Decentralized Coordination of Networked Cobots: A Graph-Theoretic Approach to Object Transportation

by

Zuguang Liu

Thesis submitted to Bradley University
in partial fulfillment of the requirements for the degree of
Master of Science
in
Electrical Engineering

© Zuguang Liu, Peoria, Illinois, USA, 2024

This Thesis for the M.S. Degree in Electrical Engineering

by

Zuguang Liu

has been approved

Date 12/20/2024

Chairman, Thesis Committee

Dr. Md Suruz Miah

Reader, Thesis Committee

Dr. Aleksander Malinowski

Reader, Thesis Committee

Dr. Jacqueline Henderson

Chairman, Department of Electrical

& Computer Engineering

Dr. Yufeng Lu

Abstract

As the deployment of collaborative robots (cobots) in industrial settings becomes more prevalent, the need for autonomous systems that can navigate and collaborate with each other to complete tasks in shared workspaces is increasing. This thesis focuses on the development of a multi-robot system designed for autonomous navigation and cooperative object transportation in industrial environments. The research addresses the problem of cooperative object transportation, requiring the robots to share information and coordinate their actions efficiently.

This thesis presents a complete package of autonomous multi-robot systems for cooperative object transportation tasks, including the design of control, navigation, and communication systems. In particular, the communication network among the robots is modeled using directed graphs, with the graph Laplacian matrix representing the connectivity among the robots. The system is validated in a simulation environment and later realized on real robots to evaluate its performance in various configurations. The main contributions of this thesis include the design and implementation of a decentralized coordination strategy based on graph Laplacian, the development of a set of technologies to implement the control and communication systems for the robots, and the validation of the system in simulation and real-world environments.

Acknowledgements

The completion of this thesis would not have been possible without the guidance of my advisor, Dr. Md Suruz Miah. Dr. Miah's expertise in the field of mechatronics and robotics has been invaluable to me throughout my research, and his invaluable feedback has been instrumental in shaping this work. I am grateful for his patience and understanding, and I am thankful for the opportunity to work with him. Moreover, thanks to the Bradley University - Detroit Area Pre-College Engineering Program (BU/DAPCEP) directed by Dr. Miah and Dr. Jacqueline Henderson, I was able to gather quality materials and resources to accelerate the implementation part of my research.

Special thanks the Faculty of Electrical Engineering at Bradley University for providing me with the resources and support necessary to complete this work. Dr. Yufeng Lu, Dr. Mohammad Imtiaz, Dr. Aleksander Malinowski have always been understanding for my situation and have been instrumental in helping me navigate the challenges of graduate school. Mr. Christopher Mattus have been helpful in sorting out the logistics of my research, and I am grateful for his assistance. Other faculty members have also been instrumental in my success, and I would like to thank them for their support. I would also like to express my gratitude to the department for providing me with Graduate Assistantship during my study. Additionally, I want to thank Dr. Mohammed Abouheaf from Bowling Green State University for his assistance in presenting my work at the 2024 IEEE International Symposium on Robotic and Sensors Environments (ROSE).

Finally, I would like to thank my family, my colleagues, as well as my friends and peers at Bradley University for their encouragement and camaraderie. Special thanks to Ty Wilkinson, a talented student in the department who provided great expertise and assistance in the mechanical design of the robot.

Table of Contents

List of Tables

List of Figures

Acronyms

CNN convolutional neural network. 8

D-H Denavit-Hartenberg. 30, 41

DDPG deep deterministic policy gradient. 9, 49, 61, 68

DDR differential drive robot. 25–28, 56, 57

DOF degrees of freedom. viii, 25, 29, 30, 32, 33, 36, 40, 49, 53, 61, 65, 71, 72

DRL deep reinforcement learning. 49, 61, 68

IIK iterative inverse kinematics. 39, 41, 42

MCU microcontroller unit. 50, 51, 56

ML machine learning. 5, 6, 9, 15, 33, 48, 61

MRS multi-robot system. 2–5, 15–20, 23–25, 35, 67

ODR omnidirectional drive robot. 25, 27, 28, 44

RL reinforcement learning. 6, 8, 9, 33

SBC single-board computer. 50, 51, 56

SLAM simultaneous localization and mapping. 14, 54

Chapter 1

Introduction

Technological advancements in the areas of computer, electrical, and mechanical engineering have led to bridge the gap between humans and robots significantly, fostering closer collaboration. This progress has spurred the widespread adoption of collaborative robots (cobots) across various industries. Cobots, designed to assist humans or to operate in collaboration with human operators in task completion within the same workspace, represent a paradigm shift from the bulky, insensate, and unintelligent robots of the past [1].

The rationale of human-robot collaboration lies in strengths and weaknesses of conventional robots and humans respectively. Robots can continuously execute pre-programmed tasks with high level of accuracy, speed and repeatability, but lack in soft skills of humans such as flexibility, decision-making and adaptability [2], [3]. With human-in-the-loop design, cobots are purposed to improve work performance and quality of humans by matching machines strengths with human soft skills [4]. Additional benefits of cobots include reducing or aiding jobs that are otherwise "dirty, dangerous or dull", and making the robot "less technical and more intuitive" for everyone [2]. A comparison of traditional robots and collaborative robots is shown in Table 1.1.

Table 1.1: Comparison of traditional robots and collaborative robots

Features	**Traditional Robots**	**Collaborative Robots**
Workspaces	Isolated	Shared (human-in-the-loop)
Controls	Tele-op (remote control), or hard programming	Soft automation by Human Robot Interaction (HRI)
Tasks	Repeatable tasks, rarely changed	Frequent task changes

1.1 Motivation

Among the main applications in the field of industrial robotics, the task of handling materials using a group of autonomous cobots is quite challenging due to various constraints and requirements pertaining to autonomy. The deployment of industrial cobots such as [5] has been impactful in manufacturing and assembly settings, making them part of the so-called "Industrial Revolution 4.0" [6]. In these settings, cobots often execute tasks in a structured environment semi-autonomously via demonstration, or autonomously with machine vision and perception [7], [8]. One of the common tasks in these settings is the transportation of materials from one point to another. The materials can be of various types, such as parts, tools, assemblies, or even delicate objects like glass sheets. As opposed to robots deployed for specific material handling, a system of multiple robots are more flexible in handling various types of materials, more adaptable to changes in the environment, and easily replaceable in case of failure [5]. To achieve this, the robots should be capable of navigating the environment, avoiding obstacles, and planning collision-free paths to goal locations, while coordinating with other robots to transport objects. As a result, the design and implementation of a multi-robot system (MRS) for cooperative object transportation tasks is a challenging and realistic research problem.

1.2 Thesis Objective and Problem Statement

In this thesis, we focus on the design and implementation of a MRS for cooperative object transportation tasks. We use wheeled mobile robots equipped with robotic arms as individual agents. While the system is primarily validated with a group of up to three robots, the design is scalable to accommodate more. The robots are required to work collaboratively, sharing information and coordinating their actions to achieve tasks efficiently.

Consider a set of N autonomous cobots $\mathcal{R} = \{R_1, R_2, \cdots, R_N\}$, working together to transport an object from a start (initial) position to a designated goal. For the i-th robot $R_i \in \mathcal{R}$, the objective is to design control inputs $\boldsymbol{u}_i(t)$ for each robot such that they cooperatively transport the object to the goal position. The motion of the robots is constrained by the requirement that they maintain a stable formation in order to avoid collisions and to ensure the object is transported safely. The formation error to be minimized is given by

$$\boldsymbol{e}(t) = \sum_{R_i, R_j \in \mathcal{R}} \|\boldsymbol{q}_i(t) - \boldsymbol{q}_j(t) - \boldsymbol{r}_{ij}\|^2, \tag{1.1}$$

where $\boldsymbol{q}_i(t)$ and $\boldsymbol{q}_j(t)$ are the positions of robots i and j at time t, respectively, and $\boldsymbol{r}_{ij}$ is the desired relative position between robots i and j. The control inputs $\boldsymbol{u}_i(t)$ are therefore designed to minimize the formation error over the course of the task. The research also addresses the issue of intermittent communication losses between the robots, which may occur due to environmental factors or hardware failures.

1.3 Significance of the Study

To achieve said objectives, the research employs graph theory to model the communication network among the robots, utilizing the graph Laplacian matrix to represent connectivity. Robot dynamics and kinematics are modeled to design the control system for the manipulator and the navigation system for the mobile base. The communication system is designed to enable the robots to share information and coordinate their actions, while

resilient to intermittent communication losses. The system is validated in a simulation environment and implemented on real robots to evaluate its performance in various scenarios. The simulation work is presented in a research conference, and the corresponding paper is submitted for publication [9].

The main contributions of this thesis feature (1) a complete package of autonomous MRS for cooperative object transportation tasks, including the design of the control, navigation, and communication systems; (2) a decentralized coordination strategy based on graph Laplacian to model connectivity and interactions among the robots; (3) a set of carefully chosen technologies to implement the control and communication systems for the robots; (4) validation of the system in simulation and real-world environments, demonstrating its performance and robustness in various scenarios.

1.4 Thesis Outline

The remainder of this thesis is organized as follows.

- Chapter 2 provides a comprehensive review of the literature on MRSs, cooperative object transportation, and path planning algorithms.
- Chapter 3 represents the multi-robot system with graph Laplacian, and discusses the graph-based coordination strategies.
- Chapter 4 shows the kinematic and dynamic models of the robots employed in the thesis work, which facilitate the design of the control and navigation systems.
- This system is evaluated in simulation environment in Chapter 5, and with real robots in Chapter 6, where the respective results are discussed.
- Finally, Chapter 7 concludes the thesis and provides directions for future work.

Chapter 2

Literature Review

To design and implement an effective MRS for cooperative transportation tasks, it is essential to understand the latest developments in key areas such as manipulator control, path planning, and multi-robot cooperative transportation. In this chapter, we review the current state of the art in these areas, focusing on the latest research and developments that are shaping the future of robotics. This literature review aims to provide a comprehensive overview of these key areas, highlighting the latest research and developments that are shaping the future of robotics.

The outline of this Chapter is as follows. Section 2.1 explores manipulator control methods, examining how machine learning (ML) techniques are enhancing the precision and adaptability of robotic manipulators. Section 2.2 delves into state-of-the-art path planning algorithms, discussing various approaches that enable robots to navigate complex environments efficiently and safely. Finally, Section 2.3 surveys multi-robot cooperative transportation studies, focusing on strategies and algorithms that facilitate effective collaboration among multiple robots for transportation tasks.

2.1 Manipulator Control Using Machine Learning

The control of robotic manipulators is a well-studied problem in robotics, with a variety of methods available for different applications. In industrial settings, manipulators are often controlled using traditional methods such as PID controllers, which are simple and effective for many applications. However, these methods are not always suitable for more complex tasks, such as object manipulation or cooperative transportation. In recent years, there has been a growing interest in using ML techniques to control robotic manipulators, due to their ability to learn complex control policies from data. Hua *et al.* highlighted deep reinforcement learning and imitation learning as two promising approaches for learning control policies for robotic manipulators [10].

Reinforcement learning (RL) is a type of machine learning paradigm alongside supervised learning, unsupervised learning, to name a few. It is different from the other ones in not requiring explicit labeling of data. Instead, it uses a reward value as feedback for solving optimization problems. This optimization is based on the Markov decision process [11]. The process models the decision-making framework at discrete time, illustrated in Fig. 2.1. At time instant t, the agent is in the state s_t, so it chooses action a_t by the policy particular to the state, such that in the next iteration $t+1$, the environment responds to the action by generating reward $r_{t|t+1}$ and next state s_{t+1}. The RL algorithm is supposed to use these variables to update policy for optimal reward, mathematically modelled by a value function defined as

$$Q_\pi(s,a) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t r_t \middle| s_t = s, a_t = a \right], \tag{2.1}$$

where $\mathbb{E}_\pi[\cdot]$ denotes the expected value for a certain policy π, $\gamma \in (0,1)$ is the discounted factor weighing the future reward. The value function $Q(s,a)$ estimates *how good* it is for the agent to perform a certain action (a) in a certain state (s). The study of reinforcement learning algorithms mainly focuses on developing different types of agents capable of mapping its state into action for optimal reward in different ways. As such, cobots can leverage the RL framework, in combination of deep learning, to iteratively train and improve the handling of the target task over time.

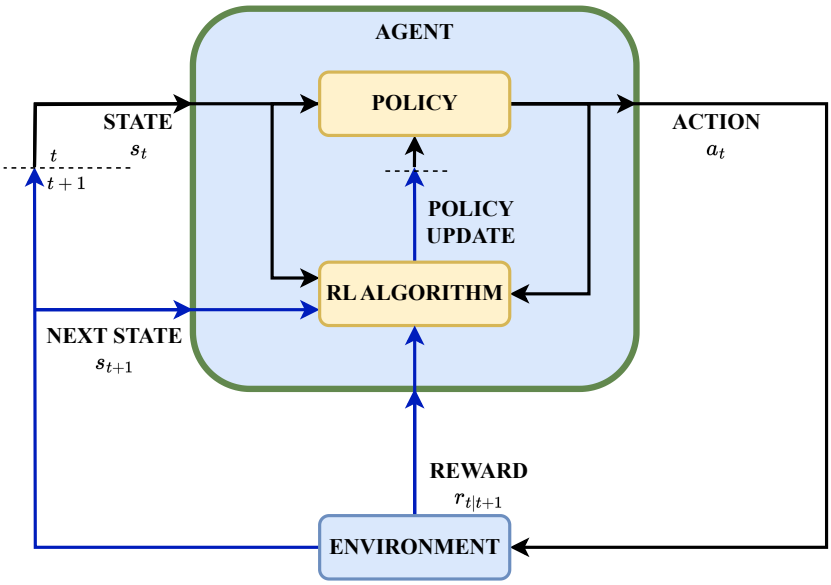


Figure 2.1: Reinforcement learning illustration [12]

SARSA, state-action-reward-state-action is a straightforward example of an on-policy algorithm. Its name is taken from the discrete sequence of events during the learning process, *i.e.*, $s_t \rightarrow a_t \rightarrow r_t \rightarrow s_{t+1} \rightarrow a_{t+1}$ [11]. Here on-policy means the agent evaluates or improves the policy it has been using to make decisions. In contrast, an off-policy agent evaluates or improves a policy different from that used to generate actions. The most basic off-policy algorithm is Q-learning agent, which requires a Q-table that keeps and updates a lookup table of Q-values for all possible states and actions over the learning process, such that the optimal action can be found by [13]

$$a^*(s) = \underset{a}{\text{argmax}} \ Q(s, a). \tag{2.2}$$

This makes it impractical when the state-space and action-space is large, which is common for robots. As a result, deep Q-network agent was developed to resolve this issue by having deep neural network approximate the Q-table as opposed to exhaustively listing it [14], [15]. Deep Q-network works based on the Bellman optimality equation, which states the optimal value function of state-action pair (s, a) is the expected reward r, plus the maximum of

"expected discounted return" for any future state-action pair (s', a'):

$$Q^*(s,a) = \mathbb{E}\Big[r + \gamma \max_{a'} Q^*(s', a')\Big]. \tag{2.3}$$

For a deep neural network to estimate the optimal action-value function, *i.e.*, $Q(s,a;\theta) \approx Q^*(s,a)$, it has to be trained to minimize the mean-squared Bellman error at iteration i

$$L_i(\theta_i) = \mathbb{E}\left[\Big(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s,a;\theta_i)\Big)^2\right], \tag{2.4}$$

where θ_i is the set of neural network weight vectors at the ith iteration. This leads to the success of many research problems addressed by cobots in different applications. A convolutional neural network (CNN) was pre-trained to transfer RGB plus depth image into the Q-table leveraged for RL process in [16]. Similarly, Ghadirzadeh *et al.* trained an unsupervised deep CNN to encode a full-body motion data captured from a tracking suit, then used the reconstructed motion representation as the input to a recurrent neural network that approximates the Q-function [17].

The aforementioned algorithms share a similarity that they are based on value iteration methods for determining actions for the cobots. Value-based methods ultimately map a discrete action space where the most optimal policy may not be approachable. On the other hand, policy-based algorithms such as policy gradient agent explicitly build a representation of a policy by parameterizing, such that it can be deterministic and optimized [11]. Mathematically, let π_θ denote a policy with parameters θ, the objective function for a policy gradient agent is given by

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} \gamma^t r_{t+1}\right]. \tag{2.5}$$

Policy parameter can be updated with stochastic gradient ascent with the update rule

$$\theta \leftarrow \theta + \alpha \boldsymbol{\nabla}_\theta J(\theta), \tag{2.6}$$

where α is learning rate, and $\boldsymbol{\nabla}_\theta J(\theta)$ is the gradient of the objective function, which can be calculated with

$$\boldsymbol{\nabla}_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\boldsymbol{\nabla}_\theta \log \pi_\theta \sum_{t=0}^{T-1} r_{t+1}\right]. \tag{2.7}$$

The disadvantage of policy-based agents, however, is its poor efficiency in high variance policies and tendency to fall into local optimum. Therefore, actor-critic approaches, such as deep deterministic policy gradient (DDPG) method, were later proposed to combine the merits of both value- and policy-based algorithms by synchronously or asynchronously train two networks over the learning process [18]. DDPG agents were used to encode task and safety requirements into the learning process of industrial cobots in [19], [20]. Zhang *et al.* employed DDPG as well for the purpose of subtask allocation between human and robots in assembly jobs [21]. There are other actor-critic agents adopted for cobot applications. For example, Soft Actor-Critic algorithm was implemented to solve a human-robot collaborative maze game within 30 minutes of real-world play session [22], [23].

Another way to improve the learning process is by modelling the environment of the agent, or ultimately the probability model of the states. This is called model-based RL, and is heavily dependent on the context of the learning as well as available tools for the programmers. As an example, to make the collaborative manipulator interact with human more friendly, Roveda *et al.* let a RL agent preemptively generate the impedance control parameters based on a pre-trained neural network that models the human-robot dynamics [24]. This is a good example where RL does not always need to be raw input/output end-to-end. Smart design of architecture with appropriate choice of models can improve the outcome of learning as well.

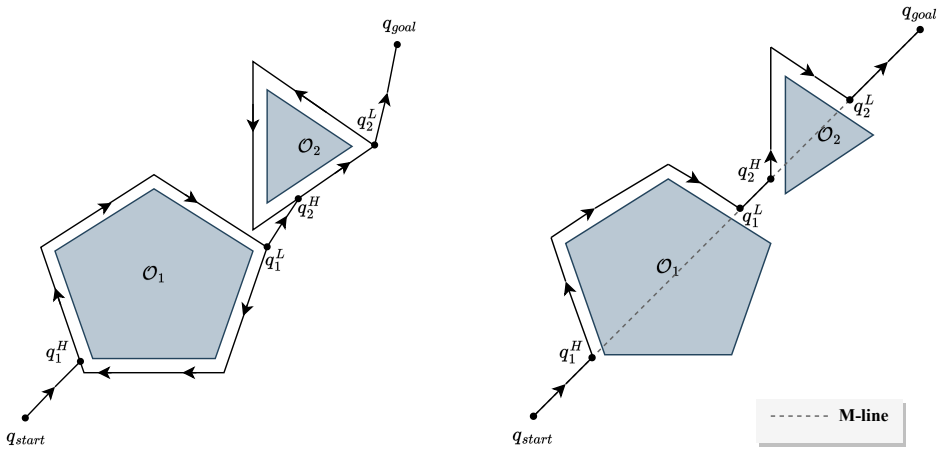
In summary, the use of ML techniques for controlling robotic manipulators has shown great promise in recent years, enabling robots to learn complex control policies from data and adapt to changing environments.

2.2 Path Planning Algorithms

When robot navigate in a complex and dynamic environment, path planning determines an optimal and feasible path from a start location to the goal, while obstacle avoidance ensures that the robot does not collide with any obstacles in its path. Here we discuss a few typical path planning and obstacle avoidance algorithms. The choice of using a particular simple or complex one depends on the navigation task.

The Bug Algorithm is a classic and intuitive method for path planning and obstacle avoidance. It is based on the behavior of a bug moving towards its destination while circumnavigating obstacles in its path. The position of the robot where it first encounters the obstacle is called *hit-point*, after which the robot starts to follow the contour of the obstacle until a certain *leave-point*. The determination of the hit-point, circumnavigation path as well as the leave-point are the key to the effectiveness of bug algorithms. The original work proposed two versions of the algorithm, Bug1 and Bug2 [25]. Bug1 lets the robot circumvents the obstacle completely, then sets the leave-point to where it is the closest to the goal. Bug2, on the contrary, establishes an M-line from the start straight to the goal, so that the robot always follows the M-line when not circumventing, and during circumventing, it leaves the obstacle whenever it touches the M-line. Bug1 and Bug2 are illustrated in Fig. 2.2, where $q_{\text{start}}, q_{\text{goal}}, q^H, q^L$ represent the start, goal, hit and leave point, respectively, and $\mathcal{O}_1$ and $\mathcal{O}_2$ are two obstacles in the way of the robot path. Following these, improvements to the bug algorithms were developed, such as Ibug and TangentBug [26], [27]. A comprehensive analysis of different kinds of bug algorithms is summarized in [28]. Bug algorithms have the advantage of not requiring a priori knowledge of the environment by sacrificing optimality and is therefore used in small-scale robot navigation problems.

When the robot working environment is known, many path finding algorithms typically take a graph-based searching approach. An occupancy grid map is particularly useful in this case, since the graph can directly build on the grid by treating each "vacant" cell in the grid as a node. Nodes are connected to neighboring nodes with weighted edges. The path from the start to the end node with the lowest total edge values along the path is therefore the optimal path. Algorithms with this approach usually study the efficiency of



(a) (b)

Figure 2.2: (a) bug1 and (b) bug2

the weight function of the edges. For example, Dijkstra's algorithm lets the weight be the distance between the nodes. The commonly used A* search algorithm uses a cost function as the weight

$$f(n) = g(n) + h(n), \tag{2.8}$$

where $g(n)$ is the actual cost from initial state to current node n, $h(n)$ is a heuristic function tunable to represent estimated cost from n to the goal [29]. As a demonstration, Fig. 2.3 shows a hex grid map where obstacles are in black, while free unexplored spaces are in gray. Green, blue, and red nodes are candidates labeled with $g(n) + h(n)$ where $g(n)$ is defined as Manhattan distance from start to node n, and $h(n)$ is Manhattan distance from node n to goal. The A* algorithm uses Manhattan distance in both cost and heuristic function to evaluate neighboring nodes in the hex grid map, and eventually discovers the path from start to goal consisting cells shown in green. Blue cells could be potential candidates, but they eventually lead to higher-weighted red cells, which leads to backtracking and finding alternative paths. D* and D* Lite are improvements of A* in dynamic edge weights at algorithm runtime [30], [31].

Artificial potential field path planning treats the robot as a particle moving in a poten-

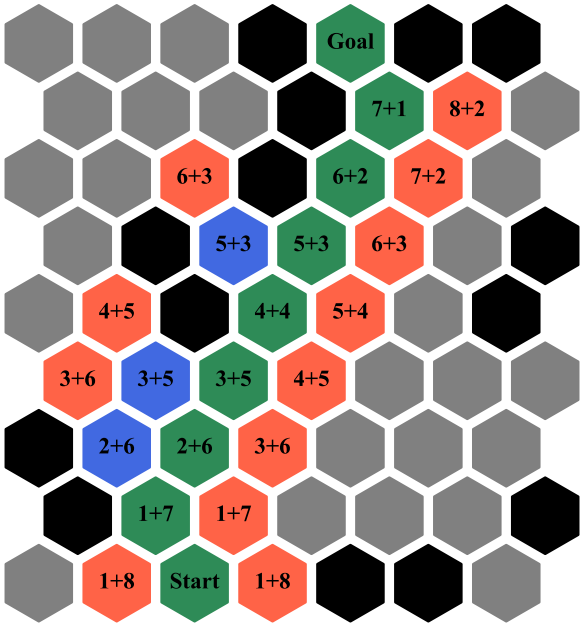


Figure 2.3: Illustration of A* path finding in a hex grid map

tial field, where the goal location exerts an attractive force on the robot and the obstacles exert a repulsive force. The potential field U is typically composed of an attractive potential and a repulsive potential, contributed by the goal and obstacles in the map, respectively in (2.9) to (2.11).

$$U = U_{\text{att}} + U_{\text{rep}} \tag{2.9}$$

$$U_{\text{att}} = \frac{k_{\text{att}}}{2}(d_{\text{goal}})^2 \tag{2.10}$$

$$U_{\text{rep}} = \frac{k_{\text{rep}}}{2}\left(\frac{1}{d_{\text{goal}}} - \frac{1}{p_0}\right)^2 \tag{2.11}$$

In these equations, d_{goal} is the distance from the robot to the goal, p_0 is the distance threshold where the repulsive potential starts to take effect, and k_{att} and k_{rep} are the attractive and repulsive potential coefficients, respectively. The robot then moves in the direction of the negative gradient of the potential field, which leads it towards the goal while

avoiding obstacles. Unlike graph-based path searching, which works with discrete map representations, the artificial potential field method transforms the map into a continuous space. This approach requires more processing power but allows for greater optimization in the resulting path. Past studies have employed different tools in this optimization problem, such as reinforcement learning, simulated annealing, and deterministic annealing [32]–[34]. Other studies attempted to fight the biggest weakness of artificial potential field – the local minima problem – by improving the field generation itself. For example, Vadakkepat *et al.* used evolutionary algorithms to smoothen the optimized path [35]. Bounini *et al.* combined A* searching and artificial potential field to not only avoid local minima problem, but also reduce the processing cost of the path searching [36].

Path planning, as one of the fundamental problems in navigation for robotic systems, has been extensively studied and applied in various applications.

2.3 Multi-Robot Cooperative Transportation Studies

Object transportation with cooperative robots is one of the practical problems often studied in the robotics field. Inoue *et al.* conducted a rigid body force analysis between up to three mobile robot manipulators and came up with a decentralized control method that requires minimal intercommunication [37]. Dong *et al.* divided the cooperative manipulator control problem into two symmetrical unconstrained subsystems disconnected at the payload center and applied the payload constraints separately with the Udwadia-Kalaba method [38]. This approach was verified by having a similar twin robotic system hold the payload to track a predefined trajectory in a simulation. In Inglett *et al.*'s work, the task was divided into three phases, namely contact, re-orientation, and cooperative transportation, but their experiment assumes the payload has wedge-shaped slots where the robot can easily lock onto [39]. When the payload is a delicate and heavy object such as a sheet of glass, assumptions like this do not apply, which is why Wu *et al.* leveraged two humanoid robots with the payload held in their hands [40]. In their experiment, the robots practice the leader-follower framework, where the leader moves by the user command, and the follower

adjusted its position with a zero-moment point (ZMP) preview controller to maintain the dynamical balance of the payload [41]. A similar study with humanoid also implemented a ZMP controller, but instead of the leader-follower approach, they embedded a local path planner into individual robots, so that robots can autonomously navigate to a target position [42].

In terms of navigation, Rioux *et al.* presented a realistic and well-thought path planner for two humanoid robots to cooperatively transport an object to a target with a depth camera of RGB mounted on their head [43]. Specifically, they encoded the poses of the two robots plus the payload pose into a single node, and executed a graph-based Anytime Repairing A* (ARA*) search algorithm in the 2D occupancy map generated with simultaneous localization and mapping (SLAM) [44]. In another simulation research that employed KUKA YouBots, a two-layer path planner was designed where the global, centralized layer executes a novel sampling-based search algorithm called optimally-connected random tree, whereas the local, decentralized layer only considers obstacle avoidance [45]. Another study conducted a simulation in a 2D polygonal environment, introducing a centralized multi-robot path planner for the cooperative transportation of a payload of arbitrary shape. The path planner is capable of instructing the robots to maneuver around obstacles while maintaining the payload's balance [46]. Compared to the two aforementioned research works, studies that focus on navigation of object transportation often simplify the task into "stick carry applications." Within this scope, the problem is often solved with meta-heuristic methods such as particle swarm optimization, bee colony optimization, evolutionary algorithm, and the likes [47]–[49].

Just like humanoid robots are used in place of simply robot manipulators, multi-robot object transportation problem can take on different other forms. For example, when the payload can be easily relocated by simply pushing and pulling, a leader-follower control can be used where the leader robot pushes and the follower robots pull by force/moment sensory data from the payload [50]. Pi *et al.* designed a transportation robot specifically for underground applications, where a different set of kinematics would be considered [51]. Lippi *et al.* even proposed a set of heterogeneous transporter robots that included ground and aerial vehicles cooperatively involved in the transportation task [52].

2.4 Summary

In this chapter, we have reviewed the current state of the art in manipulator control, path planning, and multi-robot cooperative transportation. We have discussed how ML techniques are enhancing the precision and adaptability of robotic manipulators, enabling them to learn complex control policies from data. We have also explored various path planning algorithms, including bug algorithms, graph-based searching, and artificial potential fields, which enable robots to navigate complex environments efficiently and safely. Finally, we have surveyed multi-robot cooperative transportation studies, focusing on strategies and algorithms that facilitate effective collaboration among multiple robots for transportation tasks. The insights gained from these studies will inform the design and implementation of the MRS for cooperative transportation tasks in this thesis.

Chapter 3

Graph Theoretic Model

The ultimate goal of MRS is to achieve coordinated behavior among individual robots to accomplish complex tasks. Decentralized coordination is a common approach to achieve this goal, where each robot makes decisions based on local information and interactions with its neighbors. The coordination of robots in a decentralized manner requires a robust communication network that enables the exchange of information among the robots. The network topology plays a crucial role in determining the performance of the MRS, as it affects the convergence rate, stability, and robustness of the system.

The network structure of a MRS can be represented by a graph, where the nodes correspond to robots and the edges denote the connections between them. The system's behavior is influenced by both the interactions among the nodes and the network topology. Algebraic graph theory offers tools to derive useful characteristics of the graph. In the context of MRS, these characteristics provide valuable insights into the system's overall performance. This chapter introduces the concept of graph Laplacian and its application in modeling and analyzing MRS. We present the Laplacian matrix and its properties in Section 3.1, followed by its application in cobot coordination in Section 3.2. Last but not least, we analyze three different network topologies for a system of three robots to illustrate the importance of graph Laplacian in MRS in Section 3.3.

3.1 Preliminaries

Consider a team of N autonomous cobots represented by the set of nodes $\mathcal{V} = \{1, 2, \ldots, N\}$. The topology of communication links from one robot to another one is described by a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Here, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of directed edges, where an edge $(i, j) \in \mathcal{E}$ indicates that robot i forwards information to robot j. For i-th robot, the set of neighbors is denoted by $\mathcal{N}_i = \{j : (j, i) \in \mathcal{E}; j = 1, 2, \ldots, N\}$.

The network graph can be mathematically described using the *adjacency matrix* $\mathbf{A} \in \mathbb{R}^{N \times N}$, where each element A_{ij} is defined as:

$$A_{ij} = \begin{cases} 1 & \text{if } (i, j) \in \mathcal{E}, \\ 0 & \text{otherwise.} \end{cases} \tag{3.1}$$

The second matrix, called the *degree matrix*, captures information about the connectivity of individual nodes. It is a diagonal matrix $\mathbf{D} \in \mathbb{R}^{N \times N}$ given by:

$$\mathbf{D} = \text{diag}(d_1, d_2, \ldots, d_N), \tag{3.2}$$

where d_i is the degree of node i. For a directed graph, the degree matrix is an out-degree matrix when d_i represents the number of outgoing edges from node i, and an in-degree matrix when d_i represents the number of incoming edges to node i [53]. For a MRS, the out-degree matrix is more interesting as it models a system where each robot influences others by broadcasting its state.

The *Laplacian matrix* $\mathbf{L} \in \mathbb{R}^{N \times N}$ is defined as:

$$\mathbf{L} = \mathbf{D} - \mathbf{A}, \tag{3.3}$$

where each element L_{ij} is given by:

$$L_{ij} = \begin{cases} d_i & \text{if } i = j, \\ -1 & \text{if } (i,j) \in \mathcal{E} \text{ and } i \neq j, \\ 0 & \text{otherwise.} \end{cases} \tag{3.4}$$

The Laplacian matrix is a key tool in graph theory and has several important properties that are useful for analyzing the connectivity and dynamics of a network [54].

3.2 Application in Cobot Coordination

When the message traffic between robots in MRS is modeled as a graph, the Laplacian matrix can be used to analyze the communication dynamics. One of the important question in this is how fast the robots can reach a consensus state, where all robots share the same information in real-time, including their positions, velocities, task status and so on. The process in which the robots reach a consensus state is also known as convergence within the network. For a MRS executing object transportation tasks, the convergence rate is crucial for efficient and coordinated movement, because otherwise the robots may collide or fail to reach the target in time.

One way to analyze the convergence behavior through Laplacian matrix is its eigenvalues. The eigenvalues of $\mathbf{L}$, denoted by $\lambda_1, \lambda_2, \ldots, \lambda_N$, are solutions to the equation:

$$\mathbf{L}\boldsymbol{v} = \lambda \boldsymbol{v}, \tag{3.5}$$

where $\boldsymbol{v}$ is the eigenvector corresponding to the eigenvalue λ. The eigenvalues have the following properties:

- **Non-negativity.** Since the Laplacian matrix is positive semi-definite, all its eigenvalues are non-negative.

- **Zero eigenvalue.** For a connected graph, there is exactly one zero eigenvalue ($\lambda_1 = 0$).
- **Algebraic connectivity.** The second smallest eigenvalue, λ_2, is known as the algebraic connectivity. It measures the connectivity strength of the graph and is used to determine the convergence rate of the consensus algorithm. A higher algebraic connectivity implies faster convergence, as well as higher resilience to disturbances and delays.
- **Spectral gap.** The spectral gap is the difference between the two largest eigenvalues of the Laplacian matrix. A higher spectral gap indicates better synchronizability in the dynamic network, *i.e.* information flows more uniformly over time [55]. The spectral gap belongs to a broader concept of spectral graph theory, which was not explored further in this work.

There are other properties of the Laplacian matrix that can be used to analyze the network, such as the Fiedler vector, but the eigenvalues are the most commonly used for stability and convergence analysis.

3.3 Analysis of Topologies

To illustrate the application of graph Laplacian in MRS, we consider four different network topologies for a system of three robots, A, B, and C. The adjacency matrix, degree matrix, and Laplacian matrix for each topology are presented, along with the corresponding eigenvalues. The analysis of these eigenvalues provides insights into the convergence behavior and stability of the system.

3.3.1 Fully Connected Topology

The first topology is a fully connected network, where there are direct connections between all robots, as shown in Fig. 3.1. The adjacency matrix, degree matrix, and Laplacian matrix

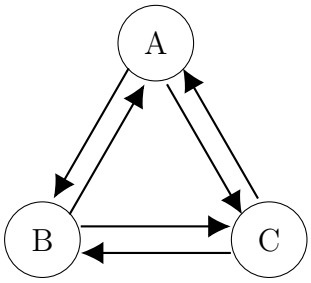


Figure 3.1: Fully Connected Topology

for this topology are given by:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}, \quad \mathbf{D} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}, \quad \mathbf{L} = \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix}. \tag{3.6}$$

The eigenvalues of the Laplacian matrix are $\lambda_1 = 0$, $\lambda_2 = 3$, and $\lambda_3 = 3$. The network represented by this topology has the highest redundancy and connectivity possible for a three-robot system. However, it also results in unnecessarily large communication throughput and computational complexity, which can be inefficient for real-time coordination tasks. As a result, the fully connected topology is not optimal for most MRS applications, but we will be using it as an "ideal" convergence rate for comparison with other topologies.

3.3.2 Cyclic Topology

In the cyclic topology, illustrated in Fig. 3.2, each robot is connected to the next one, forming a closed loop. The adjacency matrix $\mathbf{A}$, degree matrix $\mathbf{A}$, and Laplacian matrix $\mathbf{L}$ for this topology:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}, \quad \mathbf{D} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{L} = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ -1 & 0 & 1 \end{bmatrix}. \tag{3.7}$$

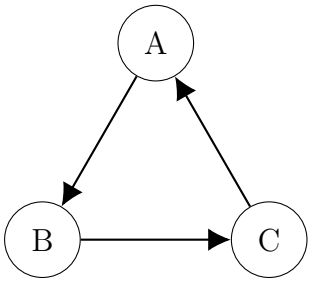


Figure 3.2: Cyclic Topology

The eigenvalues of the Laplacian matrix are $\lambda_1 = 0$, $\lambda_2 = \frac{3+\sqrt{3}i}{2}$, and $\lambda_3 = \frac{3-\sqrt{3}i}{2}$. The non-zero eigenvalues λ_2 and λ_3 are complex conjugates, indicating that the system will oscillate before reaching a consensus state. The cyclic topology is not optimal for convergence, as the oscillations can lead to instability and inefficiency in task execution.

3.3.3 Cyclic with Back Link

The stability of the cyclic topology can be improved by adding a back link from the last robot to the first one, as shown in Fig. 3.3. The back link provides redundant connectivity and improve information flow in the network. The adjacency matrix, degree matrix, and

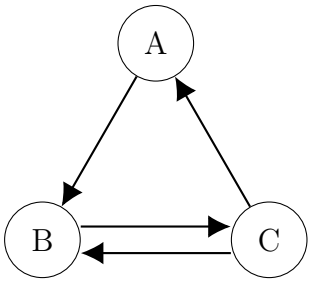


Figure 3.3: Cyclic with Back Link

Laplacian matrix for this topology are:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}, \quad \mathbf{D} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}, \quad \mathbf{L} = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ -1 & -1 & 2 \end{bmatrix}. \tag{3.8}$$

The eigenvalues of this topology are $\lambda_1 = 0$ and $\lambda_2 = 2$. The algebraic connectivity is 2, indicating a non-oscillating convergence at a reasonable rate, which is about 67% of the fully connected topology. However, the lack diverse eigenvalues indicate a low degree of redundancy in the network. Should a robot fail, the system may become disconnected, leading to a loss of coordination.

3.3.4 Star Topology

In the star topology, robot A is connected to all other robots, forming a central hub that communicates back and forth with the peripheral robots B and C, as shown in Fig. 3.4. Equation (3.9) presents the adjacency matrix, degree matrix, and Laplacian matrix for

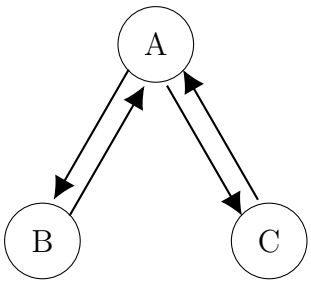


Figure 3.4: Star Topology

the star topology. The eigenvalues of the Laplacian matrix are $\lambda_1 = 0$, $\lambda_2 = 1$, and $\lambda_3 = 3$. The algebraic connectivity $\lambda_2 = 1$ indicates a slower convergence rate, which is about 33% of the fully connected topology. The star topology is less efficient in terms of convergence, as the central hub (robot A) can become a bottleneck for information flow. However, its robustness relies solely on one node (robot A), which reduces the points of

failure from three to one. If the star topology is applied to a MRS and the central hub can be identified beforehand, the robustness of the system can be further improved by increasing the reliability of the hub node. Given the hub node is reliable, the star topology provides a high level of redundancy, ensuring robot B (or C) can still operate even if robot C (or B) fails.

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \quad \mathbf{D} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{L} = \begin{bmatrix} 2 & -1 & -1 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix}. \tag{3.9}$$

Table 3.1 summarizes the key characteristics of the aforementioned topologies based on their Laplacian eigenvalues.

Table 3.1: Comparison of Different Topologies

Topology	**Number of Edges**	**Eigenvalues of Laplacian**	**Remarks**
Fully Connected	6	$\{0, 3, 3\}$	Highest redundancy & connectivity, but extremely large throughput
Cyclic	3	$\{0, \frac{3+\sqrt{3}i}{2}, \frac{3-\sqrt{3}i}{2}\}$	Oscillatory convergence
Cyclic with Back Link	4	$\{0, 2\}$	Fast convergence, low redundancy
Star	4	$\{0, 1, 3\}$	Slow convergence, high redundancy

This analysis coincides with our numerical simulations, shown in Fig. 3.5. Three topologies are initialized with $A = 0.3, B = 0.5, C = 0.8$. Over each iteration, their value is averaged with their neighbors and updated. The fully connected network in Fig. 3.5(a) converges to a consensus state at the fastest rate, but we believe it is not practical for real-time coordination due to the high communication overhead. The cyclic topology in Fig. 3.5(b) shows oscillatory behavior, while the cyclic with back link in Fig. 3.5(c) and star topology in Fig. 3.5(d) converge to a consensus state. The cyclic with back link topology converges 1-2 steps faster than the star topology, and with further analysis, we can determine the degree of redundancy and robustness of the system as well.

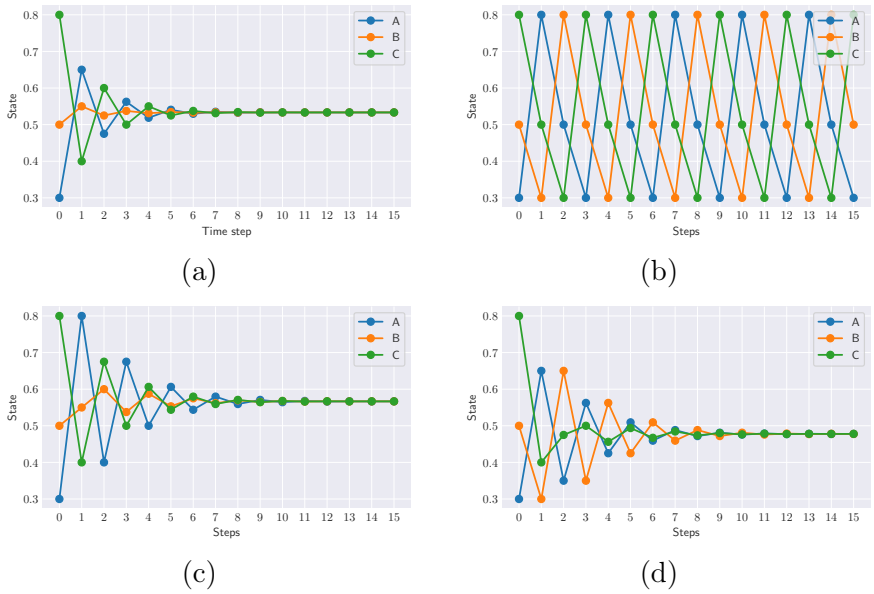


Figure 3.5: Convergence behavior over time for (a) fully-connected, (b) cyclic, (c) cyclic with back link, and (d) star topologies

As a result, in the design of networked cobots, we considered both star topology and cyclic with back link topology to achieve a balance between convergence rate and redundancy.

3.4 Summary

This chapter introduced the concept of graph Laplacian and its application in modeling and analyzing MRS. The Laplacian matrix captures the connectivity and dynamics of the network, enabling the analysis of convergence, stability, and robustness. We demonstrated the application of Laplacian matrix in cobot coordination by analyzing three different network topologies for a system of three robots. The analysis of Laplacian eigenvalues provides valuable insights into the performance of the networked cobots, guiding the design of efficient and robust systems.

Chapter 4

Cobot Kinematics and Dynamics

Apart from studying the MRS as a multi-agent system, it is also essential to understand the kinematics and dynamics of the individual agents. As a complex electrical and mechanical system, the cobot consists of a mobile platform and a manipulator, whose motion is governed by the kinematic and dynamic models. In this chapter, we present the kinematic and dynamic models of the cobot system, which are essential for the control system design.

In Section 4.1, we illustrate the kinematic models of the mobile platform, which can be either a differential drive robot or an omnidirectional drive robot. Section 4.2 presents the kinematic model of the manipulator, which is a 2-DOF or 5-DOF robot arm. Finally, in Section 4.3, the dynamic model of the manipulator is discussed.

4.1 Mobile Platform Kinematics

Two types of wheeled locomotion systems are considered for the mobile platform: differential drive robot (DDR) and omnidirectional drive robot (ODR). Let $\boldsymbol{q} = [x, y, \theta]^\top$ denote the state of the mobile platform, where x and y are the position coordinates on x- and y-axes, respectively, and θ is the orientation angle. The forward kinematic model facilitates

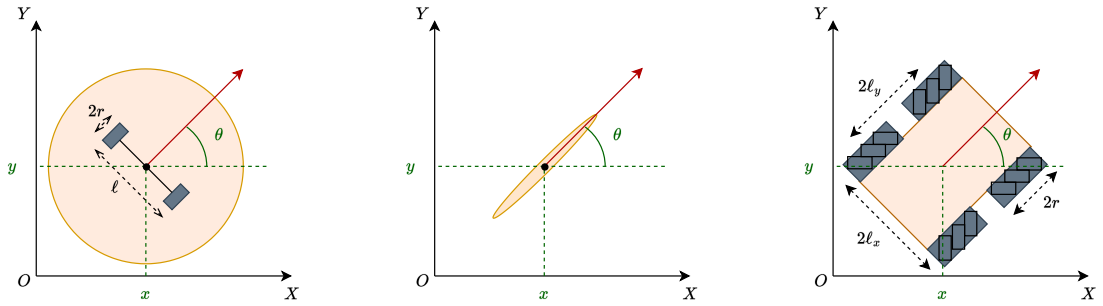
the computation of the robot's state based on the control inputs $\boldsymbol{u} = [u_1, u_2, \ldots, u_n]^\top$:

$$\dot{\boldsymbol{q}} = \mathbf{f}(\boldsymbol{u}). \tag{4.1}$$

Forward kinematic equations can be used to estimate the robot state using dead reckoning methods. The inverse kinematics, on the other hand, calculates the necessary control inputs to achieve a desired state.

$$\boldsymbol{u} = \mathbf{f}^{-1}(\dot{\boldsymbol{q}}). \tag{4.2}$$

A DDR mobile robot is mainly composed of a rigid body, a pair of driving wheels of radius r connected by an axle of length ℓ, and a castor wheel, as shown in Fig. 4.1(a). The



(a) (b) (c)

Figure 4.1: (a) Differential drive robot, (b) unicycle model, and (c) omnidirectional drive robot.

control input vector $\boldsymbol{u} = [u_l, u_r]^\top$ consists of the angular velocities of the left and right wheels, respectively. Since the system is subject to non-slip constraints:

$$\dot{x} \sin\theta - \dot{y} \cos\theta = 0, \tag{4.3}$$

it is often simplified to a unicycle model, as seen in Fig. 4.1(b), whose linear velocity v and

angular velocity ω are given by:

$$v = \sqrt{\dot{x}^2 + \dot{y}^2} = \frac{r}{2}(u_l + u_r), \tag{4.4}$$

$$\omega = \dot{\theta} = \frac{r}{\ell}(u_r - u_l). \tag{4.5}$$

The forward kinematics of the DDR system can therefore be expressed as:

$$\dot{\boldsymbol{q}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}. \tag{4.6}$$

The inverse kinematics using linear and angular velocities is given by:

$$\begin{bmatrix} u_l \\ u_r \end{bmatrix} = \frac{1}{r} \begin{bmatrix} 1 & \ell/2 \\ 1 & -\ell/2 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{4.7}$$

Because of the non-holonomic constraints imposed by Eq. (4.3), controllers that adjust $\dot{\boldsymbol{q}} = [\dot{x}, \dot{y}, \dot{\theta}]^\top$ must circumvent the constraints and therefore are not desirable. Instead, we designed the control method for the DDR robot to achieve a desired set of linear and angular velocities v and ω, which are discussed in later chapters.

The ODR system leverages Mecanum wheels to achieve omnidirectional motion. The Mecanum wheels, also known as Swedish wheels, are constructed with rollers attached at circumference at an angle of typically 45°, such that sideways, diagonal, and rotational motion is possible. Figure 4.1(c) shows the configuration of an ODR system, where r is the wheel radius, ℓ_x and ℓ_y are the distances from the center of the robot to the wheel contact points along the x- and y-axes in the robot frame.

The control input vector $\boldsymbol{u} = [u_{\text{fl}}, u_{\text{fr}}, u_{\text{bl}}, u_{\text{br}}]^\top$ consists of the angular velocities of the front-left, front-right, back-left, and back-right wheels, respectively [56]. The forward

kinematic model of the ODR system:

$$\dot{\boldsymbol{q}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \frac{r}{4} \begin{bmatrix} -1 & 1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ -1/(\ell_x + \ell_y) & 1/(\ell_x + \ell_y) & 1/(\ell_x + \ell_y) & -1/(\ell_x + \ell_y) \end{bmatrix} \begin{bmatrix} u_{\text{fl}} \\ u_{\text{fr}} \\ u_{\text{bl}} \\ u_{\text{br}} \end{bmatrix}. \tag{4.8}$$

Unlike the DDR system, the ODR system is holonomic, meaning it can move in any direction without constraints. The inverse kinematics of the ODR system is given by:

$$\begin{bmatrix} u_{\text{fl}} \\ u_{\text{fr}} \\ u_{\text{bl}} \\ u_{\text{br}} \end{bmatrix} = \frac{1}{r} \begin{bmatrix} -1 & 1 & -(\ell_x + \ell_y) \\ 1 & 1 & (\ell_x + \ell_y) \\ 1 & -1 & -(\ell_x + \ell_y) \\ -1 & -1 & (\ell_x + \ell_y) \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix}. \tag{4.9}$$

4.2 Manipulator Kinematic Model

In this section, we present the kinematics of the robot arm mounted on the mobile platform. The manipulator is assumed to be rigid and consists of links connected by joints. The joints are often characterized with joint angles for revolute joints or joint displacements for prismatic joints. The kinematic model describes the relationship between the joint parameters $\boldsymbol{\phi} = [\phi_1, \phi_2, \ldots, \phi_N]^\top$ and the end-effector position $\boldsymbol{p} = [x, y, z]^\top$.

Furthermore, since the manipulator is mounted on a mobile platform, the global position of the end-effector is offset and rotated by the mobile platform's position $\boldsymbol{p}_b = [x_b, y_b, z_b]^\top$ and orientation θ. The end-effector position in the global frame ($\boldsymbol{p} = [x, y, z]^\top$) and local coordinate frame ($\boldsymbol{p}_a = [x_a, y_a, z_a]^\top$) are related as:

$$\begin{bmatrix} \boldsymbol{p} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}(\theta) & \mathbf{p}_b \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{p_a} \\ 1 \end{bmatrix}, \tag{4.10}$$

where $\mathbf{R}(\theta) \in \mathbb{R}^{4\times4}$ is the rotation matrix from the local frame to the global frame, subject

to the orientation angle θ. This change in coordinate frames applies to all kinds of manipulators, regardless of the number of DOF. Our work relates to a 2-DOF planar manipulator and a 5-DOF articulated robot arm, so we will focus on these two cases.

First we study a 2-DOF manipulator with revolute joints, as shown in Fig. 4.2. The

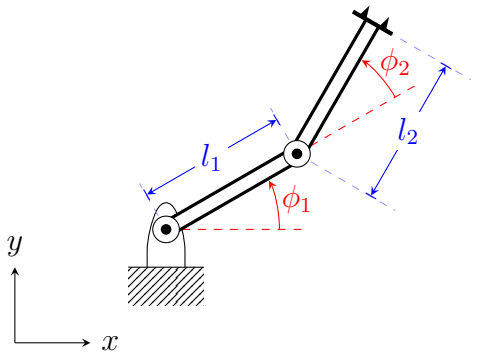


Figure 4.2: 2-DOF R-R manipulator.

joint angles are denoted by ϕ_1 and ϕ_2, and the link lengths are l_1 and l_2. The rotation matrix $\mathbf{R}(\theta)$ from the coordinate frame in the diagram to the global frame:

$$\mathbf{R}(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ \sin\theta & 0 & -\cos\theta \\ 0 & 1 & 0 \end{bmatrix}, \tag{4.11}$$

which is used in Eq. (4.10) to compute the end-effector position in the global frame.

The forward kinematics to computer the end-effector position in the local coordinate frame $\boldsymbol{p}_a$ is given by:

$$\boldsymbol{p}_a = \begin{bmatrix} x_a \\ y_a \end{bmatrix} = \begin{bmatrix} l_1 \cos\phi_1 + l_2 \cos(\phi_1 + \phi_2) \\ l_1 \sin\phi_1 + l_2 \sin(\phi_1 + \phi_2) \end{bmatrix}. \tag{4.12}$$

The inverse kinematics computes the joint angles for a given end-effector position (x_a, y_a):

$$\begin{aligned} \phi_2 &= \arccos\left(\frac{x_a^2 + y_a^2 - l_1^2 - l_2^2}{2 l_1 l_2}\right), \\ \phi_1 &= \arctan\left(\frac{y_a}{x_a}\right) - \arctan\left(\frac{l_2 \sin\phi_2}{l_1 + l_2 \cos\phi_2}\right). \end{aligned} \tag{4.13}$$

In the case of a higher DOF manipulator, the kinematics are more complex and require the use of homogeneous transformation matrices. For this, we consider a 5-DOF manipulator with revolute joints, as shown in Fig. 4.3. The control input vector

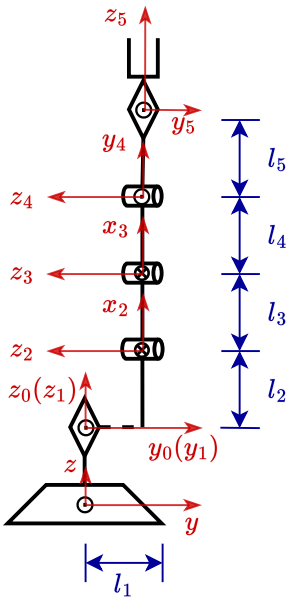


Figure 4.3: 5-DOF manipulator made of revolute joints.

$\boldsymbol{\phi} = [\phi_1, \phi_2, \phi_3, \phi_4, \phi_5]^\top$ consists of the joint angles. Frame $\{5\}$ represents the position and orientation of the end-effector. The Denavit-Hartenberg (D-H) method is used to mathematically describe joints of the articulated robot manipulator [57]. This method represents a spatial kinematic chain with four parameters, namely link length, link twist, link offset and joint angle, which are represented by a, d, α, ϕ, known as *D-H parameters*. The D-H

Table 4.1: KUKA YouBot D-H Parameters [58]

Link i	a_i	d_i	α_i	ϕ_i
1	0	0	0	ϕ_1
2	$\pi/2$	l_1	l_2	$\pi/2+\phi_2$
3	0	l_3	0	ϕ_3
4	0	l_4	0	$-\pi/2+\phi_4$
5	$-\pi/2$	0	l_5	ϕ_5

parameters of the five joints are listed in Table 4.1. They allow for convenient derivation of affine transformation matrices

$$\mathbf{T}_i^{i-1} = \begin{bmatrix} \cos\phi_i & -\sin\phi_i\cos\alpha_i & \sin\phi_i\sin\alpha_i & a_i\cos\phi_i \\ \sin\phi_i & \cos\phi_i\cos\alpha_i & -\cos\phi_i\sin\alpha_i & a_i\sin\phi_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4.14}$$

The notation $(\cdot)_i^j$ refers to the transform from frame $\{j\}$ to $\{i\}$. For example, when $i=3$, (4.14) only supports the transformation from frame $\{2\}$ to $\{3\}$. However, with all five transformations accessible, the transformation from the base coordinate frame $\{0\}$ to the end-effector $\{5\}$ can be formally calculated. This transformation is a function of the joint parameters $\boldsymbol{\phi} = [\phi_1, \phi_2, \cdots, \phi_5]^\top$:

$$\mathbf{T}_5^0(\boldsymbol{\phi}) = \prod_{i=1}^{5} \mathbf{T}_i^{i-1}. \tag{4.15}$$

As a result, the end-effector position relative to the base frame $\boldsymbol{p}_a = [x_a, y_a, z_a]^\top$ can be

extracted from the transformation matrix $\mathbf{T}_5^0$:

$$\begin{bmatrix} x_a \\ y_a \\ z_a \\ 1 \end{bmatrix} = \mathbf{T}_5^0(\boldsymbol{\phi}) \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}. \tag{4.16}$$

Using Eq. (4.10), the global end-effector position $\boldsymbol{p}$ can be computed. The base frame (frame $\{0\}$) in Fig. 4.3 aligns with the mobile platform's frame, so the rotation matrix $\mathbf{R}(\theta)$ is defined as

$$\mathbf{R}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{4.17}$$

Though the goal of inverse kinematics is to solve for a possible set of joint parameters $\boldsymbol{\phi}$ given the end-effector position $\boldsymbol{p}$, for a 5-DOF manipulator, there are multiple solutions. Realistically, the most optimal solution is chosen based on minimal amount of joint movement or energy consumption. A more practical approach in this case is to use iterative inverse kinematics, which is also a control strategy that will be discussed in Chapter 5.

4.3 Manipulator Dynamics

The kinematic equations describe the manipulator motion without considering the forces and torques acting on the system. In contrast, the dynamic model describes the relationship between the joint torques $\boldsymbol{\tau} = [\tau_1, \tau_2, \ldots, \tau_N]$, end-effector acceleration $\ddot{\boldsymbol{p}}$, velocity $\dot{\boldsymbol{p}}$, and position $\boldsymbol{p}$. Using the Euler-Lagrange equation of motion, an analytical model of the manipulator dynamics can be expressed as:

$$\mathbf{M}(\boldsymbol{\phi})\ddot{\boldsymbol{\phi}} + \mathbf{C}(\boldsymbol{\phi}, \dot{\boldsymbol{\phi}})\dot{\boldsymbol{\phi}} + \mathbf{G}(\boldsymbol{\phi}) = \boldsymbol{\tau}, \tag{4.18}$$

where $\mathbf{M}(\boldsymbol{\phi})$ is the inertial matrix, $\mathbf{C}(\boldsymbol{\phi}, \dot{\boldsymbol{\phi}})$ is the Coriolis and centrifugal matrix, $\mathbf{G}(\boldsymbol{\phi})$ is the gravity vector, and $\boldsymbol{\tau}$ is the joint torque vector.

In the case of a 2-DOF manipulator that is assumed to be a system consisting of two masses m_1 and m_2 connected by weightless links, these matrices can be expressed as:

$$\mathbf{M}(\boldsymbol{\phi}) = \begin{bmatrix} a + 2b\cos\phi_2 & b\cos\phi_2 \\ b\cos\phi_2 & b \end{bmatrix}, \quad \mathbf{C}(\boldsymbol{\phi}, \dot{\boldsymbol{\phi}}) = \begin{bmatrix} -c\sin\phi_2\dot{\phi}_2 & -c\sin\phi_2(\dot{\phi}_1 + \dot{\phi}_2) \\ c\sin\phi_2\dot{\phi}_1 & 0 \end{bmatrix} \tag{4.19}$$

where $a = m_1 l_1^2 + m_2(l_1^2 + l_2^2 + 2l_1 l_2 \cos\phi_2)$, $b = m_2 l_1 l_2$, and $c = m_2 l_1 l_2$. The gravity vector $\mathbf{G}(\boldsymbol{\phi})$ is given by:

$$\mathbf{G}(\boldsymbol{\phi}) = \begin{bmatrix} (m_1 + m_2)gl_1\cos\phi_1 + m_2 g l_2 \cos(\phi_1 + \phi_2) \\ m_2 g l_2 \cos(\phi_1 + \phi_2) \end{bmatrix}. \tag{4.20}$$

The analytical model can be used to control the inverse dynamic in an ideal environment, as seen in Fig. 4.4. However, in practice, the model is often inaccurate due to friction,

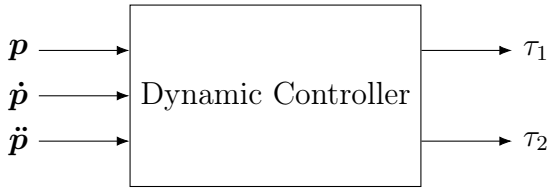


Figure 4.4: Block diagram of dynamic control system.

damping, disturbances and other uncertainties. Furthermore, such inverse dynamic control is reactive, which is not capable of predicting the future state of the system. As a result, a more advanced control strategy is required to achieve better performance in real-world scenarios. One such strategy is the use of ML algorithms. For example, Raina *et al.* proposed a controller based on deep neural network and RL to improve the manipulator control system [59].

4.4 Summary

In this chapter, we presented the kinematic and dynamic models of the cobot system. The kinematic models of the mobile platform and manipulator are essential for the control system design. The dynamic model of the manipulator is used to compute the joint torques required to achieve a desired end-effector position. Over the next two chapters, the control system utilizing these models will be discussed, followed by the simulation and implementation of the system.

Chapter 5

Computer Simulations

In the previous chapters, we presented the mathematical models around MRS, including kinematics, dynamics, and decentralized communication. To achieve a fully functional cooperative multi-cobot system, these theoretical models should be used to develop control algorithms that can be implemented on the physical robots. However, before deploying the algorithms on the physical robots, it is essential to verify the algorithms in a simulated environment.

In this chapter, we present two sets of computer simulations that were conducted to approach the problem of cooperative multi-cobot systems. The first set of simulations was conducted using the KUKA YouBot mobile manipulator in a simulated environment. Its method is presented in Section 5.1, and the results are shown in Section 5.2. The second set of simulations was conducted using a digital twin of the Romi robot, which was used to verify the implementation methods before deploying them on the physical robot.

5.1 Methods Using KUKA YouBot

In an effort to understand the challenges and opportunities on the topic of cooperative multi-cobot systems, we have conducted preliminary experiments in a simulated environment. The problem scenario is shown in Fig. 5.1 and explained as follows. A sheet of

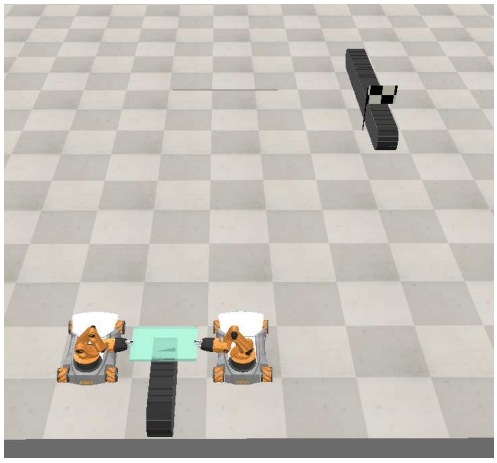


Figure 5.1: Cooperative object transportation using KUKA Youbot in the CoppeliaSim commercial robot simulator.

glass as the delicate object is to be transported as *payload* from the end of one conveyor belt to another one. The robot we selected for this task is the KUKA YouBot, a mobile manipulator with a 5-DOF arm and an omnidirectional driving base [60]. The desktop robot was used as a popular choice for research and educational purposes, so as a result, numerous resources are available for developing and testing algorithms for the robot. Two KUKA YouBot's on each side of the starting conveyor belt are supposed to coordinate with each other to carefully lift, carry and eventually drop the payload onto the target conveyor belt marked by the flag. Dropping the payload on the halfway is not permitted, since that would lead to potential damage to the payload. Given that the gripper is detachable and can be replaced with alternatives offering a range of specifications [60], we assume that the friction between the end-effector and the payload surface is made sufficiently high to overcome slippage during the robot's starting, stopping, and turning motions. The selection of a specific gripper configuration that satisfies this assumption will be determined during the experimental implementation. However, should the two cobots move beyond a certain distance from each other, the payload could still fall due to insufficient support. The proposed solution to the cooperative object transportation problem is divided into three subsystems, namely manipulator control, navigation, and distributed cooperative autonomy. This is illustrated in Fig. 5.2. The manipulator control subsystem is developed for providing ac-

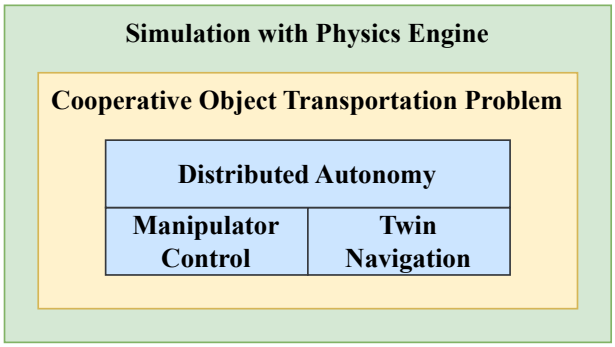


Figure 5.2: Subsystems of cooperative object transportation system used in this work.

tuator commands to the articulated manipulator arm such that the object (payload) to be transported is secured. The actuator commands of the manipulator are determined by the control algorithm which is illustrated later. Twin navigation is another subsystem that involves the locomotion of the robots. This includes the strategy to optimally move from one place to another, while synchronizing the movement to ensure an efficient and safe transportation task. The distributed autonomy subsystem integrates the above two subsystems into a streamlined process executed in an industrial simulation environment. It is distributed in the sense that each robot runs their own task scheduling, and exchanges information with each other in real time. In the following subsections, we provide details of each of these subsystems.

5.1.1 Twin Navigation

The robots practice distributed path planning where each robot has separate navigation goals. Between the two robots, leader-follower formation is applied. For the follower robot, the goal is to stay in formation with the leader, whereas for the leader robot, the goal is to transport the payload to the target location. The robots synchronize navigation information by communicating with each other.

Point-directed navigation drives the leader robot but treats the payload as the main subject in error correction. Let the target position of the payload be presented by an

appropriate point in the XY plane $(x_o^{[\text{ref}]}, y_o^{[\text{ref}]})$ at the start of the conveyor belt. The position error of the payload at an arbitrary point (x_o, y_o) can therefore be denoted with vector

$$\boldsymbol{e}_o = \begin{bmatrix} e_{o,x} \\ e_{o,y} \end{bmatrix} = \begin{bmatrix} x_o^{[\text{ref}]} - x_o \\ y_o^{[\text{ref}]} - y_o \end{bmatrix} \tag{5.1}$$

where $e_{o,x}$ and $e_{o,y}$ are the x- and y-coordinate error of the payload respectively. This can be directly transferred to the error for the leader (robot m), with additional constraint to keep at 0° orientation:

$$\boldsymbol{e}_m = \begin{bmatrix} e_{m,x} \\ e_{m,y} \\ e_{m,\theta} \end{bmatrix} = \begin{bmatrix} e_{o,x} \\ e_{o,y} \\ 0 - \theta_m \end{bmatrix} = \begin{bmatrix} x_o^{[\text{ref}]} - x_o \\ y_o^{[\text{ref}]} - y_o \\ -\theta_m \end{bmatrix} \tag{5.2}$$

Likewise, $e_{m,x}, e_{m,y}, e_{m,\theta}$ are the x-, y-coordinate and orientation error of robot m respectively. Since the Mecanum locomotion mechanism supports independent control in x and y positions, we can implement a variety of controllers to minimize $\boldsymbol{e}_m$.

We use the left robot as leader (robot m) and the right one as follower (robot n), but the choice is arbitrary. In usual cases of leader-follower formation, robot n maintains a fixed distance from robot m, or in another word, minimizes the difference between the target distance and the actual distance, establishing a simple pursuit policy. In the task of carry delicate objects, however, a specific formation needs to be applied: the follower should maintain a fixed perpendicular distance with the leader, while keeping the same orientation as the leader, as illustrated in Fig. 5.3. As presented in Chapter 4, with omnidirectional wheels, linear velocities $\dot{x}, \dot{y}$ as well as angular velocity $\dot{\theta}$ can be dictated with independent controllers. Referring to Fig. 5.3, the geometric errors of these three variables $e_{n,x}, e_{n,y}$ and $e_{n,\theta}$ are calculated with Eq. (5.3), which will be given to the feedback controllers for optimization.

$$\boldsymbol{e}_n = \begin{bmatrix} e_{n,x} \\ e_{n,y} \\ e_{n,\theta} \end{bmatrix} = \begin{bmatrix} x_m + d_{mn}\cos(\theta_m) - x_n \\ y_m + d_{mn}\sin(\theta_m) - y_n \\ \theta_m - \theta_n \end{bmatrix} \tag{5.3}$$

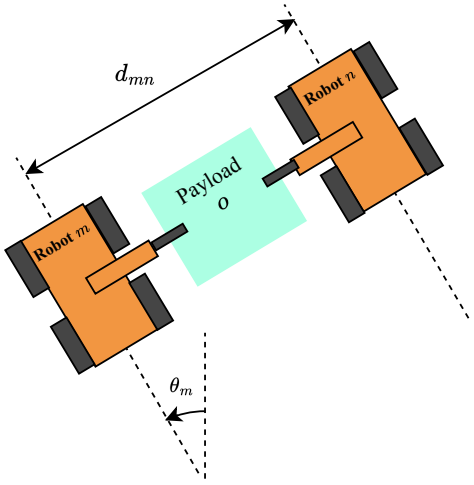


Figure 5.3: Leader-follower formation with special policies that make robot n maintain the same orientation as θ_m, and a perpendicular distance d_{mn} from robot n.

The navigation strategy of the twin robot is summarized in Fig. 5.4. The position of the payload (x_o, y_o) is compared with that of the target $(x_o^{[\text{ref}]}, y_o^{[\text{ref}]})$, which regulates the velocities $(\dot{x}_m, \dot{y}_m, \dot{\theta}_m)$ of the leader robot by three feedback controllers, and in turn, affects its new pose (x_m, y_m, θ_m). On the other hand, the pose of the leader is set as reference for the follower. With three more controllers practicing leader-follower formation, the velocities $(\dot{x}_n, \dot{y}_n, \dot{\theta}_n)$ and new pose (x_n, y_n, θ_n) of the follower is determined. Last but not least, assuming no slippage of the payload when held between the two end effectors due to enough friction, the payload moves in the same trajectory as the two robots.

5.1.2 Manipulator Control

Iterative inverse kinematics (IIK) is the core method to control the manipulator part of each cobot. This method calculates the joint parameters based on the desired and actual end-effector positions. It solves for a dynamic solution that leverages motion differentials

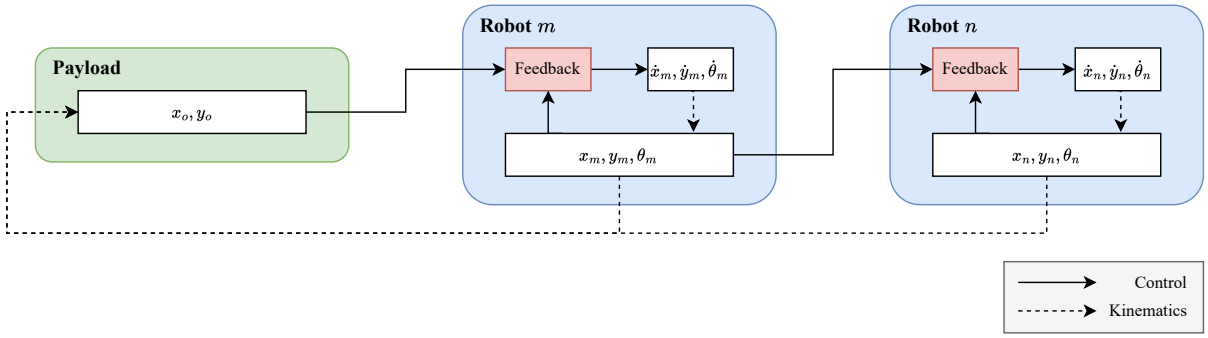


Figure 5.4: Twin robot object transport navigation scheme. Solid arrows are data flow for feedback control, whereas dash arrows are caused by kinematics.

applied iteratively over multiple sampling times. The Jacobian matrix transforms joint differentials to motion differentials of the end effector:

$$\dot{\boldsymbol{p}} = \mathbf{J}\dot{\boldsymbol{\phi}}. \tag{5.4}$$

In Eq. (5.4), $\dot{\boldsymbol{p}}$ represents the time derivative of the end-effector pose $\boldsymbol{p}$. The term $\dot{\boldsymbol{\phi}}$ denotes the first-order derivative of the joint parameters $\boldsymbol{\phi}$, which include the angular velocities for revolute joints or the linear velocities for prismatic joints. As stated in Section 4.2, the 5-DOF manipulator arm mounted on KUKA YouBot are controlled using five revolute joint angles so $\dot{\boldsymbol{\phi}} = [\dot{\phi}_1, \dot{\phi}_2, \ldots, \dot{\phi}_5]^T$. The Jacobian matrix J can be computed column by column, where each column corresponds to the motion differential of the end effector due to the motion of a single joint [61]:

$$\mathbf{J} = \begin{bmatrix} \boldsymbol{J_1} & \boldsymbol{J_2} & \cdots & \boldsymbol{J_5} \end{bmatrix}. \tag{5.5}$$

For the i-th column, $\boldsymbol{J_i}$ is given by

$$\boldsymbol{J_i} = \begin{cases} \begin{bmatrix} \boldsymbol{z}_{i-1}^0 \times (\boldsymbol{o}_n^0 - \boldsymbol{o}_{i-1}^0) \\ \boldsymbol{z}_{i-1}^0 \end{bmatrix} & \text{if } i\text{-th joint is revolute} \\ \begin{bmatrix} \boldsymbol{z}_{i-1}^0 \\ \mathbf{0} \end{bmatrix} & \text{if } i\text{-th joint is prismatic,} \end{cases} \tag{5.6}$$

where $\boldsymbol{z}_{i-1}^0$, $\boldsymbol{o}_n^0$, and $\boldsymbol{o}_{i-1}^0$ can be taken from the affine transformation matrix, which is discussed as part of the kinematic model in Section 4.2. To explain in detail, a generic transformation matrix from the i-th frame to j-th frame $\mathbf{T}_i^j$ can be decomposed into vector components in x-, y-, z-axis of the frame rotation $(\boldsymbol{x}_i^j, \boldsymbol{y}_i^j, \boldsymbol{z}_i^j)$ and the origin $(\boldsymbol{o}_i^j)$ given by

$$\mathbf{T}_i^j = \begin{bmatrix} \boldsymbol{x}_i^j & \boldsymbol{y}_i^j & \boldsymbol{z}_i^j & \boldsymbol{o}_i^j \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{5.7}$$

For KUKA YouBot, only the first case in Eq. (5.6) is used.

For a streamlined process, the method to obtain Jacobian from directly from the D-H parameters of a robot such as Table 4.1 is summarized in Algorithm 5.1. With the Jacobian

Algorithm 5.1 Streamlined algorithm of calculating Jacobian from D-H parameters

Input: D-H parameters for n joints
  for $i = 1, \ldots, n$ **do**
    Calculate $\mathbf{T}_i^{i-1}$ with D-H parameters of i-th link by Eq. (4.14)
    Calculate $\mathbf{T}_i^0$ cumulatively by Eq. (4.15), keep in memory
  Also keep $\mathbf{T}_0^0 = \mathbf{I}_4$ in memory
  Get $\boldsymbol{o}_n^0$ from T_n^0 by Eq. (5.7)
  for $i = 1, \ldots, n$ **do**
    Get $\boldsymbol{z}_{i-1}^0$ from $\mathbf{T}_{i-1}^0$ by Eq. (5.7)
    Get $\boldsymbol{o}_{i-1}^0$ from $\mathbf{T}_{i-1}^0$ by Eq. (5.7)
    Calculate $\boldsymbol{J_i}$ by Eq. (5.6), keep in memory
  Concatenate $\boldsymbol{J_i}$'s for Jacobian matrix J as per Eq. (5.5)

matrix derived, the goal of IIK is solving for the joint parameter update $\Delta\boldsymbol{\phi}$ to complement

the end-effector pose error:

$$\boldsymbol{e}_p \equiv \boldsymbol{p}^{[\text{ref}]} - \boldsymbol{p} = \mathbf{J}\Delta\boldsymbol{\phi}$$

where $\boldsymbol{p}^{[\text{ref}]}$ and $\boldsymbol{p}$ are the target and actual pose of the end effector with respect to the base coordinate frame. Two common methods of solving IIK are pseudo-inverse and damped least squares method (DLSM) [62]. Here we chose the damped least squares method, whose goal is to minimize

$$\|J\Delta\boldsymbol{\phi} - \boldsymbol{e}_p\|^2 + \lambda^2\|\Delta\boldsymbol{\phi}\|^2 \tag{5.8}$$

where λ is a non-zero damping constant set empirically. Eq. (5.8) can be eventually rewritten to evaluate $\Delta\boldsymbol{q}$ with

$$\Delta\boldsymbol{\phi} = \mathbf{J}^T(\mathbf{J}\mathbf{J}^T + \lambda^2\mathbf{I})^{-1}\boldsymbol{e}_p. \tag{5.9}$$

More detailed derivation can be found in Buss's paper, as well as an alternative calculation approach using singular value decomposition [63].

Fig. 5.5 illustrates IIK in a simplified block diagram. In this diagram, the Jacobian

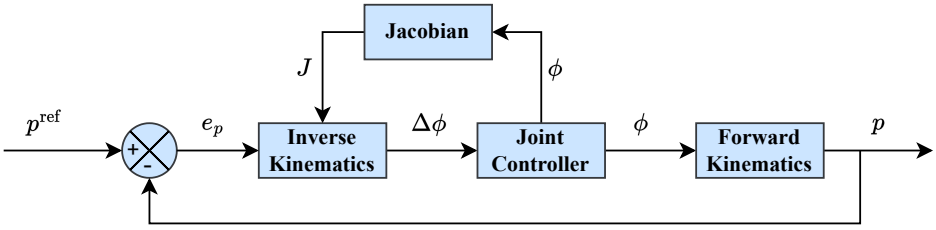


Figure 5.5: Simplified block diagram of IIK

block could be a pre-determined function that solves $\mathbf{J}(\boldsymbol{\theta})$ symbolically by Algorithm 5.1, or a runtime process that executes Algorithm 5.1 numerically. The inverse kinematics block is solved iteratively by Eq. (5.9). The joint controller is embedded in the low-level joint firmware of the KUKA YouBot[1]. The controller can be instructed to execute changes in the joint angle, denoted as $\Delta\boldsymbol{q}$. Since all joints also feature encoders, the firmware reports back the actual joint angle $\boldsymbol{\theta}$. Using this feedback, the pose of the end-effector $\boldsymbol{p}$

[1]Firmware source code and documentation available at GitHub repository https://github.com/youbot/youbot_driver

is calculated by forward kinematics as shown in Eq. (4.16). Assuming the encoders are properly calibrated, the acquired pose $\boldsymbol{p}$ is therefore sufficiently accurate. Consequently, any positional error can be effectively mitigated in the subsequent iteration of the inverse kinematics process.

Iterative inverse kinematics ultimately searches for a local solution that is closest to the target pose. This allows for smooth "natural" pose transitions when the target pose is programmed to track a trajectory rather than directly reach the final pose. For instance, at time t_0, if the end effectors are required to lift the payload by 0.5 m along the z-axis from (x_0, y_0, z_0), directly setting the target position to $(x_0, y_0, z_0 + 0.5)$ at the next sampling step $t_0 + \Delta t$ (where Δt is the sampling time) could result in abrupt movements that disrupt the payload's balance. A more realistic and stable approach involves executing a process over N steps. This is achieved by incrementally setting the target position at $[z_0 + 0.001, z_0 + 0.002, \ldots, z_0 + 0.05]$ gradually at $[t_0 + \Delta t, t_0 + 2\Delta t, \ldots, t_0 + N\Delta t]$. This method results in a more streamlined process when implemented with iterative inverse kinematics.

5.1.3 Distributed Autonomy

In this section we explain our method to automate the above manipulator and navigation control into a streamlined task. Our autonomy solution is built directly on the scenario constructed in CoppeliaSim simulation software [64]. In the simulation, both robots feature dynamic components for the manipulator axes and the Mecanum wheel rollers, each functioning as individual programmable joints. By controlling the position and speed of these joints, akin to programming a real-world robot, the arm and mobile platform can be maneuvered within the constraints set by the physics engine. Each robot adhere to separate instruction schedules and exchange information about each other via a communication channel, thereby facilitating distributed task execution. A finite-state machine consisting of five states was used to schedule the control and navigation execution.

Start State. The twin robots start at the two sides of the payload where the payload is within the reach of the manipulator arms. The manipulator control algorithm then

executes to hold the payload and pick it up.

Lift State. In this state, after the end-effectors (grippers) of the two robots securely lock the payload, the target poses for the end-effectors gradually move away to a higher position, causing the inverse kinematics to smoothly lift the end-effectors and the payload in between as a result. The inverse kinematics is implemented with the built-in `simIK` library for accuracy.

Navigate State. Once the payload is elevated, the robots move on to synchronously navigate to the goal, which is the target conveyor belt. The left robot (robot m) serves as the leader whose controller is to minimize its error in x, y, θ as described in Eq. (5.2), whereas the right robot (robot n) follows to optimize its corresponding error as in Eq. (5.3). The desired velocities are converted to individual wheel velocities by Eq. (4.9), which describes the inverse kinematic model of ODR. The wheel velocities are then given to the corresponding joints in the simulation. For proof of concept implementation, we implemented PID controller with Python programming language via the remote API with the simulation software, and we plan to explore more non-linear controller for better performance and stability.

Drop State. When the payload arrives at the desired x-y location, the robots carefully drop it with a method similar to that used to lift it, except for a lower altitude.

End State. Finally, the payload is placed on the conveyor belt, so the robots release it by loosen the gripper and driving away from the conveyor belt.

5.2 Simulation Results

This section presents how well the methods illustrated in Section 5.1 perform within a real-time robot simulator, CoppeliaSim. The manipulator control, twin navigation, and distributed autonomy subsystems are verified in real-time computer simulations running the Bullet [65] and Open Dynamics [66] physics engine. In the simulation software, the established YouBot model is set to exert a maximum force of 300 N. To mimic the realistic scenario where the payload is a sheet of glass, we set the friction coefficient at the contact

point between the end-effector and the payload to 0.2. The maximum velocity of the mobile platform was capped at 0.1 m/s to accommodate non-slippage assumption. The payload is carefully lifted by the manipulator control subsystem from the start position at $(x, y, z) = (0.23, -2.32, 0.26)$ m, and dropped to the end position at $(2.25, 0.88, 0.26)$ m, as shown in the beginning and end part from the trajectory shown in Fig. 5.6. The gray curve at the bottom is the projection of the trajectory on the XY plane. The end-effectors of the robots, illustrated by the orange dashed lines, stay side-by-side with the payload along the trajectory, and move away at the end of the transportation[2].

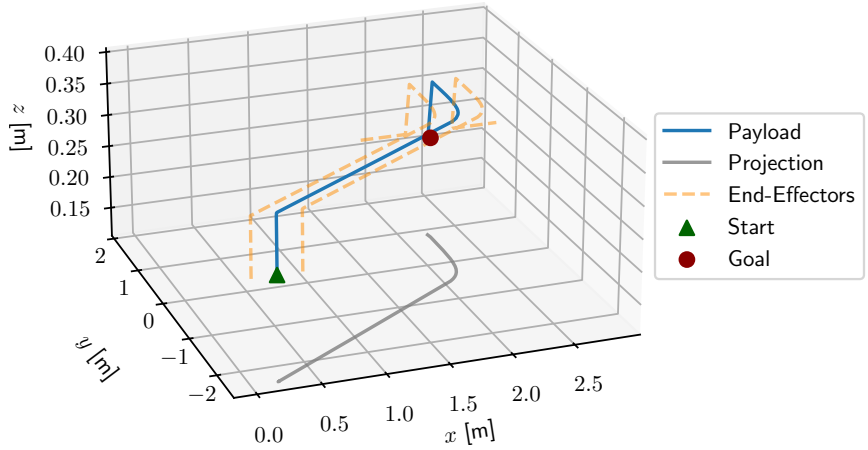
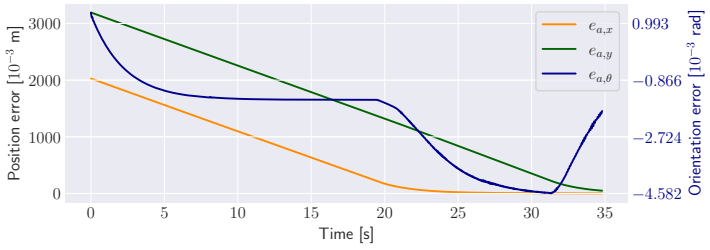


Figure 5.6: Trajectory of the payload from start to end position

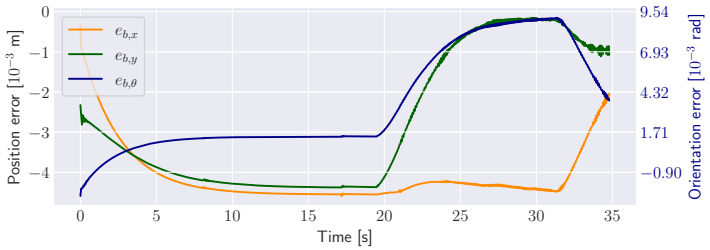
The dual navigation subsystem effectively minimizes the pose errors of mobile platforms m and n, as demonstrated in Eq. (5.2) and Eq. (5.3), respectively. The vector $\boldsymbol{e}_m$ is plotted against the simulation time, as depicted in Fig. 5.7(a). The variables $e_{m,x}$ and $e_{m,y}$ originate from approximately 2 m and 3.2 m respectively, signifying the initial distance from the start position to the goal. As the simulation progresses, both values decrease and eventually approach 0, indicating successful delivery of the payload to the goal. Conversely, $e_{m,\theta}$, represented by the blue curve and axis on the right, exhibits fluctuations throughout the simulation time. However, these fluctuations are on a scale under 10^{-3} rad, rendering them negligible. The same observations apply to $e_{n,x}, e_{n,y}$ and $e_{n,\theta}$. This is consistent with the

[2]Demonstration video of the simulation can be found at `https://youtu.be/amEwZZVkLSU`.

requirement of the object transportation task, which necessitates the follower robot b to precisely track the pose of the leader robot m within a strict tolerance. Failure to do so would result in the payload falling off, thereby failing the transportation task.



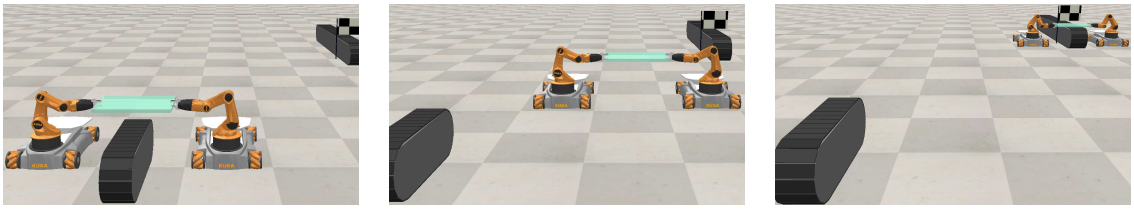
(a)



(b)

Figure 5.7: Mobile platform pose error $\boldsymbol{e}_a$ and $\boldsymbol{e}_b$ of robot a and b over time

Following five states in the distributed automation, Fig. 5.8(a) illustrates Start and Lift states, Fig. 5.8(b) demonstrates Navigate state, and Fig. 5.8(c) shows Drop and End states during the simulation.



(a) (b) (c)

Figure 5.8: Demonstration of (a) Start and Lift states, (b) Navigate state, (c) Drop and End states in simulation

The simulation methods and results using the KUKA YouBot mobile manipulator presented in this chapter are part of the preliminary experiments to understand the challenges and opportunities in cooperative multi-cobot systems. The work is formalized as a conference paper, and presented at the 2024 IEEE Robotics and Sensors Conference (ROSE) [9].

5.3 Digital Twin Simulation

As another set of simulation experiments, we created an environment in CoppeliaSim that accommodates a "one-to-one" model of the mobile robot used in Chapter 6. The purpose of this simulation is to establish a "digital twin" of the physical robot. A digital twin of a cyber-physical system is its virtual representation done in computers, which can be used to simulate and analyze the behavior of the system in a virtual environment. Grieves *et al.* rationalized that the purpose of a digital twin is to be able to understand whether the designs on paper are actually feasible, and to be able to determine the modes of failure before the physical system is actually produced [67].

We used this digital twin simulation, shown in Fig. 5.9, to verify our implementation methods before deploying them on the physical robot. The simulation environment was

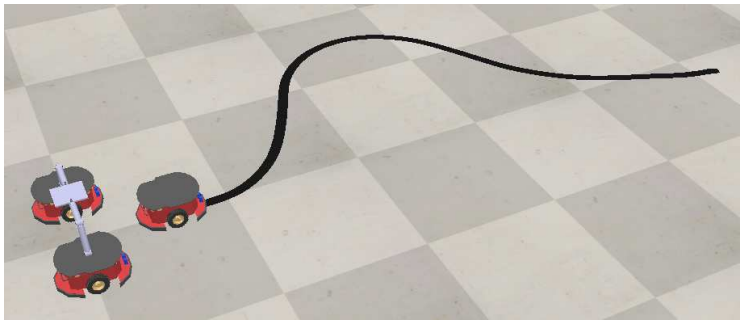


Figure 5.9: Digital twin simulation of the Romi robot in CoppeliaSim

programmed to mimic the physical robot's behavior as closely as possible, including the robot's kinematics, dynamics, and sensor readings. The digital twin simulation was programmed to follow the same interface and protocol as the real robot, allowing for seamless

integration from simulation to implementation. The control and programming of the digital twin simulation is shared with the implementation in Chapter 6, followed by the results and analysis in the same chapter. Furthermore, leveraging the physics engine in CoppeliaSim, we have an approximation of the dynamics of different parts of the robot, which is useful for generating data for ML-based algorithms.

5.4 Summary

In this chapter, we presented two sets of computer simulations to address the problem of cooperative multi-cobot systems. The first set of simulations utilized the KUKA YouBot mobile manipulator in a simulated environment. The simulation method was detailed in Section 5.1, and the results were presented in Section 5.2. The second set of simulations employed a digital twin of the Romi robot to verify the implementation methods before deploying them on the physical robot. The digital twin simulation was designed to closely replicate the physical robot's behavior, including its kinematics, dynamics, and sensor readings. Unlike the KUKA YouBot simulation, the digital twin simulation shares the exact same codebase as the physical robot except for direct hardware interface, so the detailed control and programming of the digital twin simulation can be referred to in Section 6.2 and Section 6.3, followed by the results and analysis in Section 6.5.

Chapter 6

Implementation using Autonomous Cobots

Over the last chapter, the theoretical framework of cooperative object transportation using autonomous cobots was practiced in a simulated environment. Using established KUKA YouBot models, we demonstrated the feasibility of the task thanks to the versatility of the omnidirectional mobile base and the 5-DOF manipulator arm. However, the simulation environment is a simplified representation of the real world, and the results may not be directly applicable to practical scenarios.

Considering the cost of deploying multiple KUKA YouBots, we decided to revise the robot system to a more cost-effective and customizable platform. In this chapter, we will discuss the implementation details of such a system, as well as the experimental results of the cooperative object transportation task in the real world. In Section 6.1, we will introduce the revised design based on the Pololu Romi mobile robot platform, equipped with a 3D printed manipulator arm. The hardware architecture, inter-robot communication, and the design of the manipulator arm will be discussed. In Section 6.2, we will delve into the mobile robot control system, which consists of hardware control, odometric localization, navigational control, and path planning. The manipulator control system will be discussed in Section 6.3, where we will introduce the DDPG algorithm within the deep

reinforcement learning (DRL) framework to learn the manipulator dynamics and generate control commands. Finally, we will present the experimental results of the cooperative object transportation system in the real world, and discuss the insights gained from the experiments.

6.1 Design of Cobot System

As opposed to off-the-shelf robots like KUKA YouBots, we chose to customize a robot that is more suitable for our research. Our envisioned cobot system is based on Romi mobile robots developed by Pololu Robotics and Electronics[1] and a 3D printed manipulator arm. The Romi robot is a small, low-cost robot platform made for competitive robotics, and therefore is easily customizable and expandable. The robot chassis is differential drive with two caster balls, and its circular shape allows for convenient turns or rotations in place. The bundled Romi 32U4 Control Board is a printed circuit board that includes an ATmega32U4 8-bit microcontroller unit (MCU), motor drivers, wheel encoders, battery power circuitry, and inertial sensors. For our cobot implementation, the on-board microcontroller is programmed with a firmware that communicates with a Raspberry Pi 5 single-board computer (SBC) over serial connection. The use of Raspberry Pi allows us to run high-level control algorithms with fast-prototyping programming languages such as Python, as opposed to bare-metal programming in C/C++ on the MCU. Furthermore, the SBC provides Wi-Fi connectivity, which is essential for multi-robot communication and remote monitoring. We chose the Raspberry Pi 5 in particular for its abundant online resources, community support, and future-proofing for potential upgrades such as computer vision or machine learning applications. Figure 6.1 shows part of the

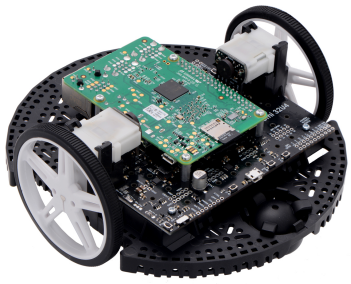


Figure 6.1: Pololu Romi Cobot

[1] `https://www.pololu.com`

Romi cobot with (from bottom to top) the chassis, the control board, and the Raspberry Pi SBC.

6.1.1 Hardware Architecture

Figure 6.2 illustrates the hardware architecture of our Romi cobot system. The on-board

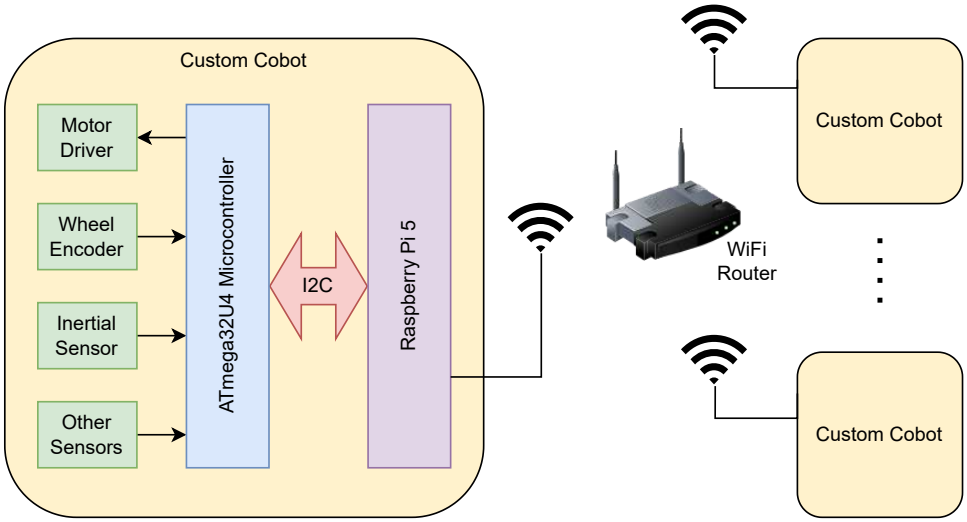


Figure 6.2: Design of Romi Cobot System

electronic sensors and actuators are routinely interfaced with the ATmega32U4 MCU, which is responsible for low-level control of the robot as a hardware abstraction layer, including motor control, encoder reading, and sensor data acquisition. Leveraging the I2C communication bus, the MCU communicates with the Raspberry Pi SBC according to a custom protocol. The Raspberry Pi runs high-level control algorithms, such as navigation, manipulation, and coordination, and sends control commands to the MCU. Finally, to facilitate multi-robot communication, the Raspberry Pi is connected to a wireless network, which allows for remote control and monitoring over the TCP/IP stack.

6.1.2 Inter-Robot Communication

MQTT technology is used to implement inter-robot communication. It is a lightweight messaging protocol heavily used in IoT applications, and recently adopted in robotics for sensory network [68], human-robot collaboration [69], and real-time industrial monitoring [70]. In MQTT, messages between clients are not transmitted with direct connections. Instead, they are addressed using a subject line called a *topic*, and clients can subscribe to topics they are interested in, so they receive a copy of the message when it is published. The topic is organized in a string-based hierarchy separated by slashes (`/`), and clients can use wildcard character (`+` or `#`) to subscribe to multiple topics at once. For example, a robot can publish its pose information to the topic `robot_a/pose`, and another robot can subscribe to all robots' pose information by subscribing to `+/pose`. This publisher-subscriber paradigm is managed by a MQTT *broker*, which is a piece of software that routes messages between clients connected to it. For our multi cobot system, we chose NanoMQ[2], which is a MQTT broker developed for low-latency performance by leveraging multi-threading and asynchronous I/O.

The communication architecture of our cobot system is illustrated in Fig. 6.3. The architecture is structured to balance bandwidth and data transmission rates effectively, ensuring seamless operation across different modules and sensors. At the top layer, the MQTT protocol operates at a low frequency of 10 Hz, which transmits high-level control commands and robot state information. The lower layer of the architecture is dedicated to the I2C bus, which operates at a higher frequency of 100 kHz to support communication between the system's hardware components. The higher bandwidth of the I2C bus is essential for real-time control of the robot's motors and sensors, while the MQTT protocol is used for high-level coordination and communication between multiple robots. Overall, the architecture optimally separates high-bandwidth, low-latency hardware interactions from low-bandwidth, high-latency communication for supervisory control, ensuring scalability and efficiency.

[2]More information about NanoMQ can be found at `https://nanomq.io/`.

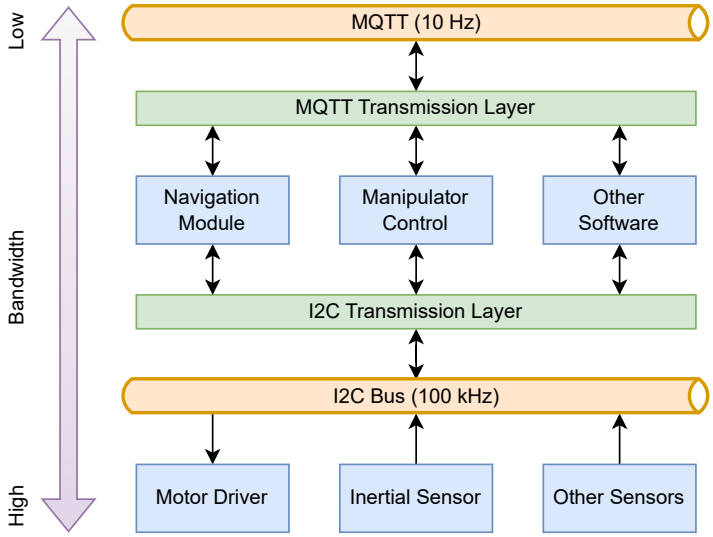


Figure 6.3: Communication Architecture of Romi Cobot System

6.1.3 Manipulator Arm Design

The manipulator arm is a 3D-printed design with 2 DOF and a gripper mounted onto the top plate of said mobile robot, as shown in Fig. 6.4. It is a lightweight, modular design manufactured from PLA material, ensuring both cost efficiency and ease of production. Key structural elements use a truss-style framework to maintain strength while minimizing material usage. The joints are equipped with servo motors, as seen with the included motor model to the left of the assembly. These servos provide the torque and angular precision needed for the arm's controlled movement. We left the gripper design open-ended to allow for future customization and expansion, such as adding sensors or actuators for more advanced manipulation tasks. This design is intended to be a proof of concept for the cobot system, demonstrating the feasibility of cooperative object transportation with a low-cost, easily customizable platform. We collaborated with Ty Wilkinson, a talented undergraduate student in the ECE department, to design and implement the manipulator arm.

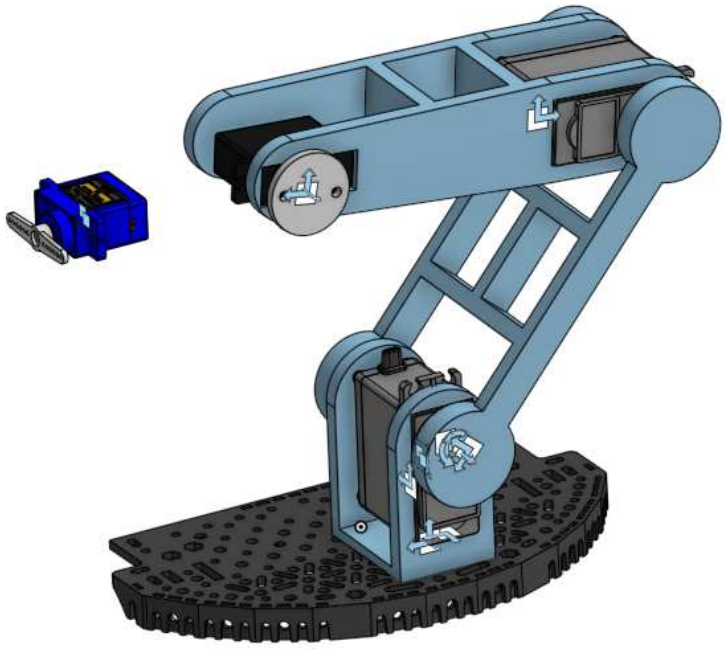


Figure 6.4: 3D Printed Manipulator Arm

6.1.4 Three-Cobot System with Scout Robot

Last but not least, compared to the 2-cobot system used in simulation (cobot m and cobot n), we have expanded the system to 3 cobots for real-world experiments. As opposed to the simulation environment where the robot state and map information are known as ground truth, the real-world system has to rely on sensors and SLAM algorithms to estimate the robot's pose and the environment map. SLAM requires the robot to move around the environment freely to build a map, which is essential for generating a path for the transportation task. However, when the two cobots actively transport the payload, their path is subject to the transportation task instead of the SLAM task. While performing the task, their observation of the environment is limited to nearby objects, such as the payload and the other cobot. This limitation may lead to a suboptimal path.

To address this issue, we have introduced a third robot in the multi cobot system. This third cobot is responsible for exploring the environment, building a map, and planning

an optimal path for future tasks, while the other two cobots focus on the transportation task. We thereby name this cobot the *scout* robot. Symbols related to the scout robot are denoted with the subscript s, such as $\boldsymbol{q}_s$ for its pose and $\boldsymbol{u}_s$ for its control input. The scout robot should be equipped with a more advanced sensor suite, such as a LiDAR or a depth camera, to build a more accurate map.

In our research, limited by time and resources, we developed a prototype scout robot that follows lines on the floor to explore the environment as a proof of concept. The scout robot is programmed to follow a visible line path that covers the entire environment, and it builds a map using the data collected by its on-board sensors. The map is then used by the other two cobots to plan their paths for the transportation task. This is illustrated in Fig. 6.5. This division of labor allows the cobots to work collaboratively and efficiently, ensuring the success of the transportation task.

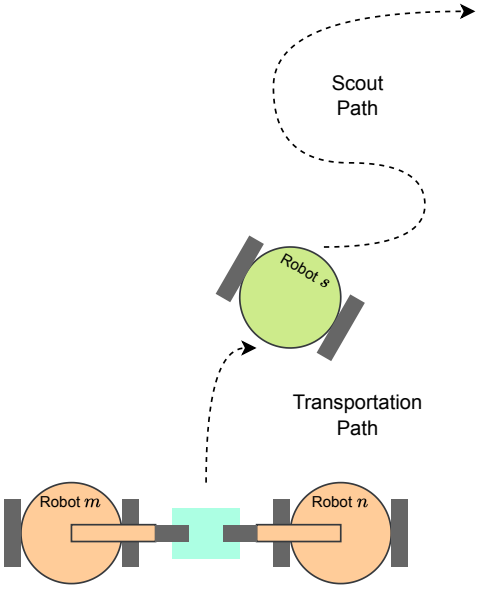


Figure 6.5: Three-Cobot System with Scout Robot

6.2 Mobile Robot Control

The Romi mobile robot platform is controlled by a combination of the on-board MCU and the Raspberry Pi SBC. In total, the control system consists of four components: hardware control, odometric localization, navigational control and path planning. In this section, we will discuss the implementation details of each component.

The **hardware control** is the only component that runs on the on-board MCU. Its job is to read the encoder sensors, and control the motors to achieve the desired velocity. The encoder is a sensor that measures the rotation of the motor shaft by counting the number of pulses generated by a disk with slots. The Romi robot uses a quadrature encoder, which generates approximately 1440 pulses per revolution of the motor shaft. Consequently, if the encoders on the left and right wheel capture c_l and c_r pulses (or ticks), respectively, within a second, the wheel speed $\boldsymbol{u} = [u_l, u_r]^\top$ can be calculated as

$$\boldsymbol{u} = \frac{2\pi}{N_{\text{enc}}}\boldsymbol{c} = \frac{2\pi}{N_{\text{enc}}}\begin{bmatrix} c_l \\ c_r \end{bmatrix} \tag{6.1}$$

where $N_{\text{enc}} = 1440$ is the number of ticks per revolution.

This wheel speed is compared against the desired speed $u^{[\text{ref}]}$ to generate a Pulse Width Modulation (PWM) signal for the motor driver, using a PI controller. Its duty cycle DC in percent is given by

$$\text{DC} = K_p(u^{[\text{ref}]} - u) + K_i \int (u^{[\text{ref}]} - u)\,\mathrm{d}t \tag{6.2}$$

where proportional gain and integral gain are set empirically as $K_p = 1.53 \times 10^{-2}$ and $K_i = 1.53 \times 10^{-3}$, respectively.

The **odometric localization** uses the encoder readings to estimate the robot's pose incrementally. Starting from an initial pose $\boldsymbol{q}_0 = [x_0, y_0, \theta_0]^\top$, the pose at time t is updated by Runge-Kutta integration of the kinematic model. Consider the DDR robot model is updated with the change in encoder readings Δc_l and Δc_r, its change in pose $\Delta\boldsymbol{q} =$

$[\Delta x, \Delta y, \Delta\theta]^\top$ can be estimated with:

$$\begin{aligned}\Delta s &= r/2(\Delta c_l + \Delta c_r)\\ \Delta\theta &= r/\ell(\Delta c_r - \Delta c_l)\\ \Delta x &= \Delta s\cos(\theta + \Delta\theta/2)\\ \Delta y &= \Delta s\sin(\theta + \Delta\theta/2),\end{aligned} \tag{6.3}$$

where r is the wheel radius and ℓ is the distance between the wheels.

The **navigational control** component is a software module that runs on the Raspberry Pi. It receives the desired pose $\boldsymbol{q}^{[\text{ref}]}$ from the path planner, uses the odometric localization to estimate the current pose $\boldsymbol{q}$, and generates a control input $\boldsymbol{u}$ to drive the robot to the desired pose. This leverages two linear controllers that generate the desired linear velocity v and angular velocity ω, with the goal of minimizing the geometric error

$$\begin{bmatrix} e_x \\ e_y \end{bmatrix} = \begin{bmatrix} x^{[\text{ref}]} - x \\ y^{[\text{ref}]} - y \end{bmatrix}. \tag{6.4}$$

The output of the controllers is given by

$$\begin{aligned} v &= K_v\sqrt{e_x^2 + e_y^2}\\ \omega &= K_\omega\Big(\text{atan2}(e_y, e_x) - \theta\Big)\end{aligned} \tag{6.5}$$

where atan2 is the four-quadrant inverse tangent function, $K_v = 4$ and $K_\omega = 4$ are the controller gains.

The control input $\boldsymbol{u} = [u_l, u_r]^\top$ is then calculated by the inverse kinematics of the DDR model according to Eq. (4.7).

The final component, **path planning**, executes least often, but has the most significant impact on the robot's behavior. The module acknowledges current poses of all robots in real time, schedules the transportation task, and assigns target poses to each robot at appropriate times. In our work, we made robot m and robot n navigate such that the

payload being transported follows the same path as the scout robot. For this, a synchronous and an asynchronous strategy were implemented.

In *synchronous path planning*, robot m and robot n follow the scout robot in real time; while in asynchronous path planning, robot m and robot n follow the scout robot's path after it has traveled a certain distance. Consider the poses of the robots $\boldsymbol{q}_m$, $\boldsymbol{q}_n$, and $\boldsymbol{q}_s$ at time t. The goal of the synchronous path planning is to deliver the payload from current position to the scout robot's position, or mathematically,

$$\boldsymbol{q}_o^{[\text{ref}]} = \begin{bmatrix} x_o^{[\text{ref}]} \\ y_o^{[\text{ref}]} \\ \theta_o^{[\text{ref}]} \end{bmatrix} = \boldsymbol{q}_s. \tag{6.6}$$

This requires robot m and robot n to be in parallel positions with the scout robot as shown in Fig. 6.6. As a result, the desired position of robot m and robot n can be expressed with

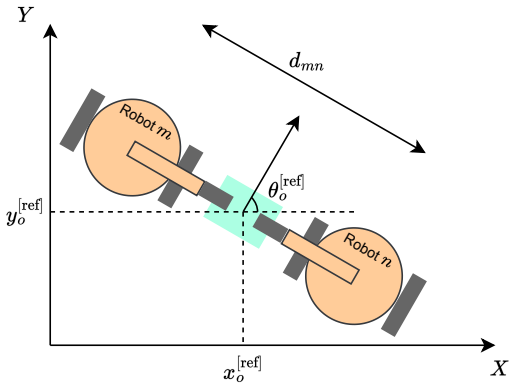


Figure 6.6: Robot m and Robot n with respect to payload

respect to the payload's desired pose as

$$\begin{aligned}
\boldsymbol{q}_m^{[\text{ref}]} &= \begin{bmatrix} x_m \\ y_m \\ \theta_m \end{bmatrix} = \begin{bmatrix} x_o^{[\text{ref}]} - \frac{d_{mn}}{2} \sin\left(\theta_o^{[\text{ref}]}\right) \\ y_o^{[\text{ref}]} + \frac{d_{mn}}{2} \cos\left(\theta_o^{[\text{ref}]}\right) \\ \theta_o^{[\text{ref}]} \end{bmatrix} \\
\boldsymbol{q}_n^{[\text{ref}]} &= \begin{bmatrix} x_n \\ y_n \\ \theta_n \end{bmatrix} = \begin{bmatrix} x_o^{[\text{ref}]} + \frac{d_{mn}}{2} \sin\left(\theta_o^{[\text{ref}]}\right) \\ y_o^{[\text{ref}]} - \frac{d_{mn}}{2} \cos\left(\theta_o^{[\text{ref}]}\right) \\ \theta_o^{[\text{ref}]} \end{bmatrix}
\end{aligned} \tag{6.7}$$

where d_{mn} is the distance between the robots. The control input $\boldsymbol{u}_m$ and $\boldsymbol{u}_n$ are then calculated by the navigational control module to drive robot m and robot n to the desired poses.

The *asynchronous strategy*, on the other hand, generates a series of waypoints along the scout robot's path, and robot m and robot n navigate to these waypoints sequentially. For this, we used spline interpolation, which is a method of constructing a smooth curve that passes through a set of points, in this case, the scout robot's XY coordinates $(x_s(i), y_s(i)), i = 0, 1, 2, \ldots, N_s$. The parametric spline representation models the curve as a pair of parametric equations for $x(t)$ and $y(t)$, where t is a parameter that defines the curve's progress [71]:

$$\begin{aligned}
x(t) &= \sum_{i=0}^{N_s+k} \gamma_{x,i} B_i^k(t) \\
y(t) &= \sum_{i=0}^{N_s+k} \gamma_{y,i} B_i^k(t),
\end{aligned} \tag{6.8}$$

where $\gamma_{x,i}$ and $\gamma_{y,i}$ are the control points along the x and y axes, and $B_i^k(t)$ are the B-spline basis functions of degree k. The basis functions $B_i^k(t)$ are defined recursively using the

Cox-de Boor formula:

$$B_i^0(t) = \begin{cases} 1 & t_i \leq t < t_{i+1} \\ 0 & \text{otherwise} \end{cases}$$
$$B_i^k(t) = \frac{t - t_i}{t_{i+k} - t_i} B_i^{k-1}(t) + \frac{t_{i+k+1} - t}{t_{i+k+1} - t_{i+1}} B_{i+1}^{k-1}(t). \tag{6.9}$$

The control points $\gamma_{x,i}$ and $\gamma_{y,i}$ are determined by solving a linear system of equations that ensures the curve passes through all the points and has continuous derivatives. This spline representation allows us to evaluate the curve at any point t to obtain the desired XY coordinates for robot m and robot n to navigate to. By sampling t at regular intervals, we can generate a series of waypoints along the scout robot's path, which, like the synchronous strategy, are treated as desired poses for the payload. The desired poses can then be calculated the same with Eq. (6.7).

A summary of the control architecture is shown in Fig. 6.7. Each robot has their

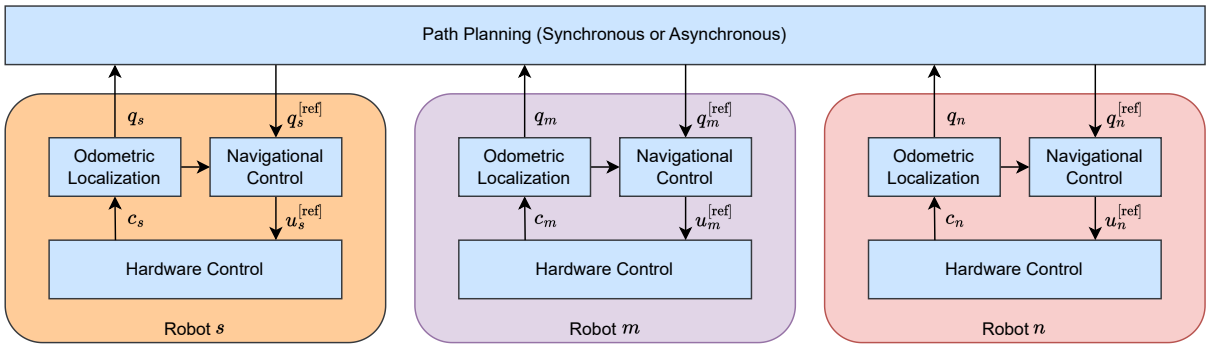


Figure 6.7: Control Architecture of Romi Mobile Robot

respective hardware control, odometric localization, and navigational control modules. The odometric localization module estimates the robot's pose $\boldsymbol{q}$ using the encoder readings $\boldsymbol{c}$ from the hardware control module. The navigational control module generates the desired control input $\boldsymbol{u}^{[\text{ref}]}$ based on the desired pose $\boldsymbol{q}^{[\text{ref}]}$ from the path planning module. The path planning module oversees all robots and generates the desired poses $\boldsymbol{q}^{[\text{ref}]}$ for each robot based on the task requirements.

6.3 Manipulator Control

The control module of the 2-DOF manipulator arm ensures a smooth and accurate movement of the end-effector. This could be easily achieved with the inverse kinematics of the manipulator arm, as described in Eq. (4.13). However, manipulator control without dynamic compensation can lead to overshoot and oscillation, especially when the manipulator arm is carrying a payload. In Section 4.3, we derived the analytical dynamics of the manipulator arm based on the Euler-Lagrange equation of motion, and suggested that it is still not the perfect model for control. Our envisioned manipulator control system leverages a ML model to learn the manipulator dynamics and compensate for the error. In particular, we selected the DDPG algorithm within the DRL framework to learn the manipulator dynamics and generate control commands. The control task therefore involves learning a policy π_θ that outputs the joint torques $\boldsymbol{\tau} = [\tau_1, \tau_2]^\top$ given the joint angles $\boldsymbol{q} = [q_1, q_2]^\top$, joint velocities $\dot{\boldsymbol{q}}$ and joint accelerations $\ddot{\boldsymbol{q}}$. We thereby define the state and action spaces as

$$\mathcal{S} = \{q_1, q_2, \dot{q}_1, \dot{q}_2, \ddot{q}_1, \ddot{q}_2\}, \quad \mathcal{A} = \{\tau_1, \tau_2\}. \tag{6.10}$$

The reward function penalizes tracking errors $\boldsymbol{e} = \boldsymbol{q}^{[\text{ref}]} - \boldsymbol{q}$ and high torques $\boldsymbol{\tau}$, and is defined as

$$r = -\|\boldsymbol{e}\|^2 - \alpha\|\boldsymbol{\tau}\|^2, \tag{6.11}$$

where α is a hyperparameter that balances the tracking error and the torque penalty.

The DDPG algorithm consists of two deep neural networks, namely the actor network and the critic network, as seen in Fig. 6.8. The *actor network* $\mu(s \mid \theta_\mu)$ parameterizes the policy by mapping states s to actions a. This network has three fully connected layers with ReLU activation functions, and the output layer uses a tanh activation function to scale the output to the action space. The hidden layers have 400 and 300 units, respectively, and the output layer has 2 units corresponding to the joint torques. The *critic network* $Q(s, a \mid \theta^Q)$ estimates the action-value function, which evaluates the quality of actions taken in given states. The network designed for the critic also has three fully connected layers. The first layer takes the state as input and outputs 400 units using ReLU activation.

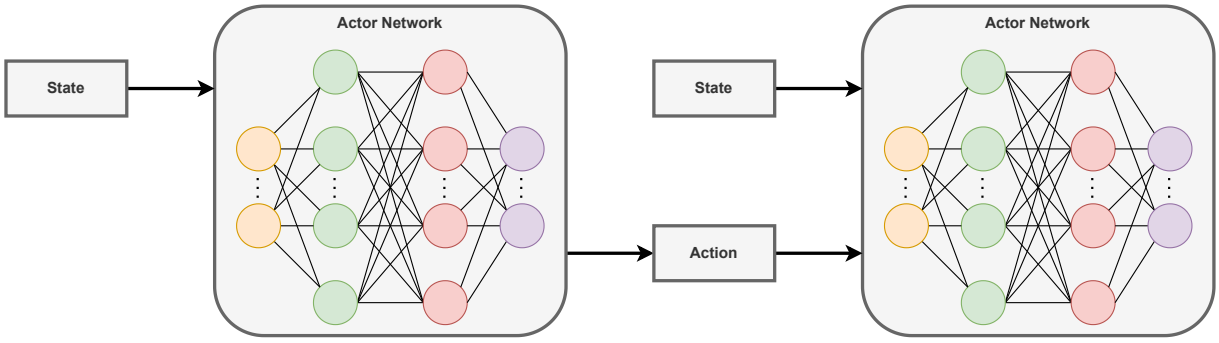


Figure 6.8: Actor and Critic Networks in DDPG

The second layer concatenates the output of the first layer with the action and outputs 300 units with ReLU activation. The third layer outputs a single value, which is the estimated action-value.

The training workflow of the DDPG algorithm for the manipulator control is as follows:

Replay Buffer. During the model interaction with the environment, the experience tuple (s, a, r, s') is cached into memory. When the value and policy networks are updated, a batch of experience tuples is sampled from the replay buffer, which breaks the temporal correlation between the samples.

Critic Update. The critic network is updated by minimizing the temporal difference error between the predicted action-value and the target action-value. The target action-value is calculated by the Bellman equation:

$$y = r + \gamma Q'(s', \mu'(s' \mid \theta_{\mu'}) \mid \theta_{Q'}), \tag{6.12}$$

where γ is the discount factor, and the prime denotes the target network.

Actor Update. The actor network is updated by maximizing the expected action-value given the state:

$$\nabla_{\theta_\mu} J(\theta) \approx \frac{1}{N} \sum_i \nabla_a Q(s, a \mid \theta_Q) \nabla_{\theta_\mu} \mu(s \mid \theta_\mu). \tag{6.13}$$

Target Network Update. To stabilize the training process, the target networks are updated by soft updating the value and policy networks:

$$\theta' = \tau\theta + (1 - \tau)\theta', \tag{6.14}$$

where τ is the soft update rate.

The training process primarily takes place in the digital twin environment illustrated in Section 5.3, where the manipulator arm is simulated in a physics engine. The trained model is then deployed to the real-world manipulator arm. Using transfer learning methods, the model is fine-tuned with real-world data to adapt to the physical dynamics of the manipulator arm.

6.4 Intermittent Communication Handling

When the robots are deployed in industrial environments, the wireless communication is often subject to packet loss, latency, and jitter, which can lead to intermittent communication. In the context of cooperative object transportation, intermittent communication can cause the robots to lose track of each other, deviate from the planned path, or even collide with each other. To address this issue, apart from employing fast converging network topologies as illustrated in Section 3.3, we implemented an algorithmic communication handling method that increases the robustness of the cooperative object transportation system.

Our method can be summarized using pseudocode in Algorithm 6.1. When robot i loses connection, it enters a loop that attempts to reconnect to the network. If the reconnection is successful, the robot returns to normal operation. If the reconnection fails, the robot estimates its pose using odometric localization according to Eq. (6.3), since the encoder readings are locally available. The other robots' poses are predicted using dead reckoning, which is a navigation technique that estimates the robot's position based on its previous position and the last known control input. This can be broken down into two steps. First, the control input $\boldsymbol{u}_j$ is used to derive the velocity $\dot{\boldsymbol{q}}_j$ of robot j using forward kinematics

Algorithm 6.1 Pseudocode when robot i loses connection

Input: $\boldsymbol{q}_s, \boldsymbol{u}_s, \boldsymbol{q}_m, \boldsymbol{u}_m, \boldsymbol{q}_n, \boldsymbol{u}_n$ ▷ *states and control inputs of all robots*

```
while true do attempt reconnect
    if success then                                          ▷ return to normal operation
        break
    else
        q_i ← OdometricLocalization(q_i, c_i)
        for robot j in all other robots do
            q_j ← DeadReckoning(q_j, u_j)
            if EuclideanDistance(q_i, q_j) < ε then          ▷ safety distance
                q̇_i ← 0
            else
                Keep q̇_i unchanged
```

expressed in Eq. (4.6). Then, the predicted pose $\boldsymbol{q}_j$ is updated with Euler integration over the time-lapse from last iteration Δt:

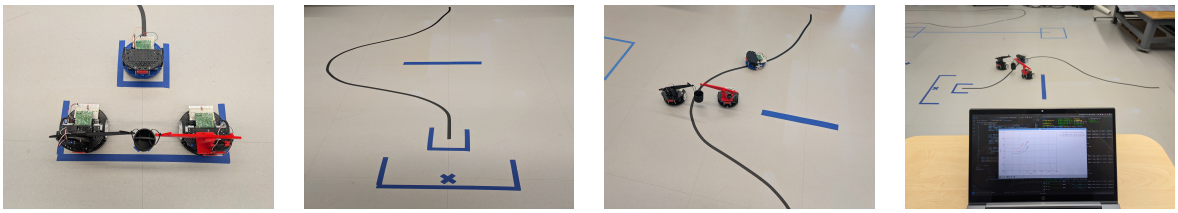
$$\boldsymbol{q}_j(t + \Delta t) = \boldsymbol{q}_j(t) + \dot{\boldsymbol{q}}_j \Delta t. \tag{6.15}$$

The predicted poses are then used to ensure that robot i maintains a safe distance from the other robots. If the Euclidean distance between robot i and robot j is less than a predefined threshold ε, the velocity of robot i is set to zero to prevent collision. Otherwise, the velocity of robot i remains unchanged.

The intermittent communication handling ensures that the robots can continue to operate even when the network is unstable, and reduces the risk of collision or deviation from the planned path. However, it relies on the assumption that the connection can be reestablished within a reasonable time frame. In the event that the connection cannot be reestablished, or the reconnection time is too long, the robots may need to be manually reset to prevent any potential hazards.

6.5 Experiment Results

Three robots described in Section 6.1 were assembled for the scout robot s and the transporter robots m and n, as seen in Fig. 6.9(a). The scout robot s is equipped with a light



(a) (b) (c) (d)

Figure 6.9: Experimental setup of implementation. (a) Transporter robots and scout robot. (b) Experimental environment and path. (c) Object transportation in action. (d) Real-time monitoring with laptop.

sensor array to follow the line path, and the transporter robots m and n are equipped with the manipulator arm. The transporter robots m and n follow the scout robot's path to transport the payload, which is a basket whose handle is grasped by the manipulator arms of the transporter robots. Limited by time and resources, the designed 2-DOF manipulator arm was not fully fabricated, so we used a simpler 1-DOF manipulator made from a servo motor and a lever arm, capable of lifting small objects. The training and deployment of the manipulator control model were unfortunately left as future work due to time constraints. On the other hand, the mobile robot control system was fully implemented and tested in the real-world environment. The path configuration of the scout robot s is shown as the black curve in Fig. 6.9(b). The scenario lets the scout robot s recognizes an obstacle ahead (the blue horizontal line) and plans a detour path to avoid it. Figure 6.9(c) shows a snapshot of the three robots in cooperative object transportation task. The last image in Fig. 6.9(d) shows the full experimental setup of the cooperative object transportation system. A laptop is connected to the wireless network, which allows for remote monitoring and control of the system in real time. The window on the laptop screen shows a real-time visualization of the robots' poses and the payload's position, which is essential for debugging and monitoring the system.

Figure 6.10 shows the experimental results of the mobile robot control system illustrated in Section 6.2. Figure 6.10(a) and Fig. 6.10(c) displays the trajectories of the three

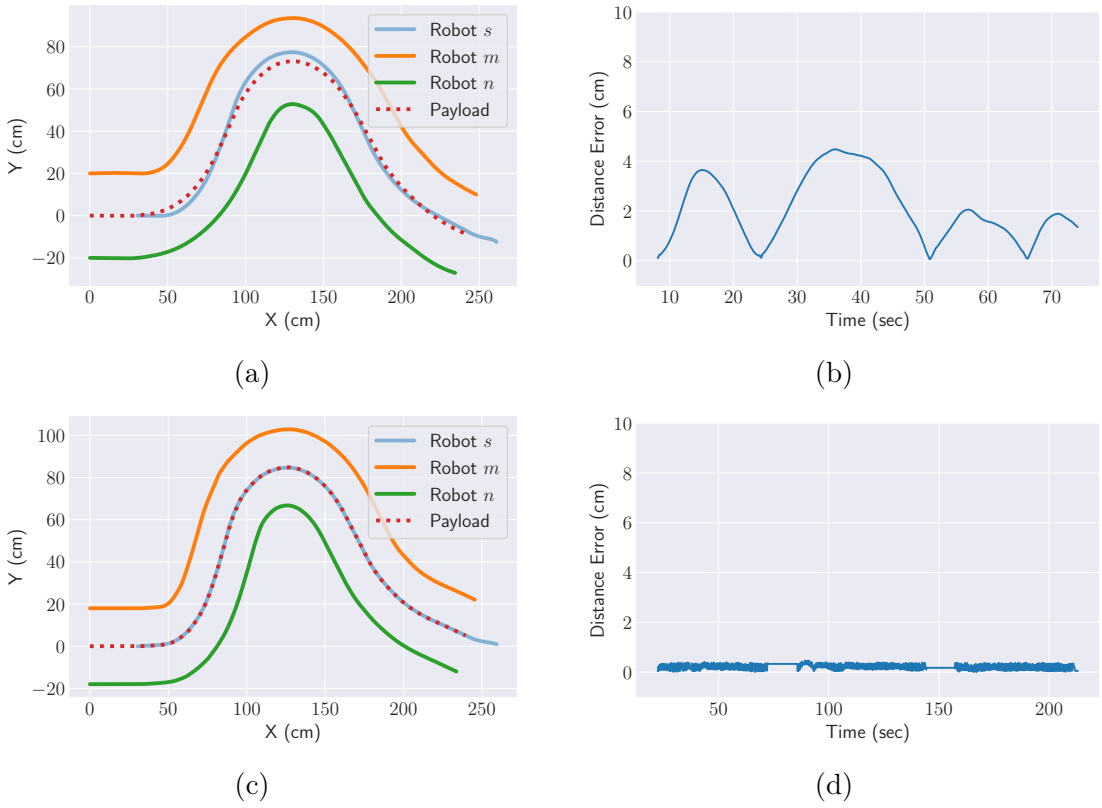


Figure 6.10: Experimental results comparing path planning strategies. (a) Trajectories and (b) tracking error in synchronous strategy. (c) Trajectories and (d) tracking error in asynchronous strategy.

robots and the payload in synchronous and asynchronous path planning, respectively[3]. The payload trajectory, shown in red dashed lines, follows the scout robot's path, shown in blue solid lines, while the transporter robots' trajectories are shown in orange and green solid lines. The payload tracking error, calculated as the Euclidean distance between the payload's position and the closest point on the scout robot's path is plotted over time in

[3]Demonstration video of the experiments can be found at `https://youtu.be/dFwgAhytarE` and `https://youtu.be/DsO6tCm7DwA`.

Fig. 6.10(b) and Fig. 6.10(d), where the former shows the synchronous strategy and the latter shows the asynchronous strategy. The results show that the asynchronous strategy has a lower tracking error compared to the synchronous strategy. This is evident from the fact that the payload trajectory aligns nearly perfectly with the scout robot's path in Fig. 6.10(c), while there is a noticeable deviation in Fig. 6.10(a). Differences in tracking error are also observed in Fig. 6.10(b) and Fig. 6.10(d), where the synchronous strategy has a significantly higher tracking error, shooting up to more than 4 cm at times, while the asynchronous strategy maintains a tracking error of less than 1 cm. However, comparing the time axis in Fig. 6.10(b) and Fig. 6.10(d), the synchronous strategy takes less time (20 s vs 220 s) to complete the task, as the transporter robots move in parallel with the scout robot. This trade-off between tracking error and completion time is a key consideration in the design of cooperative object transportation systems. When the map of the environment is known in advance, the synchronous strategy can be used to minimize completion time. On the other hand, when the map is not known or the environment is dynamic, one should consider the asynchronous strategy, which allows the MRS to adapt to the environment and minimize tracking error.

Lastly, following the discussion on multi-robot communication in Chapter 3, we have tested the resilience of the communication system by introducing packet loss. We programmatically dropped MQTT messages between the robots to simulate packet loss at a rate of 20 %, and observed the system's behavior when the cyclic with backlink and star network topology was used. We used synchronous path planning strategy for this test, since the asynchronous strategy is inherently more resilient to packet loss. The performance of the system was evaluated similarly by observing the trajectory of the robots and the payload, and the tracking error. Comparing the tracking error of the two network topologies in Fig. 6.11(b) and Fig. 6.11(d), the cyclic with backlink topology shoots higher due to lower redundancy in the network. However, it quickly recovers from the packet loss and maintains a lower tracking error compared to the star network topology. Comparatively, we see more frequent spikes in the tracking error of the star network topology, which indicates that the system is more sensitive to packet loss, due to slower convergence rate. This result stay true to the theoretical analysis in Chapter 3.

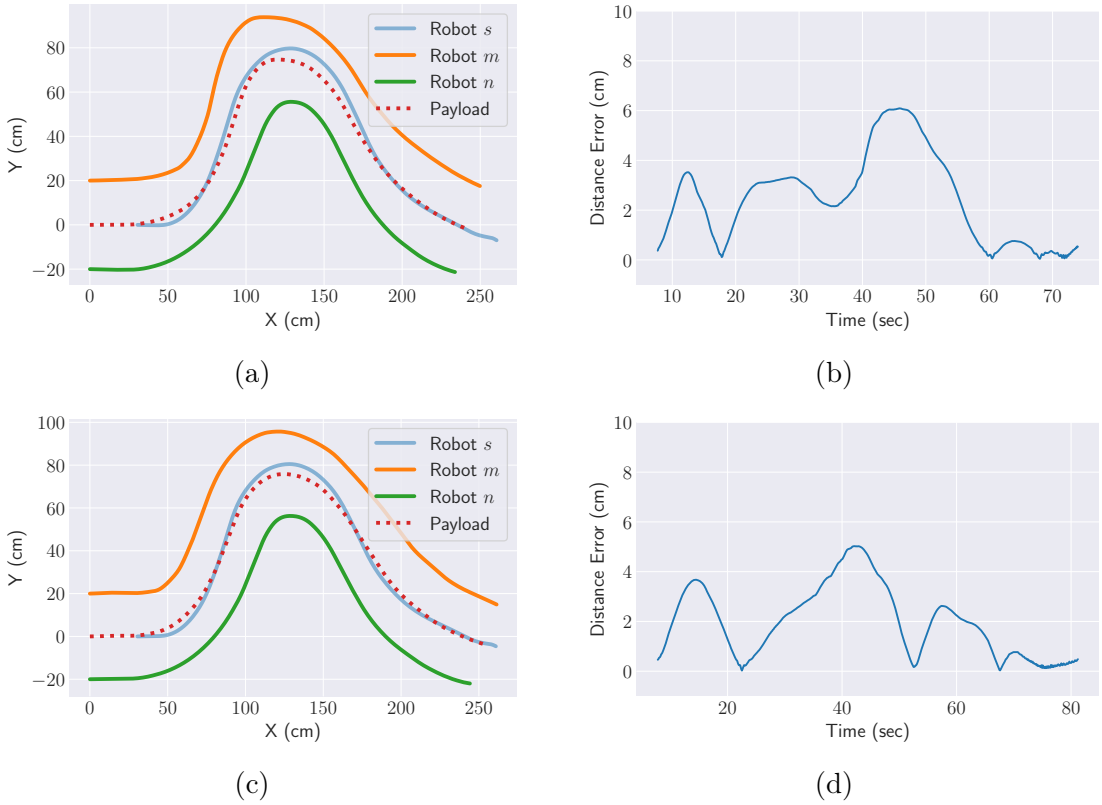


Figure 6.11: Experimental results comparing network topologies. (a) Trajectory and (b) tracking error using cyclic with backlink topology. (c) Trajectory and (d) tracking error using star network topology.

6.6 Summary

In this chapter, we discussed the implementation details of the cooperative object transportation system using autonomous cobots. We designed a cobot system based on the Pololu Romi mobile robot platform, equipped with a 3D printed manipulator arm. The system architecture was designed to balance hardware control, odometric localization, navigational control, and path planning. The manipulator arm control system was designed using the DDPG algorithm within the DRL framework to learn the manipulator dynamics and generate control commands. Furthermore, we increased the robustness of the system

by implementing an intermittent communication handling method that ensures the system can continue to operate even when the network is unstable. The mobile robot control system was fully implemented and tested in the real-world environment, and the experimental results showed the effectiveness of the cooperative object transportation system. The synchronous and asynchronous path planning strategies were compared, showing a trade-off between tracking error and completion time. Lastly, the resilience of the communication system was tested by introducing packet loss, and the cyclic with backlink topology was found to be more resilient compared to the star network topology. The results of the experiments demonstrate the feasibility of the cooperative object transportation system and provide insights into the design considerations for multi-robot systems.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, we focus on the development of a multi-robot system that can autonomously navigate and collaborate with each other to complete tasks in a shared workspace. In particular, we consider the problem of cooperative object transportation in an industrial environment, using mobile robots with manipulators attached. Such a system is often addressed as cobots, which are short for collaborative robots and designed to assist humans in completing tasks or to work simultaneously with humans in the same workspace. The robots are required to work cooperatively, sharing information and coordinating their actions to achieve tasks efficiently. The system is validated with a group of two robots, but the design is scalable to accommodate more. Reflecting on the objectives and problem statement, we have successfully designed and implemented a multi-robot system for cooperative object transportation tasks. In addition, based on the previous chapters, we can draw several detailed conclusions after evaluating the system in simulation and real-world environments.

In the graph theoretic model of the multi-robot system presented in Chapter 3, the communication network among the robots is modeled using directed graphs. The graph Laplacian matrix is used to represent the connectivity among the robots, and several

potential topologies are considered. Specifically, we compared fully connected, cyclic, cyclic with backlink and star topologies. Using the graph Laplacian and its eigenvalues, we analytically showed that the last two topologies are suitable for our envisioned multi-cobot system.

Chapter 4 presents the kinematic and dynamic models of the robots. This includes the mobile robot base, and the manipulator arm attached to it. For the mobile robot, we considered both differential drive and omnidirectional drive models, and derived the kinematic and inverse kinematic equations for each. The analysis of the manipulator arm started with a 2-DOF planar manipulator, and then extended to a 5-DOF spatial manipulator. The dynamic model of the manipulator arm was derived using the Lagrangian formulation, and the control system was designed using the computed torque method.

The theoretical models in these chapters were practiced in the simulation environment in Chapter 5. As a preliminary step, we leveraged KUKA YouBot mobile manipulators in CoppeliaSim, which is a commercial robotics simulator with embedded physics engine. The proposed solution consists of three subsystems, including the control system for the manipulator, the navigation system for the mobile base, and the communication system for the multi-robot coordination. The simulation results demonstrate the effectiveness of transferring objects from one place to another, while maintaining a stable formation and avoiding collisions.

The system was then implemented on real robots in Chapter 6. We used a custom robot based on the Pololu Romi chassis, equipped with a 3D-printed 2-DOF manipulator arm. The inter-robot communication was established using the MQTT protocol, and the robots were controlled with a microcontroller on the lower level and a Raspberry Pi on the higher level. The original two-robot system was revised into a three-robot system, where goal of the additional robot was to explore the environment and provide information to the other robots. This robot is therefore called the scout robot. The mobile robot control system of the multi-cobot system has a synchronous and asynchronous path planning strategy. Experimental results show that the robots can successfully transport objects using the scout robot's knowledge of the environment, in both synchronous and asynchronous path planning strategies. Using the implemented system, we also verified the effectiveness of

the two chosen topology configurations in the real-world environment.

In summary, we have presented a complete package of autonomous cobots for cooperative object transportation tasks. The system includes the design of the control, navigation, and communication systems, as well as a decentralized coordination strategy based on graph Laplacian to model connectivity and interactions among the robots. The system was validated in simulation and real-world environments, demonstrating its performance and robustness in various scenarios.

7.2 Future Work

The work presented in this thesis opens up several avenues for future research. First, limited by the resources and time constraints, we could not complete the planned 2-DOF manipulator arm. The infrastructure for the training of the manipulator arm controller model has been established, but we lacked details in the model to accomplish a meaningful training. The robot network could also benefit from a better-performant wireless mesh technology, such as Zigbee or batman-adv, such that the communication is distributed at the interface level instead of message level. The current system is also limited by the lack of a robust localization system, which can be addressed by integrating complex sensors such as LiDAR, and by employing advanced SLAM algorithms. The system can also be improved by incorporating more sophisticated path planning algorithms, such as RRT and A*. Finally, the system can be extended to include more robots, and tested in more complex and dynamic environments to evaluate its performance and robustness in real-world scenarios.

References

[1] C. S. Franklin, E. G. Dominguez, J. D. Fryman, and M. L. Lewandowski, "Collaborative robotics: New era of human–robot cooperation in the workplace," *Journal of Safety Research*, vol. 74, pp. 153–160, Sep. 1, 2020, ISSN: 0022-4375. DOI: 10.1016/j.jsr.2020.06.013.

[2] F. Sherwani, M. M. Asad, and B. Ibrahim, "Collaborative Robots and Industrial Revolution 4.0 (IR 4.0)," in *2020 International Conference on Emerging Trends in Smart Technologies (ICETST)*, Mar. 2020, pp. 1–5. DOI: 10.1109/ICETST49965.2020.9080724.

[3] V. D. Simone, V. D. Pasquale, V. Giubileo, and S. Miranda, "Human-Robot Collaboration: An analysis of worker's performance," *Procedia Computer Science*, 3rd International Conference on Industry 4.0 and Smart Manufacturing, vol. 200, pp. 1540–1549, Jan. 1, 2022, ISSN: 1877-0509. DOI: 10.1016/j.procs.2022.01.355.

[4] R. Gervasi, L. Mastrogiacomo, and F. Franceschini, "A conceptual framework to evaluate human-robot collaboration," *The International Journal of Advanced Manufacturing Technology*, vol. 108, pp. 841–865, 2020.

[5] K. Zhou, G. Ebenhofer, C. Eitzinger, *et al.*, "Mobile manipulator is coming to aerospace manufacturing industry," in *2014 IEEE International Symposium on Robotic and Sensors Environments (ROSE) Proceedings*, Oct. 2014, pp. 94–99. DOI: 10.1109/ROSE.2014.6952990.

[6] A. S. Sahan, S. Kathiravan, M. Lokesh, and R. Raffik, "Role of Cobots over Industrial Robots in Industry 5.0: A Review," *2nd International Conference on Advancements in Electrical, Electronics, Communication, Computing and Automation, ICAECA 2023*, 2023. DOI: 10.1109/ICAECA56562.2023.10201199.

[7] V. Villani, F. Pini, F. Leali, C. Secchi, and C. Fantuzzi, "Survey on Human-Robot Interaction for Robot Programming in Industrial Applications," *IFAC-PapersOnLine*, 16th IFAC Symposium on Information Control Problems in Manufacturing INCOM 2018, vol. 51, no. 11, pp. 66–71, Jan. 1, 2018, ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2018.08.236.

[8] A. Realyvásquez-Vargas, K. Cecilia Arredondo-Soto, J. Luis García-Alcaraz, B. Yail Márquez-Lobato, and J. Cruz-García, "Introduction and configuration of a collaborative robot in an assembly task as a means to decrease occupational risks and increase efficiency in a manufacturing company," *Robotics and Computer-Integrated Manufacturing*, vol. 57, pp. 315–328, Jun. 1, 2019, ISSN: 0736-5845. DOI: 10.1016/j.rcim.2018.12.015.

[9] Z. Liu and M. S. Miah, "Cooperative Object Transportation Using Autonomous Networked Cobots: A Distributed Approach," in *2024 IEEE International Symposium on Robotic and Sensors Environments (ROSE)*, Jun. 2024, pp. 1–7. DOI: 10.1109/ROSE62198.2024.10590934.

[10] J. Hua, L. Zeng, G. Li, and Z. Ju, "Learning for a Robot: Deep Reinforcement Learning, Imitation Learning, Transfer Learning," *Sensors*, vol. 21, no. 4, p. 1278, 4 2021, ISSN: 1424-8220. DOI: 10.3390/s21041278.

[11] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction* (Adaptive Computation and Machine Learning Series), Second edition. Cambridge, Massachusetts: The MIT Press, 2018, 526 pp., ISBN: 978-0-262-03924-6.

[12] The MathWorks Inc., *Reinforcement Learning Agents*, The MathWorks Inc., 2023. [Online]. Available: https://www.mathworks.com/help/reinforcement-learning/ug/create-agents-for-reinforcement-learning.html.

[13] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, May 1, 1992, ISSN: 1573-0565. DOI: 10.1007/BF00992698.

[14] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.* "Playing Atari with Deep Reinforcement Learning." arXiv: 1312.5602 [cs]. (Dec. 19, 2013), pre-published.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 7540 Feb. 2015, ISSN: 1476-4687. DOI: 10.1038/nature14236.

[16] N. M. Gomes, F. N. Martins, J. Lima, and H. Wörtche, "Deep Reinforcement Learning Applied to a Robotic Pick-and-Place Application," pp. 19–21, 2021. DOI: 10.3390/automation3010011.

[17] A. Ghadirzadeh, X. Chen, W. Yin, Z. Yi, M. Bjorkman, and D. Kragic, "Human-centered collaborative robots with deep reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 566–571, Apr. 1, 2021, ISSN: 23773766. DOI: 10.1109/LRA.2020.3047730. arXiv: 2007.01009.

[18] T. P. Lillicrap, J. J. Hunt, A. Pritzel, *et al.* "Continuous control with deep reinforcement learning." arXiv: 1509.02971 [cs, stat]. (Jul. 5, 2019), pre-published.

[19] M. El-Shamouty, X. Wu, S. Yang, M. Albus, and M. F. Huber, "Towards Safe Human-Robot Collaboration Using Deep Reinforcement Learning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, May 2020, pp. 4899–4905. DOI: 10.1109/ICRA40945.2020.9196924.

[20] Q. Liu, Z. Liu, B. Xiong, W. Xu, and Y. Liu, "Deep reinforcement learning-based safe interaction for industrial human-robot collaboration using intrinsic reward function," *Advanced Engineering Informatics*, vol. 49, p. 101 360, Aug. 1, 2021, ISSN: 1474-0346. DOI: 10.1016/j.aei.2021.101360.

[21] R. Zhang, Q. Lv, J. Li, J. Bao, T. Liu, and S. Liu, "A reinforcement learning method for human-robot collaboration in assembly tasks," *Robotics and Computer-Integrated Manufacturing*, vol. 73, p. 102 227, Feb. 1, 2022, ISSN: 0736-5845. DOI: 10.1016/j.rcim.2021.102227.

[22] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," 2020.

[23] A. Shafti, J. Tjomsland, W. Dudley, and A. A. Faisal, "Real-World Human-Robot Collaborative Reinforcement Learning," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2020, pp. 11 161–11 166. DOI: 10.1109/IROS45743.2020.9341473.

[24] L. Roveda, J. Maskani, P. Franceschi, *et al.*, "Model-Based Reinforcement Learning Variable Impedance Control for Human-Robot Collaboration," 2020. DOI: 10.1007/s10846-020-01183-3.

[25] V. Lumelsky and A. Stepanov, "Dynamic path planning for a mobile automaton with limited information on the environment," *IEEE Transactions on Automatic Control*, vol. 31, no. 11, pp. 1058–1063, Nov. 1986, ISSN: 1558-2523. DOI: 10.1109/TAC.1986.1104175.

[26] K. Taylor and S. M. LaValle, "Intensity-based navigation with global guarantees," *Autonomous Robots*, vol. 36, no. 4, pp. 349–364, Apr. 1, 2014, ISSN: 1573-7527. DOI: 10.1007/s10514-013-9356-x.

[27] I. Kamon, E. Rivlin, and E. Rimon, "A new range-sensor based globally convergent navigation algorithm for mobile robots," in *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 1, Apr. 1996, 429–435 vol.1. DOI: 10.1109/ROBOT.1996.503814.

[28] K. N. McGuire, G. C. H. E. de Croon, and K. Tuyls, "A comparative study of bug algorithms for robot navigation," *Robotics and Autonomous Systems*, vol. 121, p. 103 261, Nov. 1, 2019, ISSN: 0921-8890. DOI: 10.1016/j.robot.2019.103261.

[29] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, Jul. 1968, ISSN: 2168-2887. DOI: 10.1109/TSSC.1968.300136.

[30] A. Stentz, "Optimal and efficient path planning for partially-known environments," in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, May 1994, 3310–3317 vol.4. DOI: 10.1109/ROBOT.1994.351061.

[31] S. Koenig and M. Likhachev, "D* lite," in *Eighteenth National Conference on Artificial Intelligence*, 2002, pp. 476–483.

[32] Q. Yao, Z. Zheng, L. Qi, *et al.*, "Path Planning Method With Improved Artificial Potential Field—A Reinforcement Learning Perspective," *IEEE Access*, vol. 8, pp. 135 513–135 523, 2020, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3011211.

[33] Q. Zhu, Y. Yan, and Z. Xing, "Robot Path Planning Based on Artificial Potential Field Approach with Simulated Annealing," in *Sixth International Conference on Intelligent Systems Design and Applications*, vol. 2, Oct. 2006, pp. 622–627. DOI: 10.1109/ISDA.2006.253908.

[34] Z. Wu, J. Dai, B. Jiang, and H. R. Karimi, "Robot path planning based on artificial potential field with deterministic annealing," *ISA Transactions*, vol. 138, pp. 74–87, Jul. 1, 2023, ISSN: 0019-0578. DOI: 10.1016/j.isatra.2023.02.018.

[35] P. Vadakkepat, K. C. Tan, and W. Ming-Liang, "Evolutionary artificial potential fields and their application in real time robot path planning," in *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No.00TH8512)*, vol. 1, Jul. 2000, 256–263 vol.1. DOI: 10.1109/CEC.2000.870304.

[36] F. Bounini, D. Gingras, H. Pollart, and D. Gruyer, "Modified artificial potential field method for online path planning applications," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, Jun. 2017, pp. 180–185. DOI: 10.1109/IVS.2017.7995717.

[37] K. Inoue and T. Nakajima, "Cooperative object transportation by multiple robots with their own objective tasks," *Journal of the Robotics Society of Japan*, vol. 19, no. 7, pp. 888–896, 2001.

[38] F. Dong, B. Yu, X. Zhao, S. Chen, and H. Liu, "An Evenly Partition Approach to the Modeling and Constraint-Following Control for the Spatial Cooperative Dual-Robot-System," *Journal of Dynamic Systems, Measurement, and Control*, vol. 145, no. 111003, Oct. 5, 2023, ISSN: 0022-0434. DOI: 10.1115/1.4062956.

[39] J. E. Inglett and E. J. Rodríguez-Seda, "Object transportation by cooperative robots," in *SoutheastCon 2017*, Mar. 2017, pp. 1–6. DOI: 10.1109/SECON.2017.7925348.

[40] M.-H. Wu, A. Konno, and M. Uchiyama, "Cooperative object transportation by multiple humanoid robots," in *2011 IEEE/SICE International Symposium on System Integration (SII)*, Dec. 2011, pp. 779–784. DOI: 10.1109/SII.2011.6147547.

[41] S. Kajita, F. Kanehiro, K. Kaneko, *et al.*, "Biped walking pattern generation by using preview control of zero-moment point," in *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, vol. 2, IEEE, 2003, pp. 1620–1626.

[42] L. Hawley and W. Suleiman, "Control framework for cooperative object transportation by two humanoid robots," *Robotics and Autonomous Systems*, vol. 115, pp. 1–16, May 1, 2019, ISSN: 0921-8890. DOI: 10.1016/j.robot.2019.02.003.

[43] A. Rioux, C. Esteves, J.-B. Hayet, and W. Suleiman, "Cooperative Vision-Based Object Transportation by Two Humanoid Robots in a Cluttered Environment," *International Journal of Humanoid Robotics*, Aug. 25, 2017. DOI: 10.1142/S0219843617500189.

[44] M. Likhachev, G. J. Gordon, and S. Thrun, "ARA*: Anytime A* with provable bounds on sub-optimality," *Advances in neural information processing systems*, vol. 16, 2003.

[45] T. Hekmatfar, E. Masehian, and S. J. Mousavi, "Cooperative object transportation by multiple mobile manipulators through a hierarchical planning architecture," in *2014 Second RSI/ISM International Conference on Robotics and Mechatronics (ICRoM)*, Oct. 2014, pp. 503–508. DOI: 10.1109/ICRoM.2014.6990952.

[46] P. Chamoun and M. Lanthier, "Rigorous movement of convex polygons on a path using multiple robots," in *2012 IEEE International Symposium on Robotic and Sensors Environments Proceedings*, Nov. 2012, pp. 174–179. DOI: 10.1109/ROSE.2012.6402624.

[47] B. Sahu, P. K. Das, and R. Kumar, "A modified cuckoo search algorithm implemented with SCA and PSO for multi-robot cooperation and path planning," *Cognitive Systems Research*, vol. 79, pp. 24–42, Jun. 1, 2023, ISSN: 1389-0417. DOI: 10.1016/j.cogsys.2023.01.005.

[48] P. Das, A. K. Sadhu, R. R. Vyas, A. Konar, and D. Bhattacharyya, "Arduino based multi-robot stick carrying by Artificial Bee Colony optimization algorithm," in *Proceedings of the 2015 Third International Conference on Computer, Communication, Control and Information Technology (C3IT)*, Feb. 2015, pp. 1–6. DOI: 10.1109/C3IT.2015.7060152.

[49] A. K. Sadhu, P. Rakshit, and A. Konar, "A modified Imperialist Competitive Algorithm for multi-robot stick-carrying application," *Robotics and Autonomous Systems*, vol. 76, pp. 15–35, Feb. 1, 2016, ISSN: 0921-8890. DOI: 10.1016/j.robot.2015.11.010.

[50] Z. Wang, Y. Takano, Y. Hirata, and K. Kosuge, "Decentralized Cooperative Object Transportation by Multiple Mobile Robots with a Pushing Leader," in *Distributed Autonomous Robotic Systems 6*, R. Alami, R. Chatila, and H. Asama, Eds., Tokyo: Springer Japan, 2007, pp. 453–462, ISBN: 978-4-431-35873-2. DOI: 10.1007/978-4-431-35873-2_44.

[51] R. Pi, P. Cieślak, P. Ridao, and P. J. Sanz, "TWINBOT: Autonomous Underwater Cooperative Transportation," *IEEE Access*, vol. 9, pp. 37 668–37 684, 2021, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3063669.

[52] M. Lippi and A. Marino, "Cooperative Object Transportation by Multiple Ground and Aerial Vehicles: Modeling and Planning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018, pp. 1084–1090. DOI: 10.1109/ICRA.2018.8460778.

[53] F. Chung, L. Lu, and V. Vu, "Spectra of random graphs with given expected degrees," *Proceedings of the National Academy of Sciences*, vol. 100, no. 11, pp. 6313–6318, May 27, 2003. DOI: 10.1073/pnas.0937490100.

[54] F. R. Chung, *Spectral Graph Theory*. American Mathematical Soc., 1997, vol. 92.

[55] M. Abdi, E. Ghorbani, and W. Imrich. "Regular Graphs with Minimum Spectral Gap." arXiv: 1907.03733. (Aug. 7, 2020), pre-published.

[56] H. Taheri, B. Qiao, and N. Ghaeminezhad, "Kinematic model of a four mecanum wheeled mobile robot," *International journal of computer applications*, vol. 113, no. 3, pp. 6–9, 2015.

[57] J. Denavit and R. S. Hartenberg, "A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices," *Journal of Applied Mechanics*, vol. 22, no. 2, pp. 215–221, Jun. 4, 2021, ISSN: 0021-8936. DOI: 10.1115/1.4011045.

[58] Y. Zhang, Y. Li, and X. Xiao, "A novel kinematics analysis for a 5-DOF manipulator based on KUKA youBot," in *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Dec. 2015, pp. 1477–1482. DOI: 10.1109/ROBIO.2015.7418979.

[59] D. Raina and S. K. Saha, "AI-Based Modeling and Control of Robotic Systems: A Brief Tutorial," in *2021 3rd International Conference on Robotics and Computer Vision (ICRCV)*, Aug. 2021, pp. 45–51. DOI: 10.1109/ICRCV52986.2021.9546974.

[60] R. Bischoff, U. Huggenberger, and E. Prassler, "KUKA youBot - a mobile manipulator for research and education," in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 1–4. DOI: 10.1109/ICRA.2011.5980575.

[61] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. John Wiley & Sons, 2020.

[62] D. Di Vito, C. Natale, and G. Antonelli, "A Comparison of Damped Least Squares Algorithms for Inverse Kinematics of Robot Manipulators *," *IFAC-PapersOnLine*, 20th IFAC World Congress, vol. 50, no. 1, pp. 6869–6874, Jul. 1, 2017, ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2017.08.1209.

[63] S. R. Buss, "Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods," *IEEE Journal of Robotics and Automation*, vol. 17, no. 1-19, p. 16, 2004.

[64] E. Rohmer, S. P. N. Singh, and M. Freese, "CoppeliaSim (formerly V-REP): A versatile and scalable robot simulation framework," in *Proc. of the International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[65] E. Coumans and Y. Bai, *PyBullet, a Python module for physics simulation for games, robotics and machine learning*, 2016–2021. [Online]. Available: http://pybullet.org.

[66] R. Smith *et al.*, "Open dynamics engine," 2005.

[67] M. Grieves and J. Vickers, "Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems," in *Transdisciplinary Perspectives on Complex Systems: New Findings and Approaches*, F.-J. Kahlen, S. Flumerfelt, and A. Alves, Eds., Cham: Springer International Publishing, 2017, pp. 85–113, ISBN: 978-3-319-38756-7. DOI: 10.1007/978-3-319-38756-7_4.

[68] S. Ahmed, A. Topalov, and N. Shakev, "A robotized wireless sensor network based on MQTT cloud computing," in *2017 IEEE International Workshop of Electronics, Control, Measurement, Signals and Their Application to Mechatronics (ECMSM)*, May 2017, pp. 1–6. DOI: 10.1109/ECMSM.2017.7945897.

[69] C. A. Garcia, W. Montalvo-Lopez, and M. V. Garcia, "Human-Robot Collaboration Based on Cyber-Physical Production System and MQTT," *Procedia Manufacturing*, International Conference on Industry 4.0 and Smart Manufacturing (ISM 2019), vol. 42, pp. 315–321, Jan. 1, 2020, ISSN: 2351-9789. DOI: 10.1016/j.promfg.2020.02.088.

[70] R. A. Atmoko and D. Yang, "Online Monitoring & Controlling Industrial Arm Robot Using MQTT Protocol," in *2018 IEEE International Conference on Robotics, Biomimetics, and Intelligent Computational Systems (Robionetics)*, Aug. 2018, pp. 12–16. DOI: 10.1109/ROBIONETICS.2018.8674672.

[71] L. L. Schumaker and P. Dierckx, "Curve and Surface Fitting with Splines," in *Mathematics of Computation*, vol. 63, Jul. 1994, p. 427. DOI: 10.2307/2153590. JSTOR: 2153590.

APPENDICES

Appendix A

Computer Simulation Walkthrough

This chapter walks through setting up and running simulations used in the thesis work. Specifically, the PC requirements are listed, and the process to set up the Python environment is explained. Then we walk through steps needed to run the simulation using KUKA YouBot and the digital twin simulation of the Romi robot. The related files are contained in the `src/sim` directory of the thesis repository. All commands used in this chapter are assumed to be run from this folder as the working directory.

A.1 Computer Requirements

The tools needed for the simulation requires an x86-64 (also known as amd64) architecture PC with GNU/Linux operating system. For other operating systems, we suggest using a virtual machine provided by VirtualBox or VMware. In particular, Ubuntu 22.04 LTS is used for the development and testing of the simulation. The dependent packages can be installed using the following command:

```
sudo apt install -y python3 python3-virtualenv python3-tk
```

Similar packages can be found on other distributions or installed using other installation methods.

In addition, the simulation uses CoppeliaSim, a proprietary software whose education version is free to use and available to download at `https://www.coppeliarobotics.com/`. In our thesis work, we use CoppeliaSim Edu 4.7.0 rev 4, but the simulation should work with other versions as well.

Lastly, the digital twin simulation connects to a MQTT broker. In our work, we use NanoMQ launched from a Docker container. This is automated with the `mqttbroker.sh` script. To install Docker, follow the instructions at `https://docs.docker.com/engine/install/`.

A.2 Setting up Python Environment

To avoid affecting Python packages used in other projects and the operating system, we use a virtual environment. After navigating to the `src/sim` directory, run

```
virtualenv venv --clear
```

This establishes a `venv` folder, which contains the interpreter and packages specific to the simulation. To complete the setup, install all the required packages using the `requirements.txt` file, which lists the packages and their versions.

```
venv/bin/python3 -m pip install -r requirements.txt
```

A.3 Running KUYA YouBot Simulation

The KUKA YouBot simulation is automated with the `youbot_sim.py` script. After launching CoppeliaSim, simply run the script with

```
./youbot_sim.py
```

This script loads the correct scene in the simulation software, as seen in Fig. A.1 and executes the main routine. After the transportation task is finished, the simulation stops

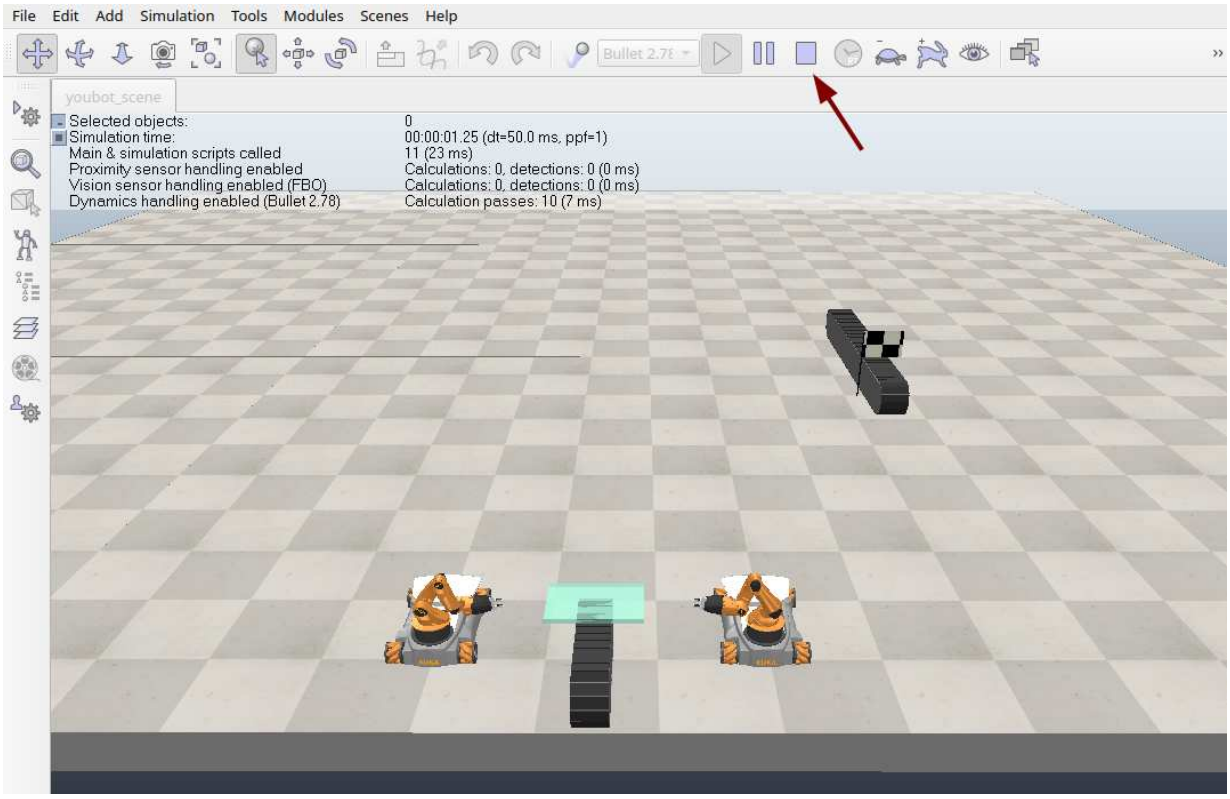


Figure A.1: KUKA YouBot simulation in CoppeliaSim

and the Python script exits. Simulation can also be stopped manually by pressing the stop button in CoppeliaSim, which is pointed out by an arrow in Fig. A.1. This is preferred over hitting Ctrl+C in the Python script, as it prevents leaving the simulation in an unstable state.

A.4 Running Digital Twin Simulation

Similarly, the digital twin simulation of the Romi robot uses the `romi_sim.py` script, which shows a scene with the Romi robot and the environment like in Fig. 5.9.

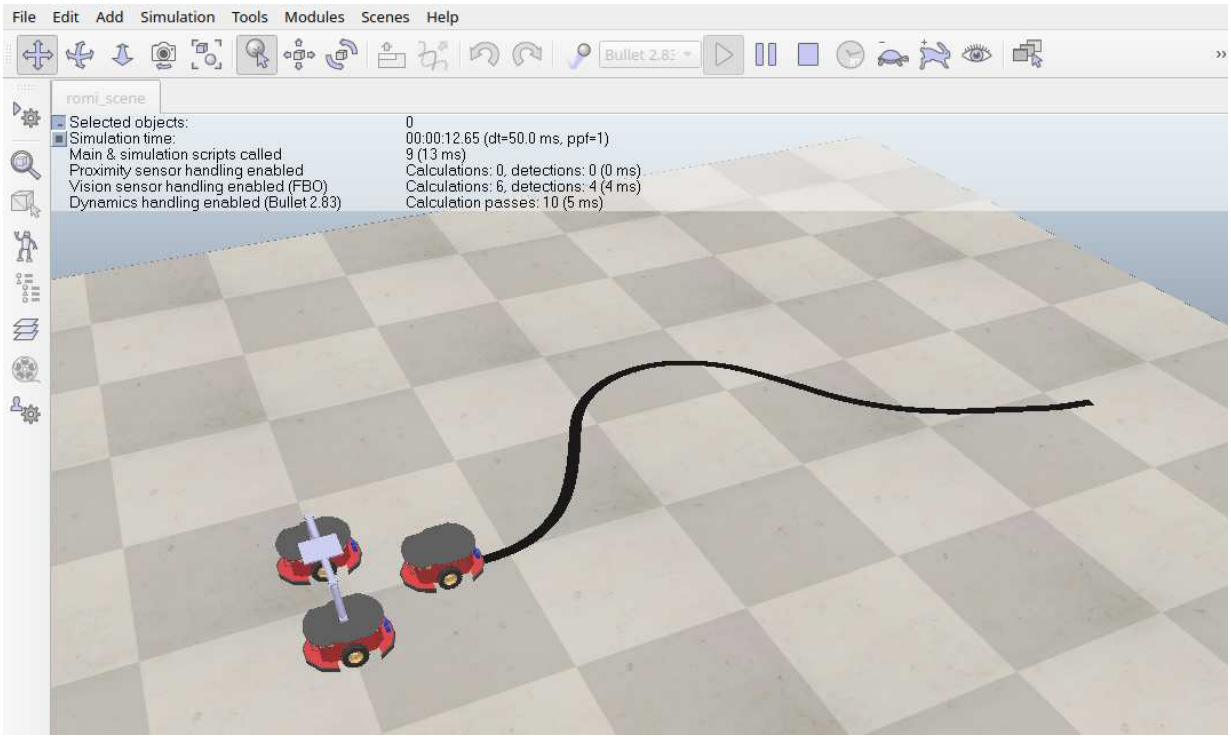


Figure A.2: Digital twin simulation of the Romi robot in CoppeliaSim

```
./romi_sim.py
```

However, simply running `romi_sim.py` does not set the simulation in motion. Two more processes need to be running in separate terminal sessions. The first process is the MQTT broker, which can be launched using the `mqtt_broker.sh` script.

```
./mqtt_broker.sh
```

The other process is the top-level path planner from `main_sync.py` or `main_async` from `src/impl_rpi` codebase, whose execution is explained in Appendix B. The three processes should be running simultaneously to see the Romi robot moving in the simulation. They would also need to be stopped manually when the simulation is no longer needed. For `romi_sim.py` in particular, the GUI stop button is also preferred over Ctrl+C in the Python script.

Appendix B

Robot Implementation Walkthrough

This chapter walks through preparing and replicating the robot implementation used in the thesis work.

B.1 Required Parts

Table B.1 lists the parts needed to realize the robot implementation. The quantities for each item are minimum numbers required for a group of three robots, two of which are equipped with manipulator arms. Items listed under *Romi Robot Assembly* can be purchased from Pololu Robotics and Electronics at `https://www.pololu.com/category/202/romi-chassis-and-accessories`. The parts for *Manipulator Arm Assembly* are for two 1-DOF lifter arms as shown in demonstration pictures and videos, since the design of the desired 2-DOF arm is not finalized at the time of writing. Detailed information on the 3D printing is explained in later sections. Parts listed under *Single Board Computer Expansion* and *Optional Tools and Accessories* are available from various vendors and retailers.

Finally, all experiments using the robotic system require a computer that kicks off the main routine and shuts off the system at certain times. The computer should be an x86-64

Table B.1: Parts List for Robot Implementation

Item	Quantity
Romi Robot Assembly	
Romi Chassis kit	3
Romi 32U4 control board	3
Pre-assembled gearmotor and encoder	6
Romi Chassis expansion plate	3
Extra chassis ball caster kit	3
Reflectance sensor array	3
Breadboard with mounting holes	3
3/4" standoffs	6
1" standoffs	6
1-1/2" standoffs	6
AA batteries	18
#2-56 screws of various lengths	20
#2-56 nuts	20
USB A to Micro-B cable	1
Manipulator Arm Assembly	
Micro servo motor with feedback	2
Set of 3D-printed parts for one arm	2
Single Board Computer Expansion	
Raspberry Pi 5	3
2×20 stacking header	3
16GB microSD card	3
Optional Tools and Accessories	
27W USB-C power supply	1
USB to Ethernet adapter	1
WiFi router	1

(also known as amd64) architecture machine with GNU/Linux operating system, and have at least one WiFi interface. In our experiments, we installed Ubuntu 22.04 LTS natively instead of using a virtual machine, since extra steps would be needed to tunnel the WiFi network to the virtual machine. The dependent Python packages can be installed with:

```
sudo apt install -y python3 python3-virtualenv python3-tk
```

Furthermore, the computer also needs Docker, whose installation instructions can be found at `https://docs.docker.com/engine/install/`, and Visual Studio Code (VSCode), which can be installed at `https://code.visualstudio.com/docs/setup/linux`.

B.2 Install Microcontroller Firmware

We start by installing the firmware to the Romi 32U4 control board by compiling the source code and uploading it to the board. The source code of the firmware is under the `src/impl_mcu` directory of the thesis repository. The compilation of the source code uses VSCode with the PlatformIO extension, which can be installed with

```
code --install-extension platformio.platformio-ide
```

After opening the `src/impl_mcu` folder in VSCode, the extension activates then downloads the necessary tools and libraries. The PlatformIO logo also appears on the left sidebar as seen in Fig. B.1.

Using the USB cable, connect the micro USB port on the Romi 32U4 control board to the computer. Then, click on the *Upload* button on the PlatformIO toolbar to compile and upload the firmware to the board. We recommend installing the firmware on all three Romi 32U4 control boards before assembling the robots.

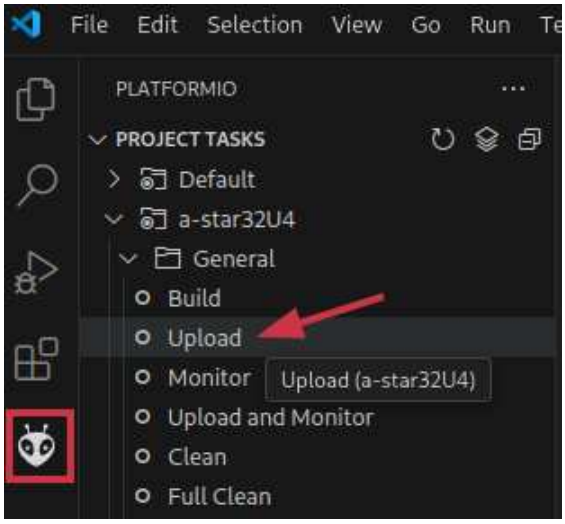


Figure B.1: PlatformIO Extension in Visual Studio Code

B.3 Mobile Robot Assembly

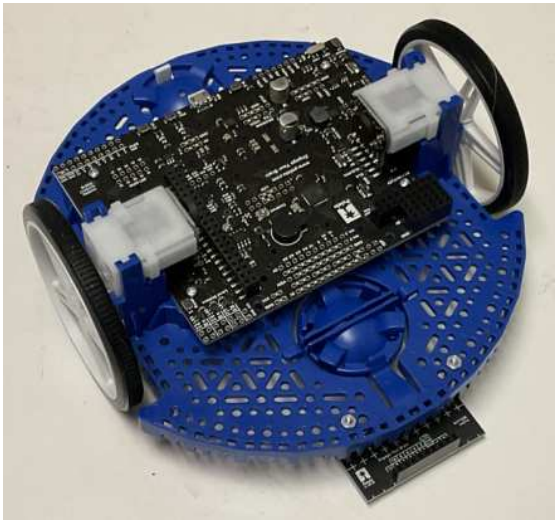
The assembly of the base chassis of the Romi robot is straightforward and can be done by following the instructions provided by Pololu at `https://www.pololu.com/docs/0J68/4`. Apart from the manipulator arm, additional parts for the robots used in the thesis work are the reflectance sensor array, top plate, and the breadboard that keeps the correct wiring connections.

The reflectance sensor array is mounted on the front of the robot under the top plate, such that it barely touches the ground, as seen in Fig. B.2(a). This can be done using two 3/4" standoffs, where the screws are on the bottom, so it does not drag on the ground.

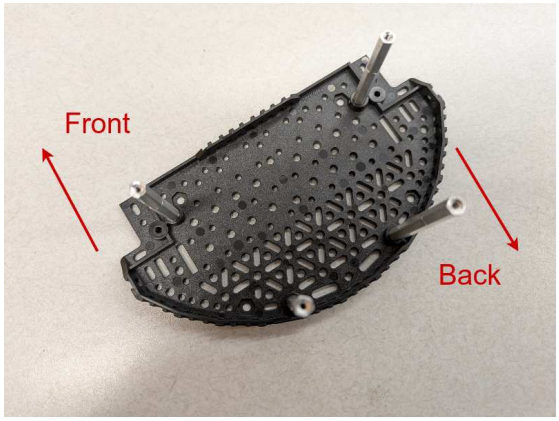
The top plate is elevated using 1" combined with 3/4" standoffs in the front, and 1-1/2" combined with 3/4" standoffs in the back, as shown in Fig. B.2(b). Use screw to connect the top plate, the standoffs, and the chassis expansion plate in the back, since the front standoffs touches the control board.

The breadboard is mounted on the base chassis directly in front of the control board, as seen in Fig. B.2(c). Along the center notch of the breadboard, there are eight mounting holes, and we found the second and seventh holes align with the base chassis plate, so we

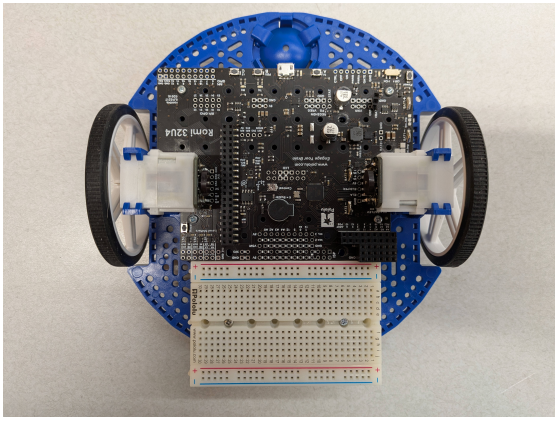
used two sets of 3/4" screws and nuts to secure the breadboard. Additionally, since there is a caster ball directly under the breadboard, the screws should be loose enough to allow the ball to move freely.



(a)



(b)



(c)

Figure B.2: (a) Reflectance sensor array mounted on romi Robot, (b) top plate with standoffs attached, (c) breadboard mounted on Romi Robot

Lastly, the connection between the reflectance sensor array and the control board is accomplished using the breadboard and jumper wires. Specifically, the wiring should connect

four nodes, including the 5V power, ground, and the two sensor outputs. The pins on the control board for the nodes are shown in Fig. B.3, where the wire labeled with "LEFT" and "RIGHT" should connect to pins that correspond to the left and right phototransistor outputs. The "SERVO" wire is used for the manipulator arm explained in the next section. On the reflectance sensor breakout board, the 5V and ground should go to pins labeled

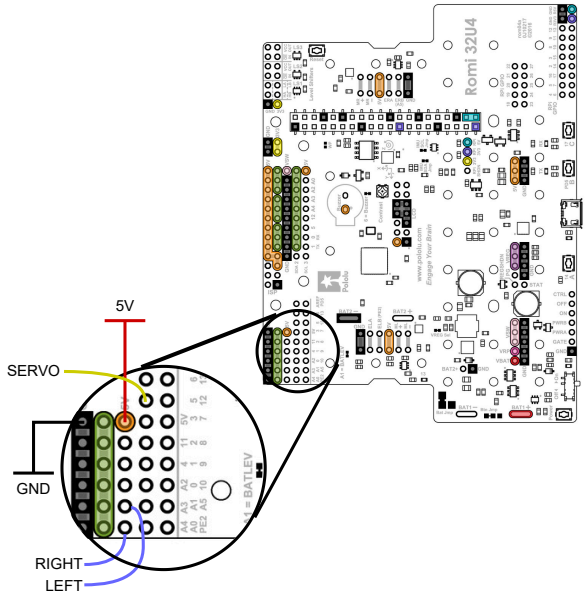


Figure B.3: Control board layout with wiring connections annotated

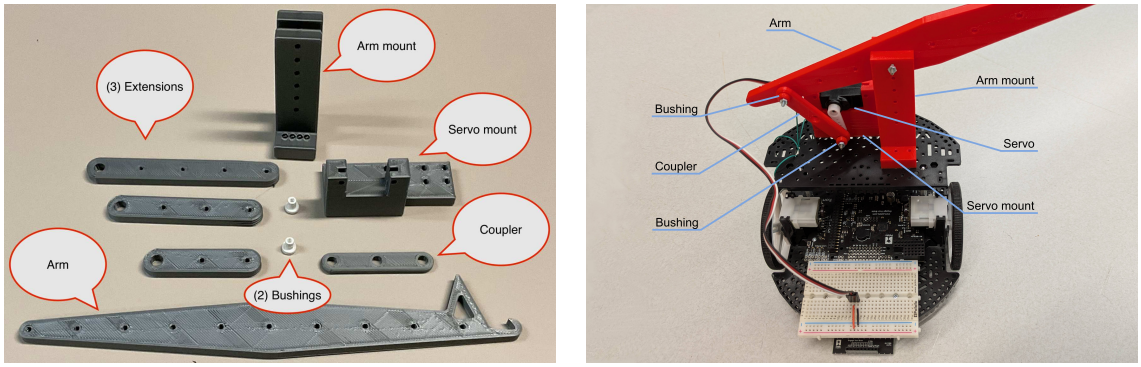
with VCC and GND, respectively. Among the 11 sensor pins, we found the No. 5 and 7 phototransistors work the best for the left and right sensors, respectively.

The assembly of the mobile robot can be repeated for the other two robots.

B.4 Manipulator Arm Assembly

Figure B.4(a) shows the 3D-printed parts for the manipulator arm assembly. For our robot implementation, the extension parts are not used. The step files for the parts are attached under the `stl` directory of the thesis repository.

The assembly of the manipulator arm is shown in Fig. B.4(b). The servo mount holds the servo motor in the bracket, while the arm mount has the arm piece connected with a screw, and the two parts are mounted next to each other. The end of the servo horn is attached to one end of the coupler, and the other end is attached to one of the holes on the arm piece. The attachment of the two ends is done using screws and nuts that are loosely tightened within the bushing, so the arm can move freely. The servo motor has standard three-pin connections, where the red wire is for power (5V), the black wire is for ground, and the white wire is for signal, labeled as "SERVO" in Fig. B.3.



(a) (b)

Figure B.4: (a) 3D-printed parts for manipulator arm assembly, (b) manipulator arm assembly on Romi Robot

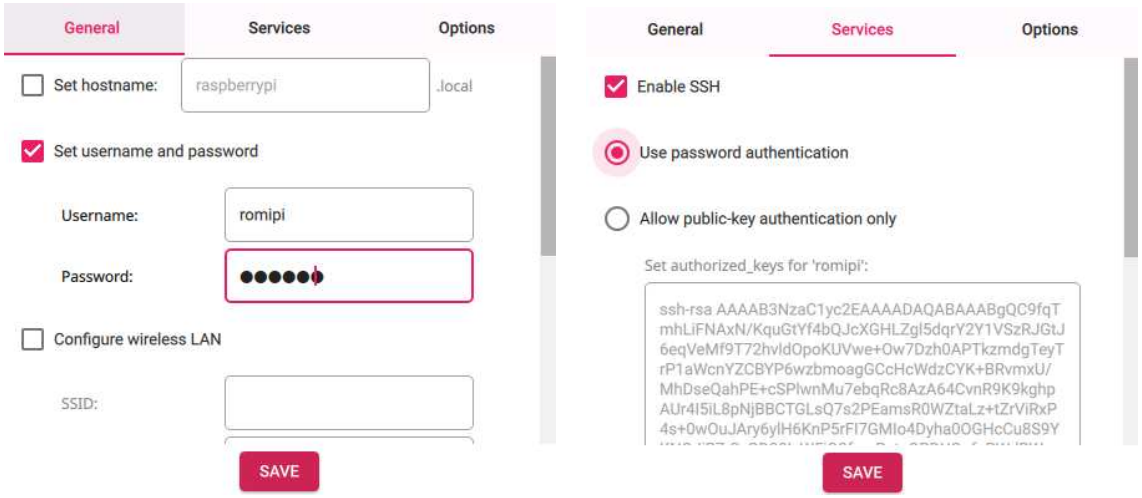
In the three-robot system shown in the thesis, two robots have the arm mounted facing opposite directions, while the third robot does not have the arm mounted.

B.5 Single Board Computer Setup

The Raspberry Pi 5 SBC is configured separately before mounting it on the robot.

The first step is to prepare the microSD card with the Raspberry Pi OS. Plug the microSD card into the computer and flash the Raspberry Pi OS Lite image using the Raspberry Pi Imager software, which can be downloaded from `https://www.raspberrypi.`

`com/software/`. We choose "Raspberry Pi OS Lite (64-bit)" under "Rapsberry Pi OS (other)" as the operating system. Clicking on "Next" pops up a dialog where we choose "Edit Settings" to customize the OS. Under "General", set username and password both to `romipi`, and under "Services", enable SSH that uses password authentication, as seen in Fig. B.5. After flashing the image, remove the microSD card from the computer and plug



(a) (b)

Figure B.5: Raspberry Pi Imager custom settings

it into the Raspberry Pi. Connect the Raspberry Pi to a USB-C power supply and wait for it to boot up.

Configure a USB to Ethernet adapter to share the internet connection from the computer to the Raspberry Pi. This can be done by running `nmtui` on the computer, selecting "Edit a connection", choosing the USB to Ethernet adapter, and making "IPv4 Configuration" to "Shared", as shown in Fig. B.6. The IP address of the Raspberry Pi can be found with `arp -a`. Then copy the SSH key to the Raspberry Pi for passwordless login:

```
ssh-copy-id romipi@xxx.xxx.xxx.xxx
```

After SSH session is established with `ssh romipi@xxx.xxx.xxx.xxx`, run `sudo nmtui`

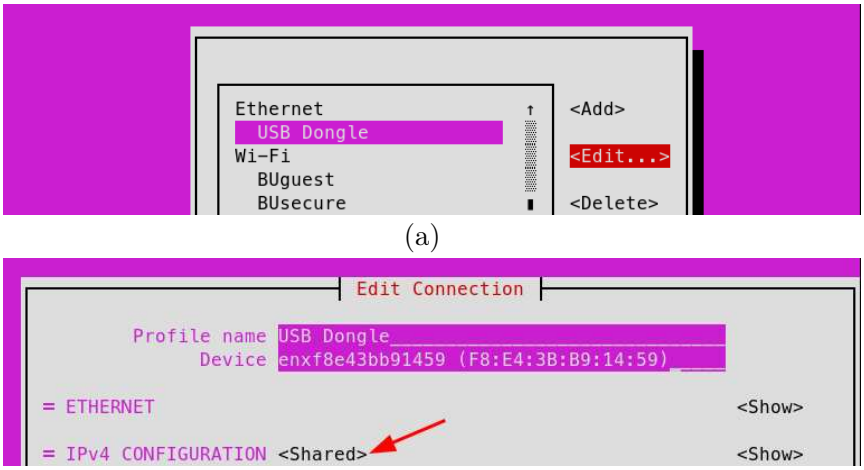
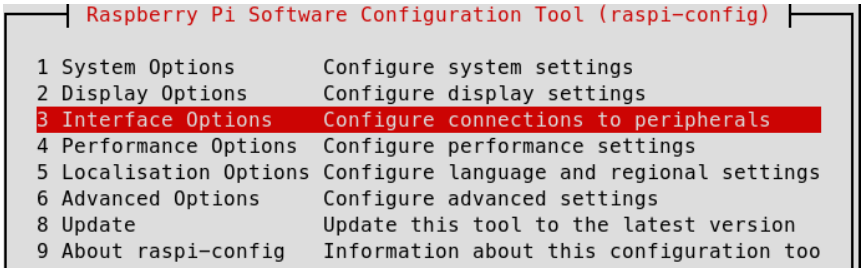


Figure B.6: USB to Ethernet adapter configuration

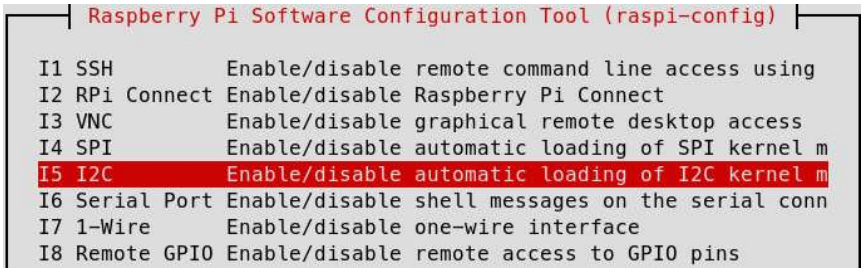
and select "Activate a connection" to connect to the desired WiFi network. Upon successful connection, exit the program, run `ip addr` and remember the IP address over the WiFi.

Next we configure the I2C interface on the Raspberry Pi. Within the same SSH session, run `sudo raspi-config`, and select "Interface Options" following "I2C" to enable the I2C interface, shown in Fig. B.7. Reboot the Raspberry Pi with `sudo reboot` to apply the changes. To check if the target I2C is enabled, run `ls /dev/i2c*` and `/dev/i2c-1` should be present.

Lastly, we configure the Python environment needed for the robot implementation. This part requires the Raspberry Pi to be connected to the Internet, from the WiFi router or the computer sharing the internet connection. Similar to the simulation, we use a virtual environment to avoid affecting Python packages used in the operating system. The source code needed for the Raspberry Pi is under the `src/impl_rpi` directory of the thesis repository. It also contains the `requirements.txt` file, which describes the dependencies for the Python environment. It is therefore more convenient to copy the whole folder to the Raspberry Pi. In a terminal session of the host PC (make sure not the SSH session), run



(a)



(b)

Figure B.7: Raspberry Pi Configuration for I2C Interface

```
scp -r src/impl_rpi romipi@xxx.xxx.xxx.xxx:/home/romipi/
```

Then enter an SSH session to the Raspberry Pi, enter the same `impl_rpi` folder, run

```
sudo apt install -y i2c-tools python3 python3-virtualenv
virtualenv venv --clear
venv/bin/python3 -m pip install -r requirements.txt
```

The Raspberry Pi is now ready to be mounted on the robot. Align the GPIO pins of the Raspberry Pi with the stacking header, and connect them to the 2×20 pins on the Romi 32U4 control board. Figure B.8 shows the Raspberry Pi mounted on the Romi robot.

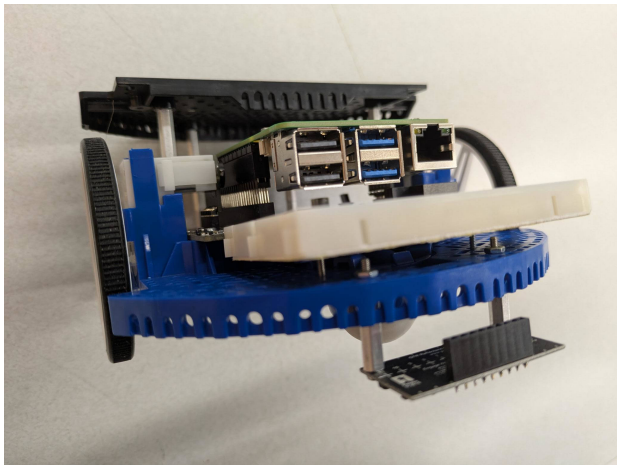


Figure B.8: Raspberry Pi Mounted on Romi Robot

The setup process can be repeated for the other two robots.

B.6 Running the Robot System

Check all the necessary parts are prepared and assembled as described in the previous sections. Figure B.9 shows a complete set of the components for the robot system. We used a standalone WiFi router and a 3D-printed cup as the payload for the transporter robots, which are optional.

Turn on all three robots and make sure they are connected to the same WiFi network as the computer. The execution of a complete robot system uses five terminal sessions, including the MQTT broker, SSH sessions to the SBC on each robot, and the main routine on the computer. All the relevant source code is under the `src/impl_rpi` directory, so we use this as the default working directory.

The MQTT broker is launched using the `mqtt_broker.sh` script.

```
./mqtt_broker.sh
```

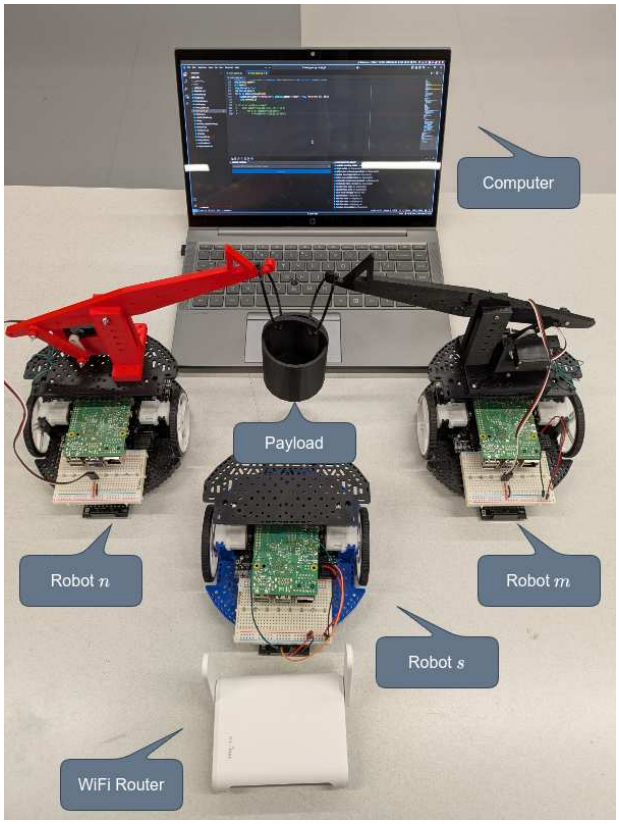


Figure B.9: Complete Robot System

Within the SSH session to the Raspberry Pi, a subroutine program is executed. For the scout robot (robot s), run

```
./scout.py romi_front
```

For the left transporter robot (robot m), run

```
./transporter.py romi_left
```

For the right transporter robot (robot n), run

```
./transporter.py romi_right
```

Finally, the main routine is executed on the computer, which has a synchronous and an asynchronous version. There should be only one main routine running at a time. To execute with the synchronous path planning, run

```
./main_sync.py
```

To execute with the asynchronous path planning, run

```
./main_async.py
```

The main routine serve as the entry point of the autonomy, so the robots should start moving as designed. On the computer, a window also pops up showing the real-time visualization of the robots' positions, paths and orientations, as seen in Fig. B.10. The robot system can be stopped by simply closing the live plot window on the computer.

The two main routine programs also serve as the entry point for the digital twin simulation, and running either of them completes the missing piece as explained in Appendix A.

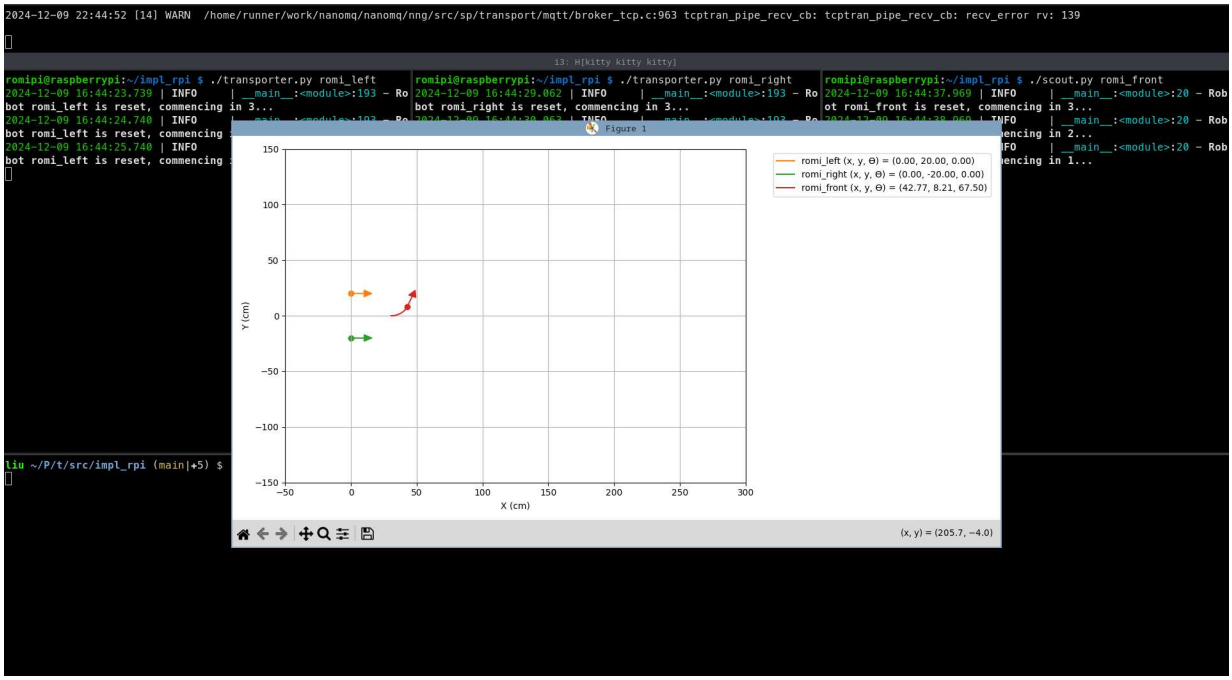


Figure B.10: Real-time Visualization of Robot System