtlebot3 sbc setup," https://emanual.robotis.com/docs/en/platform/turtlebot3/sbc_setup/, accessed: 2026-01-30.

[8] ROBOTIS, "Dynamixel xl430-w250-t e-manual," https://emanual.robotis.com/docs/en/dxl/x/xl430-w250/, accessed: 2026-01-30.

[9] ROBOTIS, "Turtlebot3 features," https://emanual.robotis.com/docs/en/platform/turtlebot3/features/, accessed: 2026-01-30.

[10] ROBOTIS, "Turtlebot3 appendix: Lds-01," https://emanual.robotis.com/docs/en/platform/turtlebot3/appendix_lds_01/, accessed: 2026-01-30.

[11] ROBOTIS, "Turtlebot3 appendix: Lds-02," https://emanual.robotis.com/docs/en/platform/turtlebot3/appendix_lds_02/, accessed: 2026-01-30.

[12] Open Robotics, "Ros 2 concepts: About nodes," https://docs.ros.org/en/jazzy/Concepts/Basic/About-Nodes.html, accessed: 2026-01-30.

[13] Open Robotics, "Ros 2 concepts: About discovery," https://docs.ros.org/en/jazzy/Concepts/Basic/About-Discovery.html, accessed: 2026-01-30.

[14] Open Robotics, "Ros 2 concepts: About executors," https://docs.ros.org/en/jazzy/Concepts/Intermediate/About-Executors.html, accessed: 2026-01-30.

[15] Open Robotics, "Adding physical and collision properties to a urdf model (ros 2 jazzy tutorial)," https://docs.ros.org/en/jazzy/Tutorials/Intermediate/URDF/Adding-Physical-and-Collision-Properties-to-a-URDF-Model.html, accessed: 2026-01-30.

[16] Open Robotics, "rviz2 (ros 2 package documentation)," https://docs.ros.org/en/jazzy/p/rviz2/, accessed: 2026-01-30.

[17] Open Robotics, "nav_msgs/occupancygrid message (ros api documentation)," https://docs.ros.org/en/noetic/api/nav_msgs/html/msg/OccupancyGrid.html, accessed: 2026-01-30.

[18] ROS Wiki, "Rviz displaytypes: Map," https://wiki.ros.org/rviz/DisplayTypes/Map, accessed: 2026-01-30.

[19] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based slam," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43, 2010.

[20] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2d lidar slam," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1271–1278.

[21] ROBOTIS, "Turtlebot3 e-manual: Slam," https://emanual.robotis.com/docs/en/platform/turtlebot3/slam/, accessed: 2026-01-30.

[22] ROS Enhancement Proposals, "Rep-105: Coordinate frames for mobile platforms," https://www.ros.org/reps/rep-0105.html, 2010, accessed: 2026-01-30.

[23] Navigation2 Project, "Nav2 setup guide: Transform setup (tf)," https://docs.nav2.org/setup_guides/transformation/setup_transforms.html, accessed: 2026-01-30.

[24] S. Macenski and I. Jambrecic, "Evaluation of the google cartographer performance using an mpu-9250," in *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XLII-2/W3, 2017, pp. 543–547.

[25] Open Robotics, "Ros 2 how-to: Node arguments (including remapping rules)," https://docs.ros.org/en/jazzy/How-To-Guides/Node-arguments.html, accessed: 2026-01-30.

[26] Navigation2 Project, "Nav2 concepts," https://docs.nav2.org/concepts/index.html, accessed: 2026-01-30.

[27] Navigation2 Contributors, "nav2_regulated_pure_pursuit_controller: Readme," 2026, accessed: 2026-02-12. [Online]. Available: https://github.com/ros-navigation/navigation2/blob/main/nav2_regulated_pure_pursuit_controller/README.md

[28] Open Robotics, "nav2_controller (ros 2 package documentation)," https://docs.ros.org/en/jazzy/p/nav2_controller/, accessed: 2026-01-30.

[29] Navigation2 Project, "Nav2: Configuring bt navigator," https://docs.nav2.org/configuration/packages/configuring-bt-navigator.html, accessed: 2026-01-30.

[30] Navigation2 Project, "Nav2: Configuring amcl," https://docs.nav2.org/configuration/packages/configuring-amcl.html, accessed: 2026-01-30.

[31] Navigation2 Project, "Nav2: Configuring costmaps," https://docs.nav2.org/configuration/packages/configuring-costmaps.html, accessed: 2026-01-30.

[32] Navigation2 Project, "Nav2 costmap plugins: Obstacle layer," https://docs.nav2.org/configuration/packages/costmap-plugins/obstacle.html, accessed: 2026-01-30.

[33] Navigation2 Project, "Nav2 setup guide: Setting up sensors for mapping and localization," https://docs.nav2.org/setup_guides/sensors/mapping_localization.html, accessed: 2026-01-30.

[34] Open Robotics, "Using ros 2 launch for large projects," https://docs.ros.org/en/jazzy/Tutorials/Intermediate/Launch/Using-ROS2-Launch-For-Large-Projects.html, accessed: 2026-01-30.

[35] Open Robotics, "nav2_bringup (ros 2 package documentation)," https://docs.ros.org/en/jazzy/p/nav2_bringup/, accessed: 2026-01-30.