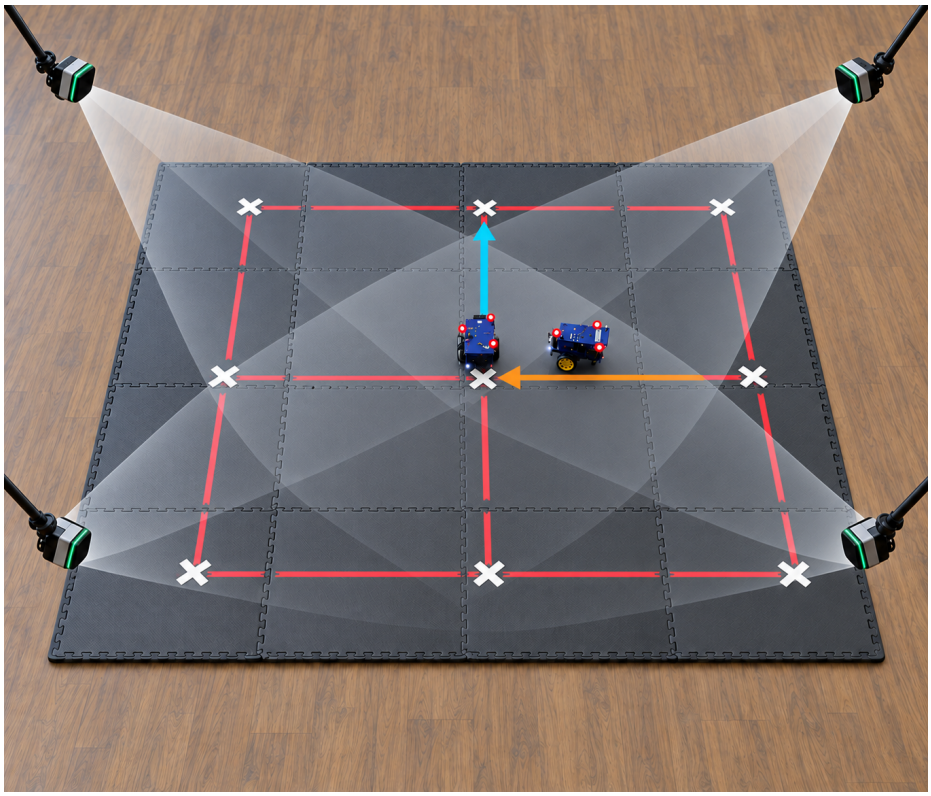


Execution Monitoring and Local Coordination for Multi-Agent Fleet Management

From Simulation to Hardware: Implementation and Evaluation on a Multi-Robot Platform

Master's thesis in MPSYS

Kim Hotvedt
Toshith Manglani

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026
www.chalmers.se

Master's thesis 2026

Execution Monitoring and Local Coordination for Multi-Agent Fleet Management

From Simulation to Hardware: Implementation and Evaluation on a Multi-Robot Platform

Kim Hotvedt
Toshith Manglani

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Systems, Control and Mechatronics
Chalmers University of Technology
Gothenburg, Sweden 2026

Execution Monitoring and Local Coordination for Multi-Agent Fleet Management
From Simulation to Hardware: Implementation and Evaluation on a Multi-Robot Platform

Kim Hotvedt
Toshith Manglani

Supervisor: Sabino Roselli, Department of Electrical Engineering
Examiner: Martin Fabian, Department of Electrical Engineering

Master's Thesis 2026
Department of Electrical Engineering
Division of Systems, Control and Mechatronics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Illustration of the experimental setup, showing motion-capture cameras tracking two Duckiebots along their planned trajectories.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2026

Execution Monitoring and Local Coordination for Multi-Agent Fleet Management From Simulation to Hardware: Implementation and Evaluation on a Multi-Robot Platform

Kim Hotvedt
Toshith Manglani
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Robot fleet-management systems combine high-level scheduling with local motion control to coordinate multiple robots operating in shared environments. However, during execution robots can deviate from a pre-computed schedule, resulting in delays or conflicts. This thesis extends an existing scheduling–control framework by introducing an execution-monitoring local coordinator and implementing the framework on a physical multi-robot platform. The coordinator monitors robot states and predicted trajectories and handles identified conflicts through waiting, priority changes, and re-planning. The proposed method is evaluated in simulation under execution-time disturbances. The distributed Model Predictive Control (MPC) framework and coordinator were also implemented on physical robots (Duckiebots).

Hardware experiments showed mean tracking error values between 0.039 m and 0.076 m, while schedule-adherence experiments revealed accumulated delays caused by differences between idealized scheduling assumptions and physical robot behavior. In simulation, the coordinator resolved interactions that otherwise caused large delays or incomplete execution.

The results demonstrate that execution monitoring and online coordination can complement high-level scheduling and distributed MPC, while highlighting the trade-off between conservative offline scheduling and dynamic conflict resolution during execution.

Keywords: robot fleet management, multi-robot systems, model predictive control, execution monitoring, robot coordination, scheduling, sim-to-real.

Acknowledgments

We would like to thank our supervisor, Dr. Sabino Francesco Roselli from the Division of Systems and Control at Chalmers University of Technology, for his guidance throughout the project. His support with project ideas, implementation, feedback, and his detailed knowledge of the scheduling framework were very valuable during the development of this thesis.

We would also like to thank Dr. Ze Zhang for his input on the MPC formulation and MPC software implementation, which helped us better understand both the theoretical and practical parts of the controller.

AI-based tools were used during the preparation of this thesis as support for improving language, clarity, and text structure. They were also used as a discussion aid during parts of the writing process. All technical content, analysis, results, and conclusions were reviewed and remain the responsibility of the authors. Codex was used as a supporting tool during the software development.

[From Toshith]

To my parents, thank you. Thank you for your advice, constant guidance, and loving me for being myself. Thank you for trusting my decisions, and for being the inspiration for me to keep going through all the difficult times. I love you both dearly.

To Raunak, I do not know where to begin. Thank you for being the constant, annoying, and impossible-to-live-up-to presence throughout my childhood. Despite all our differences, you and I were more alike than either of us probably admitted. I hope you would be proud to see me finish my education, and in some ways, I hope you completed yours through me. I hope you are finally enjoying your much deserved rest. This is for you.

To my friends Johannes, Hailan, Juan, Kanishk, Richard, Adrien, and Ilenia, thank you for being there with me through the good and the bad, and for being a constant presence through my time here in Göteborg. All of you helped me through the toughest phase of my life. You made this place feel like a home.

To Nipun, Nihal, Ria, Rahul, and Shorouq, thank you for being my 3 hour calls. Though you were not here with me, the constant chats, the calls, and your online companionship meant a lot to me.

I would also like the thank Tim Bergling, Armando Christian Perez, Kyrre Gørvell-Dahll, Kanye Omari West, and Marshall Bruce Mathers. Though they do not know me, they produced the most important pieces of perfection to help me be myself.

Lastly, to Kim, thank you for being my thesis partner. It was really nice working with you. The lunch time chats, and the mutual understanding of work distribution made this one of the best group projects I have done.

[From Kim]

I would like to give a special thanks to my wife, who was the one who pushed me to study at Chalmers and pursue an engineering degree in the first place. We have been together for many years, and this journey has not always been easy. My studies have at times taken a lot of time and energy from our relationship, but she has continued to support me, motivate me, and believe in me when I have struggled

to do so myself. I am very grateful that we made it through this journey together, and her support has played a large part in allowing me to reach this point. Looking back now, I feel very proud of what I have managed to accomplish and reaching this point would not feel the same without her.

Tosh, it has been a pleasure getting to know you and doing this thesis together. It has been a long journey, both academically and personally, and I am very glad that we got to share it. I am grateful for all the work we have done together, but also for the friendship that has grown during the process.

Kim Hotvedt and Toshith Manglani, Gothenburg, 2026

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

DMPC	Distributed Model Predictive Control
ETA	Estimated Time of Arrival
IMU	Inertial Measurement Unit
MAE	Mean Absolute Error
MOCAP	Motion Capture
MPC	Model Predictive Control
NMPC	Nonlinear Model Predictive Control
PANOC	Proximal Averaged Newton-type Method for Optimal Control
PI	Proportional-Integral
QTM	Qualisys Track Manager
RFM	Robot Fleet Management
RMSE	Root Mean Square Error
ROS	Robot Operating System
ToF	Time of Flight
UDP	User Datagram Protocol
NC	Not Completed

Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that have been used throughout this thesis.

Indices

i, j	Robot indices
k	Discrete time or sample index

Sets

N	Set of nodes in the road-network graph
E	Set of edges in the road-network graph
V	Set of robots in the fleet
A	Set of tasks
S	Overall fleet schedule
s_i	Individual schedule for robot i

Parameters

T	Scheduling horizon
$\Omega_{\max}$	Maximum robot velocity
μ	Temporal safety margin in the scheduling constraints
ϕ	Temporal safety margin for opposing edge traversal
d_{fleet}	Minimum inter-robot distance used by the MPC
ρ_{pen}	Penalty parameter used in the CasADi MPC formulation
Δt_s	Safety margin used during replanning
q_{rpd}	Reference-path deviation penalty

q_{pos}	Position-tracking penalty
q_{vel}	Velocity-reference penalty
q_{θ}	Heading-error penalty
q_{v}	Linear-velocity penalty
$q_{\Delta v}$	Linear-acceleration penalty
q_{ω}	Angular-velocity penalty
$q_{\Delta \omega}$	Angular-acceleration penalty
q_{stc}	Static-obstacle penalty
q_{dyn}	Dynamic-obstacle penalty
$q_{p,N}$	Terminal-position penalty
$q_{\theta,N}$	Terminal-heading penalty

Variables

x_i, y_i	Planar position of robot i
θ_i	Heading of robot i
r_i	Robot i
Ω_k	Desired velocity at time step k
L_{next}	Distance to the next scheduled node
t	Current time
t_n	Scheduled arrival time at the next node
$\mathbf{v}$	Vector of positive constraint violations
t_{current}	Current execution time during replanning
$t^v_{i,0}$	Starting time of the replanned route
$t^v_{i,k}$	Entry time of the replanned robot at node k
$t^l_{i,k}$	Exit time of the replanned robot at node k
t^f_k	Reserved time of a frozen robot at node k
$t^f_{e,\text{start}}$	Start time of frozen-robot occupancy of edge e
$t^f_{e,\text{end}}$	End time of frozen-robot occupancy of edge e
$t^r_{e,\text{start}}$	Start time of replanned-robot occupancy of edge e
$t^r_{e,\text{end}}$	End time of replanned-robot occupancy of edge e
e_k	Signed lateral tracking error at sample k
x_{ref}	Reference x-coordinate for a vertical path segment
y_{ref}	Reference y-coordinate for a horizontal path segment

v	Linear velocity command
ω	Angular velocity command

Contents

List of Figures

List of Tables

1
Introduction

Mobile robots are used in a wide variety of fields and can improve operational efficiency, precision, and increased productivity by automating tasks such as transportation, logistics, and material handling. Examples include transporting medical equipment in hospitals, supporting search-and-rescue missions in exploration environments, and moving goods and materials in warehouses. When multiple mobile robots operate in the same environment, the need to manage and coordinate them arises, which is commonly referred to as Robot Fleet Management (RFM)[1]. RFM concerns the coordinated operation of multiple autonomous or semi-autonomous robots in a shared environment and may include functions such as task allocation, path planning, resource management, and communication.

Managing a fleet of robots requires coordination at different levels. At a high level, tasks, routes, and the timing of robot movements need to be coordinated to avoid conflicts and deadlocks between robots [2]. However, a high-level schedule alone does not control the actual motion of each robot. During execution, each robot also requires a low-level controller to follow its reference trajectory while accounting for obstacles and controllable and uncontrollable agents [2]. This creates a natural separation between fleet-level scheduling and the continuous motion control of the individual robots.

This thesis builds on the fleet-management framework proposed by Roselli et al. [2], which integrates high-level scheduling with local trajectory generation and distributed MPC-based tracking to coordinate multiple mobile robots. While the framework demonstrates feasibility in simulation, real deployments introduce sensing, actuation, and communication uncertainties that can lead to deviations from the nominal schedule and trajectory execution. This motivates an additional execution layer that continuously monitors the schedule using real measurements and applies supervisory interventions when needed. In this thesis, such a local coordinator is designed, implemented, and validated for a small fleet of real-world moving robots at Chalmers.

Despite the growing amount of research on robot fleet management, experimental validation on physical robots remains relatively limited. Morais et al. [1] reviewed 259 studies in detail, of which only 24 presented experimental validation involving real robots.

1.1 Background

Although simulation enables quick development, transferring robotic systems from simulation to physical platforms often exposes a 'reality gap' due to sensing, actuation and communication effects that are difficult to model perfectly [11]. The execution of a pre-computed schedule is affected by uncertainties that are difficult to capture at the planning stage. Sensor noise, communication delays, modeling errors, and unmodeled disturbances can cause deviations between the planned trajectories, the control commands produced by the MPC layer, and the robots' actual motion. If such deviations are not monitored and handled explicitly during execution, the resulting schedule violations may propagate through the fleet, leading to increased delays and, in the worst case, conflicts such as deadlocks at shared resources or safety-critical situations despite the existence of a collision-free plan. These considerations motivate the introduction of an additional execution layer: a **local coordinator** that monitors schedule execution using real measurements and intervenes when necessary.

1.2 Motivation

One way to address execution uncertainties would be to increase model fidelity, redesign the MPC controller, or frequently recompute global schedules. However, these approaches do not fully resolve fleet-level coordination during execution. The MPC controller operates locally and primarily addresses continuous-time deviations, whereas conflicts and deadlocks typically arise from the interaction between multiple robots at shared resources. Conversely, global rescheduling can be computationally and organizationally costly if triggered too often, and may introduce unnecessary changes when deviations are small.

These considerations motivate a layered approach in which most deviations are handled locally through lightweight supervisory interventions (e.g. yielding, slowing down, or temporary re-ordering at critical resources), while larger deviations trigger a request for global rescheduling. The local coordinator proposed in this thesis is designed around this principle.

1.3 Purpose and Aim

The purpose of this thesis is to design, implement, and experimentally evaluate an execution-monitoring local coordinator, and to demonstrate the complete fleet-management architecture on a physical multi-robot platform (Duckiebots at Chalmers). The aim is to improve robustness and schedule adherence under execution-time uncertainties by adding a supervisory layer between the scheduler/local planner and the low-level controller.

1.3.1 Research Questions

(I) How can a local coordinator be designed to detect and handle conflicts during schedule execution?
(II) How can the coordinator be integrated into the existing framework, and how does it perform in simulation and on physical robots?
(III) What practical challenges arise in the sim-to-real transfer of the existing framework, particularly for trajectory tracking and schedule adherence?

1.3.2 Objectives

To achieve this aim, the thesis will:

- integrate the coordinator with the existing framework;
- monitor the execution using robot state and MPC prediction data;
- detect conflicts at shared nodes during execution;
- implement local actions such as waiting, priority changes, and replanning;
- implement the existing framework and the coordinator on physical Duckiebots; and
- evaluate the coordinator in simulation and on hardware, together with the tracking and schedule-adherence performance of the physical robots.

1.4 Contributions

The main contributions of this thesis are:

- implementation and deployment of the scheduling–planning–control framework of Roselli et al. on a physical multi-robot platform. To the best of our knowledge, this is the first hardware implementation of this particular distributed NMPC framework;
- design and implementation of a local coordinator that monitors the execution of the schedule and handles conflicts through waiting, priority changes, and replanning;
- implementation of the coordinator in simulation for all three identified conflict scenarios, and on physical Duckiebots for Scenarios 1 and 2;
- evaluation of the physical implementation through trajectory-tracking and schedule-adherence experiments, together with hardware demonstrations of the distributed MPC and coordinator.

The remainder of the thesis is structured as follows. Chapter 2 presents the theoretical background and related work. Chapter 3 describes the experimental platform, software implementation, coordinator design, and evaluation method. Chapter 4 presents the simulation and hardware results and discussions of the results. Finally, Chapter 5 summarizes the main conclusions, and presents directions for future work.

2

Theoretical Background and Related Work

2.1 Theoretical background

In RFM, problems related to task allocation, path planning, resource management, and communication arise when multiple robots operate together. Rema et al. [3] describe task scheduling as the problem of determining the timing and order in which tasks should be executed while optimizing different objectives. Several classes of methods can be used for task scheduling, including heuristic methods [4], metaheuristic methods [5], hybrid methods [6], machine-learning methods [7], and optimization-based methods [8].

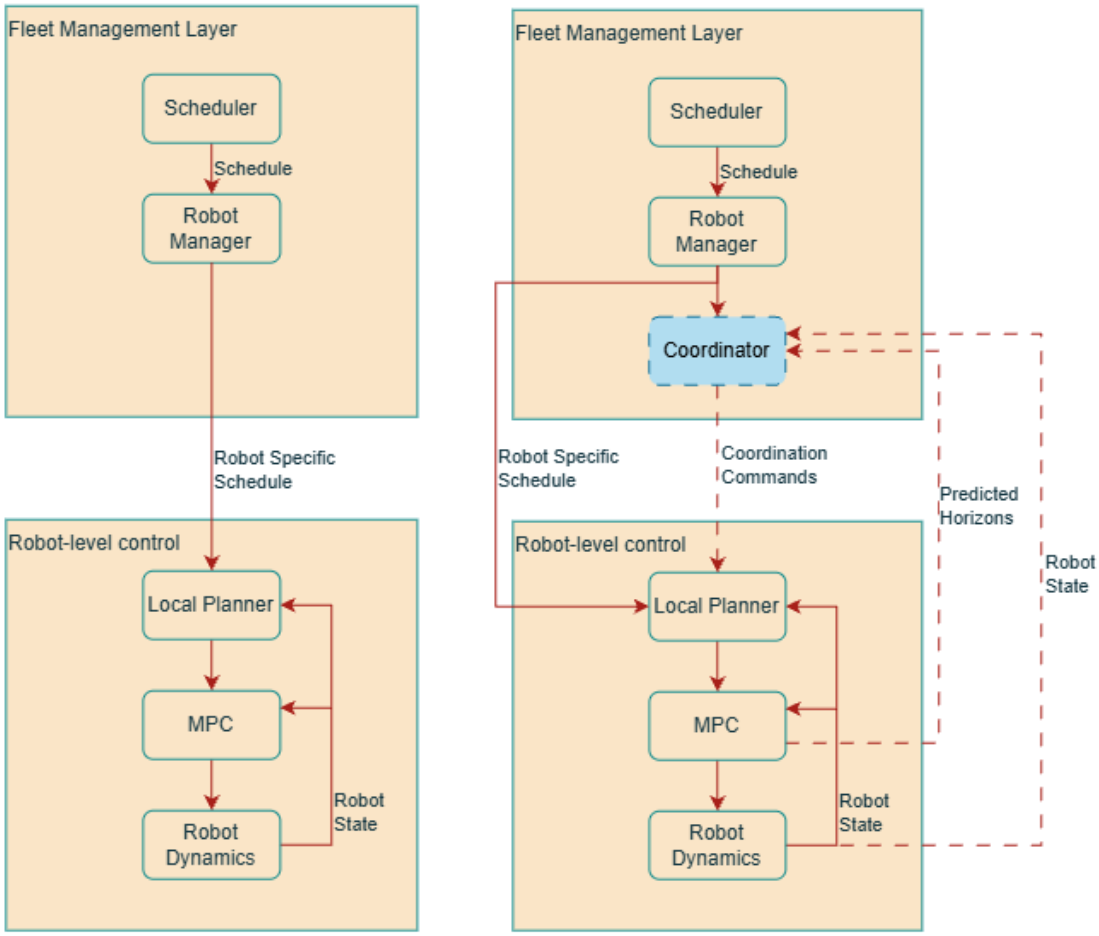
At the lower control level, mobile robots use feedback controllers to convert reference trajectories or desired motion into actuator commands. Several control approaches are used for mobile-robot navigation, including PID control, model predictive control (MPC), fuzzy-logic control, and reinforcement-learning-based control [9].

This thesis builds on the scheduling–planning–control architecture proposed by Roselli et al. [2]. In order to position the contribution of the local coordinator, this section first summarizes the baseline framework and its main components.

2.1.1 Multi-robot fleet-management architecture

The original architecture in this thesis is the fleet-management framework proposed by Roselli et al. [2]. The framework combines a centralized high-level scheduler with robot-specific local planning and distributed low-level control. At a high level, the scheduler assigns tasks to robots, determines routes through the road network, and computes the times at which robots should reach specific locations. These schedules are then used by a local planner on each robot to generate reference trajectories, while a distributed MPC computes collision-free control actions in real time. This layered design allows the computationally demanding task-allocation and scheduling problem to be handled centrally, while the continuous motion control problem is handled locally by each robot.

In the problem formulation of Roselli et al., the workspace is assumed to be two-dimensional and is abstracted as a weighted directed graph $G(N, E)$, where nodes represent task locations, depots or intersections and edges represent road segments or lanes. The road segments can have different capacities: narrow segments do not allow overtaking and may lead to impasses when robots travel in opposite directions,



(a) Current framework of [2] **(b)** Proposed Coordinator Framework

Figure 2.1: Changes in the project framework. Adding a coordinator would add execution-monitoring layer that makes it a **closed-loop system**.

while wide segments permit more flexible traffic. The robot fleet is denoted by the set V, and each task $a \in A$ is associated with a node, a time window, and potentially precedence constraints. The scheduler takes the graph G, the task set A, and the robot set V as input and produces a schedule S, i.e. an ordered set of triplets specifying when each robot should reach particular nodes along its route.

The local planner and the controller operate on top of the schedule rather than replacing it. For each robot, the local planner requests the individual schedule s_i and generates a local reference trajectory at every sampling instant, while the controller computes a collision-free control action based on this reference. The overall architecture therefore separates the discrete fleet-level coordination problem from the continuous trajectory-tracking problem.

The resulting architecture contains two distinct levels: a centralized scheduling layer and a distributed local planning and control layer. While feedback is present within the local control layer, there is no intermediate component that continuously monitors fleet execution and uses this information to influence fleet-level decisions. The

connection between the two levels is primarily established through the schedule generated prior to execution.

This thesis addresses this architectural gap by introducing an execution-monitoring coordinator between the two levels. The role of this layer and its relation to supervisory control are discussed in Section 2.1.4. The proposed supervisor will change the framework as seen in Figure 2.1.

2.1.2 High-level scheduling with ComSat

The high-level scheduling layer in the framework is based on the compositional algorithm ComSat, originally developed for conflict-free electric vehicle routing problems and adopted by Roselli et al. for mobile robot fleet scheduling [2]. The role of ComSat is to compute a feasible schedule over a finite horizon $[0, T]$, taking into account the road network, the available robots, and the set of tasks to be executed. In this setting, tasks are associated with time windows and precedence constraints, and robots may also be subject to operational restrictions such as battery limitations and depot returns.

ComSat decomposes the overall scheduling problem into subproblems. First, shortest paths between relevant task and depot locations are computed using Dijkstra's algorithm. These paths are used in the so-called E-Routing subproblem, which computes feasible routes for the robots while accounting for task time windows, precedence constraints, robot-task compatibility, and battery limitations. If the routing stage is feasible, the resulting routes are passed to a Capacity Verification step, where timing variables are introduced to determine when each robot reaches each node and when it starts traversing each edge. This verification stage enforces conflict-avoidance constraints that prevent two robots from occupying the same node or edge too closely in time. If the capacity constraints are not feasible for the current paths, the algorithm invokes a Paths Changing step to search for alternative paths, and iterates until a feasible schedule is found or until an infeasible solution is found.

An important feature of the version used by Roselli et al. is that the capacity constraints are augmented with explicit temporal safety margins. In particular, the parameters μ and ϕ are introduced to prevent robots from reaching the same node or traversing the same or opposite edges too close in time. In the reported experiments, these margins are set empirically to 20 seconds in order to account for non-zero sized agents, absorb delays and reduce the risk of conflicts during execution. The output of ComSat is therefore not only an assignment of robots to tasks and routes, but a time-parameterized schedule S specifying when each robot should reach the nodes along its route.

For the present thesis, ComSat is treated as the baseline global scheduler. The proposed coordinator does not replace ComSat, but instead monitors the execution of the time-parameterized schedule it produces and intervenes when real-world execution deviates from the assumptions under which the schedule was computed.

2.1.3 Model Predictive Control for multi-robot trajectory tracking

Apart from the scheduler, Roselli et al. introduce a robot-specific local planning and control layer that bridges the discrete schedule with the continuous motion of the robots [2]. Since the schedule computed by ComSat assumes constant robot speed and only specifies when robots should reach nodes along the graph, it is not directly suitable for real-time motion control. To address this, each robot has a local planner that dynamically generates a local reference trajectory and a desired speed based on its current schedule and its current location along the route. A global reference trajectory is first precomputed by sampling the scheduled path. During execution, a local reference is extracted and resampled according to the urgency of reaching the next scheduled node. The desired speed is chosen as

$$\Omega_k = \min\left(\Omega_{\max}, \frac{L_{\text{next}}}{\max(0, t_n - t)}\right),$$

where L_{next} is the distance to the next scheduled node and t_n is the scheduled arrival time at that node. In this way, the local planner adapts the continuous reference to the high-level schedule online.

The low-level controller is formulated as a distributed MPC problem, solved independently for each robot while explicitly accounting for static obstacles and the current and predicted future positions of the other robots. The objective function consists of three main parts: reference tracking, obstacle avoidance, and fleet collision avoidance. The reference tracking term penalizes deviations between the predicted robot state and the reference trajectory, as well as deviations between the control input and the desired speed reference and rapid changes in the control signal. The fleet-collision term penalizes predicted inter-robot distances below a prescribed safety distance d_{fleet}. The resulting optimization problem is solved over a finite horizon subject to the robot motion model and physical input constraints.

This control architecture is important for the present thesis for two reasons. First, it provides the continuous collision-avoidance and trajectory-tracking capabilities that the coordinator will build upon rather than replace. Second, it also highlights the limitation that motivates the coordinator: although the distributed MPC can avoid collisions locally, it does not by itself guarantee that the global schedule remains efficient or deadlock-free under execution-time disturbances. The coordinator proposed in this thesis is therefore intended to complement the MPC layer by monitoring schedule adherence and introducing higher-level supervisory decisions when local control alone is insufficient.

2.1.4 Supervisory control and discrete-event coordination

While the low-level MPC layer handles continuous trajectory tracking and local collision avoidance, the coordination problem addressed in this thesis can be viewed as hybrid in nature. The robots move continuously in real time, while several of the coordination decisions are discrete. Examples include deciding which robot should enter an intersection first, whether a robot should wait before entering a

narrow corridor, or whether the execution has deviated sufficiently from the nominal schedule that rescheduling should be requested. Systems combining continuous, time-driven dynamics with event-driven decision making can be described from a hybrid-systems perspective [10].

In supervisory control of discrete-event systems, the system is described in terms of states and events, and a supervisor restricts controllable events in order to satisfy a desired specification [10]. In the context of this thesis, this viewpoint is applied at the coordination level rather than to low-level motor commands. Relevant coordination events can, for example, correspond to *enter intersection*, *wait before corridor*, *release robot*, or *request reschedule*. Whether a robot is allowed to proceed or instructed to wait can be regarded as a controllable coordination decision, while deviations in the physical execution arise from disturbances that are outside the direct control of the coordinator. Intersections and narrow corridors can similarly be viewed as shared resources. A coordination rule can therefore restrict simultaneous access to such resources, for example by allowing only one conflicting robot to occupy a critical region at a time or by enforcing a required temporal separation between robots.

In the proposed architecture, the coordinator performs this role by monitoring the robot states, predicting their arrival at shared resources, and applying local priority or waiting decisions when potential conflicts are detected. The supervisory-control perspective is used here primarily as a way of structuring the coordination problem. In particular, it separates the continuous motion-control problem handled by the MPC from the discrete decisions associated with access to shared resources and deviations from the schedule. This provides a useful interpretation of the coordinator: the measured execution is compared with the expected execution, potential coordination conflicts are identified, and a finite set of predefined interventions can be applied.

The coordinator developed in this thesis is inspired by supervisory control, but it is not obtained through formal supervisor synthesis from an automaton model. Instead, the implementation is algorithmic: supervisory rules and local ordering policies are defined for shared resources in the road network and are triggered online based on the measured robot states and predicted arrival times. Supervisory control is thus used as a conceptual framework for the execution-monitoring and coordination layer rather than as a formal synthesis method.

The scheduling and control layers described in the previous sections operate at different levels of the fleet-management framework. The scheduler determines the routes and timing of the robots at a discrete level, while the MPC continuously controls the motion of each robot during execution. However, interactions between robots during execution can require decisions that are not purely continuous control actions. Examples include deciding which robot should be allowed to enter a shared region first, whether a robot should temporarily wait, or whether the current plan can no longer be executed as intended.

These types of decisions can be viewed from a discrete-event perspective. Systems that combine continuous, time-driven dynamics with event-driven decision making can be described as hybrid systems [10]. In the considered fleet-management prob-

lem, the physical motion of the robots represents the continuous part of the system, while coordination actions such as allowing a robot to proceed, instructing it to wait, or initiating replanning represent discrete events.

Supervisory control provides a useful conceptual framework for this type of coordination. In supervisory control of discrete-event systems, a supervisor observes the evolution of a system and restricts controllable events in order to satisfy a desired specification [10]. For a multi-robot system, shared nodes and links can be interpreted as resources whose access must be coordinated. Decisions concerning whether a robot is allowed to enter or continue through such a resource can therefore be treated as supervisory decisions. This interpretation separates the role of execution-level coordination from that of the local controller. The MPC remains responsible for continuous trajectory tracking and local control, while a supervisory layer can monitor interactions between robots and determine whether an additional coordination action is required. The supervisory layer therefore does not replace the scheduler or MPC, but complements them by addressing discrete decisions that arise during the execution of the scheduled plans.

The coordinator developed in this thesis follows this supervisory interpretation. However, it is not obtained through formal supervisor synthesis from an automaton model. Instead, predefined coordination rules are evaluated online using information obtained during execution. Supervisory control is therefore used as a conceptual framework for separating continuous robot control from discrete fleet-level coordination. The specific conflict conditions, coordination rules, and implementation of the coordinator are presented in Section 3.3.

2.2 Related Work

The following sections aim to position this thesis within the existing literature by reviewing related work in execution monitoring, multi-robot coordination, re-planning, fleet-management architectures, and hardware validation.

2.2.1 Execution monitoring in robotic systems

Execution monitoring is a well-established concept in robotics and autonomous systems. Pettersson [12] describes it as an agent that can identify deviations from observed state and expected state and to be able to classify the errors and recover from them.

The survey provides a broad robotics perspective on execution monitoring and serves as a useful conceptual starting point for the present thesis. In contrast, the present work studies execution monitoring specifically in a scheduled multi-robot fleet-management architecture.

2.2.2 Coordination at shared resources and intersection management

Cáp et al. presents in [13] a control law that deals with safe and deadlock-free execution of preplanned trajectories under delaying disturbances, given an initial feasible collision-free trajectory for multiple robots. The robots state is defined as the position of each robot along the pre-planned trajectory. The goal of the multi-robot controller is to take the current state of each robot and return what they refer to as the advancement decision for every robot such that the system is collision-and deadlock-free, so that no single robot having a disturbance stops the entire system.

The control law is referred to as the Robust Multi-Robot Trajectory Tracking Strategy (RMTRACK). Their interpretation of collisions is based on the notion of coordination space. Coordination space is a geometric representation of the joint progress of multiple robots, where collisions in the physical workspace correspond to shared collision region in a higher-dimensional abstract space. For the special case of two robots, this coordination space becomes two-dimensional, and the collision region can be visualized as an shared collision region in that space.

The purpose of RMTRACK apart from collision-and deadlock-free is also to ensure that under disturbances the trajectories in coordination space avoids these collision regions while still reaching the final goal. RMTRACK allows the robots to deviate from the exact planned trajectory in coordination space, but preserves the original passing order around each collision region.

A more intuitive example is that given that a collision region has been detected at an intersection due to a delay from robot R_1. From the initial trajectory R_1 must pass before R_2. Then R_2 can not pass before R_1 and must wait until R_1 has passed the collision obstacle.

A related approach is presented by Soltero et al. [14], who consider multiple robots following predefined intersecting trajectories. Potential conflict regions are represented as collision zones, and collisions are avoided by stopping robots before entering an occupied zone and allowing them to continue once the zone is released. The method also includes a deadlock-avoidance mechanism for situations where several robots are waiting for each other.

In comparison, the coordinator developed in this thesis detects potential conflicts at shared resources based on execution and schedule information and applies local waiting or ordering decisions.

2.2.3 Online plan repair and discrepancy handling

In Coskun and O'Kane's paper "Online Plan Repair in Multi-robot Coordination with Disturbances" [15] they build their work around the previously mentioned RMTRACK, but extends it by allowing a robot to swap the order in which a pair of robots pass through the shared collision region, instead of having to wait for the other delayed robot to pass as initially planned. This allows the system to do a online repair by estimating future disturbances and deciding whether flipping the obstacle/order is likely to improve travel time / overall performance.

In another paper, Tong et al. [16] propose a trajectory-repairing framework for

autonomous vehicles. If the system detects a potential collision risk with another vehicle, it first attempts a speed adjustment. If this adjustment is insufficient and the risk remains, the framework triggers trajectory repair.

At a high level, this is similar to the present thesis in that lightweight local corrections are attempted before stronger intervention is applied. However, their work focuses on continuous trajectory repair for a single vehicle, whereas this thesis considers multi-robot coordination repair, for example through waiting, reordering, local replanning, or rescheduling. In addition, the overall architecture differs: our work is based on a scheduler–trajectory generator–MPC fleet-management framework.

In this literature review, Heselden and Das [17], review the problems and challenges for multi-robot systems when it comes to coordination and path-planning that is relying on prioritized planning. This kind of method solves the path-planning for robots with higher priority order first.

Our scheduler, ComSat, does not use prioritised planning in this sense. However, the review highlights that initial plans can become infeasible or suboptimal, which motivates the need for replanning.

The review therefore discusses rescheduling methods that modify the priority ordering, either globally or locally, in order to recover feasibility or improve performance. This is relevant to the present thesis, since the proposed coordinator also operates on the principle that an initially planned ordering at a shared resource may need to be revised online through waiting, reordering, or replanning when execution deviates from the nominal plan.

2.2.4 Fleet-management and multi-robot coordination architectures

Forte et al. [18] proposes an architecture for a heterogeneous fleet of robots which includes task assignment, motion planning, coordination, control and localization. Tasks are assigned online and only idle robots can be given an assignment. Paths are generated to the robots, which takes them from the current position to the position of where the assignment should be completed. The authors defines critical section as the overlapping path regions were robots could potentially collide.

The paper propose a coordinator, MRTA-Extended Priority-Based Coordination, where MRTA stands for Multi-Robot Task Assignment. The purpose of this coordinator is to make robots yield to each other in critical sections by introducing precedence constraints.

If there is a risk of, for example two robots colliding, the framework estimates how much delay would be introduced if robot A yields to robot B and vice versa. In the interference cost model, the smaller of these two possible yielding delays is considered as an approximation of the local coordination cost. The coordinator does not account for the effect this may have on future critical sections, since doing so would lead to a combinatorial explosion.

Hažík et al. [19] propose a fleet-management architecture in which route planning and robot synchronization are treated separately. Access to shared parts of the road network is coordinated using a semaphore-based mechanism, where robots may be

stopped until the required nodes or critical regions become available.

Souto et al. [20] present a fleet-management system combining dynamic task assignment, execution monitoring, and traffic management. The system coordinates access to spatially restricted areas and can temporarily redirect robots to waiting locations when routes or shared resources are unavailable.

While these approaches address fleet coordination through online task assignment, route synchronization, or restricted-area management, the present thesis focuses on execution monitoring of a pre-computed schedule and on applying local corrective actions when the actual execution deviates from the nominal plan.

2.2.5 Hardware validation of multi-robot control frameworks

Hardware validation remains relatively uncommon in robot fleet-management research. In the review by Morais et al. [1], 259 papers were selected for detailed analysis, while only 24 of the reviewed studies presented experimental validation involving real robots. The authors therefore identify a significant gap between simulation-based research and experimental validation.

Distributed MPC has been demonstrated on physical multi-robot systems, although fully distributed hardware implementations are less common. For example, Gräfe et al. [21] implemented DMPC-Swarm on a physical swarm of up to 16 Crazyflie nano-quadcopters using distributed computation and wireless mesh communication. To the best of our knowledge, however, the specific distributed MPC framework proposed by Roselli et al. [2] has previously only been evaluated in simulation.

3
Methods

This chapter describes the experimental platform, the software architecture and implementation, and the evaluation methodology used in this thesis. It covers the hardware setup, the software modifications required to deploy the fleet-management framework on the Duckiebots, the design and implementation of the proposed coordinator, and the metrics and experimental scenarios used for evaluation.

3.1 Experimental platform

3.1.1 Duckiebot platform

The experiments were carried out using Duckiebots, see figure 3.1, a small autonomous mobile robot platform from Duckietown. The robots are equipped with several sensors, such as a camera, IMU, time-of-flight (ToF) sensors, and wheel encoders. They use a differential-drive configuration powered by two DC motors. The platform is also equipped with NVIDIA Jetson Nano board 4Gb. The robotics ecosystem uses Docker containers and supports both ROS and Python [22].

Figure 3.1: The robot used in the experiment, Duckiebots from Duckietown.

The Duckiebots provide onboard sensing through the IMU and wheel encoders. During the development of the hardware implementation, these sensors were inves-

tigated for state estimation and wheel-speed feedback, including experiments with encoder-based PI control. However, the final experimental setup did not rely on the onboard sensors for robot pose estimation or low-level feedback. Instead, the robot pose used by the fleet-management framework was obtained from the external motion-capture system described in the section below.

3.1.2 Motion-capture and computing setup

To obtain accurate pose information for the robots, the motion-capture laboratory (MOCAP lab) at Chalmers was used. The laboratory is equipped with eight Arqus motion-capture cameras (Figure 3.2 from Qualisys [23] and uses Qualisys Track Manager (QTM) for motion capture. The setup with the cameras is as seen in figure 3.3.

The MOCAP computer used in the experiments was a Dell Pro 16 Plus PB16250 running Windows 11 Enterprise, equipped with an Intel Core Ultra 7 255U processor with 12 cores and 32 GB of RAM.

To capture the real-time pose of all four robots, QTM collects and processes the 6-DOF motion-capture data. The real-time data are retrieved using the QTM real-time protocol [24]. The MOCAP computer then transmits the pose data via UDP to the host computer at a frequency of 120 Hz.

The host computer receives the UDP data and extracts the planar coordinates (x_i, y_i, θ_i) for each robot r_i, which are used as the robot state in the implemented framework.

Figure 3.2: The Arqus cameras used in the MOCAP lab.

Figure 3.3: Lab environment setup: the robots will work on the black mats, MOCAP cameras can be seen on the top left and top right in the picture.s

3.2 Software implementation

This section describes the software implementation used in the thesis. It first explains the network architecture used for the distributed MPC, followed by the implementation of the original PANOC-based MPC framework in CasADi. The coordinator implementation is then described, together with how it was integrated into the existing framework. Finally, the experimental setup used to evaluate the implemented system is presented.

3.2.1 Network architecture

The communication setup that was used in the experiments is illustrated in Figure 3.4.

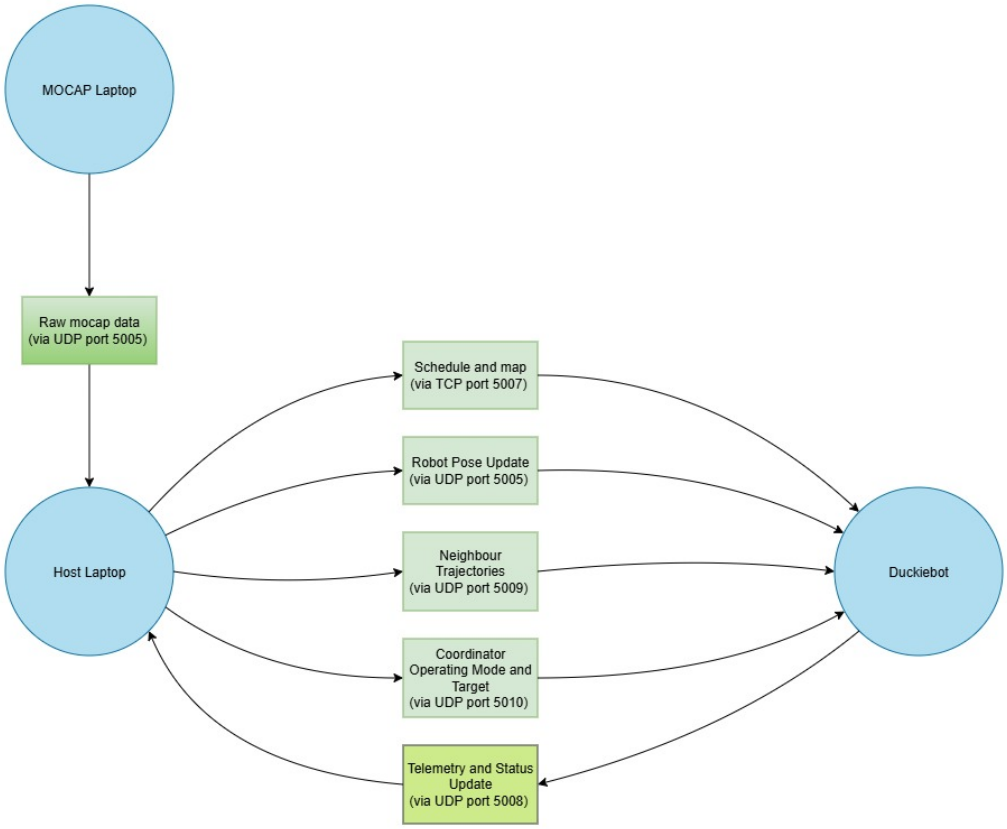


Figure 3.4: Communication setup used in the experiments, showing the data exchange between the motion-capture laptop, host computer, and Duckiebot.

The distributed control framework is organized around two computers and a distributed MPC that runs locally on each robot. The host laptop is responsible for sending the schedule, collecting runtime feedback, and relaying predictions based on current robot state, while a separate motion-capture (MOCAP) laptop provides

real-time pose measurements. Each Duckiebot executes its own MPC locally and therefore does not rely on the host laptop for online optimization.

At system start-up, the host laptop sends a robot-specific planning package to each robot r_i via TCP on port 5007. This package contains the map required for local planning, together with a schedule describing the assigned route of the robot. More specifically, the schedule includes the initial state of r_i, the path coordinates, and, estimated arrival times (ETAs). Hence, the schedule defines the path and temporal targets to be followed, rather than a fully computed control trajectory.

Real-time state estimation is provided through the motion-capture system. The MOCAP laptop sends pose measurements for all tracked robots to the host laptop via UDP on port 5005. The host laptop then forwards the corresponding pose estimate to each Duckiebot, also via UDP on port 5005. In this way, each robot receives only its own pose measurement, which is used as the current state estimate for onboard control.

Each Duckiebot solves its MPC problem locally at every control step. After each update, the robot send telemetry data to the host laptop via UDP on port 5008. This telemetry contains the robot's current pose, the applied control input, and the predicted future states generated by the MPC over the prediction horizon. It also include additional monitoring information, such as reference states, target information, solver timing, and controller status.

The host laptop uses the received telemetry to reconstruct the predicted motion of the robot fleet. For each robot r_i, it collects the most recent predicted trajectories of the other robots $r_j \neq r_i$ and combines them into a neighboring-state message. This message represents the expected motion of surrounding robots over the prediction horizon and is sent back to robot r_i via UDP on port 5009. Each Duckiebot takes into consideration the anticipated motion of neighboring robots into its next local MPC solve.

The coordinator, which is on the host laptop uses information from the constructed neighboring-state message for its calculations. It relays the operation mode for the robots to stop, cross, or continue execution normally using a coordinator packet via UDP on port 5010. Further details of coordinator communication are explained in section 3.3.6.

This communication structure enables a distributed MPC architecture in which optimization remains decentralized, while coordination is achieved through prediction of the other robots. The host laptop acts as a communication and bridges different layers, whereas the actual control computation is performed onboard each individual robot.

3.2.2 Implementation of the MPC on Duckiebots

The software for the MPC implementation was available from the original Git repository, where a complete implementation using the PANOC solver through OpEn was provided. The repository also contained an initial skeleton for a CasADi-based implementation. However, this version was not fully integrated with the trajectory-tracking framework and could therefore not be used as a direct replacement for the

PANOC implementation.

In this thesis, the CasADi implementation was completed and integrated into the framework. The implementation was made to follow the original PANOC formulation as closely as possible, including the parameter structure, tracking and control objectives, acceleration penalties, and the constraints and penalty terms related to static obstacles and interactions between robots.

One important difference between the two implementations is the optimization formulation. The PANOC implementation uses a single-shooting formulation, where the control inputs are the optimization variables and the corresponding state trajectory is obtained by forward simulation of the system dynamics. In contrast, the CasADi implementation uses direct multiple shooting, where both the states and control inputs are optimization variables and the system dynamics are imposed as equality constraints between consecutive states.

Another difference is how the obstacle-related penalty constraints are handled. In the PANOC/OpEn implementation, these are handled internally by OpEn through its penalty-constraint formulation. Since this mechanism is specific to OpEn, a similar penalty approach was implemented explicitly in the CasADi version. The positive constraint violations were collected in a vector v and added to the objective function through the quadratic penalty term

$$\rho_{\text{pen}}\|v\|_2^2, \tag{3.1}$$

where ρ_{pen} is the penalty parameter. The value of ρ_{pen} was increased between repeated solver calls to try to mimic the outer penalty-update procedure used in the PANOC/OpEn implementation.

The complete CasADi implementation, including both the static obstacle-avoidance constraints and the dynamic fleet-collision constraints, was successfully tested in simulation and on the physical Duckiebots. This showed that the complete MPC formulation could also be used on the hardware.

For the final experimental evaluation, a simplified configuration was used. The static obstacle-avoidance constraints and dynamic fleet-collision constraints were omitted since they required additional tuning on the physical robots, while robot interactions were instead handled by the coordinator. The final experiments therefore focused on the execution monitoring and coordination provided by the coordinator, while the complete constrained MPC implementation had already been tested on the hardware.

3.2.2.1 Tuning in simulation

The MPC was first tuned in simulation. The main parameters adjusted during tuning were the weights associated with reference-path tracking, position and heading tracking, velocity tracking, control effort, acceleration, and obstacle avoidance. The parameters were adjusted through repeated simulation runs while observing the resulting trajectory tracking and control behaviour. The final simulation parameters are shown in Table 3.1.

3.2.2.2 Tuning for hardware

The simulation parameters were used as an initial configuration for the hardware implementation. However, additional empirical tuning was required for the physical Duckiebots. The weights were adjusted separately for the individual robots through repeated trajectory-tracking tests.

The parameters mainly affected the trade-off between following the reference path, tracking individual reference points, maintaining the desired heading and velocity, and producing smooth control actions. Table 3.1 compares the final parameter values used in simulation and for the physical robots.

Below is the description of what the parameters mean, and aims to clarify Table 3.1.

Table 3.1: MPC parameters and values used in simulation and on the physical Duckiebots.

Parameter	Description	Simulation	Duck2	Duck6
q_{rpd}	Reference-path deviation penalty	0.0	20.0	5.0
q_{pos}	Position-tracking penalty	8.0	0.1	2.0
q_{vel}	Velocity-reference penalty	1.0	0.0	0.50
q_{θ}	Heading-error penalty	1.0	2.0	1.0
q_{v}	Linear-velocity penalty	0.0	0.25	0.25
$q_{\Delta v}$	Linear-acceleration penalty	0.0	0.125	0.125
q_{ω}	Angular-velocity penalty	0.0	0.0	0.0
$q_{\Delta \omega}$	Angular-acceleration penalty	0.0	0.001	0.05
q_{stc}	Static-obstacle penalty	10.0	0.0	0.0
q_{dyn}	Dynamic-obstacle penalty	10.0	0.0	0.0
$q_{p,N}$	Terminal-position penalty	0.0	0.1	0.1
$q_{\theta,N}$	Terminal-heading penalty	0.0	0.15	0.15

3.3 The Coordinator

A central objective of this thesis was to develop a coordination strategy capable of handling potential deadlocks during schedule execution. For this purpose, an additional coordination layer was introduced between the scheduler and the controller.

The coordinator monitors the current state and MPC prediction horizon of each robot. In addition, it tracks three nodes associated with the current execution of each robot: the previous node (which the robot has most recently left), the current target node (towards which the robot is traveling), and the next target node in its scheduled route.

During the initial evaluation of the existing framework, a significant source of delays was observed when multiple robots approached the same node simultaneously. Although the MPC can prevent direct collisions, these interactions can result in robots blocking each other and significantly delaying execution. The first implementation of the coordinator therefore focuses on conflicts arising when multiple robots approach a common node.

By analysing these interactions, three recurring conflict scenarios were identified:

- Scene 1: Simple node crossing with different exit paths.
- Scene 2: Node crossing with exit link of one robot being the entry link of the other.
- Scene 3: Opposing traversal using the same consecutive links.

The coordination strategies were first developed and evaluated in simulation before being transferred to the physical robots. The following subsections describe to each scenario and the respective solutions in detail. Finally, the strategies are combined into the complete coordinator and integrated with the simulation and hardware implementations.

A simplified map was created for the development and initial testing of the coordinator, as shown in Figure 3.5. The environment consists of nine nodes arranged in a square grid and represents a reduced version of the larger map used in the framework. The layout was selected such that all three identified conflict scenarios could be reproduced in a controlled environment. This allowed the coordination strategies to be developed and tested independently before being applied to larger simulation setups.

3.3.1 Scenario 1 (Mutual Exclusion)

Scenario 1 represents a mutual-exclusion problem in which two robots are scheduled to traverse a common node, but enter and exit towards different nodes. A conflict may occur when deviations from the schedule cause the robots to approach the common node at approximately the same time. In the example shown in Figure 3.6a, robot $A1$ follows the path $N10 \rightarrow N11 \rightarrow N12$, while robot $A2$ follows $N01 \rightarrow N11 \rightarrow N21$. The common node $N11$ represents the shared resource.

In this example, both robots are scheduled to reach $N11$ at $t = 4.00$s. Without intervention from the coordinator, the existing collision-avoidance mechanism attempts to resolve the interaction locally by adjusting the motion of the robots. Instead, the

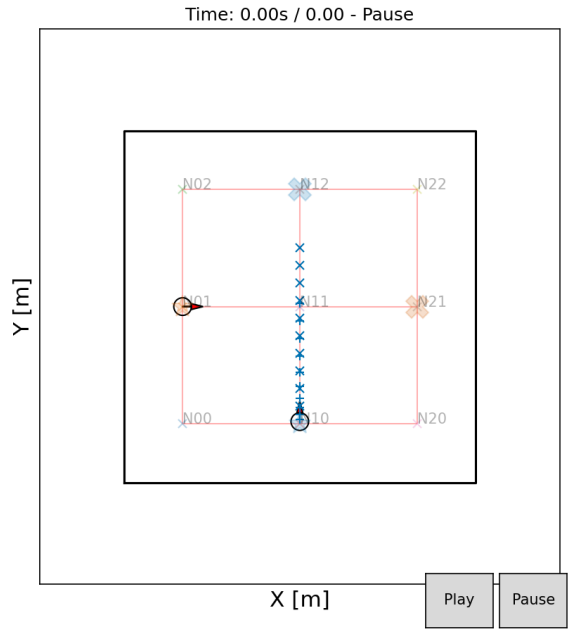
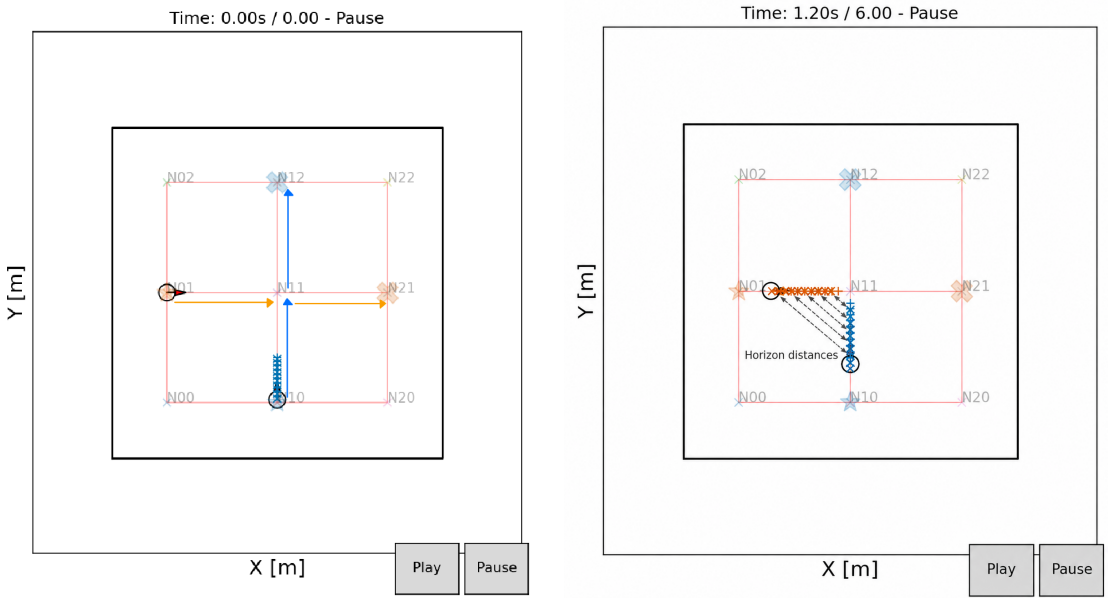


Figure 3.5: Development environment for the coordinator

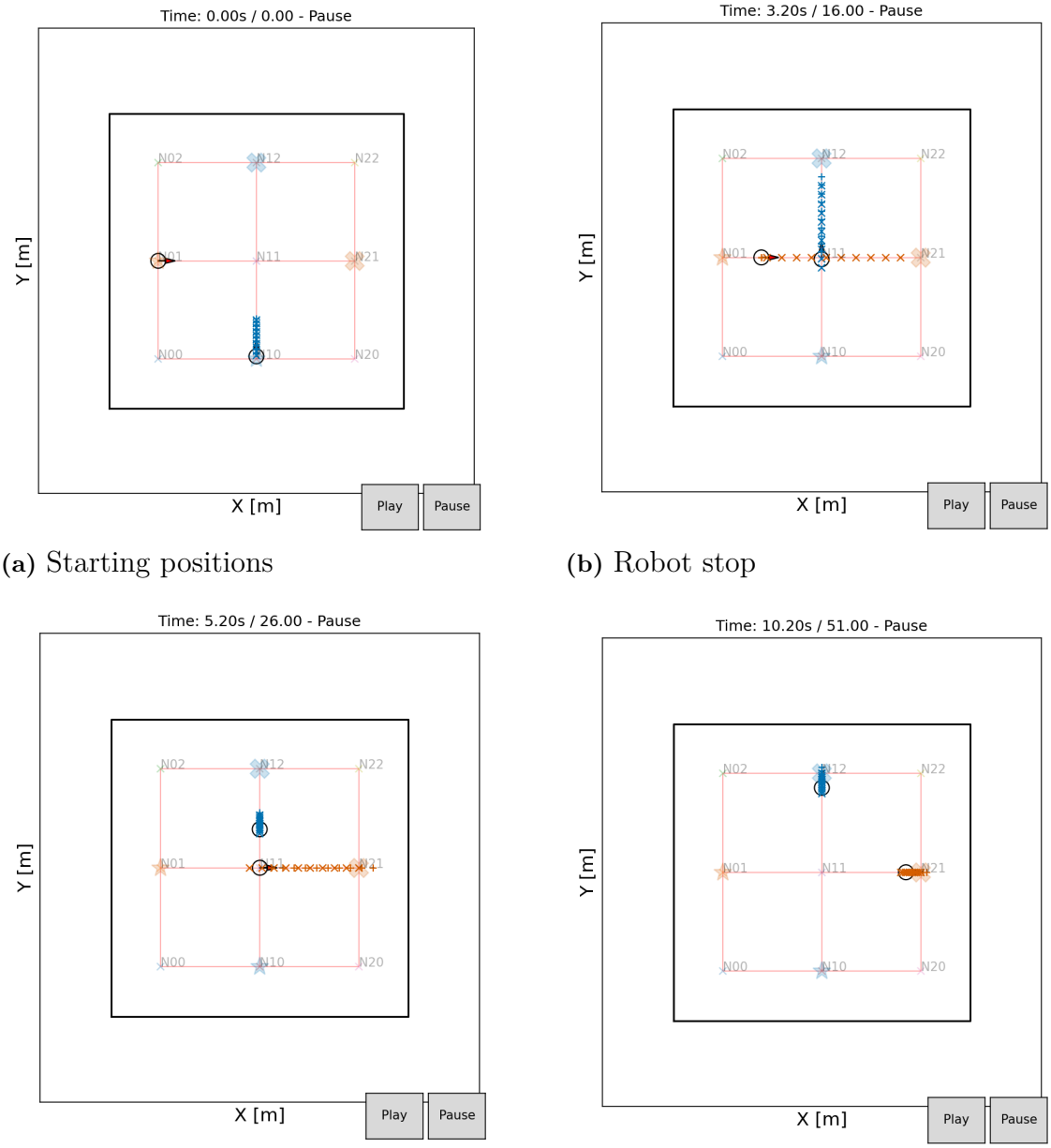


(a) Expected path for Scenario 1 **(b)** MPC Horizon Distance Calculation

Figure 3.6: Coordinator Scenario 1

coordinator applies a mutual-exclusion strategy in which one robot is temporarily stopped while the other is allowed to cross the shared node.

Approaching the same node does not necessarily indicate that coordinator intervention is required. For example, one robot may have only recently started travelling towards $N11$, while the other may already be close to completing its traversal to the node. Stopping a robot solely because the current target nodes are equal would



(a) Starting positions

(b) Robot stop

(c) Shifted Coordinate Stop

(d) Crossing Completed

Figure 3.7: Snapshots of scene 1 process.

therefore introduce unnecessary waiting.

To determine whether intervention is required, the coordinator compares the MPC prediction horizons of the conflicting robots. Each predicted position in the horizon of $A1$ is compared with each predicted position in the horizon of $A2$, as illustrated in Figure 3.6b. If any pair of predicted positions is separated by less than a predefined safety threshold, the interaction is considered a potential conflict and the coordinator intervenes.

Once a conflict has been detected, the coordinator determines which robot should be given priority. The decision is based on the robots' schedule status and their

distance to the conflicting node. If one robot is delayed while the other remains on schedule, priority is given to the robot that is on schedule. This prevents a delay affecting one robot from unnecessarily propagating to another robot. If both robots are delayed, or neither robot is delayed, priority is instead given to the robot closest to the conflicting node.

Allowing the selected robot to reach $N11$ is not sufficient to resolve the conflict. Depending on its scheduled arrival time at the following node, the robot may reach $N11$ and remain there while waiting to continue its route. This would leave the shared node occupied and prevent the stopped robot from proceeding.

To avoid this, the target of the crossing robot is temporarily shifted beyond the conflicting node in the direction of its next scheduled node. The shifted coordinate is selected such that the robot clears the shared node while remaining on its original route. The local plan is then updated by replacing the target position at $N11$ with this shifted coordinate. Once the crossing robot has cleared the shared node, the stopped robot is released and normal schedule execution resumes. The complete sequence is illustrated in Figure 3.7.

3.3.2 Scenario 2 (Priority Inversion)

Scenario 2 represents a priority-inversion problem in which the order of traversal through a shared node cannot be selected using the priority criteria from Scenario 1 alone. In the example shown in Figure 3.8, robot $A1$ follows the path $N10 \rightarrow N11 \rightarrow N01$, while robot $A2$ follows $N01 \rightarrow N11 \rightarrow N21$. Both robots therefore approach the common node $N11$, but the next node of $A1$, $N01$, is the node from which $A2$ is approaching.

If $A1$ is allowed to cross $N11$ first, it proceeds towards $N01$ while $A2$ is traversing

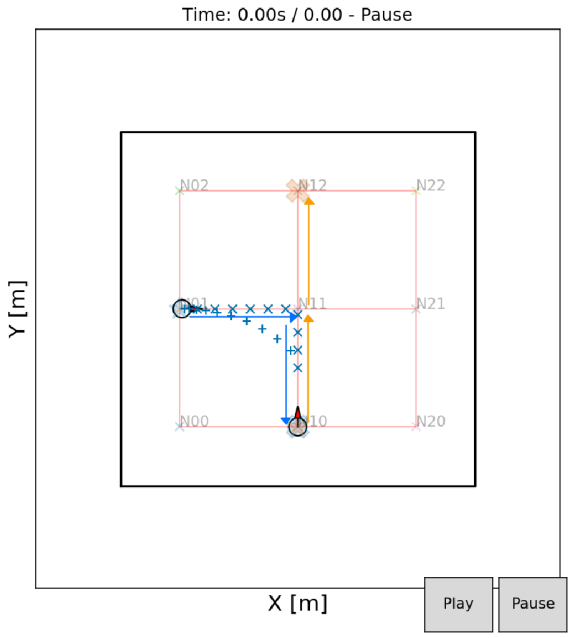
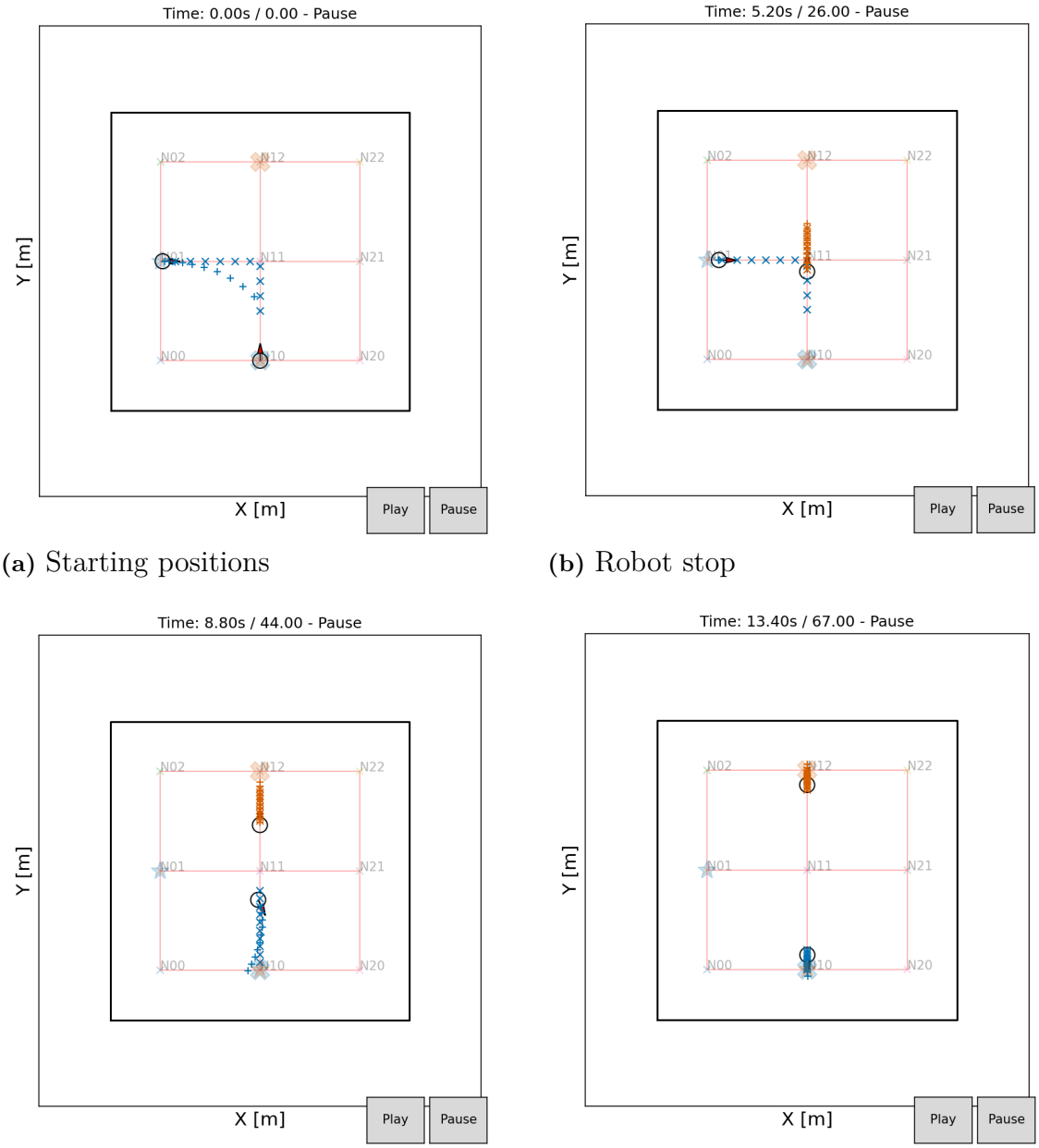


Figure 3.8: Expected paths for scenario 2.

from $N01$ towards $N11$. As a result, the robots would travel towards each other along the same link. In this configuration, allowing $A1$ to cross based only on its schedule status or distance to $N11$, as in Scenario 1, may create a blocking situation immediately after the shared node.

To detect this configuration, the coordinator uses the previous, current target, and next target nodes of each robot. For two robots approaching the same current target node, it checks whether the next target node of one robot is equal to the previous node of the other robot. In the example above, the next target of $A1$ and the previous node of $A2$ are both $N01$. This identifies $A1$ as the robot that would enter



(a) Starting positions

(b) Robot stop

(c) Shifted Coordinate Stop

(d) Crossing Completed

Figure 3.9: Snapshots of scenario 2 process.

the link currently being used by $A2$. The coordinator therefore overrides the priority selection used in Scenario 1. $A1$ is instructed to wait before the conflicting node, while $A2$ is given priority to cross $N11$. Once $A2$ has cleared the shared node, $A1$ can resume execution and continue towards $N01$.

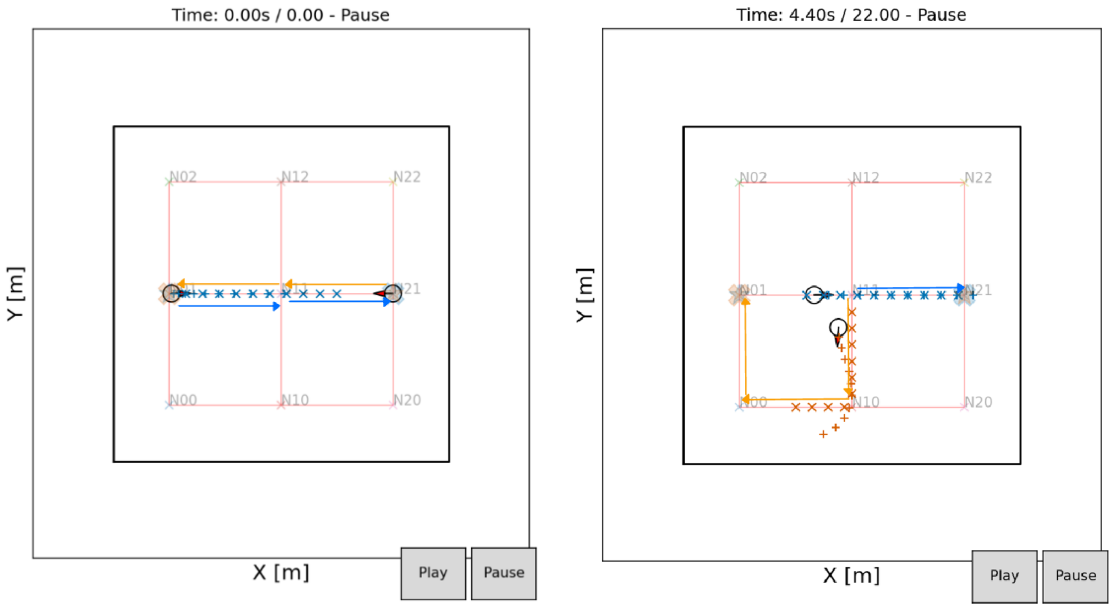
Allowing the crossing robot to reach only the conflicting node may cause it to stop at that node while waiting for its next scheduled target. The shifted-coordinate strategy is therefore also applied in Scenario 2. The target of $A2$ is temporarily shifted beyond $N11$ in the direction of $N21$, ensuring that $A2$ completely clears the shared node before $A1$ is released. The complete execution sequence is shown in Figure 3.9.

3.3.3 Scenario 3 (Replanning)

Scenario 3 represents a conflict that cannot be resolved by changing the order in which the robots traverse the shared node. In the example shown in Figure 3.10a, robot $A1$ follows the path $N01 \rightarrow N11 \rightarrow N21$, while robot $A2$ follows $N21 \rightarrow N11 \rightarrow N01$. The robots are therefore scheduled to traverse the same two links in opposite directions.

Unlike Scenarios 1 and 2, allowing either robot to cross $N11$ first does not resolve the conflict. After crossing the shared node, the selected robot would continue along the link currently occupied by the other robot. A waiting strategy would therefore only postpone the conflict. To resolve this situation, the planned route of one of the robots must instead be changed.

The coordinator detects this scenario using the same previous, current target, and next target node information used in Scenario 2. For two robots approaching the



(a) Expected path for scenario 3 **(b)** Re-planned paths

Figure 3.10: Scenario 3.

same current target node, Scenario 3 is identified when the previous node of each robot is equal to the next target node of the other robot. In the example in Figure 3.10a, the previous and next nodes are exchanged between $A1$ and $A2$, indicating that the robots intend to traverse the same sequence of links in opposite directions.

Once the conflict is detected, one robot is stopped while the other is allowed to reach the conflicting node. Assuming $A1$ is stopped, $A2$ is allowed to reach $N11$. Replanning is initiated from the conflicting node rather than from an additional intermediate position. Introducing an intermediate node without modifying the underlying route representation does not correctly represent the occupied link to the scheduler and can result in the same conflicting route being generated.

After reaching the conflicting node, $A2$ enters a handoff mode in which execution of its original schedule is discontinued. The coordinator then constructs a new scheduling problem using the current execution state. This includes the current position and execution time, the remaining jobs of $A2$, and the existing schedules of the other robots. Tasks that have already been completed by $A2$ are removed so that they are not included again in the new schedule.

The scheduler is then invoked to generate a new route and schedule for $A2$ while accounting for the existing plan of $A1$ and the link currently occupied by it. The resulting route therefore avoids the conflict that triggered replanning. An example of the replanned paths is shown in Figure 3.10b.

Once a feasible schedule has been generated, the new route and schedule replace the original plan of $A2$ and execution resumes using the updated trajectory. $A1$ is subsequently released and continues along its original schedule. The modifications required to enable this execution-time rescheduling are described in Sections 3.3.3.1 and 3.3.3.2.

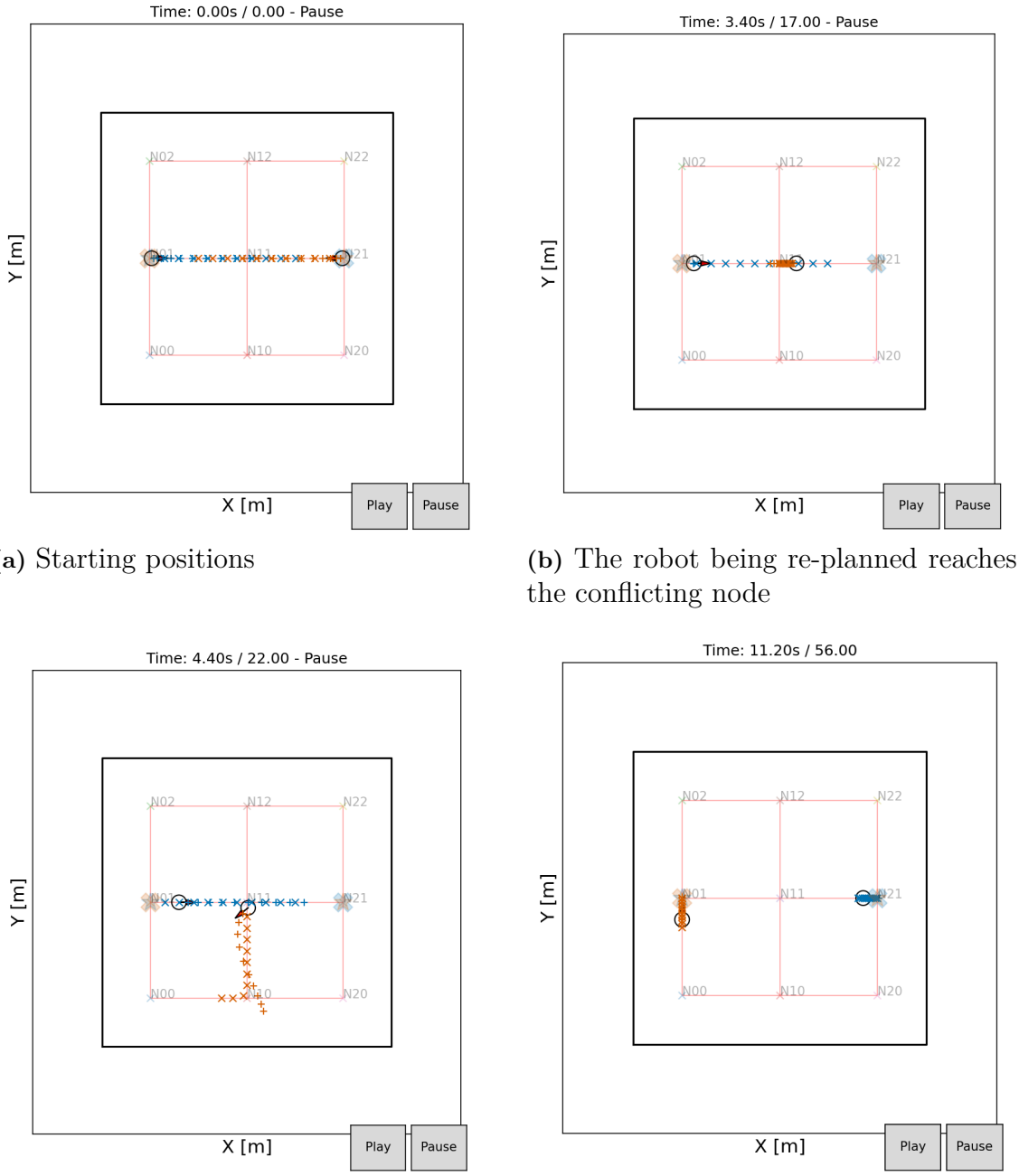
3.3.3.1 Changes made in the scheduler

The original scheduler was designed to generate a complete schedule before execution and therefore required several modifications to support re-planning during execution. These modifications allow a new schedule to be generated from the current state of the fleet without changing the schedules of robots that are not being re-planned.

In the original implementation, schedule generation always starts at $t = 0.0$s. This is not suitable for re-planning, since a new schedule may be requested at any point during execution. The scheduling model was therefore extended to accept the current execution time t_{current}, which is maintained by the coordinator and provided to the scheduler when re-planning is initiated.

The schedules of robots that are not being replanned are treated as frozen schedules. These schedules reserve nodes and links during the time intervals in which they are expected to be occupied. The scheduler must therefore generate the new plan around these existing space-time reservations rather than modifying the plans of the remaining robots.

In addition to the frozen schedules, the link occupied by the stopped robot is treated as unavailable to the replanned robot. This prevents the scheduler from selecting the same route that produced the original conflict and ensures that an alternative path is considered.



(a) Starting positions

(b) The robot being re-planned reaches the conflicting node

(c) Re-plan has happened and robot follows the new path

(d) Conflict solved

Figure 3.11: Snapshots of scenario 3 process.

The scheduling problem is also initialized using the current state of the replanned robot rather than its original starting state. Only the remaining jobs are included in the new scheduling problem, while tasks that have already been completed are removed. Together, these modifications allow the existing scheduling framework to be reused for execution-time replanning using updated initial conditions, execution time, remaining jobs, and the current plans of the other robots.

To enforce these conditions during optimization, additional constraints were introduced into the existing Z3 scheduling model. These constraints are described in Section 3.3.3.2.

3.3.3.2 Constraints added to optimizer

To support replanning during execution, additional constraints were introduced into the Z3 scheduling model. These constraints account for the current execution time and prevent the replanned robot from conflicting with the existing schedules of the other robots.

First, the starting time of the replanned route is constrained to the current execution time. If $t^v i, 0$ represents the starting time of the new route and t_{current} is the current execution time, the constraint is defined as

$$t^v_{i,0} = t_{\text{current}}. \tag{3.2}$$

The schedules of robots that are not being replanned are treated as fixed space-time reservations. The replanned robot must therefore avoid nodes during the time intervals in which they are reserved by another robot. If a node is expected to be occupied by a frozen robot at time t^f_k, the replanned robot must either leave the node before the frozen robot reaches it or enter the node after the frozen robot has cleared it. With $t^v_{i,k}$ and $t^l_{i,k}$ representing the entry and exit times of the replanned robot at node k, respectively, and Δt_s representing the safety margin, this is expressed as

$$\left(t^l_{i,k} \leq t^f_k - \Delta t_s\right) \vee \left(t^v_{i,k} \geq t^f_k + \Delta t_s\right). \tag{3.3}$$

A similar constraint is introduced for links shared between the replanned robot and a frozen schedule. Let $[t^f_{e,\text{start}}, t^f_{e,\text{end}}]$ represent the time interval during which a frozen robot is expected to occupy link (e), and $[t^r_{e,\text{start}}, t^r_{e,\text{end}}]$ represent the corresponding interval for the replanned robot. The replanned robot must complete its traversal before the frozen robot enters the link or wait until the frozen robot has completely cleared it. This is expressed as

$$\left(t^r_{e,\text{end}} + \Delta t_s \leq t^f_{e,\text{start}}\right) \vee \left(t^r_{e,\text{start}} \geq t^f_{e,\text{end}} + \Delta t_s\right). \tag{3.4}$$

These logical constraints are implemented using the Z3 `Or()` operator. Together, the additional constraints allow the scheduler to generate a new route and schedule for the affected robot while treating the existing motion of the remaining robots as fixed space-time reservations.

3.3.4 Combining all scenarios

The three coordination strategies described above are combined into a single conflict-detection procedure. Since all three scenarios involve robots approaching the same target node, the coordinator first groups robots according to their current target node. Further conflict detection is performed only when two or more robots share the same target node.

The conditions defining the three scenarios are not mutually exclusive when evaluated independently. For example, a configuration requiring replanning also satisfies the more general condition that the robots are approaching a common node. The scenarios are therefore evaluated from the most restrictive to the most general condition. Scenario 3 is checked first, followed by Scenario 2 and finally Scenario 1. This ensures that a configuration requiring replanning is not incorrectly handled using a simpler waiting strategy.

Once a conflict has been detected and classified, the coordinator maintains the corresponding coordination decision until the conflict is resolved. For this purpose, an active conflict contains the robots involved, the identified scenario, and the action assigned to each robot. Three actions are used by the coordinator: *stopped*, where the robot is prevented from moving; *crossing*, where the robot is allowed to proceed through the conflicting node; and *handoff*, used in Scenario 3 when the execution of the current schedule is discontinued and a new schedule is generated for the robot. For Scenarios 1 and 2, the shifted target coordinate associated with the crossing robot is also retained.

Maintaining an active conflict prevents the coordinator from repeating the conflict-classification and priority-selection procedure at every update. Instead, the previously selected actions continue to be applied while the conflict remains active. This is particularly important because the relative distances and schedule delays of the robots may change after one robot has been stopped; re-evaluating the priority at every update could otherwise cause the selected crossing robot to change during the resolution of the same conflict.

The conflict remains active until the target-node information indicates that the crossing robot has cleared the conflicting node. The stored coordination actions are then removed and normal execution resumes. The complete conflict-detection and resolution procedure is summarized in Figure 3.12.

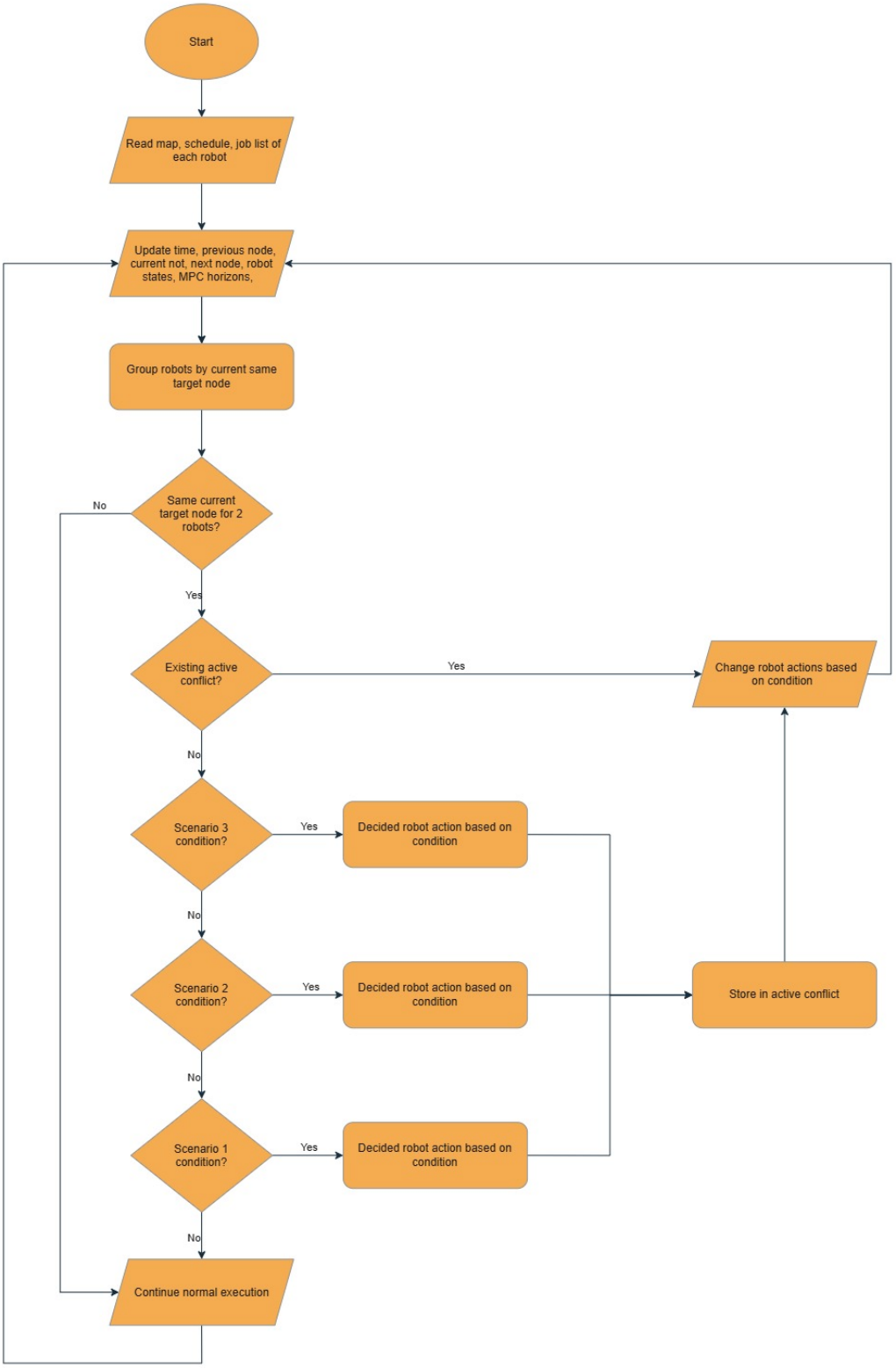


Figure 3.12: Flowchart of the coordinator

3.3.5 Software Implementation of the Coordinator

The coordinator was first integrated into the existing simulation framework. At each simulation step, the coordinator receives the current robot states, simulation time, current target nodes, and MPC prediction horizons. Once the required information from all robots is available, the conflict-detection procedure described in Section 3.3.4 is performed. If no conflict is detected, the existing scheduling, planning, and MPC execution continues without intervention.

When a conflict is detected, the coordinator modifies the execution according to the action assigned to each robot. For a robot assigned the *stopped* action, the control commands sent to the simulated robot are set to zero. The MPC continues to operate, but its computed control input is not applied until the conflict has been resolved.

For a robot assigned the *crossing* action, the shifted coordinate described in Section 3.3.1 is introduced into its local plan. The current target coordinate is temporarily replaced by the shifted coordinate, causing the robot to move beyond the conflicting node before continuing along its original route. The node-level representation of the route is retained during this process so that the coordinator continues to associate the robot with the original conflicting node until it has been cleared.

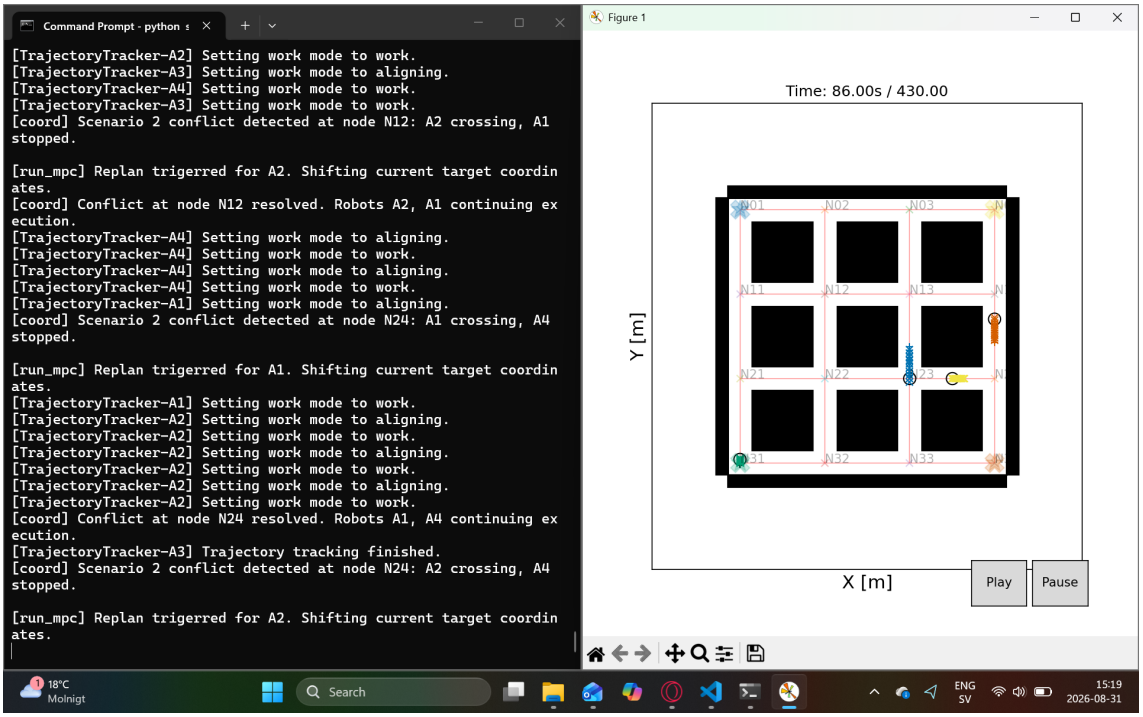


Figure 3.13: Screenshot of coordinator simulations.

The *handoff* action is used only for Scenario 3. In this case, the newly generated route and schedule described in Section 3.3.3 replace the previous plan of the re-planned robot. The local planner is then updated using the new schedule and generates a new reference trajectory for the MPC. Subsequent control actions are therefore computed using the re-planned route.

In this way, the coordinator changes the execution of the robots only when intervention is required. Trajectory generation and control remain part of the existing distributed MPC framework, while the coordinator provides the higher-level decisions required to resolve conflicts between robots. The coordinator outputs were recorded for further analysis as well. The simulations looked as seen in Figure 3.13. The decisions made by the coordinator are also seen on the terminal.

3.3.6 Hardware Implementation of the Coordinator

After validating the coordinator in simulation, it was integrated into the physical multi-robot framework. The hardware experiments focused on Scenarios 1 and 2, which require the coordinator to determine which robot should wait and which robot should proceed through the conflicting node.

The coordination principles remain the same as in simulation. However, in the physical implementation, the coordinator operates on the host computer while the MPC and local planning are executed independently on each Duckiebot. The information required for conflict detection must therefore be collected from the robots, and the resulting coordination decisions must be communicated back to them through the network architecture described in Section 3.2.1. Each Duckiebot sends its current state and MPC prediction horizon to the host computer. The host uses this information to construct the neighbouring-state information required by the distributed MPC. The coordinator uses the same received information along with the current target-node information to perform conflict detection. This avoids introducing an additional communication channel for collecting robot states and predictions.

To communicate the resulting coordination decisions back to the robots, an additional coordinator packet is transmitted from the host computer to each Duckiebot via UDP on port 5010. The packet specifies the execution mode assigned to the robot and, when required, the shifted target coordinate used to clear the conflicting node.

Three execution modes are used in the physical implementation: `WAIT`, `WORK`, and `CROSSING`. In `WAIT` mode, the robot is prevented from moving. In `WORK` mode, the robot follows its schedule and applies the control commands generated by its MPC normally. In `CROSSING` mode, the robot proceeds through the conflicting node using the shifted target coordinate described in Section 3.3.1. These modes provide the interface between the coordination decision made on the host computer and the local execution on each Duckiebot.

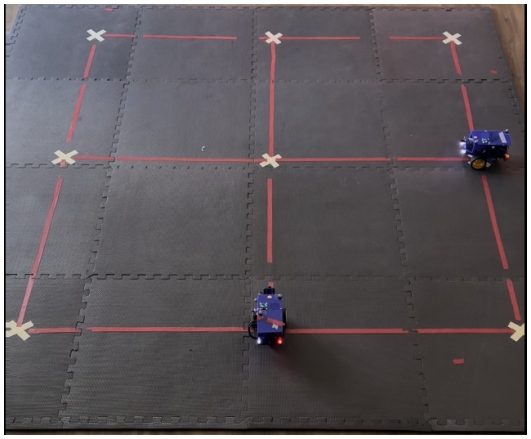
At start-up, the robots are initially assigned `WAIT` mode to prevent movement before the required state and coordination information is available. When the coordinator determines that no conflict is present, the robots are switched to `WORK` and execute their schedules normally.

When a conflict corresponding to Scenario 1 or Scenario 2 is detected, the robot selected to yield is assigned `WAIT`, while the robot selected to proceed is assigned `CROSSING`. The shifted target coordinate is transmitted together with the `CROSSING` command, allowing the robot to clear the conflicting node before returning to its original route. Once the coordinator determines that the shared node has been cleared and the conflict is resolved, the affected robots return to `WORK` mode and

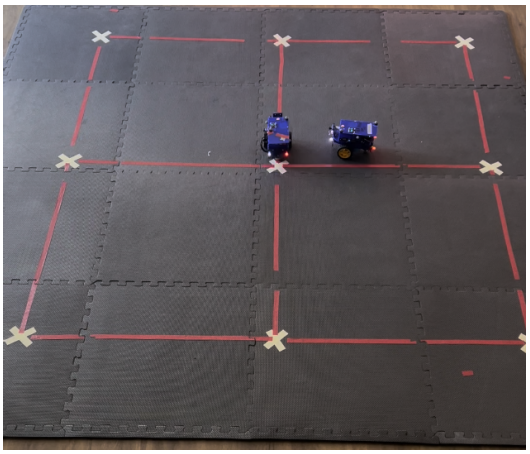
normal execution continues.

This implementation preserves the distributed structure of the original framework: trajectory generation and MPC optimization remain onboard each Duckiebot, while the host-based coordinator only intervenes in the execution when a shared-node conflict is detected.

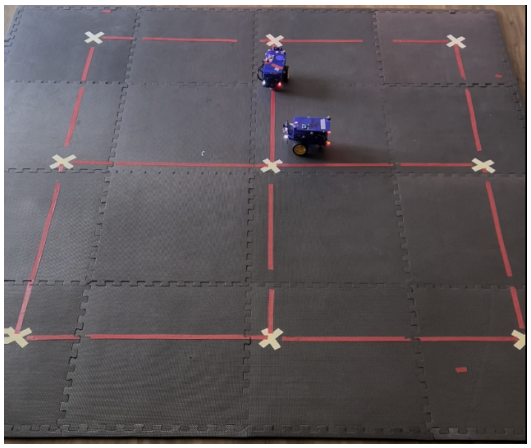
Scenario 1 of the coordinator can be seen as implemented in Figure 3.14. Scenario 2 was also implemented on the coordinator. Videos for both are available in the the git repository. [25]



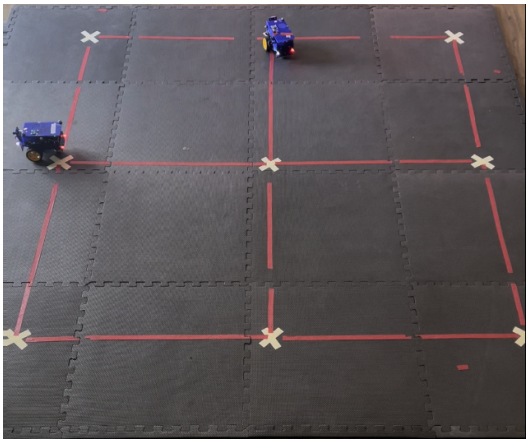
(a) Starting positions of robots.



(b) One robot stopping and letting the other cross.



(c) Crossing robot reaches shifted coordinate before stopped robot starts moving again



(d) Conflict recolved

Figure 3.14: Snapshots of scenario 1 on hardware.

3.4 Experimental methodology

3.4.1 Simulation experiments

The simulation experiments were designed to evaluate the ability of the coordinator to handle conflicts caused by deviations from the nominal schedule. A total of 22 plans were evaluated: six with two robots, six with three robots, and ten with four robots. All plans were executed on the same map shown in Figure 3.15, while the starting positions and number of assigned tasks were varied between experiments.

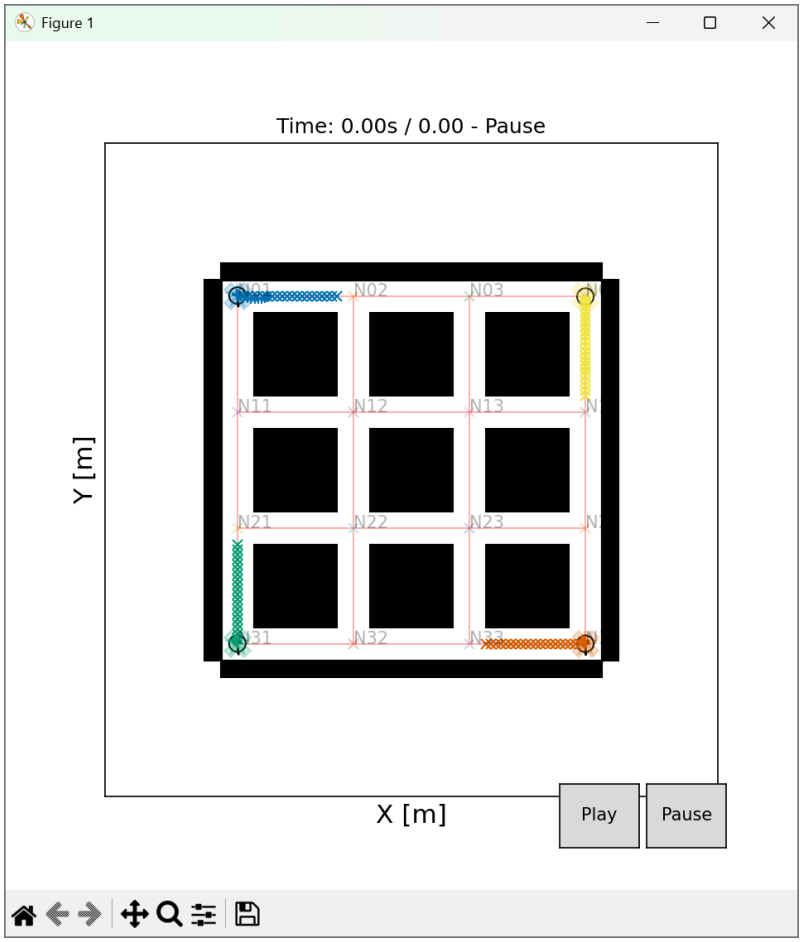


Figure 3.15: Map of coordinator testing in simulations

The scheduler generates conflict-free schedules based on predefined task orders and temporal safety margins. Under nominal execution, these margins can prevent robots from reaching shared resources at approximately the same time, thereby reducing the need for online intervention. The experiments were conducted using scheduling safety coefficients of 0, 1, and 5. A safety coefficient of 10 was additionally evaluated for one of the four-robot configurations. This allowed the relationship between robustness introduced during schedule generation and the amount of online coordination required during execution to be investigated.

The generated schedules assume that the planned traversal times are followed during execution. To evaluate the system when this assumption is violated, disturbances

were introduced after schedule generation by increasing the traversal times of selected links. The original schedule remained unchanged, while its execution deviated from the conditions under which it had been generated. These disturbances created situations in which conflicts could arise during execution despite the existence of an initially feasible schedule.

Three execution configurations were compared. The first used the original MPC formulation without the coordinator. The second used the revised CasADi-based MPC formulation without the coordinator. The third combined the revised MPC formulation with the proposed coordinator. The comparison between the first two configurations was used to evaluate the effect of the MPC modifications, while the comparison between the latter two was used to evaluate the additional contribution of the coordinator.

The recorded metrics included whether the plan was successfully completed, the simulated completion time, the delay relative to the scheduled completion time, and, when the coordinator was enabled, the number and type of coordinator interventions. A run was classified as not completed (NC) when the robots were unable to complete the scheduled plan because of an unresolved conflict.

The reported completion times represent the progression of the simulated task execution and are intended to evaluate schedule adherence and conflict resolution. They should not be interpreted as measurements of computational efficiency. Solver computation time was therefore considered separately when assessing the suitability of the framework for real-time physical deployment.

3.4.2 Hardware experiments

Four types of hardware tests were conducted, one consisting of a single robot completing four different paths to evaluate trajectory-tracking and schedule adherence, one verification of the complete distributed MPC implementation, and two coordinator experiments.

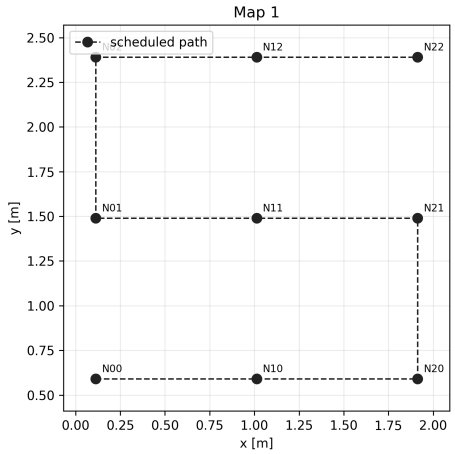
Videos were used to demonstrate the distributed MPC implementation as well as the two coordinator experiments and is presented in section 4.1.3.

3.4.2.1 Singe Robot Tracking

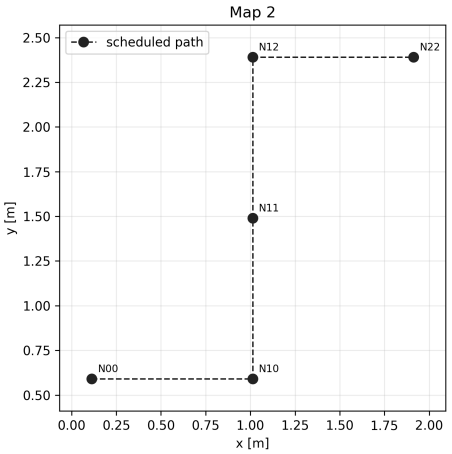
The single-robot experiments were designed to evaluate trajectory tracking and schedule adherence on the physical Duckiebots. Four different maps were used, see Figure 3.16 and each with its own schedule. The paths were selected to include both straight segments and turns of different complexity.

Each map was executed five times, resulting in a total of 20 hardware runs. During each run, the robot pose was recorded using the motion-capture system together with the corresponding schedule information. The recorded data were later used to calculate the trajectory-tracking error and schedule-adherence metrics described in section 4.1.1 and 4.1.2 respectively.

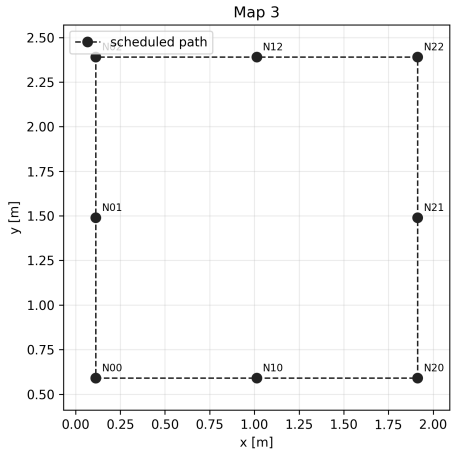
One of the paths was also recorded on video to illustrate the physical execution of the trajectory.



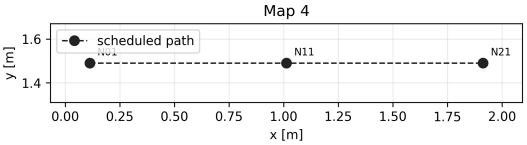
(a) Map 1



(b) Map 2



(c) Map 3



(d) Map 4

Figure 3.16: Scheduled paths used in the single-robot hardware experiments.

4

Results

The following chapter will present the results found throughout the thesis. It will be divided into hardware results and software result. The hardware results will include tracking performance and schedule adherence, videos of a single robot completing it's scheduled trajectory but also videos demonstrating the DMPC with multiple robots and the coordinator scene one and two implementation on hardware. The software results present the performance of the coordinator in simulation, including its effect on execution time and the occurrence of the different coordination scenarios.

4.1 Experiments on physical robots

4.1.1 Error Tracking

The tracking error was evaluated using the logged pose and control data from the physical Duckiebot experiments. For each run, the recorded robot state (x, y, θ), the applied MPC command (v, ω), the experiment time t, and the currently tracked target node were saved in logs. Each map was evaluated over five repeated runs.

The scheduled path consists of horizontal and vertical graph edges. For each logged sample, the active scheduled edge was determined from the current MPC target node and the corresponding previous node in the schedule. The lateral tracking error was then computed as the perpendicular deviation from this edge. For a horizontal edge at $y = y_{\text{ref}}$, the signed tracking error was

$$e_k = y_k - y_{\text{ref}},$$

while for a vertical edge at $x = x_{\text{ref}}$, it was

$$e_k = x_k - x_{\text{ref}}.$$

Only samples with linear velocity $|v| \geq 0.001\,\text{m/s}$ were included, in order to exclude when the robot has reached the final node. The signed errors were stored for inspection, while the reported metrics were computed using the absolute tracking error.

For each run, the root mean square error (RMSE), mean absolute error (MAE), and 95th percentile absolute error were computed as

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{k=1}^{N} e_k^2},$$

$$\mathrm{MAE} = \frac{1}{N} \sum_{k=1}^{N} |e_k|,$$

where N is the number of included samples. The 95th percentile describes the error below which 95% of the included absolute errors fall. Finally, the results were aggregated over the five repetitions of each map and reported as mean ± standard deviation. Table 4.1 - 4.4 below presents the results for each map.

It should also be noted that the position reported by the motion-capture system corresponds approximately to the centre of the Duckiebot. The robot is about 0.13 m wide, corresponding to approximately 0.065 m from the center to either side. The reported lateral tracking error therefore represents the deviation of the robot center from the scheduled path, rather than the distance between the entire robot footprint and the path.

Table 4.1: Path tracking error for Map 1 over five runs.

Run	RMSE [m]	MAE [m]	95th percentile [m]
011	0.065	0.052	0.120
012	0.079	0.063	0.155
013	0.063	0.043	0.136
014	0.091	0.074	0.157
015	0.082	0.066	0.155
Mean	0.076 ± 0.012	0.060 ± 0.012	0.144 ± 0.016

Table 4.2: Path tracking error for Map 2 over five runs.

Run	RMSE [m]	MAE [m]	95th percentile [m]
016	0.083	0.067	0.153
017	0.076	0.064	0.141
018	0.055	0.043	0.089
019	0.075	0.060	0.159
020	0.067	0.055	0.122
Mean	0.071 ± 0.011	0.058 ± 0.010	0.133 ± 0.028

Table 4.3: Path tracking error for Map 3 over five runs.

Run	RMSE [m]	MAE [m]	95th percentile [m]
021	0.065	0.047	0.138
022	0.062	0.050	0.119
023	0.064	0.050	0.137
024	0.071	0.049	0.176
025	0.065	0.048	0.127
Mean	0.065 ± 0.003	0.049 ± 0.001	0.139 ± 0.022

Table 4.4: Path tracking error for Map 4 over five runs.

Run	RMSE [m]	MAE [m]	95th percentile [m]
026	0.044	0.040	0.075
027	0.025	0.021	0.049
028	0.032	0.024	0.059
029	0.033	0.028	0.066
030	0.061	0.040	0.138
Mean	0.039 ± 0.014	0.031 ± 0.009	0.077 ± 0.035

The hardware experiments showed that the MPC was able to track the scheduled paths with relatively small lateral errors. Across the four maps, the mean RMSE ranged from approximately 0.039 m to 0.076 m, while the mean absolute error ranged from 0.031 m to 0.060 m. The tracking performance was therefore good enough for the hardware experiments, although the results varied between the different maps.

Map 4, which consisted of a straight path, gave the lowest tracking error. The maps with more turns generally resulted in larger errors. This was mainly seen around turns and when the robot changed from one path segment to the next. The tracking was still sufficient for the experiments, but these parts could have been improved further.

A large amount of tuning was needed to reach this level of performance. The parameters that worked well in simulation could not be used directly on the physical robots. As shown in Table 3.1, the final hardware parameters were quite different from the simulation parameters. The parameters also differed between the individual Duckiebots. This shows that the physical robots behaved differently from the simulated model and also that the different Duckiebots did not behave exactly the same.

During the hardware development, wheel-speed feedback using a PI controller and system identification were also tested to improve the tracking. Neither approach gave a consistent enough improvement to be included in the final experiments, and both required additional tuning. A more systematic identification of each robot could potentially improve the low-level model, but the available time was instead focused on obtaining sufficiently reliable tracking for the hardware experiments and coordinator evaluation.

The gain and trim calibration of the Duckiebots also affected the MPC tuning. A higher gain made the robots harder to control, while a gain that was too low made them weak when turning. A lower value of around 0.6 gave the best overall behaviour in the final experiments. The trim was also inconsistent over time. A trim value that initially corrected a drift to one side and made the robot drive acceptably straight could later result in a drift in the opposite direction, requiring a new trim value. Since gain and trim affected how the robots responded to the MPC commands, they also increased the amount of tuning required.

4.1.2 Schedule Adherence

As mentioned above the recorded data included the pose, command and time. Schedule adherence was evaluated by comparing the planned arrival time for each scheduled node with the measured arrival time from the physical experiment. For each map, the actual arrival time was averaged over the five repeated runs, and the timing difference was computed as actual arrival time minus planned arrival time. Positive values therefore indicate that the Duckiebot reached the node later than scheduled.
For each scheduled node, the timing difference was calculated as

$$\Delta t_k = \bar{t}_{\text{actual},k} - t_{\text{planned},k},$$

where $\bar{t}_{\text{actual},k}$ is the mean measured arrival time over the five repeated runs and $t_{\text{planned},k}$ is the scheduled arrival time at node k. Positive values therefore indicate that the robot reached the node later than scheduled.
The mean absolute timing error was calculated as

$$\text{MAE}_{\text{schedule}} = \frac{1}{M-1} \sum_{k=2}^{M} |\Delta t_k|,$$

where M is the number of scheduled nodes. The initial node was excluded since its arrival time is fixed at $t = 0$ and does not represent schedule execution.
In Table 4.5 - 4.8 is the averaged arrival time for each map and in Table 4.9 is a summary of the mean absolute error for all nodes and the final node delay.

Table 4.5: Schedule adherence results for Map 1.

Node	Planned [s]	Actual mean [s]	Difference mean [s]
N00	0.00	0.00	0.00
N10	3.59	5.71	2.12
N20	7.19	9.84	2.65
N21	10.79	15.13	4.34
N11	14.39	20.99	6.60
N01	17.99	24.94	6.95
N02	21.59	30.01	8.42
N12	25.19	38.38	13.19
N22	28.79	46.17	17.38

Table 4.6: Schedule adherence results for Map 2.

Node	Planned [s]	Actual mean [s]	Difference mean [s]
N00	0.00	0.00	0.00
N10	3.59	6.99	3.40
N11	7.19	10.97	3.78
N12	10.79	14.99	4.20
N22	14.39	22.11	7.72

Table 4.7: Schedule adherence results for Map 3.

Node	Planned [s]	Actual mean [s]	Difference mean [s]
N00	0.00	0.00	0.00
N10	3.59	5.63	2.04
N20	7.19	9.70	2.51
N21	10.79	15.04	4.25
N22	14.39	21.64	7.25
N12	17.39	25.64	8.25
N02	21.59	29.59	8.00
N01	25.19	34.73	9.54
N00	28.79	39.10	10.31

Table 4.8: Schedule adherence results for Map 4.

Node	Planned [s]	Actual mean [s]	Difference mean [s]
N01	0.00	0.00	0.00
N11	3.59	5.63	2.04
N21	7.19	10.52	3.33

Table 4.9: Schedule adherence summary for all maps.

Map	Mean absolute error [s]	Final node delay [s]
Map 1	7.71	17.38
Map 2	4.78	7.72
Map 3	6.52	10.31
Map 4	2.69	3.33

The schedule-adherence results were less successful than the trajectory-tracking results. For all four maps, the robot reached the scheduled nodes later than planned. The delay also generally increased as the robot progressed along the route, which resulted in a considerable delay at the final node.

One reason for this is that the schedule was generated using an idealized travel time. The scheduled edges were approximately 0.9 m long and had a planned travel time of 3.59–3.60 s. The scheduler therefore assumed an average traversal velocity of approximately 0.25 m/s for each edge.

In practice, the robot does not travel at a constant velocity over an entire edge. It accelerates from lower speeds, slows down when approaching a node, and changes direction when moving between path segments. The average physical velocity over an edge can therefore be lower than the velocity assumed by the scheduler, even if the commanded velocity is higher during parts of the traversal.

In addition, the velocity generated by the MPC does not correspond directly to the physical velocity of the Duckiebot. The MPC does not control the wheel motors directly. Instead, the commanded linear and angular velocities are passed to the low-level motor controller inside the Duckietown container, which converts them into left- and right-wheel commands. This low-level controller was not modified as part of this thesis.

The wheel response is adjusted through the Duckiebot calibration parameters gain and trim. The gain scales the overall motor command and therefore affects how strongly both wheels are driven. The trim is used to compensate for differences between the left and right motors so that the robot drives straighter. These parameters are calibrated on the individual Duckiebot and are separate from the MPC tuning.

Because of this additional low-level layer, the physical motion of the robot does not necessarily correspond exactly to the velocity requested by the MPC. The time required to traverse an edge can therefore differ from the travel time assumed by the scheduler.

The results show that the physical Duckiebot generally required more than the scheduled 3.59–3.60 s to travel between nodes. As a result, timing errors accumulated over the route and the robot reached the later nodes increasingly behind schedule. This can be seen particularly clearly in Maps 1–3, where the largest delays generally occurred towards the end of the route.

4.1.3 Video demonstration

Videos demonstrating the hardware implementation of the distributed MPC and the coordinator scenarios are available in the project repository [25]. The files `3robot_1.mov`, `3robot_2.mov`, and `3robot_3.mov` show three Duckiebots executing scheduled trajectories using the distributed MPC framework. These recordings provide a qualitative demonstration that the distributed MPC could be operated with three physical robots. In these videos the full NMPC implementation, including both the static obstacle-avoidance constraints and the dynamic fleet-collision constraints is used.

A screenshot from `3robot_3.mov` is presented in Figure 4.1 and Figure 4.2 shows the monitor node that acts as a real-time virtual representation of the physical robot fleet, combining the planned schedules with MOCAP measurements and MPC predictions.

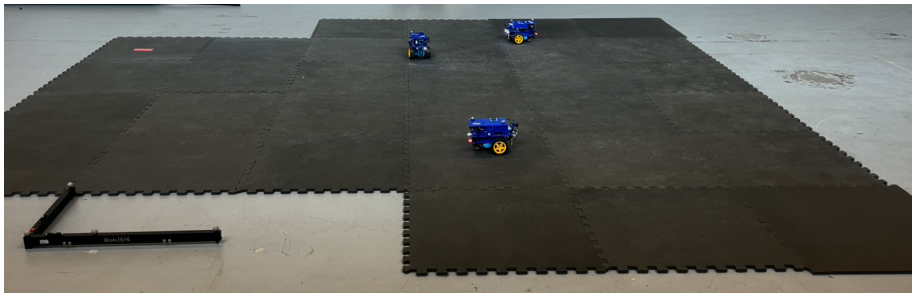


Figure 4.1: Screenshot from the video `3robot_3.mov` which demonstrates the full DMPC on three Duckiebots.

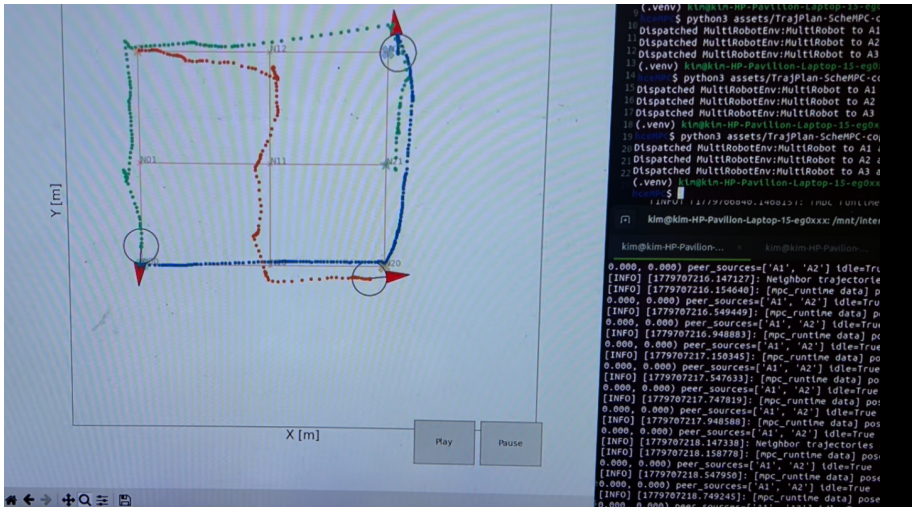


Figure 4.2: Tracking result as seen in the monitor node from the run in video`3robot_3.mov`.

The file `DistributedMPC.mov` shows the distributed MPC operating with two Duckiebots.

The three-robot videos show that the distributed MPC was successfully operated on three Duckiebots. These recordings were made before the final controller tuning was completed. One of the robots later became unavailable due to a hardware failure, so the final hardware experiments were carried out with two robots.

The file `Single_Robot_Tracking.mov`, is a video from the recorded data used in error tracking analysis.

4.2 Experiments in simulated environment

The simulation experiments compared three configurations: the original MPC, the revised MPC, and the revised MPC combined with the coordinator. The purpose of this comparison was to distinguish the improvements resulting from the revised MPC formulation from those introduced specifically by the coordinator. The experiments also evaluated the ability of each configuration to complete the scheduled plans when execution-time disturbances caused the robots to deviate from the nominal schedule.

Table 4.10 presents the completion time and final delay for each configuration. A run is reported as NC (not completed) when the robots were unable to complete the scheduled plan because of an unresolved conflict. The completion times represent the simulated execution of the plans and should not be interpreted as measurements of solver computation time.

Table 4.10: Schedule execution using the old MPC, the new MPC, and the new MPC with the coordinator for different scheduling safety coefficients.

Run	Robots	Safety	ETA	Old MPC		New MPC		New MPC + Coordinator	
		Coeff.	[s]	Completion [s]	Delay [s]	Completion [s]	Delay [s]	Completion [s]	Delay [s]
1	2	0	114.0	118.8	4.8	115.8	1.8	128.6	14.6
2	2	1	131.0	137.8	6.8	132.0	1.0	131.4	0.4
3	2	5	134.0	217.0	83.0	135.0	1.0	134.4	0.4
4	2	0	154.0	NC	NC	NC	NC	156.4	2.4
5	2	1	158.0	NC	NC	165.4	7.4	NC	NC
6	2	5	161.0	165.0	4.0	168.6	7.6	161.6	0.6
7	3	0	112.0	NC	NC	113.6	1.6	129.2	17.2
8	3	1	112.0	237.0	125.0	113.6	1.6	112.6	0.6
9	3	5	115.0	163.6	48.6	116.4	1.4	115.6	0.6
10	3	0	115.0	NC	NC	NC	NC	115.8	0.8
11	3	1	132.0	NC	NC	133.6	1.6	132.6	0.6
12	3	5	141.0	NC	NC	142.6	1.6	141.4	0.4
13	4	0	105.0	NC	NC	107.2	2.2	107.4	2.4
14	4	1	121.0	NC	NC	NC	NC	121.8	0.8
15	4	5	131.0	362.0	231.0	133.0	2.0	131.6	0.6
16	4	0	133.0	NC	NC	NC	NC	152.2	19.2
17	4	1	154.0	197.8	43.8	158.2	4.2	154.6	0.6
18	4	5	161.0	NC	NC	163.0	2.0	161.6	0.6
19	4	0	140.0	187.2	47.2	NC	NC	141.2	1.2
20	4	1	141.0	NC	NC	197.0	56.0	141.6	0.6
21	4	5	148.0	150.8	2.8	204.0	56.0	148.6	0.6
22	4	10	163.0	NC	NC	212.0	49.0	163.6	0.6

The comparison between the original and revised MPC formulations shows that the modifications made to the MPC resolved several problems present in the original implementation. The revised MPC generally completed the schedules with smaller delays and produced more consistent execution. For example, in Run 3, the delay decreased from 83.0s with the original MPC to 1.0s with the revised MPC. Similarly, in Run 15, the delay decreased from 231.0s to 2.0s. These results show that part of the improvement observed in the complete framework resulted from the revised MPC formulation.

The comparison between the revised MPC with and without the coordinator demonstrates the additional contribution of online coordination. In Runs 4, 10, 14, 16, and 19, the revised MPC without the coordinator resulted in NC, whereas the coordinator framework successfully completed the corresponding schedules. In Run 19,

for example, the revised MPC was unable to complete the plan, while the coordinator framework completed it with a delay of $1.2s$. In Run 20, the revised MPC completed with a delay of $56.0s$, whereas the coordinator reduced the delay to $0.6s$. These cases demonstrate that the coordinator can resolve interactions that either prevent the MPC from progressing or cause substantial delays.

The coordinator did not improve every individual execution. In Run 1, the revised MPC completed with a delay of $1.8s$, while the coordinator framework completed with a delay of $14.6s$. The additional delay was caused by the waiting and re-planning actions introduced during conflict resolution. Run 5 was the only configuration in which the revised MPC completed while the coordinator framework resulted in NC. The contribution of the coordinator should therefore be interpreted primarily as improving the robustness and recoverability of execution rather than consistently reducing the completion time of every plan.

A specific limitation of the revised MPC was observed when two robots approached each other from opposite directions on the same link. In these situations, the revised MPC could cause both robots to remain stationary because neither robot found a motion that resolved the interaction. With the original formulation, the robots sometimes moved slightly forward or backward and eventually found a feasible solution. Although the revised formulation was more successful overall, it was less capable of resolving this particular type of head-on interaction without higher-level intervention. This behaviour further motivates the re-planning and conflict-resolution mechanisms provided by the coordinator.

Table 4.11 presents the number of coordinator interventions during each run, separated according to the three conflict scenarios.

The experiments with different scheduling safety margins further illustrate the relationship between offline scheduling and online coordination. Runs 13-15, Runs 16-18, and Runs 19-22 use the same respective task configurations while varying the temporal safety margin used during schedule generation. Increasing this margin generally increases the temporal separation between robots and therefore produces a more conservative schedule.

This effect can be observed in Runs 16-18. Increasing the safety coefficient from 0 to 1 and then to 5 increased the scheduled completion time from $133s$ to $154s$ and $161s$, respectively. At the same time, the number of coordinator interventions decreased from 9 to 7 and 5. The final execution delay also decreased from $19.2s$ with a coefficient of 0 to $0.6s$ with coefficients of both 1 and 5. A similar reduction in coordinator activity can be observed in Runs 19-22, where the number of interventions decreased from 11 with a coefficient of 0 to 5 with the larger coefficients.

The reduction in coordinator interventions is expected because a larger safety margin introduces additional separation during schedule generation. Consequently, some interactions that require execution-time coordination with a smaller margin are avoided by the scheduler. The scheduler and coordinator therefore provide complementary mechanisms for handling potential conflicts: the scheduling safety margin handles them proactively, while the coordinator responds to conflicts that arise during execution.

However, reducing the number of coordinator interventions does not necessarily

Table 4.11: Coordinator scenario activations during each experimental run.

Run	Robots	Safety Coeff.	Case 1	Case 2	Case 3	Total
1	2	0	1	1	1	3
2	2	1	1	2	0	3
3	2	5	0	2	0	2
4	2	0	3	2	0	5
5	2	1	–	–	–	–
6	2	5	0	2	0	2
7	3	0	1	2	1	4
8	3	1	1	3	0	4
9	3	5	0	3	0	3
10	3	0	2	4	0	6
11	3	1	2	4	0	6
12	3	5	0	4	0	4
13	4	0	0	1	2	3
14	4	1	0	7	0	7
15	4	5	0	4	0	4
16	4	0	4	3	2	9
17	4	1	0	7	0	7
18	4	5	0	5	0	5
19	4	0	5	6	0	11
20	4	1	4	6	0	10
21	4	5	0	5	0	5
22	4	10	0	5	0	5

result in a shorter overall execution time. The additional temporal separation introduced by the scheduler also increases the nominal duration of the schedule. For example, in Runs 19-22, the schedule with a safety coefficient of 0 had an ETA of $140s$ and was completed by the coordinator framework in $141.2s$ despite requiring 11 interventions. Increasing the safety coefficient to 10 reduced the number of interventions to 5 but increased the scheduled completion time to $163s$. The number of interventions should therefore not be interpreted independently as a measure of execution performance. A tighter schedule may require more online coordination while still completing the assigned tasks earlier than a more conservative schedule.

An important limitation of these results is that the completion times in Table 4.10 describe the simulated progression of the scheduled tasks rather than the computational performance of the controller. When robots approached conflicting areas, the revised CasADi-based MPC could require more than one second to calculate a single control step for an individual robot. When several robots simultaneously encountered conflicts, this computational load prevented the system from maintaining the intended real-time control rate.

Consequently, the smaller delays shown in Table 4.10 indicate that the simulated robot motion remained closer to the schedule, but they do not demonstrate that the revised framework was computationally faster. The computational delay can be overlooked when evaluating the logical behavior of the framework in simulation,

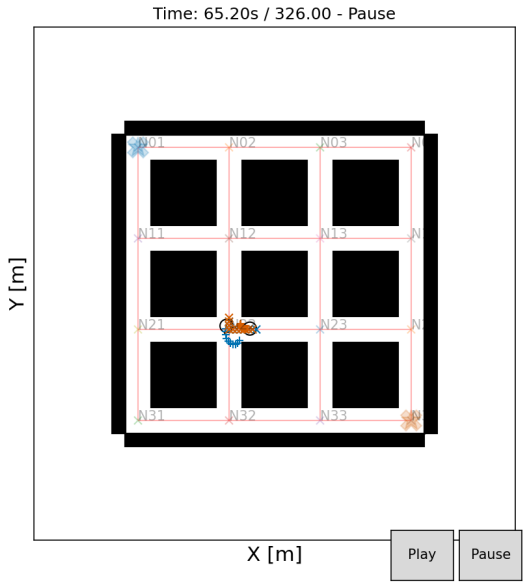
but it would be a significant limitation for deployment on a physical fleet. Further optimization would therefore be required before the complete collision-avoidance formulation could be used reliably in real time.

A further limitation concerns the geometric representation of the robots. Collision avoidance in the simulation is based on a disc-shaped approximation of each robot. In some interactions, the MPC resolved a conflict only after the robots moved very close to one another and navigated around each other. Although such a maneuver remained feasible according to the circular collision model, physical robots generally have non-circular footprints and orientation-dependent swept areas. A maneuver classified as collision-free in the simulation may therefore be infeasible or unsafe for the corresponding physical robots.

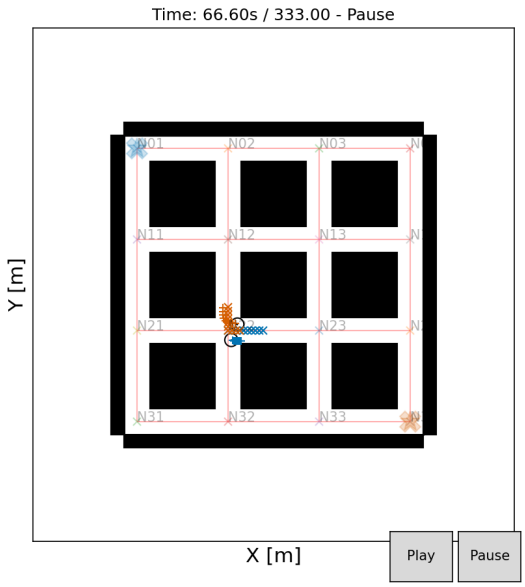
Figure 4.3 illustrates an example of this close-proximity behaviour. The simulation results consequently demonstrate the logical conflict-resolution capability of the framework but do not establish that every generated maneuver could be performed safely by a physical robot fleet.

The revised MPC provided a more successful solution than the original formulation and resolved several of the problems encountered during its implementation. The coordinator further improved execution robustness by resolving conflicts that remained problematic for the revised MPC, particularly interactions requiring waiting, priority changes, or re-planning. Nevertheless, the solver computation time, the difficulty of resolving head-on interactions, and the simplified disc-shaped representation of the robots remain important limitations when considering physical deployment.

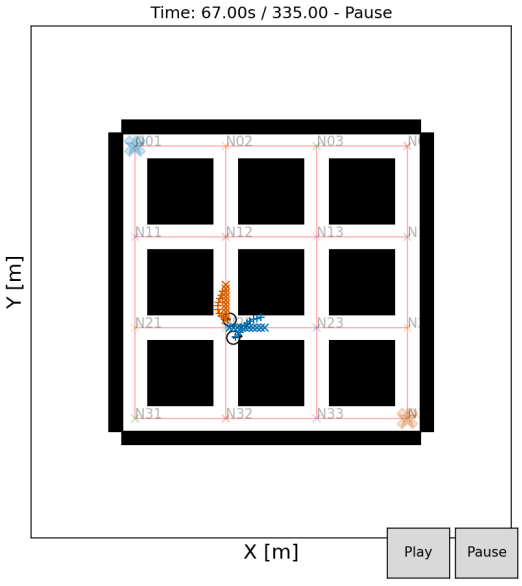
Overall, the original MPC completed 10 of the 22 runs, the revised MPC completed 17 of the 22 runs, and the coordinator framework completed 21 of the 22 runs. The experiments show that increasing the scheduling safety margin can reduce the frequency of execution-time conflicts, but at the cost of more conservative schedules. The coordinator provides an alternative mechanism in which tighter schedules can be retained and conflicts can instead be handled when they occur. The results therefore support the use of the coordinator as an execution-monitoring and recovery layer that complements, rather than replaces, the robustness provided by the high-level scheduler.



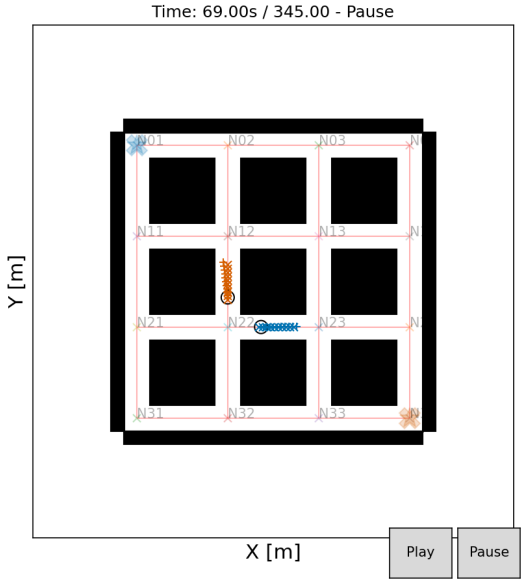
(a) Robots entering a conflict



(b) Robots navigating around each other



(c) Robots finding a path that works for simulations



(d) Conflict solved

Figure 4.3: snapshots of how MPC solves the conflict. This cannot be used on a physical fleet as robot geometry will not allow that.

5

Conclusion

The framework proposed by Roselli et al. [2], together with the proposed coordinator, was successfully implemented on the Duckiebots. The complete distributed MPC formulation was also shown to work on the physical robots, although the static obstacle-avoidance and dynamic fleet-collision constraints were omitted from the final experiments due to the additional tuning required. The hardware experiments showed that the MPC could track the scheduled paths with relatively small lateral errors. The results also showed a clear sim-to-real difference, since the parameters used in simulation could not be transferred directly to the physical robots and had to be tuned separately. The behaviour of the Duckiebots, particularly the gain and trim calibration, also affected the final tracking performance. Schedule adherence was less successful than the trajectory tracking. The robots generally reached the scheduled nodes later than planned. This indicates that the simplified timing assumptions used by the scheduler do not fully represent the motion of the physical robots, where acceleration, slowing down, and turning affect the actual traversal time between nodes. The coordinator was successfully integrated into the existing framework. Scenarios 1 and 2 were demonstrated on physical Duckiebots, while Scenario 3, which required replanning, was demonstrated in simulation. The results therefore show that an additional execution-monitoring layer can be integrated between the scheduler and local control level to handle conflicts that arise during execution.

As shown in Table 4.10, the original MPC completed 10 of the 22 runs, the revised MPC completed 17, and the coordinator framework completed 21. This comparison shows that the revised MPC resolved several limitations of the original formulation, while the coordinator provided a further improvement in execution robustness. However, the revised MPC experienced greater difficulty when robots were close to each other. The solution time for a single MPC step increased from the typical $20 - 50ms$ to as much as $1.5s$ per robot. Additionally, when two robots approached each other from opposite directions on the same link, they remained stationary, whereas the original formulation sometimes produced small forward or backward movements through which a solution was eventually found. The coordinator further improved the robustness of schedule execution by resolving interactions through waiting, priority changes, or re-planning. With the coordinator in place, the new CasADi-based MPC formulation could omit the dynamic fleet-collision constraints, reducing the computational load on the solver and consistently producing solutions in less than $50ms$ per step.

Some conflict-resolution maneuvers relied on the disc-shaped robot approximation used in the simulation. The close-proximity movements considered feasible by this

model may not be feasible or safe for physical robots with non-circular designs. A more representative robot model should therefore be used to produce more realistic simulations.

The experiments with different scheduling safety margins also showed a trade-off between conservative scheduling and online coordination. Increasing the temporal safety margin reduced the number of coordinator interventions, since greater separation between robots was introduced already during schedule generation. However, this also increased the nominal completion time of the schedules. Conversely, smaller safety margins produced tighter schedules and required more frequent coordinator intervention. The coordinator therefore complements the scheduler by allowing execution-time conflicts to be handled dynamically rather than relying entirely on conservative margins during schedule generation. The results suggest that combining the coordinator with smaller scheduling safety margins may improve execution efficiency by allowing tighter nominal schedules while resolving conflicts dynamically when required.

The original MPC also showed variation between repeated executions of the same schedule, with some robot interactions producing different outcomes between runs. The revised MPC showed more repeatable execution behaviour and resolved several of the problems encountered with the original formulation. The additional comparison without the coordinator indicates that part of the improvement resulted from the revised MPC itself, while the coordinator provided a further improvement by resolving conflicts that the revised MPC could not handle independently.

Future work should focus on improving the low-level control and system identification of the robots, as well as using a more realistic estimate of physical traversal times in the scheduler. The computation time of the revised MPC with dynamic fleet-collision constraints should also be reduced to ensure that the required control frequency can be maintained when multiple robots encounter conflicts. The collision model should be extended to represent the actual robot design, and the handling of robots travelling in opposite directions on the same link should be improved. It would also be interesting to evaluate the framework on another mobile-robot platform and to extend the hardware evaluation of the coordinator to more robots and the re-planning scenario. The most appropriate safety coefficient should also be evaluated to achieve a suitable balance between offline scheduling and online coordination. Future simulation studies should also consider stochastic execution-time disturbances rather than predefined delays, providing a closer approximation of the uncertainties encountered during physical fleet operation. Finally, future work should also extend the quantitative coordinator evaluation to the physical Duckiebots by performing repeated experiments with and without the coordinator. This would allow its ability to handle real execution-time disturbances, such as communication delays, wheel slip, and actuator variations, to be evaluated directly.

Bibliography

[1] P. H. C. Morais, K. C. T. Vivaldini, E. R. R. Kato, and R. S. Inoue, "A Review of Robot Fleet Management," *IEEE Access*, vol. 13, 2025. doi: 10.1109/ACCESS.2025.3586564.

[2] S. F. Roselli, Z. Zhang, and K. Åkesson, "Combining High Level Scheduling and Low Level Control to Manage Fleets of Mobile Robots," *arXiv preprint* arXiv:2510.23129, 2025. doi: 10.48550/arXiv.2510.23129. Available: `https://arxiv.org/abs/2510.23129`

[3] C. Rema, P. Costa, M. Silva, and E. J. S. Pires, "Task Scheduling with Mobile Robots—A Systematic Literature Review," *Robotics*, vol. 14, no. 6, Art. no. 75, 2025. doi: 10.3390/robotics14060075.

[4] S. Yi and J. Luo, "Heuristic Scheduling for Robotic Job Shops Using Petri Nets and Artificial Potential Fields," *IEEE Transactions on Automation Science and Engineering*, vol. 22, pp. 7556–7568, 2025. doi: 10.1109/TASE.2024.3464857.

[5] F. Wang, Y. Zhang, and Z. Su, "A Novel Scheduling Method for Automated Guided Vehicles in Workshop Environments," *International Journal of Advanced Robotic Systems*, vol. 16, no. 3, pp. 1–13, 2019. doi: 10.1177/1729881419844152.

[6] J. Lu, C. Ren, Y. Shao, J. Zhu, and X. Lu, "An Automated Guided Vehicle Conflict-Free Scheduling Approach Considering Assignment Rules in a Robotic Mobile Fulfillment System," *Computers & Industrial Engineering*, vol. 176, Art. no. 108932, 2023. doi: 10.1016/j.cie.2022.108932.

[7] P. Durst, X. Jia, and L. Li, "Multi-Objective Optimization of AGV Real-Time Scheduling Based on Deep Reinforcement Learning," in *Proceedings of the 42nd Chinese Control Conference (CCC)*, Tianjin, China, 2023, pp. 5535–5540.

[8] S. T. Atik, A. S. Chavan, D. Grosu, and M. Brocanelli, "A Maintenance-Aware Approach for Sustainable Autonomous Mobile Robot Fleet Management," *IEEE Transactions on Mobile Computing*, vol. 23, no. 6, pp. 7394–7407, 2024. doi: 10.1109/TMC.2023.3334589.

[9] R. Raj and A. Kos, "Discussion on Different Controllers Used for the Navigation of Mobile Robot," *International Journal of Electronics and Telecommunications*, vol. 70, no. 1, pp. 229–239, 2024. doi: 10.24425/ijet.2024.149535.

[10] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. New York, NY, USA: Springer, 2008.

[11] E. Aljalbout, J. Xing, A. Romero, I. Akinola, C. R. Garrett, E. Heiden, A. Gupta, T. Hermans, Y. Narang, D. Fox, D. Scaramuzza,

and F. Ramos, "The Reality Gap in Robotics: Challenges, Solutions, and Best Practices," *arXiv preprint* arXiv:2510.20808, 2025. Available: https://arxiv.org/abs/2510.20808

[12] O. Pettersson, "Execution Monitoring in Robotics: A Survey," *Robotics and Autonomous Systems*, vol. 53, no. 2, pp. 73–88, 2005. doi: 10.1016/j.robot.2005.09.004. Available: https://www.sciencedirect.com/science/article/pii/S092188900500134X

[13] M. Čáp, J. Gregoire, and E. Frazzoli, "Provably Safe and Deadlock-Free Execution of Multi-Robot Plans Under Delaying Disturbances," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 5113–5118. doi: 10.1109/IROS.2016.7759750.

[14] D. E. Soltero, S. L. Smith, and D. Rus, "Collision Avoidance for Persistent Monitoring in Multi-Robot Systems with Intersecting Trajectories," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, San Francisco, CA, USA, 2011, pp. 3645–3652.

[15] A. Coskun and J. M. O'Kane, "Online Plan Repair in Multi-robot Coordination With Disturbances," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 3333–3339. doi: 10.1109/ICRA.2019.8793522.

[16] K. Tong, M. Steinberger, M. Horn, S. Solmaz, and D. Watzenig, "Anytime Optimal Trajectory Repairing for Autonomous Vehicles," *IEEE Open Journal of Intelligent Transportation Systems*, vol. 6, pp. 537–553, 2025. doi: 10.1109/OJITS.2025.3563823.

[17] J. R. Heselden and G. P. Das, "Heuristics and Rescheduling in Prioritised Multi-Robot Path Planning: A Literature Review," *Machines*, vol. 11, no. 11, Art. no. 1033, 2023. doi: 10.3390/machines11111033.

[18] P. Forte, A. Mannucci, H. Andreasson, and F. Pecora, "Online Task Assignment and Coordination in Multi-Robot Fleets," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4584–4591, 2021. doi: 10.1109/LRA.2021.3068918.

[19] J. Hažík, M. Dekan, P. Beňo, and F. Duchoň, "Fleet Management System for an Industry Environment," *Journal of Robotics and Control*, vol. 3, no. 6, pp. 779–789, 2022. doi: 10.18196/jrc.v3i6.16298.

[20] A. Souto, P. A. Prates, A. Lourenço, M. S. Al Maamari, F. Marques, D. Taranta, L. do Ó, R. Mendonça, and J. Barata, "Fleet Management System for Autonomous Mobile Robots in Secure Shop-Floor Environments," in *2021 IEEE 30th International Symposium on Industrial Electronics (ISIE)*, 2021. doi: 10.1109/ISIE45552.2021.9576269.

[21] A. Gräfe, J. Eickhoff, M. Zimmerling, and S. Trimpe, "DMPC-Swarm: Distributed Model Predictive Control on Nano UAV Swarms," *Autonomous Robots*, vol. 49, Art. no. 28, 2025. doi: 10.1007/s10514-025-10211-w.

[22] Duckietown, "Duckietown Platform," Available: https://duckietown.com/platform/ [Accessed: 05-May-2026].

[23] Qualisys AB, "Arqus," [Online]. Available: https://www.qualisys.com/cameras/arqus/ [Accessed: Aug. 24, 2026].

[24] Qualisys AB, "QTM Real-time Server Protocol," [Online]. Available: https://docs.qualisys.com/qtm-rt-protocol/ [Accessed: Aug. 24, 2026].

[25] T. Manglani and K. Hotvedt, "Coordinator shceMPC – Demonstration Videos," GitHub repository. [Online]. Available: https://github.com/tosht0sh/coordinator_shceMPC/tree/v3/docs/demos [Accessed: Aug. 26, 2026].

A

Appendix 1

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY