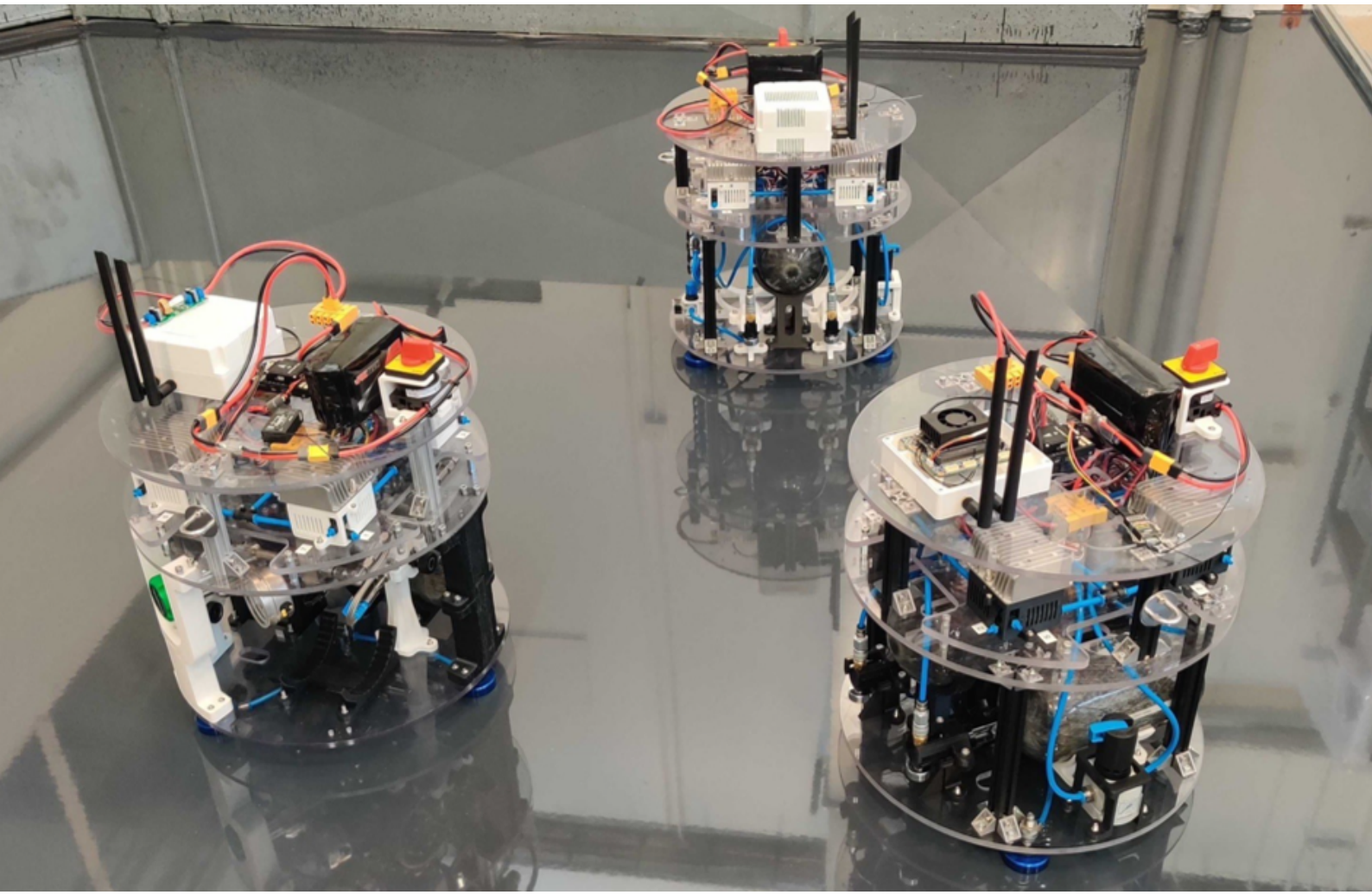
KTH ROYAL INSTITUTE
OF TECHNOLOGY

Degree Project in Technology

Second cycle, 30 credits

Predictive Controllers for Load Transportation in Microgravity Environments

SUJET PHODAPOL



Stockholm, Sweden, 2023

Predictive Controllers for Load Transportation in Microgravity Environments

SUJET PHODAPOL

Master's Programme, Systems, Control and Robotics, 120 credits
Date: November 18, 2023

Supervisor: Pedro Roque
Examiner: Dimos V. Dimarogonas
School of Electrical Engineering and Computer Science

Abstract

Space activities have been increasing dramatically in the past decades. As a result, the number of space debris has also increased significantly. Therefore, it is necessary to clean up and remove them to prevent a collision between space debris and spacecraft. In this thesis, we focus on load transportation using tethers, which connect multiple robots and loads together with lightweight cables. We propose a generalized framework to model and calculate the interaction force for the tethered multi-robot system. Then, we develop centralized and decentralized non-linear Model Predictive Control (MPC) controllers to complete a transportation task. Two simulators, a numerical and physical simulator, are presented and used to evaluate the performance of the controllers. The numerical simulator is used to verify the proposed model and evaluate the controllers for the ideal case. The physical simulator is then used to validate the performance of both centralized and decentralized controllers in real-time settings. Finally, we demonstrate how the proposed controllers perform in two and three-dimensional experiments.

Keywords

Model Predictive Control, Decentralized control, Multi-robot systems, Space technology,

Sammanfattning

Rymdaktiviteter har ökat dramatiskt under de senaste årtiondena. Som en följd av detta har mängden rymdskräp också ökat avsevärt. Därför är det nödvändigt att rensa upp och avlägsna detta skräp för att förhindra kollisioner mellan rymdskräp och rymdfarkoster. I denna rapport fokuserar vi på transporter av rymdobjekt som är sammanbundna via en lätt kabel. Vi föreslår en allmän metod för att modellera och beräkna interaktionskraften för det förenade multirobotsystemet. Sedan utvecklar vi centraliserad och decentraliserad icke-linjär modell-prediktiv reglering, MPC (eng. Model Predictive Control), för att uppnå transportuppgiften. Två simulatorer, en numerisk och en fysisk simulator, presenteras och används för att utvärdera styrsystemets prestanda. Den numeriska simuleringen används för att verifiera den föreslagna modellen och utforma styrsystemet för det idealiska fallet. Den fysiska simuleringen används sedan för att validera prestandan för både det centraliserade och decentraliserade styrsystem i realtid. Slutligen demonstrerar vi hur de föreslagna styrsystemen utför sig i tre- respektive två-dimensionella experiment.

Nyckelord

Modell-prediktiv reglering, Decentraliserad reglering, Multirobotsystem, Rymdteknologi

Acknowledgments

First of all, I would like to express my gratitude to Professor Dimos V. Dimarogonas for giving me the opportunity to do research on this interesting topic. I especially would like to thank my supervisor, Pedro Roque. He provided me with a lot of helpful feedback and suggestions during the project, which helped me to get the most out of this project. I also would like to thank PhD students in the DISCOWER project, especially, Elias Krantz, for helping me with the thruster system and experiments.

This project is done under the DISCOWER (Distributed Control in Weightless Environments) project in Space Robotics Laboratory.

Space (Robotics Lab), Stockholm
October 2023
Sujet Phodapol

Contents

List of Figures

List of Tables

List of acronyms and abbreviations

3DOF	three-degree-of-freedom
ADR	Active Debris Removal
BVP	Boundary Value Problem
CFHOC	Constrained Finite Horizon Optimal Control
DCM	Direction Cosine Matrix
ISS	International Space Station
LEO	Low Earth Orbit
LQR	Linear-Quadratic Regulator
MPC	Model Predictive Control

PID	Proportional–Integral–Derivative
PWM	Pulse Width Modulation
RK4	explicit Runge-Kutta 4th Order
ROS 2	Robot Operating System
SMC	Sliding-Mode Controller

Chapter 1

Introduction

1.1 Overview

"Imagination is infinite but space has its limits"

After more than 60 years of space missions, men have left a lot of space debris in our Low Earth Orbit (LEO) as shown in Fig. 1.1. In the near future, the number of space debris will continue to increase dramatically [1, 2]. As a result, there will be a higher chance of spacecraft and debris colliding. This risk can eventually cause the phenomena called Kessler syndrome [3]. This syndrome is a chain reaction of collisions between space debris and spacecrafts, which will generate more space debris and eventually make the space environment unusable. Therefore, it is an unavoidable mission to capture and remove space debris. There are several proposed methods for Active Debris Removal (ADR), such as using a net, harpoon, or robotic arm, and also contactless methods [4, 5]. Although contactless methods are promising, they are currently still in the early stage of development. Therefore, in this thesis, we will focus on contact-based methods, specifically the tethered method. With the contact method, it is more challenging to control the robot since it has to directly contact the space debris, which increases the likelihood of accidentally colliding and creating more debris fragments. In addition, the space debris is usually in an unknown state, which makes it more difficult to predict the states and then use them to control the robot. Another main challenge in controlling the robot is to manoeuvre the robot in a microgravity environment, where there is no air and residual gravity can be ignored [6]. In this specific environment, robots can only move by ejecting small masses in the opposite direction, through their thrusters, to the environment. Therefore,

the resources of the robot are limited and need to be optimized and allocated properly.

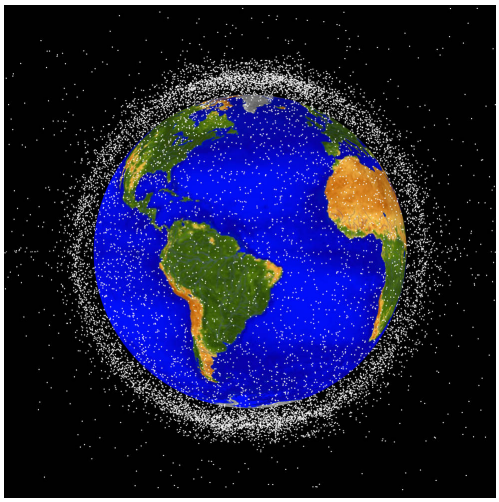


Figure 1.1: Visualization of space debris around LEO [7]

Due to the high cost of conducting experiments in space, it is necessary to test and verify the controller in ground-based testbeds before deployment. Therefore, a high-fidelity simulator and testbeds that can replicate the microgravity environment have been developed. The main challenge of imitating this environment comes from the presence of gravity and friction. Therefore, the testbed needs to be designed to eliminate these effects. Air-bearing-based designs are frequently utilized in floating platforms to minimize the friction between the platform on the substantially flat and smooth surface [8, 9]. These air bearings generate an air cushion by uniformly releasing the compressed air and levitating the platform. As a result, this floating platform can provide space-like dynamics in a planar motion.

Furthermore, the usage of multiple robots demonstrates the potential advantage of working together to transport a large payload due to the limited power of each robot [10, 11]. In this work, we focus primarily on achieving a collaborative task of load transportation. However, with a higher number of robots, the more complex the control problem is. Therefore, we need to develop a suitable controller to solve this problem. Additionally, cables are used to connect the load and the robot due to a number of advantages; first, compared to a rigid link, cables can be stored inside the robot's tight space and subsequently extended as required. Second, cables are significantly lightweight and flexible, which is suitable for the limited load capacity in the

space mission [12, 13]. However, the cable is not rigid, which makes the control problem more challenging.

1.2 Related work

Collaborative transportation is a challenging problem in the field of robotics [14, 15, 16, 17]. In this section, we will review the related work in the field of collaborative transportation and multi-agent control.

To achieve ADR and load transportation in space, many works have been presented. Philipp Behruzi et al [18] displays the tethered transportation on the slosh object on International Space Station (ISS). However, the controller is preprogrammed to actuate two active robots in order to move a passive load and the functional controller design has not yet been presented. Another research from H.T. Linskens et al. [19] proposes a simple Sliding-Mode Controller (SMC) method for the guidance and control system of the tethered system. The results demonstrate that SMC performs better than the classic Linear-Quadratic Regulator (LQR) in terms of propellant consumption. This work also shows the analysis of the tether model using lumped-mass model discretizing. However, the controller in this work is designed only for one robot. Also, the tether used in this work is very long (200 m), but we use only short cables in our work. Thus, we can ignore the model and vibration of the cable. Although there is still not much work on collaborative transportation in the space environment, there are several works from other applications, including ground, aerial, and legged robots, which can be applied to our system.

In aerial robotics system, the work from Roberto C. Sundin et al. [10] proposes predictive control frameworks, including centralized and decentralized controllers and analytical solutions for the equilibrium point for the load transportation of the drone systems. The experiment demonstrates the advantages of the centralized scheme in better tracking error but it needs more computational time compared to the decentralized one. Although this system has many similarities to our system, we can not directly use the method because the absence of gravity in the microgravity environment induces infinite equilibrium points. Another research from Taeyoung Lee [20] uses Proportional–Integral–Derivative (PID) control to achieve the transportation task; however, with PID, we cannot directly include desired constraints such as input limit in the controller.

For the legged robotics system, Flavio De Vincenti et al. [21] propose a centralized Model Predictive Control (MPC) controller for the locomotion and

manipulation task. The centralized controller is designed to calculate the states of each robot and send them to the local whole-body controller. Each robot and the payload is assumed to be a single rigid body, which can reduce the complexity of the problem. However, the controller paradigm is designed for direct contact between the robot and the payload. Therefore, it is not suitable for the tethered system. Another work from Jeeseop Kim et al. [22] proposes a centralized and distributed controller for the load transportation task. The centralized controller is designed to solve the whole interconnected system as a single optimal control problem and then calculate the control input for each robot. The distributed controller is designed to use information sharing between two robots to reduce the size of the problem. However, the controller is designed for the rigid link between the robot and the payload, which needs to be modified for the tethered system.

In this thesis, we will focus primarily on the problem of collaborative load transportation. To achieve this task, we need to first model the dynamics and interaction forces of the system. We propose a generalized framework to calculate the interaction force in the cable between the robot and load based on the number of robots. Then, a MPC controller, which is an optimization-based controller, is developed to solve the control problem. The main advantage of this controller is that it allows the controller to achieve the optimal control input within given constraints, which are important for space applications. Two simulators, a numerical simulator in Python and a physical simulator in Gazebo, are developed to evaluate the performance of the controller. The former simulator is used to verify the model and test the controller in an ideal environment. The latter is used to assess how well the controller performs in a more realistic setting, where the length of the cable can change and a communication protocol is included. The outcome of this project can be extended to the application for debris removal in space and also be applied to distributed control in multi-agent schemes.

The contributions of this project can be summarised as follows:

- We propose a generalized framework to model and calculate the interaction force for the n-agent tethered system.
- We propose a centralized and decentralized nonlinear MPC controller to achieve a load transportation task.
- We demonstrate the comparative study of the performance of the two controllers.

1.3 Structure of the thesis

The thesis is structured as follows: the theoretical concepts of attitude descriptions and MPC are presented in Chapter 2. In Chapter 3, the kinematic and dynamic models of the system are derived and explained. Control designs for both centralized and decentralized schemes are presented in Chapter 4. Chapter 5 shows the experimental setup and results. Next, in Chapter 6, the hardware description of the air carriages that will be used in future experiments is explained in detail. Chapter 7 summarizes our main results and outlines future research directions.

Chapter 2

Background

2.1 Quaternion

There are several ways to represent the orientation of the spacecraft or robot, such as Euler angles, rotation matrix, and quaternion. In this section, we will discuss a quaternion representation. The main advantage of using quaternion is that it does not suffer from the singularity and gimbal lock problem [23], which is the case for Euler angles. The quaternion is a four-element vector that is defined in scalar-vector quaternion convention as follows [24]:

$$\boldsymbol{q} = q_1\hat{\boldsymbol{i}} + q_2\hat{\boldsymbol{j}} + q_3\hat{\boldsymbol{k}} + q_4 \tag{2.1}$$

$$\boldsymbol{q} = \begin{bmatrix} q_1 & q_2 & q_3 & q_4 \end{bmatrix}^T \tag{2.2}$$

where q_1, q_2, q_3 are the vector part, and q_4 is the scalar part. Note that vectors are represented by small, bold letters. The hat symbol denotes the unit vector. Matrices are denoted by bold, capital letters. The quaternion can also be represented as an angle of rotation and axis of rotation as follows:

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{a}sin(\alpha/2) & cos(\alpha/2) \end{bmatrix}^T \tag{2.3}$$

where $\boldsymbol{a}$ is a normalized unit vector representing the axis of rotation and α is the angle of rotation. Also, a unit quaternion satisfies the following property:

$$||\boldsymbol{q}|| = 1 \tag{2.4}$$

In order to represent the attitude of the robot in the 3D space, one can convert a quaternion to a rotation matrix or Direction Cosine Matrix (DCM) $\boldsymbol{\Lambda}(\boldsymbol{q})$ as

follows [24]:

$$\boldsymbol{\Lambda}(\boldsymbol{q}) = (q_4^2 - \boldsymbol{q}_{1:3}^T \boldsymbol{q}_{1:3})\boldsymbol{I}_3 - 2q_4\boldsymbol{q}_{\boldsymbol{3}}^{\times} + 2\boldsymbol{q}_{\boldsymbol{1:3}}\boldsymbol{q}_{\boldsymbol{1:3}}^T \tag{2.5}$$

$$= \begin{bmatrix} q_1^2 - q_2^2 - q_3^2 + q_4^2 & 2(q_1q_2 + q_3q_4) & 2(q_1q_3 - q_2q_4) \\ 2(q_1q_2 - q_3q_4) & -q_1^2 + q_2^2 - q_3^2 + q_4^2 & 2(q_2q_3 + q_1q_4) \\ 2(q_1q_3 + q_2q_4) & 2(q_2q_3 - q_1q_4) & -q_1^2 - q_2^2 + q_3^2 + q_4^2 \end{bmatrix} \tag{2.6}$$

where $\boldsymbol{I}_3$ is a 3×3 identity matrix. The superscript $\times$ denotes a skew-symmetric of the vector, which is defined as follows:

$$\boldsymbol{\omega}^{\times} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \tag{2.7}$$

Additionally, quaternion can also be converted from the Euler angles in a 3-2-1 sequence as follows:

$$\begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = \begin{bmatrix} sin(\phi/2)cos(\theta/2)cos(\psi/2) - cos(\phi/2)sin(\theta/2)sin(\psi/2) \\ cos(\phi/2)sin(\theta/2)cos(\psi/2) + sin(\phi/2)cos(\theta/2)sin(\psi/2) \\ cos(\phi/2)cos(\theta/2)sin(\psi/2) - sin(\phi/2)sin(\theta/2)cos(\psi/2) \\ cos(\phi/2)cos(\theta/2)cos(\psi/2) + sin(\phi/2)sin(\theta/2)sin(\psi/2) \end{bmatrix} \tag{2.8}$$

where ϕ, θ, ψ are roll, pitch, and yaw angles, respectively. The conversion from the quaternion to the Euler angle is not unique. Therefore, one can use the following equation to convert the quaternion to the Euler angles:

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} atan2(2(q_4q_1 + q_2q_3), 1 - 2(q_1^2 + q_2^2)) \\ -\pi/2 + 2atan2(\sqrt{1 + 2(q_4q_2 - q_1q_3)}, \sqrt{1 - 2(q_4q_2 - q_1q_3)}) \\ atan2(2(q_4q_3 + q_1q_2), 1 - 2(q_2^2 + q_3^2)) \end{bmatrix} \tag{2.9}$$

2.1.1 Quaternion Algebra

In this section, we will discuss two operations of quaternion: addition and multiplication [25]. The addition of two quaternions is performed element-wise. It can be done by adding each element of the scalar and vector parts as

follows:

$$\boldsymbol{p} + \boldsymbol{q} = (p_1 + q_1)\hat{\boldsymbol{i}} + (p_2 + q_2)\hat{\boldsymbol{j}} + (p_3 + q_3)\hat{\boldsymbol{k}} + (p_4 + q_4) \tag{2.10}$$

This operation is commutative. Also, it can represent a composite rotation only for a very small angle. Thus, it is more common to use multiplication to represent the composite rotation. The multiplication of two quaternions follows the Hamilton product as follows:

$$\boldsymbol{p} \otimes \boldsymbol{q} = p_4 q_4 - \boldsymbol{p}_{1:3} \cdot \boldsymbol{q}_{1:3} + p_4 \boldsymbol{q}_{1:3} + q_4 \boldsymbol{p}_{1:3} + \boldsymbol{p}_{1:3} \times \boldsymbol{q}_{1:3} \tag{2.11}$$

The multiplication of two quaternions can also be represented in a matrix form as follows:

$$\boldsymbol{p} \otimes \boldsymbol{q} = \begin{bmatrix} -p_1 & -p_2 & -p_3 & p_4 \\ p_4 & -p_3 & p_2 & p_1 \\ p_3 & p_4 & -p_1 & p_2 \\ -p_2 & p_1 & p_4 & p_3 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} \tag{2.12}$$

2.1.2 Quaternion Differentiation

To measure the change in the orientation of the robot at any time step $\boldsymbol{q}(t)$, one can use the quaternion differentiation [25]. To illustrate, we consider the robot is rotating with angular velocity $\boldsymbol{\omega}$ at time t. Let $\Delta\boldsymbol{q}$ be the rotation change during time Δt in a local frame. The new orientation after rotation is $\boldsymbol{q}(t+\Delta t)$ at time $t + \Delta t$. Then, $\Delta\theta = ||\boldsymbol{\omega}||\Delta t$ is the rotation angle around rotation axis $\hat{\boldsymbol{\omega}} = \boldsymbol{\omega}/||\boldsymbol{\omega}||$. The orientation change can be described as follows:

$$\Delta\boldsymbol{q} = \hat{\boldsymbol{\omega}} \sin\frac{\Delta\theta}{2} + \cos\frac{\Delta\theta}{2} \tag{2.13}$$

$$= \hat{\boldsymbol{\omega}} \sin\frac{||\boldsymbol{\omega}||\Delta t}{2} + \cos\frac{||\boldsymbol{\omega}||\Delta t}{2} \tag{2.14}$$

Then, we can determine the new orientation as a composite rotation as follows:

$$\boldsymbol{q}(t + \Delta t) = \Delta\boldsymbol{q} \otimes \boldsymbol{q}(t) \tag{2.15}$$

We can derive the difference as follows:

$$\boldsymbol{q}(t+\Delta t) - \boldsymbol{q}(t) = \left(\hat{\boldsymbol{\omega}} \sin \frac{||\boldsymbol{\omega}||\Delta t}{2} + \cos \frac{||\boldsymbol{\omega}||\Delta t}{2}\right) \otimes \boldsymbol{q}(t) - \boldsymbol{q}(t) \quad (2.16)$$

$$= \sin \frac{||\boldsymbol{\omega}||\Delta t}{2} \hat{\boldsymbol{\omega}} \otimes \boldsymbol{q}(t) + \cos \frac{||\boldsymbol{\omega}||\Delta t}{2} \otimes \boldsymbol{q}(t) - \boldsymbol{q}(t) \quad (2.17)$$

$$= \sin \frac{||\boldsymbol{\omega}||\Delta t}{2} \hat{\boldsymbol{\omega}} \otimes \boldsymbol{q}(t) - 2\sin^2 \frac{||\boldsymbol{\omega}||\Delta t}{4} \boldsymbol{q}(t) \quad (2.18)$$

Then, the derivation of the quaternion $\dot{\boldsymbol{q}}$ can be derived as follows:

$$\dot{\boldsymbol{q}} = \lim_{\Delta t \to 0} \frac{\boldsymbol{q}(t+\Delta t) - \boldsymbol{q}(t)}{\Delta t} \quad (2.19)$$

$$= \lim_{\Delta t \to 0} \left(\frac{\sin(||\boldsymbol{\omega}||\Delta t/2)}{\Delta t} \hat{\boldsymbol{\omega}} \otimes \boldsymbol{q}(t) - \frac{2\sin^2(||\boldsymbol{\omega}||\Delta t/4)}{\Delta t} \boldsymbol{q}(t)\right) \quad (2.20)$$

$$= \left(\frac{d}{dt} \sin\left(\frac{||\boldsymbol{\omega}||t}{2}\right)\Big|_{t=0}\right) \hat{\boldsymbol{\omega}} \otimes \boldsymbol{q}(t) \quad (2.21)$$

$$= \frac{||\boldsymbol{\omega}||}{2} \hat{\boldsymbol{\omega}} \otimes \boldsymbol{q}(t) \quad (2.22)$$

$$= \frac{1}{2} \boldsymbol{\omega} \otimes \boldsymbol{q}(t) \quad (2.23)$$

One can define omega operator $\boldsymbol{\Omega}(\boldsymbol{\omega})$ as:

$$\boldsymbol{\Omega}(\boldsymbol{\omega}) \equiv \begin{bmatrix} -\boldsymbol{\omega}^{\times} & \boldsymbol{\omega} \\ -\boldsymbol{\omega}^T & 0 \end{bmatrix} \quad (2.24)$$

$$= \begin{bmatrix} 0 & \omega_z & -\omega_y & \omega_x \\ -\omega_z & 0 & \omega_x & \omega_y \\ \omega_y & -\omega_x & 0 & \omega_z \\ -\omega_x & -\omega_y & -\omega_z & 0 \end{bmatrix} \quad (2.25)$$

Then, an equivalent matrix expression for the derivative can be represented as:

$$\dot{\boldsymbol{q}} = \frac{1}{2} \boldsymbol{\Omega}(\boldsymbol{\omega}) \boldsymbol{q} \quad (2.26)$$

2.1.3 Quaternion Distance

In order to measure the difference between two orientations, one can use the quaternion distance, which defines the closest angle between two orientations.

To measure this distance $d(\boldsymbol{p}, \boldsymbol{q})$, one can use the following equation [26]:

$$d(\boldsymbol{p}, \boldsymbol{q}) = \theta = cos^{-1}(2\langle \boldsymbol{p}, \boldsymbol{q} \rangle^2 - 1) \tag{2.27}$$

where $\langle \boldsymbol{p}, \boldsymbol{q} \rangle$ is an inner product between two quaternions, which can be defined as follows:

$$\langle \boldsymbol{p}, \boldsymbol{q} \rangle = p_1 q_1 + p_2 q_2 + p_3 q_3 + p_4 q_4 \tag{2.28}$$

Eq. 2.27 can give the distance between two quaternions in a range between $[0, \pi]$; however, it is numerically expensive from the trigonometric function. Therefore, one can use the following equation to compute the distance between two quaternions:

$$d(\boldsymbol{p}, \boldsymbol{q}) = 1 - \langle \boldsymbol{p}, \boldsymbol{q} \rangle^2 \tag{2.29}$$

Eq. 2.29 can provide the value between $[0, 1]$ which can also be scaled by π to give the value between $[0, \pi]$. This approximated distance can represent a good approximation of Eq. 2.27 as shown in Fig. 2.1.

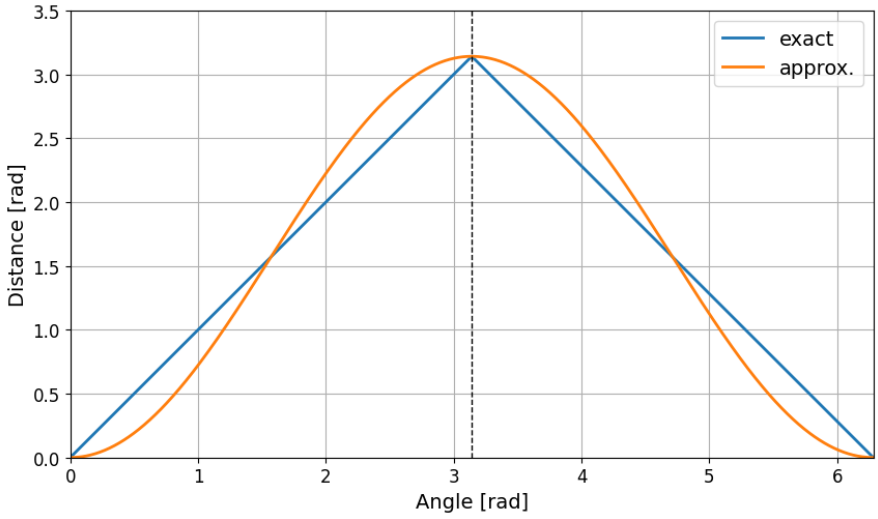


Figure 2.1: Comparison between exact solution and approximation solution for quaternion distance. The x-axis shows the actual difference between the two orientations and the y-axis shows the calculated distance.

2.2 Model Predictive Control

Model Predictive Control is an optimization-based controller that solves the trajectory optimization problem at each time step in a receding horizon fashion. In each time step, the predictive controller solves the open loop Constrained Finite Horizon Optimal Control (CFHOC) and applies only the first control input to the system. The feedback is introduced to the controller by measuring the current states of the system and using it as the initial condition of the next optimization problem. The optimization problem is then solved again in the next time step. This process is repeated until the system reaches the desired point or trajectory.

2.2.1 Constrained Finite Horizon Optimal Control

Consider the following discrete-time nonlinear system:

$$x_{k+1} = f(x_k, u_k) \tag{2.30}$$

where $x_k \in \mathbb{R}^n$ is the state of the system at time step k and $u_k \in \mathbb{R}^m$ is the control input at time step k. $f(\cdot)$ is the dynamic of the system. To control this system, in an ideal case, one would like to solve the infinite horizon optimal control problem, which can be formulated as follows:

$$\begin{aligned} J^*_\infty = \min_{u(\cdot)} \quad & \sum_{k=0}^{\infty} l(x_k, u_k) \\ \text{subject to} \quad & x_{k+1} = f(x_k, u_k), k = 0, \dots, \infty \\ & x_k \in \mathcal{X}, k = 0, \dots, \infty \\ & u_k \in \mathcal{U}, k = 0, \dots, \infty \\ & x_0 = x(0) \end{aligned} \tag{2.31}$$

Where $l(\cdot)$ is a stage or running cost and $x_{(\cdot)}$ indicates a prediction state and $x(\cdot)$ indicates a measured state. $\mathcal{X}$ and $\mathcal{U}$ are state and input constraints, respectively. With the limitation of computation, we cannot directly solve this problem due to the infinite number of optimization variables. In order to handle this issue, one can truncate the infinite horizon optimal control problem into the CFHOC problem. One can introduce a terminal set $x_N \in \mathcal{X}_f$, which is invariant, to ensure that we can always find the feasible control input to keep the system within the set. The infinite cost can be approximated to the terminal

cost $l_f(x_N)$. Thus CFHOC can be formulated as follows:

$$\begin{aligned} \boldsymbol{J_N^*} = \min_{u(\cdot)} \quad & \sum_{k=0}^{N-1} l(x_i, u_i) + \boldsymbol{l_f(x_N)} \\ \text{subject to} \quad & x_{k+1} = f(x_k, u_k), k = 0, \ldots, N-1 \\ & x_k \in \mathcal{X}, k = 0, \ldots, N-1 \\ & u_k \in \mathcal{U}, k = 0, \ldots, N-1 \\ & \boldsymbol{x_N \in \mathcal{X}_f} \\ & x_0 = x(0) \end{aligned} \tag{2.32}$$

The goal of this predictive controller is to find the optimal control input sequence $u^* = \{u_0^*, u_1^*, ..., u_{N-1}^*\}$ that minimizes the cost function J.

2.2.2 Receding Horizon

From solving CFHOC, we can obtain the optimal control input sequence. However, in practice, if we apply the whole sequence to the system, the system may not be able to track the desired point or trajectory. This is due to the fact that the model of the system is not perfect and also the effect of external disturbance. Therefore, the controller needs to be able to handle the uncertainty of the system. In order to do so, the feedback is incorporated into this open-loop controller and translated to the closed-loop controller. This can be done by solving the optimization problem in an iterative manner or a receding horizon fashion. In other words, in each time step, the controller will solve CFHOC problem to get the optimal solution (i.e., control input sequence), then apply only the first input to the system. The system is then evolved for one step, the current states are measured and used as an initial condition to recompute the optimization problem again for the next time step (i.e., $x_0 = x(k)$). The system goes through this procedure again and again until it achieves the target point or trajectory. Furthermore, due to the effect of the horizon, a longer horizon will give a more optimal solution, however, it needs more computation power. On the other hand, a short-horizon controller experiences a short-sighted problem, resulting in probably not an optimal solution and also an unstable system. Using the proper horizon length plays a crucial role in MPC controller. As a result, we can then formulate the optimal

control problem in each time step as follows:

$$\begin{aligned} J_N^* = \min_{u(\cdot)} \quad & \sum_{k=0}^{N-1} l(x_i, u_i) + l_f(x_N) \\ \text{subject to} \quad & x_{k+1} = f(x_k, u_k), k = 0, \ldots, N-1 \\ & x_k \in \mathcal{X}, k = 0, \ldots, N-1 \\ & u_k \in \mathcal{U}, k = 0, \ldots, N-1 \\ & x_N \in \mathcal{X}_f \\ & \boldsymbol{x_0 = x(k)} \end{aligned} \tag{2.33}$$

2.2.3 Direct Single and Direct Multiple Shooting

To solve an optimization problem effectively, one needs to consider the way to construct the problem. There are several factors that play crucial roles in the computation time. One is a shooting method. There are two popular shooting methods, which are single shooting and multiple shooting [27]. These methods are numerical techniques used to solve optimal control and trajectory optimization problems while satisfying certain constraints.

2.2.3.1 Single Shooting Method

Single shooting is a simple method to solve the optimization problem, which is formulated as a Boundary Value Problem (BVP). The main idea is to find the solution for the entire trajectory by integrating the differential equations forward in time. Next, the obtained trajectory is examined to see if the given boundary conditions are satisfied. Therefore, one can formulate the state of the system in each time step as follows:

$$\begin{aligned} x_1 &:= f(x_0, u_0) \\ x_2 &:= f(x_1, u_1) \\ &\vdots \\ x_{N-1} &:= f(x_{N-2}, u_{N-2}) \\ x_N &:= f(x_{N-1}, u_{N-1}) \end{aligned} \tag{2.34}$$

where x_0 is the initial state of the system and x_N is the final state of the system, which can also be formulated as a function of x_0 as follows:

$$x_N := \mathcal{F}(x_0, u_0, u_1, \ldots, u_{N-1}) \tag{2.35}$$

where $\mathcal{F}$ is the function that represents the whole trajectory of the system. From this structure, despite being straightforward to use, the single-shooting approach can experience convergence problems when the optimization problem is very nonlinear.

2.2.3.2 Multiple Shooting Method

In the multiple shooting method, on the other hand, the trajectory is divided into smaller segments. The main idea is to solve the smaller optimization problem for each segment separately. The solution of each segment is then enforced to be continuous with the solution of the adjacent segments. Therefore, one can formulate the state of the system in each segment as follows:

$$x_k = f(s_k, u_k), k = 0, \ldots, N-1 \tag{2.36}$$

where s_k is an artificial initial value and x_k is a trajectory piece. Thus, each trajectory piece is only a function of the local artificial initial value s_k and control input u_k. Then, one can connect each piece with continuity as follows:

$$x_k = s_{k+1}, k = 0, \ldots, N-1 \tag{2.37}$$

As a result, MPC with multiple shooting method can be formulated by incorporating the state of the system in each segment as the optimization variable as follows:

$$\begin{aligned} J_N^* = \min_{u(\cdot), x(\cdot)} \quad & \sum_{k=0}^{N-1} l(x_i, u_i) + l_f(x_N) \\ \text{subject to} \quad & x_{k+1} = f(x_k, u_k), k = 0, \ldots, N-1 \\ & x_k \in \mathcal{X}, k = 0, \ldots, N-1 \\ & u_k \in \mathcal{U}, k = 0, \ldots, N-1 \\ & x_N \in \mathcal{X}_f \\ & x_0 = x(k) \end{aligned} \tag{2.38}$$

Although the multiple shooting method creates a larger optimization problem, due to more decision variables, this method displays better convergence properties than single shooting, especially when the problem is highly nonlinear [28]. Also, this method will give both optimal control input sequence $u^* = \{u_0^*, u_1^*, ..., u_{N-1}^*\}$ and optimal state sequence $x^* = \{x_0^*, x_1^*, ..., x_N^*\}$ that minimizes the cost function J. Therefore, we will use

this method to formulate the MPC controller in the following Chapters.

Chapter 3

Systems Dynamics

To understand the behaviour of the system, one needs to first model the system. The evolution of the dynamics system can be modelled using differential equations. Also, it is necessary to derive a precise model for designing a model-based controller. This chapter will describe the derivation of the equation of motion of our tethered system. First, the dynamics of a single free flyer will be explored. Then, the kinematics, dynamics and force interaction of the multi-agent system will be discussed. Finally, the equation of motion of the n-agent tethered system will be derived.

3.1 Modelling of a single free flyer

In order to get the dynamics of multiple robots, one first needs to understand the dynamics of a single free flyer. A single free-flying robot can be modelled as a thruster-controlled spacecraft [29]. With this type of spacecraft, each thruster is a one-directional thruster, meaning that it cannot produce a negative thrust. In order to generate thrust in both directions, we pair two thrusters together in the opposite direction. Also, the robot needs to have at least six pairs of thrusters to be able to move in all directions in three-dimensional space. The thrusters are placed in a specific way that they can produce thrust, which will exert both force and torque on the robot in all directions as illustrated in Fig. 3.1. The arrangement of the thrusters can be described using matrix $\boldsymbol{L} \in \mathbb{R}^{3\times 6}$ as follows:

$$\boldsymbol{L} = \begin{bmatrix} \boldsymbol{l}_1 & \boldsymbol{l}_2 & \cdots & \boldsymbol{l}_6 \end{bmatrix}^T \tag{3.1}$$

where $\boldsymbol{l}_i \in \mathbb{R}^3$ denotes the lever arm of i^{th} thruster pairs from the center of mass of the robot in the body frame $\{\mathcal{B}\}$. Thus, the thrust directions are given by a matrix $\boldsymbol{D} \in \mathbb{R}^{3\times 6}$ as follows:

$$\boldsymbol{D} = \begin{bmatrix} \hat{\boldsymbol{d}}_1 & \hat{\boldsymbol{d}}_2 & \cdots & \hat{\boldsymbol{d}}_6 \end{bmatrix} \tag{3.2}$$

where $\hat{\boldsymbol{d}}_i \in \mathbb{R}^3$ is a unit vector defining the direction of the i^{th} thruster. We can describe the amount of force provided by each thruster pair with a vector $\boldsymbol{u} \in \mathbb{R}^6$

$$\boldsymbol{u} = \begin{bmatrix} u_1 & u_2 & \cdots & u_6 \end{bmatrix}^T \tag{3.3}$$

Thus, the body force $\boldsymbol{F} \in \mathbb{R}^3$ and torque $\boldsymbol{\tau} \in \mathbb{R}^3$ in body frame $\{\mathcal{B}\}$ of the robot are calculated as following equations:

$$\boldsymbol{F} = \boldsymbol{D}\boldsymbol{u} = \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_6 \end{bmatrix} \tag{3.4}$$

$$\boldsymbol{\tau} = \boldsymbol{L}\boldsymbol{u} = \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = l_{arm} \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & -1 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_6 \end{bmatrix} \tag{3.5}$$

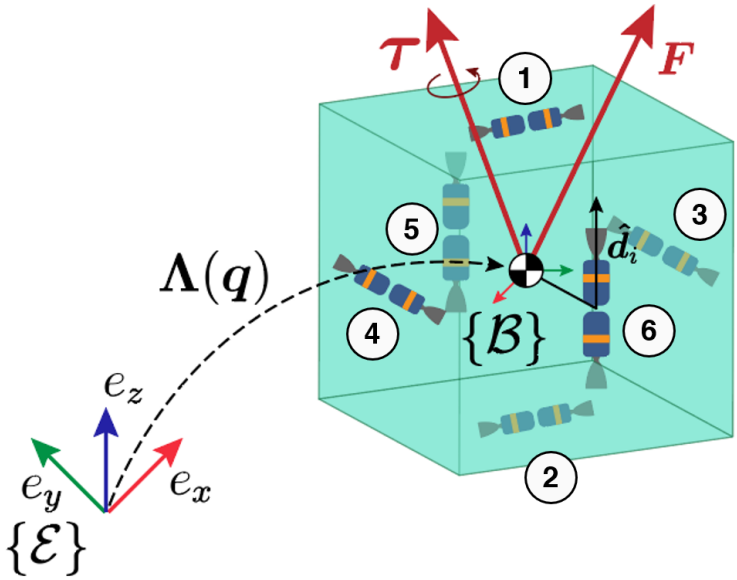


Figure 3.1: Thruster locations of the robot.

The model of the single robot is then derived using the Newton-Euler method. This method is used to derive the equation of motion of a rigid body by separating translation (Newton), and rotation (Euler). By using the conversion of the linear momentum of the robot and the summation of all the forces acting on the robot, the change in position and velocity of the robot can be computed. Similarly, by considering all the torques exerted on the robot, one can calculate the change in orientation and angular velocity. The equation of motion of a single robot is given by:

$$\dot{\boldsymbol{p}} = \boldsymbol{v} \tag{3.6}$$

$$\dot{\boldsymbol{v}} = m^{-1}(\mathbf{\Lambda}(\boldsymbol{q})^T \boldsymbol{F}) \tag{3.7}$$

$$\dot{\boldsymbol{q}} = \frac{1}{2}(\mathbf{\Omega}(\boldsymbol{\omega}))\boldsymbol{q} \tag{3.8}$$

$$\dot{\boldsymbol{\omega}} = \boldsymbol{J}^{-1}(-\boldsymbol{\omega}^{\times}\boldsymbol{J}\boldsymbol{\omega} + \boldsymbol{\tau}) \tag{3.9}$$

where m is mass of the robot and $\mathbf{\Lambda}(\boldsymbol{q})^T$ is the DCM that transforms the thruster force ($\boldsymbol{F}$) from the body frame $\{\mathcal{B}\}$ to the inertial frame $\{\mathcal{E}\}$. DCM $\mathbf{\Lambda}(\boldsymbol{q})$ can be calculated from the current attitude of the robot, which is described by a quaternion from Eq. 2.6. $\mathbf{\Omega}(\boldsymbol{\omega})$ can be derived from quaternion kinematics from Eq. 2.26 as a function of robot angular velocity.

3.2 Modelling of an n-agent tethered system

In this section, we will connect multiple robot models together with the load to form a tethered system. The tethered system can be modelled as a multi-body system. A cable will be modelled similarly to a rigid rod. This cable is connected to the robot and the load with a ball-and-socket joint. Thus, this model can also be used for the system connected together with a massless rod. For our system, we will constrain the system to always have the tension force in the cable using the controller. This will be explained in Chapter 4. The model of the tethered system is shown in Fig. 3.2. The dynamics of the system can be described using the Newton-Euler method. The equation of motion of the system is given by:

$$\dot{\boldsymbol{p}}_i = \boldsymbol{v}_i \qquad i = 1, 2, \ldots, n \quad (3.10)$$

$$\dot{\boldsymbol{v}}_i = m_i^{-1}(\boldsymbol{\Lambda}_i^T(\boldsymbol{q}_i)\boldsymbol{F}_i - \boldsymbol{T}_i) \qquad i = 1, 2, \ldots, n \quad (3.11)$$

$$\dot{\boldsymbol{q}}_i = \frac{1}{2}(\boldsymbol{\Omega}_i(\boldsymbol{\omega}_i))\boldsymbol{q}_i \qquad i = 1, 2, \ldots, n \quad (3.12)$$

$$\dot{\boldsymbol{\omega}}_i = \boldsymbol{J}_i^{-1}(-\boldsymbol{\omega}_i^{\times}\boldsymbol{J}_i\boldsymbol{\omega}_i + \boldsymbol{\tau}_i - \boldsymbol{r}_i \times (\boldsymbol{\Lambda}_i(\boldsymbol{q}_i)\boldsymbol{T}_i)) \qquad i = 1, 2, \ldots, n \quad (3.13)$$

$$\dot{\boldsymbol{p}}_L = \boldsymbol{v}_L \quad (3.14)$$

$$\dot{\boldsymbol{v}}_L = m_L^{-1}\sum_{i=1}^{n}\boldsymbol{T_i} \quad (3.15)$$

$$\dot{\boldsymbol{q}}_L = \frac{1}{2}(\boldsymbol{\Omega}_L(\boldsymbol{\omega}_L))\boldsymbol{q}_L \quad (3.16)$$

$$\dot{\boldsymbol{\omega}}_L = \boldsymbol{J}_L^{-1}(-\boldsymbol{\omega}_L^{\times}\boldsymbol{J}_L\boldsymbol{\omega}_L + \sum_{i=1}^{n}\boldsymbol{R}_i \times (\boldsymbol{\Lambda}_L(\boldsymbol{q}_L)\boldsymbol{T}_i)) \quad (3.17)$$

where $\boldsymbol{p}_i, \boldsymbol{v}_i, \boldsymbol{p}_L, \boldsymbol{v}_L, \boldsymbol{T}_i$ are defined in inertial frame $\{\mathcal{E}\}$, while $\boldsymbol{J}_i, \boldsymbol{\omega}_i, \boldsymbol{\tau}_i, \boldsymbol{F}_i, \boldsymbol{r}_i$ are defined in body frame $\{\mathcal{B}_i\}$ of the robot i^{th}. $\boldsymbol{J}_L, \boldsymbol{\omega}_L, \boldsymbol{R}_i$ are in the load frame $\{\mathcal{B}_L\}$. Where $\boldsymbol{\Lambda}_i(\boldsymbol{q}_i), \boldsymbol{\Lambda}_L(\boldsymbol{q}_L)$ are DCM that performs the coordinate transformation from the inertial frame $\{\mathcal{E}\}$ into the body frame $\{\mathcal{B}_i\}$ and load frame $\{\mathcal{B}_L\}$, respectively. Also, $\boldsymbol{\Lambda}_{(\cdot)}^T(\boldsymbol{q}_{(\cdot)})$ transform the coordinate from the body or load frame back to the inertial frame. n is a number of robots.

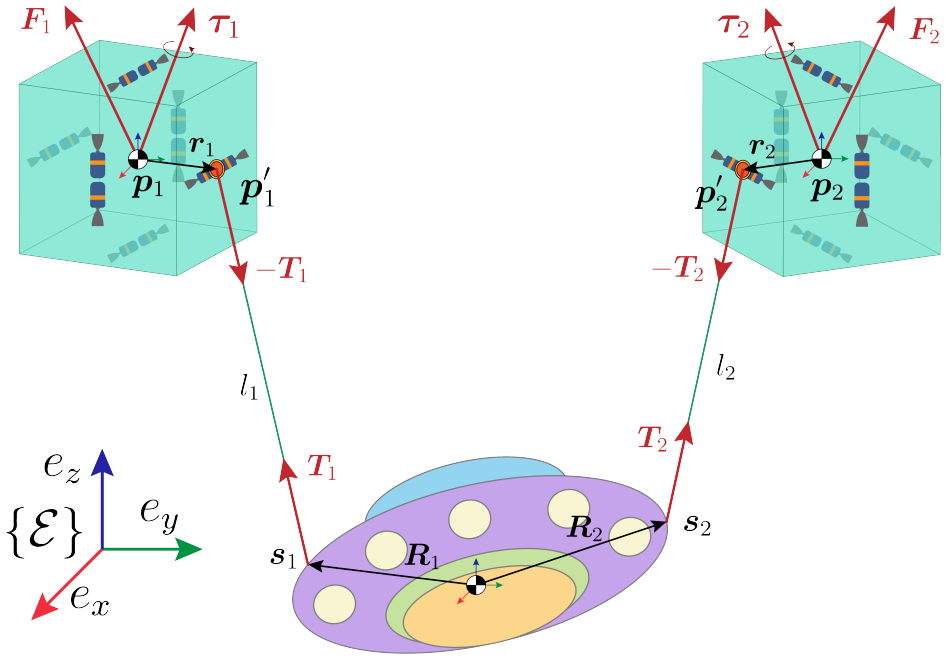


Figure 3.2: An n-agent tethered system.

In order the solve the equation of motion of the system, we need to formulate all interaction forces (i.e., cable tension, $\boldsymbol{T}_i$) in terms of states and inputs. By considering the holonomic constraints in this system, the length of all cables is always constant. Thus, we get the following holonomic constraints ψ_i equations:

$$\psi_i = l_i^2 = ||\boldsymbol{p}'_i - \boldsymbol{s}_i||^2 \qquad i = 1, 2, \ldots, n \tag{3.18}$$

$$\dot{\psi}_i = 0 = (\boldsymbol{p}'_i - \boldsymbol{s}_i) \cdot (\boldsymbol{v}'_i - \boldsymbol{v}_{s_i}) \qquad i = 1, 2, \ldots, n \tag{3.19}$$

$$\ddot{\psi}_i = 0 = (\boldsymbol{p}'_i - \boldsymbol{s}_i) \cdot (\boldsymbol{a}'_i - \boldsymbol{a}_{s_i}) + (\boldsymbol{v}'_i - \boldsymbol{v}_{s_i}) \cdot (\boldsymbol{v}'_i - \boldsymbol{v}_{s_i}) \qquad i = 1, 2, \ldots, n \tag{3.20}$$

In the following, for brevity, we will show the detailed derivation only for robot 1 and also introduce these shorthands:

$$\boldsymbol{\Lambda}_{(\cdot)} := \boldsymbol{\Lambda}_{(\cdot)}(\boldsymbol{q}_{(\cdot)}) \qquad (\cdot) = 1, 2, \ldots, n, L \tag{3.21}$$

$$\boldsymbol{\gamma}_{(\cdot)} := -\boldsymbol{\omega}_{(\cdot)}^{\times} \boldsymbol{J}_{(\cdot)} \boldsymbol{\omega}_{(\cdot)} \qquad (\cdot) = 1, 2, \ldots, n, L \tag{3.22}$$

$$||\cdot||_i^2 := (\boldsymbol{v}'_i - \boldsymbol{v}_{s_i}) \cdot (\boldsymbol{v}'_i - \boldsymbol{v}_{s_i}) \qquad i = 1, 2, \ldots, n \tag{3.23}$$

Also, we will introduce a unit directional vector of the tension that can be computed using the following equations:

$$\hat{\boldsymbol{n}}_i = \frac{\boldsymbol{p}'_i - \boldsymbol{s}_i}{l_i} \qquad i = 1, 2, \ldots, n \tag{3.24}$$

$$\boldsymbol{T}_i = T_i \hat{\boldsymbol{n}}_i \qquad i = 1, 2, \ldots, n \tag{3.25}$$

From the kinematics of the system, we can compute the position, velocity and acceleration of each specific point in inertial frame $\{\mathcal{E}\}$ as follows:

$$\boldsymbol{p}'_1 = \boldsymbol{p}_1 + \boldsymbol{\Lambda}_1^T \boldsymbol{r}_1 \tag{3.26}$$

$$\begin{aligned} \boldsymbol{v}'_1 &= \boldsymbol{v}_1 + \boldsymbol{\Lambda}_1^T (\boldsymbol{\omega}_1 \times \boldsymbol{r}_1) \\ &= \boldsymbol{v}_1 + \boldsymbol{\Lambda}_1^T (\boldsymbol{\omega}_1^\times \boldsymbol{r}_1) \end{aligned} \tag{3.27}$$

$$\begin{aligned} \boldsymbol{a}'_1 &= \boldsymbol{a}_1 + \boldsymbol{\Lambda}_1^T (\dot{\boldsymbol{\omega}}_1 \times \boldsymbol{r}_1 + \boldsymbol{\omega}_1 \times (\boldsymbol{\omega}_1 \times \boldsymbol{r}_1)) \\ &= \boldsymbol{a}_1 + \boldsymbol{\Lambda}_1^T (-\boldsymbol{r}_1 \times \dot{\boldsymbol{\omega}}_1 + \boldsymbol{\omega}_1 \times (\boldsymbol{\omega}_1 \times \boldsymbol{r}_1)) \\ &= \boldsymbol{a}_1 + \boldsymbol{\Lambda}_1^T (-\boldsymbol{r}_1^\times \dot{\boldsymbol{\omega}}_1 + \boldsymbol{\omega}_1^\times (\boldsymbol{\omega}_1^\times \boldsymbol{r}_1)) \end{aligned} \tag{3.28}$$

$$\boldsymbol{s}_1 = \boldsymbol{p}_L + \boldsymbol{\Lambda}_L^T \boldsymbol{R}_1 \tag{3.29}$$

$$\begin{aligned} \boldsymbol{v}_{s_1} &= \boldsymbol{v}_L + \boldsymbol{\Lambda}_L^T (\boldsymbol{\omega}_L \times \boldsymbol{R}_1) \\ &= \boldsymbol{v}_L + \boldsymbol{\Lambda}_L^T (\boldsymbol{\omega}_L^\times \boldsymbol{R}_1) \end{aligned} \tag{3.30}$$

$$\begin{aligned} \boldsymbol{a}_{s_1} &= \boldsymbol{a}_L + \boldsymbol{\Lambda}_L^T (\dot{\boldsymbol{\omega}}_L \times \boldsymbol{R}_1 + \boldsymbol{\omega}_L \times (\boldsymbol{\omega}_L \times \boldsymbol{R}_1)) \\ &= \boldsymbol{a}_L + \boldsymbol{\Lambda}_L^T (-\boldsymbol{R}_1 \times \dot{\boldsymbol{\omega}}_L + \boldsymbol{\omega}_L \times (\boldsymbol{\omega}_L \times \boldsymbol{R}_1)) \\ &= \boldsymbol{a}_L + \boldsymbol{\Lambda}_L^T (-\boldsymbol{R}_1^\times \dot{\boldsymbol{\omega}}_L + \boldsymbol{\omega}_L^\times (\boldsymbol{\omega}_L^\times \boldsymbol{R}_1)) \end{aligned} \tag{3.31}$$

From holonomic constraints in Eq. 3.20, states of the system $\boldsymbol{p}'_1, \boldsymbol{v}'_1, \boldsymbol{a}'_1, \boldsymbol{s}_1, \boldsymbol{v}_{s_1}, \boldsymbol{a}_{s_1}$ can be substituted with above kinematics and dynamics equations to get followings:

$$
\begin{aligned}
0 = &\left[\left(\frac{\overbrace{\boldsymbol{\Lambda}_1^T\boldsymbol{F}_1 - \boldsymbol{T}_1}^{\boldsymbol{a}_1}}{m_1} + \boldsymbol{\Lambda}_1^T\left(-\boldsymbol{r}_1^\times\left(\overbrace{\boldsymbol{J}_1^{-1}(\boldsymbol{\gamma}_1 + \boldsymbol{\tau}_1 - \boldsymbol{r}_1^\times\boldsymbol{\Lambda}_1\boldsymbol{T}_1)}^{\dot{\boldsymbol{\omega}}_1}\right) + (\boldsymbol{\omega}_1^\times)^2\boldsymbol{r}_1\right)\right)\right. \\
&\left. - \left(\frac{\overbrace{\sum_{i=1}^n \boldsymbol{T_i}}^{\boldsymbol{a}_L}}{m_L} + \boldsymbol{\Lambda}_L^T\left(-\boldsymbol{R}_1^\times\left(\overbrace{\boldsymbol{J}_L^{-1}\left(\boldsymbol{\gamma}_L + \sum_{i=1}^n \boldsymbol{R}_i \times (\boldsymbol{\Lambda}_L\boldsymbol{T}_i)\right)}^{\dot{\boldsymbol{\omega}}_L}\right) + (\boldsymbol{\omega}_L^\times)^2\boldsymbol{R}_1\right)\right)\right] \\
&\cdot\left(\frac{\boldsymbol{p}_1' - \boldsymbol{s}_1}{l_1}\right) + \frac{||\cdot||_1^2}{l_1} \\
0 = &\left[\boldsymbol{\Lambda}_1^T\left(\frac{\boldsymbol{F}_1}{m_1} - \boldsymbol{r}_1^\times\boldsymbol{J}_1^{-1}(\boldsymbol{\gamma}_1 + \boldsymbol{\tau}_1) + (\boldsymbol{\omega}_1^\times)^2\boldsymbol{r}_1\right) - \left(\frac{\boldsymbol{I}_3}{m_1} - \boldsymbol{\Lambda}_1^T\boldsymbol{r}_1^\times\boldsymbol{J}_1^{-1}\boldsymbol{r}_1^\times\boldsymbol{\Lambda}_1\right)\boldsymbol{T}_1\right. \\
&- \left(\frac{\boldsymbol{I}_3}{m_L} - \boldsymbol{\Lambda}_L^T\boldsymbol{R}_1^\times\boldsymbol{J}_L^{-1}\boldsymbol{R}_1^\times\boldsymbol{\Lambda}_L\right)\boldsymbol{T}_1 - \sum_{i=2}^n\left(\frac{\boldsymbol{I}_3}{m_L} - \boldsymbol{\Lambda}_L^T\boldsymbol{R}_1^\times\boldsymbol{J}_L^{-1}\boldsymbol{R}_i^\times\boldsymbol{\Lambda}_L\right)\boldsymbol{T}_i \\
&\left. - \boldsymbol{\Lambda}_L^T\left(-\boldsymbol{R}_1^\times\boldsymbol{J}_L^{-1}\boldsymbol{\gamma}_L + (\boldsymbol{\omega}_L^\times)^2\boldsymbol{R}_1\right)\right]\cdot\hat{\boldsymbol{n}}_1 + \frac{||\cdot||_1^2}{l_1} \\
0 = &\left[\underbrace{\boldsymbol{\Lambda}_1^T\left(\frac{\boldsymbol{F}_1}{m_1} - \boldsymbol{r}_1^\times\boldsymbol{J}_1^{-1}(\boldsymbol{\gamma}_1 + \boldsymbol{\tau}_1) + (\boldsymbol{\omega}_1^\times)^2\boldsymbol{r}_1\right) - \boldsymbol{\Lambda}_L^T\left(-\boldsymbol{R}_1^\times\boldsymbol{J}_L^{-1}\boldsymbol{\gamma}_L + (\boldsymbol{\omega}_L^\times)^2\boldsymbol{R}_1\right)}_{\boldsymbol{\mu}_1}\right. \\
&\underbrace{- \left(\left(\frac{m_1 + m_L}{m_1 m_L}\right)\boldsymbol{I}_3 - \boldsymbol{\Lambda}_1^T\boldsymbol{r}_1^\times\boldsymbol{J}_1^{-1}\boldsymbol{r}_1^\times\boldsymbol{\Lambda}_1 - \boldsymbol{\Lambda}_L^T\boldsymbol{R}_1^\times\boldsymbol{J}_L^{-1}\boldsymbol{R}_1^\times\boldsymbol{\Lambda}_L\right)}_{\boldsymbol{\phi}_1}\boldsymbol{T}_1 \\
&\left. - \sum_{i=2}^n\underbrace{\left(\frac{\boldsymbol{I}_3}{m_L} - \boldsymbol{\Lambda}_L^T\boldsymbol{R}_1^\times\boldsymbol{J}_L^{-1}\boldsymbol{R}_2^\times\boldsymbol{\Lambda}_L\right)}_{\boldsymbol{\sigma}_{1,i}}\boldsymbol{T}_i\right]\cdot\hat{\boldsymbol{n}}_1 + \frac{||\cdot||_1^2}{l_1} \\
0 = &\left[\boldsymbol{\mu}_1 - \boldsymbol{\phi}_1\boldsymbol{T}_1 - \sum_{i=2}^n\boldsymbol{\sigma}_{1,i}\boldsymbol{T}_i\right]\cdot\hat{\boldsymbol{n}}_1 + \frac{||\cdot||_1^2}{l_1} \\
0 = &\underbrace{\boldsymbol{\mu}_1\cdot\hat{\boldsymbol{n}}_1 + \frac{||\cdot||_1^2}{l_1}}_{\zeta_1} - \sum_{i=2}^n\boldsymbol{\sigma}_{1,i}\boldsymbol{T}_i\cdot\hat{\boldsymbol{n}}_1 - \boldsymbol{\phi}_1\boldsymbol{T}_1\cdot\hat{\boldsymbol{n}}_1 \\
0 = &\zeta_1 - \sum_{i=2}^n\underbrace{\boldsymbol{\sigma}_{1,i}\hat{\boldsymbol{n}}_i\cdot\hat{\boldsymbol{n}}_1}_{\alpha_{1,i}}T_i - \underbrace{\boldsymbol{\phi}_1\hat{\boldsymbol{n}}_1\cdot\hat{\boldsymbol{n}}_1}_{\alpha_{1,1}}T_1 \\
0 = &\zeta_1 - \sum_{i=2}^n\alpha_{1,i}T_i - \alpha_{1,1}T_1
\end{aligned}
$$

In a similar procedure, one can compute the internal force of other robots in the system. As a result, we can formulate all internal forces in the system and stack them together in a single matrix form as follows:

$$\begin{bmatrix} \alpha_{1,1} & \alpha_{1,2} & \cdots & \alpha_{1,n-1} & \alpha_{1,n} \\ \alpha_{2,1} & \alpha_{2,2} & \cdots & \alpha_{2,n-1} & \alpha_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \alpha_{n-1,1} & \alpha_{n-1,2} & \cdots & \alpha_{n-1,n-1} & \alpha_{n-1,n} \\ \alpha_{n,1} & \alpha_{n,2} & \cdots & \alpha_{1,n-1} & \alpha_{n,n} \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ \vdots \\ T_{n-1} \\ T_n \end{bmatrix} = \begin{bmatrix} \zeta_1 \\ \zeta_2 \\ \vdots \\ \zeta_{n-1} \\ \zeta_n \end{bmatrix} \tag{3.32}$$

where each component is defined as follows:

$$\boldsymbol{\phi_i} = \left(\left(\frac{m_i + m_L}{m_i m_L} \right) \boldsymbol{I}_3 - \boldsymbol{\Lambda}_i^T \boldsymbol{r}_i^\times \boldsymbol{J}_i^{-1} \boldsymbol{r}_i^\times \boldsymbol{\Lambda}_i - \boldsymbol{\Lambda}_L^T \boldsymbol{R}_i^\times \boldsymbol{J}_L^{-1} \boldsymbol{R}_i^\times \boldsymbol{\Lambda}_L \right) \tag{3.33}$$

$$\boldsymbol{\mu}_i = \boldsymbol{\Lambda}_i^T \left(\frac{\boldsymbol{F}_i}{m_i} - \boldsymbol{r}_i^\times \boldsymbol{J}_i^{-1} (\boldsymbol{\gamma}_i + \boldsymbol{\tau}_i) + (\boldsymbol{\omega}_i^\times)^2 \boldsymbol{r}_i \right) - \boldsymbol{\Lambda}_L^T \left(-\boldsymbol{R}_i^\times \boldsymbol{J}_L^{-1} \boldsymbol{\gamma}_L + (\boldsymbol{\omega}_L^\times)^2 \boldsymbol{R}_i \right) \tag{3.34}$$

$$\boldsymbol{\sigma_{i,j}} = \left(\frac{\boldsymbol{I}_3}{m_L} - \boldsymbol{\Lambda}_L^T \boldsymbol{R}_i^\times \boldsymbol{J}_L^{-1} \boldsymbol{R}_j^\times \boldsymbol{\Lambda}_L \right) \tag{3.35}$$

$$\alpha_{i,i} = \boldsymbol{\phi}_i \hat{\boldsymbol{n}}_i \cdot \hat{\boldsymbol{n}}_i \tag{3.36}$$

$$\alpha_{i,j} = \boldsymbol{\sigma_{i,j}} \hat{\boldsymbol{n}}_j \cdot \hat{\boldsymbol{n}}_i \tag{3.37}$$

$$\zeta_i = \boldsymbol{\mu}_i \cdot \hat{\boldsymbol{n}}_i + \frac{||\cdot||_i^2}{l_i} \tag{3.38}$$

As a result, from Eq. 3.32, the tension force in each cable of each robot can be calculated by solving linear equations and formulating it in terms of states and inputs.

3.3 Modelling of a tethered system in planar coordinates

In planar coordinates, a robot is constrained to be able to move only in the x, y direction and rotate around the z axis, which can be represented by rotation angle θ. Thus, the dynamics of the system can be simplified. First, we can represent the rotation around the z-axis by only two quaternion components: q_3 and q_4 as follows:

$$\boldsymbol{q} = \begin{bmatrix} 0 & 0 & q_3 & q_4 \end{bmatrix}^T = \begin{bmatrix} 0 & 0 & sin(\theta/2) & cos(\theta/2) \end{bmatrix}^T \tag{3.39}$$

Next, DCM $\mathbf{\Lambda}(\theta)$ can be simplified to the following form:

$$\mathbf{\Lambda}(\theta) = \begin{bmatrix} -sin^2(\theta/2) + cos^2(\theta/2) & 2sin(\theta/2)cos(\theta/2) & 0 \\ -2sin(\theta/2)cos(\theta/2) & -sin^2(\theta/2) + cos^2(\theta/2) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.40}$$

$$= \begin{bmatrix} cos(\theta) & sin(\theta) & 0 \\ -sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.41}$$

One can observe that the simplified DCM is similar to a standard rotation matrix around the z-axis. Additionally, we can get simpler forms in angular components as follows:

$$\omega^{\times} = \begin{bmatrix} 0 & -\omega \\ \omega & 0 \end{bmatrix} \tag{3.42}$$

$$\boldsymbol{\gamma}_{(\cdot)} := -\boldsymbol{\omega}_{(\cdot)}^{\times} J_{(\cdot)} \boldsymbol{\omega}_{(\cdot)} = 0 \qquad (\cdot) = 1, 2, \ldots, L \tag{3.43}$$

Also, the thruster body force $\boldsymbol{F}$ and torque $\boldsymbol{\tau}$ in the robot are also simplified as follows:

$$\boldsymbol{F} = \begin{bmatrix} F_x & F_y & 0 \end{bmatrix}^T \tag{3.44}$$

$$\boldsymbol{\tau} = \begin{bmatrix} 0 & 0 & \tau_z \end{bmatrix}^T \tag{3.45}$$

For planar coordinates, the system requires at least two robots to manoeuvre the load in all directions. In our system, there are two robots equipped with eight thrusters on the robot with the arrangement shown in Fig. 3.3. Thus, body forces and torque can be calculated as follows:

$$\boldsymbol{F} = \boldsymbol{D}\boldsymbol{u} = \begin{bmatrix} F_x \\ F_y \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} \tag{3.46}$$

$$\boldsymbol{\tau} = \boldsymbol{L}\boldsymbol{u} = \begin{bmatrix} \tau_z \end{bmatrix} = l_{arm} \begin{bmatrix} 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} \tag{3.47}$$

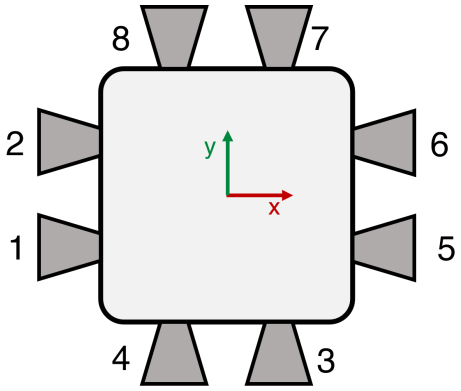


Figure 3.3: Thruster configuration of the robot.

As a result, the equation of motion of the system can be simplified as follows:

$$\dot{\boldsymbol{p}}_i = \boldsymbol{v}_i \qquad i = 1, 2 \tag{3.48}$$

$$\dot{\boldsymbol{v}}_i = m_i^{-1}(\boldsymbol{\Lambda}_i^T(\theta_i)\boldsymbol{F}_i - \boldsymbol{T}_i) \qquad i = 1, 2 \tag{3.49}$$

$$\dot{\theta}_i = \omega_i \qquad i = 1, 2 \tag{3.50}$$

$$\dot{\omega}_i = J_i^{-1}(\boldsymbol{\tau}_i - \boldsymbol{r}_i \times (\boldsymbol{\Lambda}_i(\theta_i)\boldsymbol{T}_i)) \qquad i = 1, 2 \tag{3.51}$$

$$\dot{\boldsymbol{p}}_L = \boldsymbol{v}_L \tag{3.52}$$

$$\dot{\boldsymbol{v}}_L = m_L^{-1}(\boldsymbol{T_1} + \boldsymbol{T_2}) \tag{3.53}$$

$$\dot{\theta_L} = \omega_L \tag{3.54}$$

$$\dot{\omega}_L = J_L^{-1}\left[\sum_{i=1}^{2} \boldsymbol{R}_i \times (\boldsymbol{\Lambda}_L(\boldsymbol{\theta}_L)\boldsymbol{T}_i)\right] \tag{3.55}$$

3.4 Equilibrium of the system

3.4.1 Equilibrium of the n-agent tethered system

The tethered system with three to n robots has more than one equilibrium point due to redundancy as shown in Fig. 3.4. In this work, we propose the generalized method to find one specific equilibrium point which is the center of geometry created by connecting all the anchor points. At this point, we assume all robot attitudes to be the same as the load. Thus, we can find the

center of equilibrium $\boldsymbol{c}$ from the geometry as follows:

$$\boldsymbol{c} = \frac{\sum_{i=1}^{n} \boldsymbol{s}_i}{n} \tag{3.56}$$

Next, the unit direction vector from the center of equilibrium and the poses of each transporter can be derived as follows:

$$\hat{\boldsymbol{b}}_i = \frac{\boldsymbol{s}_i - \boldsymbol{c}}{||\boldsymbol{s}_i - \boldsymbol{c}||} \qquad i = 1, 2, \ldots, n \tag{3.57}$$

$$\boldsymbol{p}_i = \boldsymbol{s}_i + (l_i + ||\boldsymbol{r}_i||)\hat{\boldsymbol{b}}_i \qquad i = 1, 2, \ldots, n \tag{3.58}$$

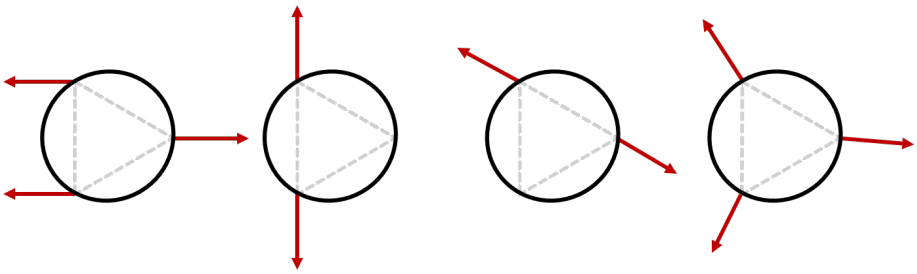


Figure 3.4: Equilibrium point of the n-agents tethered system. The equilibrium point on the right is used in this work.

3.4.2 Equilibrium of the planar tethered system

From the nature of the system, we can analytically derive the states of the two transporters from only the states of the load at an equilibrium point. At this specific point, all derivatives of the states in Eq. 3.48 - Eq. 3.55 become zero (i.e., $\dot{\boldsymbol{x}} = 0$) and the summation of force and torque at the load is zero (i.e., $\sum \boldsymbol{F} = 0$ and $\sum \boldsymbol{\tau} = 0$). Also, it can imply that the tension force $\boldsymbol{T}_1$ and $\boldsymbol{T}_2$ have to be on the same line of action but in the opposite direction (i.e., $\boldsymbol{T}_1 = -\boldsymbol{T}_2$). Therefore, we define a unit directional vector from the anchor point 2 $\boldsymbol{s}_2$ to anchor point 1 $\boldsymbol{s}_1$ as shown in Fig. 3.5 as followings:

$$\hat{\boldsymbol{b}} = \frac{\boldsymbol{s}_1 - \boldsymbol{s}_2}{||\boldsymbol{s}_1 - \boldsymbol{s}_2||} = \hat{\boldsymbol{n}}_1 = -\hat{\boldsymbol{n}}_2 \tag{3.59}$$

Then, the states of transporters can be derived as follows:

$$\begin{aligned} \boldsymbol{p}_1 &= \boldsymbol{s}_1 + l_1\hat{\boldsymbol{n}}_1 + ||\boldsymbol{r}_1||\hat{\boldsymbol{n}}_1 \\ &= \boldsymbol{s}_1 + (l_1 + r_1)\hat{\boldsymbol{b}} \end{aligned} \tag{3.60}$$

$$\begin{aligned} \boldsymbol{p}_2 &= \boldsymbol{s}_2 + l_2\hat{\boldsymbol{n}}_2 + ||\boldsymbol{r}_2||\hat{\boldsymbol{n}}_2 \\ &= \boldsymbol{s}_2 - (l_2 + r_2)\hat{\boldsymbol{b}} \end{aligned} \tag{3.61}$$

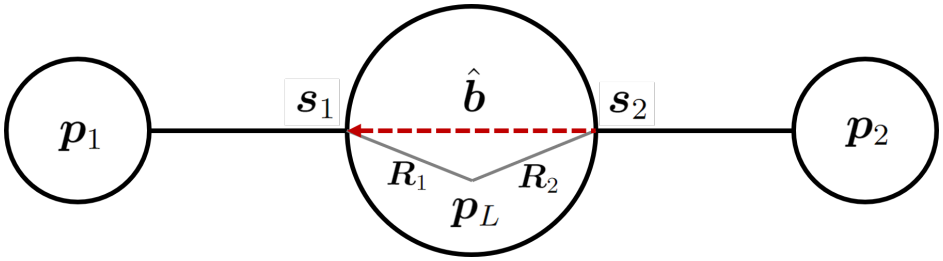


Figure 3.5: Equilibrium point of the planar tethered system.

Chapter 4

Controller Design

4.1 Centralized Control

In this section, we will describe the centralized control scheme, which is the control scheme in which all robots are controlled by a single controller. This controller is implemented by using the MPC controller to control a tethered system that consists of multiple robots and a single load. The MPC controller will solve the optimization problem to find the optimal control input for all robots. Here, we will focus on solving a point-tracking problem.

4.1.1 Discretization

To implement MPC controller, which is a discrete controller, the system needs to be first discretized to be a discrete-time system. There are several methods to discretize the system, such as Forward Euler, Backward Euler, Trapezoidal, and Runge-Kutta. In this implementation of MPC controller, the Forward Euler method is used since this method is the most simple and also uses small computational power. The Forward Euler method is defined as follows:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + f(t_k, \boldsymbol{x}_k) \times h \tag{4.1}$$

where h is the time step size, t_k is the current time, $\boldsymbol{x_k}$ is the current state, $\boldsymbol{x_{k+1}}$ is is the state in the next time step and f is the dynamics of the system in continuous time. After discretizing the system, the MPC controller can be implemented by solving the optimization problem in each time step.

4.1.2 MPC Formulation

The naive approach to implementing a controller is to solve the full model of the nonlinear optimization problem directly by using the dynamics model from Eq. 3.48 - 3.55. By substituting the tension force from Eq. 3.32 into the dynamics model, we can implement the centralized MPC for the point tracking problem as follows:

$$\begin{aligned} J_N^* = \min_{\boldsymbol{x},\boldsymbol{u}} \quad & \sum_{k=0}^{N-1} \left(||\boldsymbol{x}_k - \boldsymbol{r}_k||_{\boldsymbol{Q}}^2 + ||\boldsymbol{u}_k||_{\boldsymbol{R}}^2 \right) + ||\boldsymbol{x}_N - \boldsymbol{r}_{\boldsymbol{N}}||_{\boldsymbol{P}}^2 \\ \text{subject to} \quad & \boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k) \quad k = 0, \dots, N-1 \\ & \boldsymbol{u}_{min} \leq \boldsymbol{u}_k \leq \boldsymbol{u}_{max} \quad k = 0, \dots, N-1 \\ & \boldsymbol{x}_0 = \boldsymbol{x}(k) \end{aligned} \tag{4.2}$$

where f is the dynamics of the system in discrete time. u_{min} and u_{max} are the minimum and maximum thruster force, respectively. $||\boldsymbol{x}||_{\boldsymbol{A}}$ denotes the weighted vector norm $\sqrt{\boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x}}$. $\boldsymbol{r}_{(\cdot)}$ is the reference point. In our approach, we assume $\boldsymbol{r_k} = \boldsymbol{r_N}$. However, this approach suffers from the nonlinearity of the formulation of the tension force in the model, which is computationally expensive. Thus, we propose another way to formulate the optimization problem. In order to improve the computation efficiency [22], instead of explicitly computing the tension force in terms of states and input $\boldsymbol{T} = f(\boldsymbol{x}, \boldsymbol{u})$, one can reduce the nonlinearity and complexity of the original nonlinear dynamics of the system by introducing a Lagrange formulation. By using the principle of virtual work, cable tension in the tethered system can be reformulated in terms of the interaction force between an agent and load with Lagrange multiplier $\lambda \in \mathcal{R}$ as follows:

$$\boldsymbol{T}_i = \lambda_i (\boldsymbol{p}_i' - \boldsymbol{s}_i) \qquad\qquad i = 1, 2, \dots, n \tag{4.3}$$

where n is the number of robots. To be specific, λ_i will be included as additional decision variables. Also, Eq. 3.20 $\ddot{\psi}_i = 0$, which are the holonomic constraints, are then considered as equality constraints in the optimal control problem. Moreover, in practice, it is better to implement this equality constraint as inequality constraints as follows:

$$|\ddot{\psi}_i| \leq \epsilon_i \qquad\qquad i = 1, 2, \dots, n \tag{4.4}$$

where ϵ_i is a very small number. Therefore, the new MPC formulation can be implemented as follows:

$$\begin{aligned} J_N^* = \min_{\boldsymbol{x},\boldsymbol{u},\lambda} \quad & \sum_{k=0}^{N-1} \left(||\boldsymbol{x}_k - \boldsymbol{r}_k||_{\boldsymbol{Q}}^2 + ||\boldsymbol{u}_k||_{\boldsymbol{R}}^2 \right) + ||\boldsymbol{x}_N - \boldsymbol{r}_{\boldsymbol{N}}||_{\boldsymbol{P}}^2 \\ \text{subject to} \quad & \boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k) \quad k = 0, \ldots, N-1 \\ & \boldsymbol{u}_{min} \leq \boldsymbol{u}_k \leq \boldsymbol{u}_{max} \quad k = 0, \ldots, N-1 \\ & |\ddot{\psi}_i| \leq \epsilon_i, i = 1, 2, \ldots, n \\ & \boldsymbol{x}_0 = \boldsymbol{x}(k) \end{aligned} \tag{4.5}$$

There is one main challenge in the tethered system that is different from the system connected with rigid rods, which is the direction of the tension force. By nature, the tension force is always directed outward the load (i.e., pulling the load). On the other hand, in a rigid system, the force can be applied in any direction. It means that if the cable is not in tension, the robot will be unable to move or control the load. Therefore, we need to add a constraint to the optimization problem to ensure that the cable is always in tension. First, one can constrain the direction of the force from the thruster to be always outward the load or the same directional side as the cables as shown in Fig. 4.1. This constraint can be derived from the inner product as follows:

$$\hat{\boldsymbol{n}}_i \cdot \boldsymbol{F}_i \geq 0 \qquad i = 1, 2, \ldots, n \tag{4.6}$$

$$\hat{\boldsymbol{n}}_i^T \boldsymbol{F}_i \geq 0 \qquad i = 1, 2, \ldots, n \tag{4.7}$$

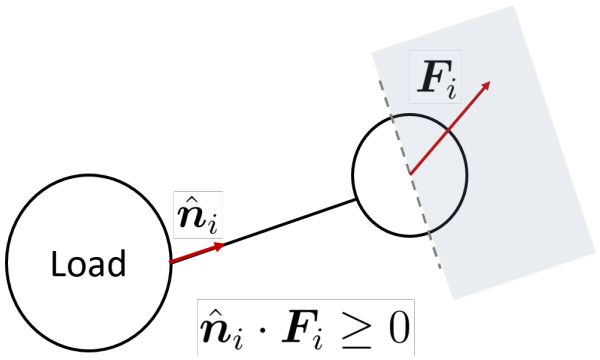


Figure 4.1: Directional constraints on the control input.

Additionally, the system is also subjected to oscillation due to the tension force, which can lead to unstable behaviour. Therefore, we add

another heuristic cost to the optimization problem to prevent the system from oscillating, by maximizing the distance between the robot and the load S. This heuristic cost is defined as follows:

$$\boldsymbol{S}_i = ||\boldsymbol{p}_i - \boldsymbol{p}_L|| \qquad i = 1, 2, \ldots, n \tag{4.8}$$

$$J_i = -\lambda_{reg_i} \boldsymbol{S}_i \qquad i = 1, 2, \ldots, n \tag{4.9}$$

where λ_{reg} is a tuning parameter for the heuristic function. It should be noted that, in equilibrium, this heuristic cost will not converge to zero, resulting in a constant cost value. As a result, the final centralized MPC formulation is as follows:

$$\begin{aligned}
J_N^* = \min_{\boldsymbol{x},\boldsymbol{u},\lambda} \quad & \sum_{k=0}^{N-1} \left(||\boldsymbol{x}_k - \boldsymbol{r}_k||_{\boldsymbol{Q}}^2 + ||\boldsymbol{u}_k||_{\boldsymbol{R}}^2 \right) + ||\boldsymbol{x}_N - \boldsymbol{r}_N||_{\boldsymbol{P}}^2 - \sum_{i=1}^{n} \lambda_{reg_i} \boldsymbol{S}_i \\
\text{subject to} \quad & \boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k) \quad k = 0, \ldots, N-1 \\
& \boldsymbol{u}_{min} \leq \boldsymbol{u}_k \leq \boldsymbol{u}_{max} \quad k = 0, \ldots, N-1 \\
& |\ddot{\psi}_i| \leq \epsilon_i, i = 1, 2, \ldots, n \\
& \hat{\boldsymbol{n}}_i^T \boldsymbol{F}_i \geq 0 \quad i = 1, 2, \ldots, n \\
& \boldsymbol{x}_0 = \boldsymbol{x}(k)
\end{aligned} \tag{4.10}$$

4.2 Decentralized Control

As shown in the previous section, the centralized controller can suffer from the curse of dimensionality. In other words, the computational power required to solve the optimization problem increases exponentially with the number of robots. In order to solve this problem, the decentralized control is proposed. The idea is to decompose the large problem of controlling the whole system with a single centralized controller into several sub-problems, which are controlled by several decentralized controllers. For this system, we design the decentralized controller for each robot that is connected to the load. The idea is that each robot will control the load that is subject to the tension force from the other robots $\boldsymbol{T}_{vir}$ as shown in Fig. 4.2.

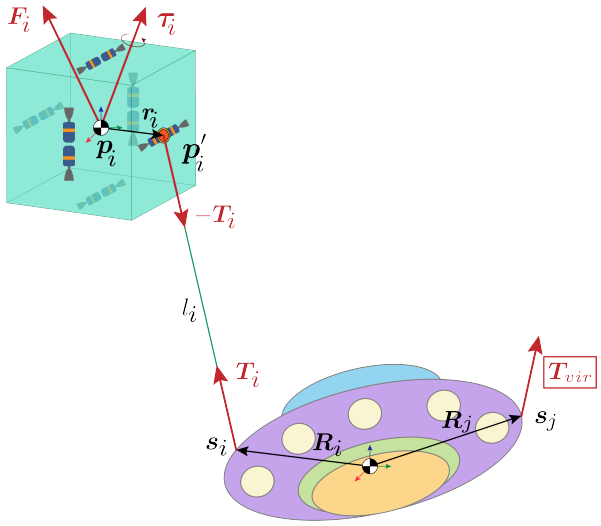


Figure 4.2: A single robot with virtual tension force.

The main challenge lies in determining the force applied by the other robots. In the centralized controller, we can calculate this force directly from the position of the other robots. However, in the decentralized controller, we cannot do that because each robot does not know the position of the other robots. Thus, we need to estimate this force. In this work, we propose the simplest method to estimate this force by utilizing the equilibrium force. To clarify, the equilibrium force is the force that the robot needs to apply to the load in order to maintain the load in the equilibrium position. This force can be calculated by assuming the constant force in the equilibrium direction $\hat{\boldsymbol{b}}$ as described in Eq. 3.57, which can be calculated as follows:

$$\boldsymbol{T}_{vir} = \gamma \hat{\boldsymbol{b}} \tag{4.11}$$

where γ is the tuning parameter for the constant force. The idea is that if the controlled robot assumes that other robots are keeping the load in the equilibrium position, then this robot will solve the optimization problem as if it is the only robot that will apply the force to move the load to the desired position. As a result, all the robots will collaborate to move the load to the desired position. Additionally, each robot is assumed to control only a partial of the load, which can be calculated as follows:

$$m_L' = m_L/n \tag{4.12}$$
$$\boldsymbol{J}_L' = \boldsymbol{J}_L/n \tag{4.13}$$

where n is the number of robots. Therefore, the equation of motion of i^{th} subsystem can be described as follows:

$$\dot{\boldsymbol{p}}_i = \boldsymbol{v}_i \tag{4.14}$$
$$\dot{\boldsymbol{v}}_i = m_i^{-1}(\boldsymbol{\Lambda}_i^T \boldsymbol{F}_i - \boldsymbol{T}_i) \tag{4.15}$$
$$\dot{\boldsymbol{q}}_i = \frac{1}{2}\boldsymbol{\Omega}_i \boldsymbol{q}_i \tag{4.16}$$
$$\dot{\boldsymbol{\omega}}_i = \boldsymbol{J}_i^{-1}(-\boldsymbol{\omega}_i^{\times}\boldsymbol{J}_i\boldsymbol{\omega}_i + \boldsymbol{\tau}_i - \boldsymbol{r}_i \times (\boldsymbol{\Lambda}_i \boldsymbol{T}_i)) \tag{4.17}$$
$$\dot{\boldsymbol{p}}_L = \boldsymbol{v}_L \tag{4.18}$$
$$\dot{\boldsymbol{v}}_L = m_L'^{-1}\left(\boldsymbol{T}_i + \sum_{j=1}^{n-1} \boldsymbol{T}_{\boldsymbol{vir_j}}\right) \tag{4.19}$$
$$\dot{\boldsymbol{q}}_L = \frac{1}{2}\boldsymbol{\Omega}_L \boldsymbol{q}_L \tag{4.20}$$
$$\dot{\boldsymbol{\omega}}_L = \boldsymbol{J}_L'^{-1}(-\boldsymbol{\omega}_L^{\times}\boldsymbol{J}_L'\boldsymbol{\omega}_L + \boldsymbol{R}_i \times (\boldsymbol{\Lambda}_L \boldsymbol{T}_i) + \sum_{j=1}^{n-1} \boldsymbol{R}_j \times (\boldsymbol{\Lambda}_L \boldsymbol{T}_{vir_j})) \tag{4.21}$$

Thus, the decentralized MPC controller of i^{th} robot can be formulated with the new discretized dynamic model f_{dec} as follows:

$$\begin{aligned} J_N^* = \min_{\boldsymbol{x},\boldsymbol{u},\lambda} \quad & \sum_{k=0}^{N-1} \left(||\boldsymbol{x}_k - \boldsymbol{r}_k||_{\boldsymbol{Q}}^2 + ||\boldsymbol{u}_k||_{\boldsymbol{R}}^2\right) + ||\boldsymbol{x}_N - \boldsymbol{r}_N||_{\boldsymbol{P}}^2 - \lambda_{reg_i}\boldsymbol{S}_i \\ \text{subject to} \quad & \boldsymbol{x}_{k+1} = f_{dec}(\boldsymbol{x}_k, \boldsymbol{u}_k) \quad k = 0, \ldots, N-1 \\ & \boldsymbol{u}_{min} \leq \boldsymbol{u}_k \leq \boldsymbol{u}_{max} \quad k = 0, \ldots, N-1 \\ & |\ddot{\psi}_i| \leq \epsilon_i \\ & \hat{\boldsymbol{n}}_i^T \boldsymbol{F}_i \geq 0 \\ & \boldsymbol{x}_0 = \boldsymbol{x}(k) \end{aligned} \tag{4.22}$$

4.3 Pulse Width Modulation controller

In order to control the thruster system, which is not continuous, the proper signal is needed to formulate and send to the solenoid valve. There are two controllers on each robot: MPC controller and Pulse Width Modulation (PWM) controller. The MPC controller will send the force command for each thruster to the PWM controller. PWM controller will translate the force into the time duration for the solenoid to open in one duty cycle. This translation

works by normalising the force value into the value between -1 and 1. In other words, 1 means open the valve for the whole duty cycle and 0.5 means open for half and close for half of the duty cycle as shown in Fig. 4.3. In our system, one duty cycle is 0.1 s and the smallest time step is 0.0005 s. This controller will be implemented on both centralized and decentralized schemes.

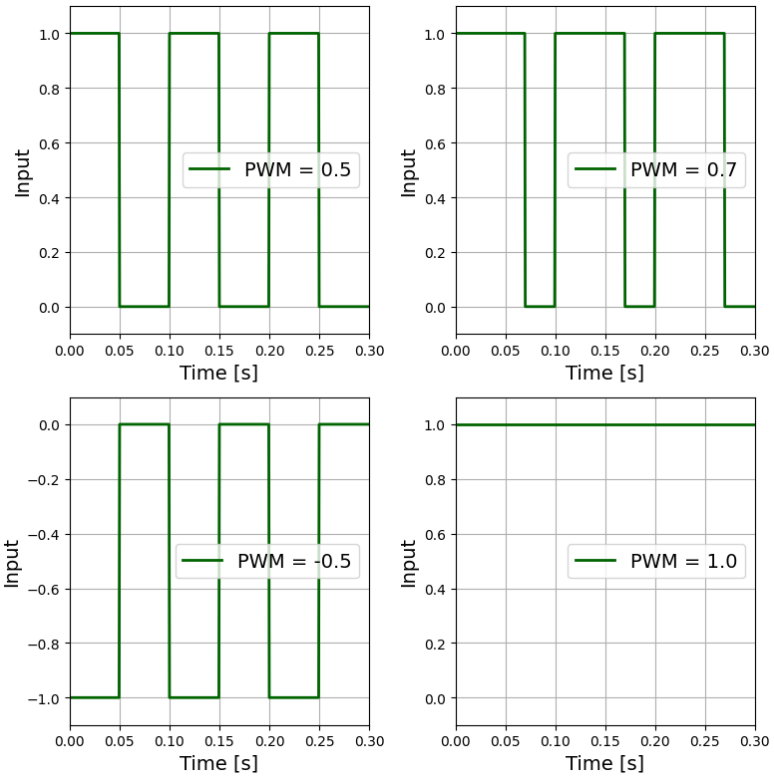


Figure 4.3: PWM signal for different control inputs.

Chapter 5

Experiments and Results

5.1 Overview

In order to verify and evaluate the proposed model and controller, we developed two simulators: numerical and physical simulators. The numerical simulator is used mainly for verifying the model and testing the controller in the ideal scenario. Then, a physical simulator is utilized to replicate the actual behaviour of the system in real-time, including the more accurate behaviour of the cable and the communication protocol. The overview is summarized in Fig. 5.1.

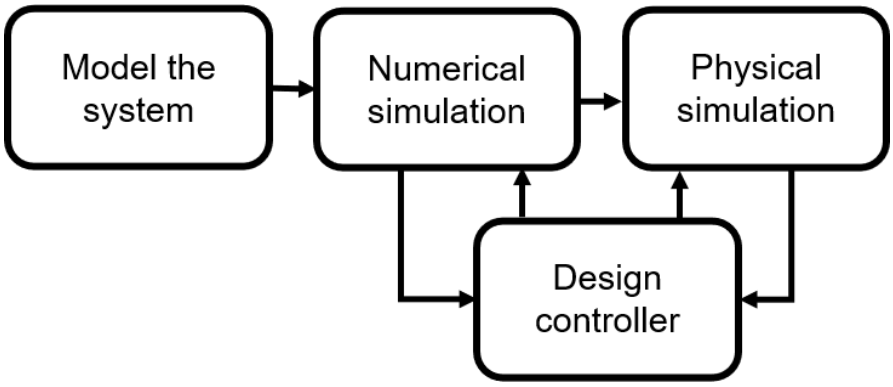


Figure 5.1: Overview of the methods.

5.2 Numerical Simulation

From the model and equations of motion in Chapter 3, we can use the dynamic model to simulate the system in the numerical simulator. The simulator is

developed in Python as shown in Fig. 5.2. To simulate the continuous system, we need to discretize the system first. In this section, we will first describe the method to discretize the equations of motion of the system. Then, we will show how to simulate and implement the controller in the simulator.

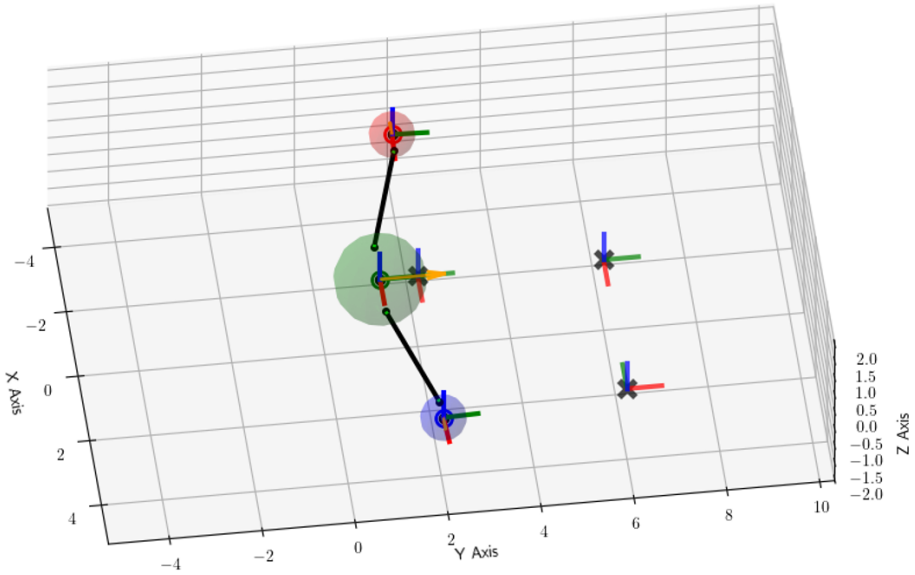


Figure 5.2: Numerical simulator developed in Python.

To get the precise discrete-time model, we use the explicit Runge-Kutta 4th Order (RK4). This method is a member of the Runge-Kutta family for solving ordinary differential equations, specifically initial value problems. This method is given by the following equations:

$$k_1 = f(t_k, x_k) \tag{5.1}$$

$$k_2 = f(t_k + \frac{h}{2}, x_k + \frac{h}{2}k_1) \tag{5.2}$$

$$k_3 = f(t_k + \frac{h}{2}, x_k + \frac{h}{2}k_2) \tag{5.3}$$

$$k_4 = f(t_k + h, x_n + hk_3) \tag{5.4}$$

$$x_{k+1} = x_k + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{5.5}$$

where h is the step size, t_k is the current time and x_k is the state at the k^{th} step, and f is the dynamics of the system. After discretizing the system, we can simulate the system by solving the initial value problem in one step. In

each time step, the current states of the system are observed and used in MPC to calculate the control input. CasADi [30] is used to construct controllers. Then, the control input is applied to the system to get the next states. Then, the results of the current step are used as the initial condition for the next step. This process is repeated until the final time is reached.

We conducted two experiments to verify the proposed model and controller. The first experiment is to control the load to follow the desired points in three-dimensional space. Arbitrary values of the mass and inertia are used for both the robots and the load. The goal of this experiment is to display the capability of a centralized controller to control the load to follow the desired points. The second experiment is to control the load in planar coordinates. The mass and inertia of the load are set following the actual values of the design platform described in Chapter 6. This experiment aims to achieve the goal of transporting the load in a real-time scenario. Thus, the computation time is one of the main concerns in this experiment. Both centralized and decentralized controllers are implemented in this experiment. The experimental setup and results of both experiments are shown in the following sections.

5.2.1 Three-dimensional Load Transportation

In the first experiment, we demonstrate the capability of our controller to control multiple robots in three-dimensional space. Three robots are the minimum number of robots to have full control over the load in all translation and rotation axes. There are 52 states and 18 inputs for the system. The experiment is conducted in numerical simulation. The system is controlled to move the load from the origin (0,0,0) following three desired poses as shown in Fig. 5.4. The first target is a simple translation task in the x direction to position (1,0,0). Next, the system performs the rotation task by rotating 90 degrees clockwise around the z-axis. Then three robots will complete the combination of translation and rotation by moving the load to the final position (1,3,-1) with the attitude of 30 degrees around the y-axis. As shown in Fig. 5.4a, the controller requires 9.03 s and 4.34 s maximum and average computing time, respectively. Thus, the experiment cannot be done in a real-time manner. As a consequence, we will perform this experiment only in the numerical simulator, which can demonstrate that the controller is capable of finding the solution for the desired manoeuvre.

Parameter	Value	Unit
dt	0.1	s
m_1	1	kg
m_2	1	kg
m_3	1	kg
m_L	2	kg
J_1	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	kgm^2
J_2	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	kgm^2
J_3	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	kgm^2
J_L	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	kgm^2
r_1	$0.5 \begin{bmatrix} 0 & -1 & 0 \end{bmatrix}$	m
r_2	$0.5 \begin{bmatrix} cos(\pi/6) & sin(\pi/6) & 0 \end{bmatrix}$	m
r_3	$0.5 \begin{bmatrix} -cos(\pi/6) & sin(\pi/6) & 0 \end{bmatrix}$	m
R_1	$\begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$	m
R_2	$\begin{bmatrix} -sin(\pi/3) & -cos(\pi/3) & 0 \end{bmatrix}$	m
R_3	$\begin{bmatrix} sin(\pi/3) & -cos(\pi/3) & 0 \end{bmatrix}$	m
l_1	3	m
l_2	3	m
l_3	3	m
u_{lim}	10	N
D_1	$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$	-
L_1	$0.5 \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & -1 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 \end{bmatrix}$	-
D_2	$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$	-
L_2	$0.5 \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & -1 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 \end{bmatrix}$	-
D_3	$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$	-
L_3	$0.5 \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & -1 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 \end{bmatrix}$	-

Table 5.1: Parameters of the system

Centralized	
Parameter	**Value**
N_t	15
Q	$10 \times$ diag(10,10,10,1,1,1,10,1,1,1, 10,10,10,1,1,1,10,1,1,1, 10,10,10,1,1,1,10,1,1,1, 10,500,10,1,1,1,500,1,1,1)
R	$0.1 \times$ eye(18)
P	$10 \times Q$

Table 5.2: Parameters of the controller

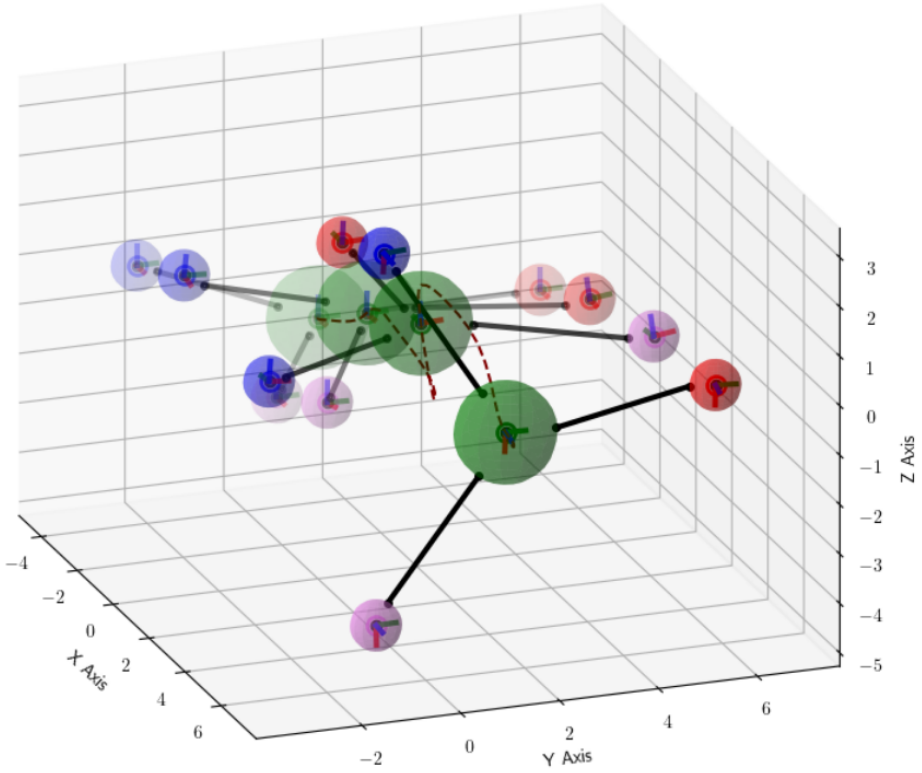
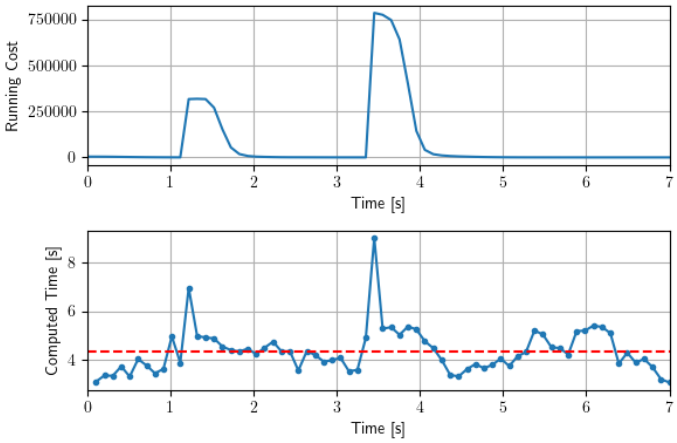
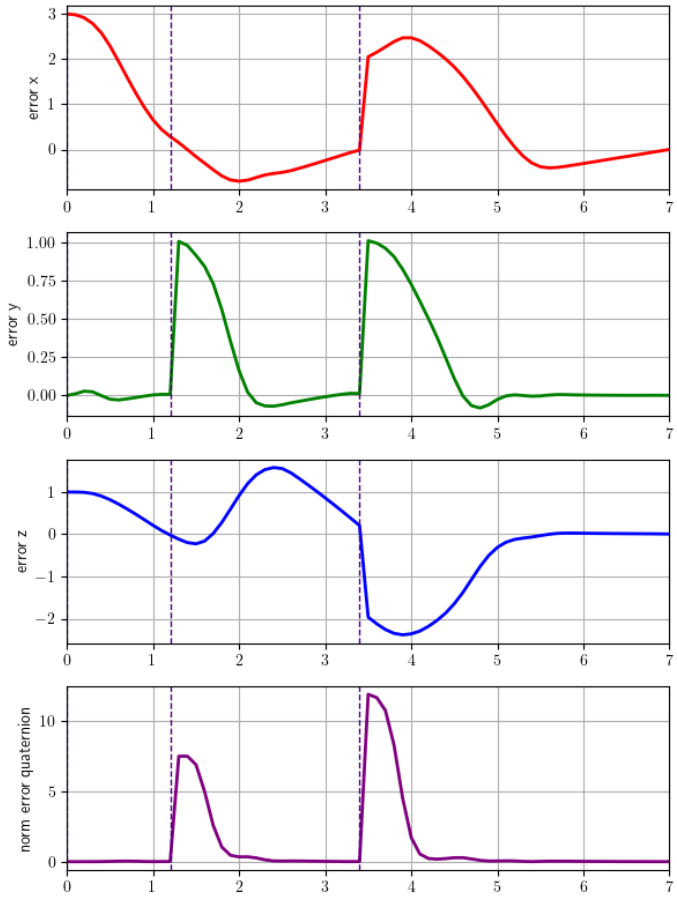


Figure 5.3: The whole trajectory of the load transportation.



(a) Computation time and cost during the motion.



(b) Error during motion.

Figure 5.4: Three dimensional experiments.

5.2.2 Two-dimensional Load Transportation

Because our testing facility is only capable of testing in planar or two-dimensional space, the real-time system experiments will be focused on this particular scenario. Also, only two robots are required in order to fully control the position and orientation of the load in two-dimensional space. Furthermore, the orientation requires only one parameter instead of four parameters in quaternion for the three-dimensional system. As a result, the number of the system's states and inputs is reduced to 18 and 8, respectively. Next, centralized and decentralized will be implemented and verified the performance. Experiments are conducted in a similar way for both the simulation and the actual robot. Two experiments are conducted: point and trajectory tracking. In the following experiments, the parameters of the system are set as shown in Tab. 5.4. The parameters for controllers are set according to Tab. 5.3. There are 438 decision variables, 36 parameters and 468 constraints for the centralized controller, while there are 267 decision variables, 24 parameters and 267 constraints for the decentralized controller.

Centralized	
Parameter	**Value**
N_t	15
Q	5 $\times$ diag(100,100,100,75,75,10,15 100,100,100,75,75,10,15 1000,1000,750,750,1000,100)
R	0.01 $\times$ eye(8)
P	10 $\times Q$
λ_{reg}	1000

Decentralized	
Parameter	**Value**
N_t	15
Q	10 $\times$ diag(100,100,80,80,5,10, 3000,3000,7500,7500,200,100
R	eye(8)
P	10 $\times$ Q
λ_{reg}	1000

Table 5.3: Parameters of the controller

Parameter	Value	Unit
dt	0.1	s
m_1	14.5	kg
m_2	14.5	kg
m_L	18.3	kg
J_1	0.370	kgm^2
J_2	0.370	kgm^2
J_L	0.412	kgm^2
r_1	0.2	m
r_2	0.2	m
R_1	0.2	m
R_2	0.2	m
l_1	0.5	m
l_2	0.5	m
u_{lim}	1.4	N
D_1	$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$	-
L_1	$0.12\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$	-
D_2	$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$	-
L_2	$0.12\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$	-

Table 5.4: Parameters of the system

5.2.2.1 Point tracking experiment

This experiment is conducted to demonstrate the capability of the controller to move the load to a desired pose. Four experiments are conducted to demonstrate the translation in the x and y direction, rotation around the z-axis and a combination of translation and rotation as follows:

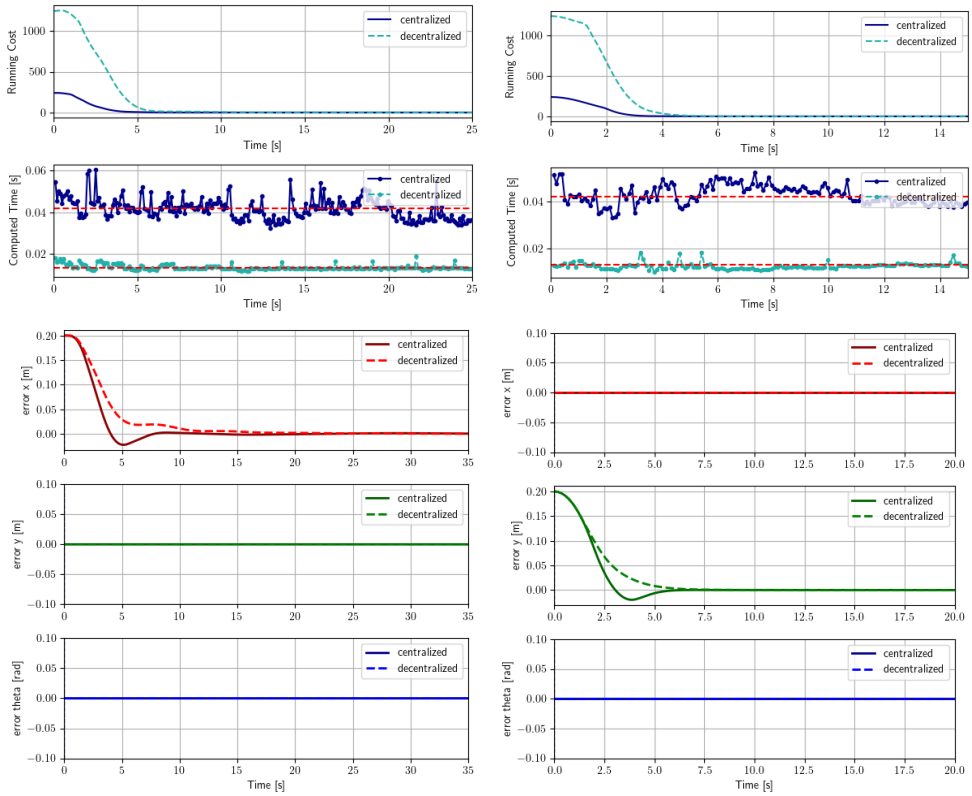
1. Translation in x-direction for 0.2 m
2. Translation in y-direction for 0.2 m
3. Rotation around the z-axis for 15 degrees
4. Translation in x and y direction for 0.2 m and rotation around the z-axis for 15 degrees

In this experiment, we define the threshold for the position error as 0.005 m and the threshold for the orientation error as 0.01 radians. The controller is considered to have achieved the task if the position error and the orientation error are less than the threshold for 5 seconds. We also compare the performance of the centralized and decentralized controller in terms of computing time and the time to achieve the task. These experiments are first conducted in numerical simulation and then in physical simulation. The results of the numerical simulation are shown in Fig. 5.5 and Tab. 5.5. Both centralized and decentralized controllers can achieve all given tasks. The decentralized controller has a better performance than the centralized controller in computing time by reducing the maximum and average computation time by 63.6% and 68.6%, respectively. This is because the decentralized controller has fewer decision variables than the centralized controller. However, the decentralized controller requires more time to move the load to the target position with 58.9% more. We can observe that the decentralized controller is not very effective in rotating tasks.

Task	**Centralized**		
	Max. computation time [ms]	**Avg. computation time [ms]**	**Convergence time [s]**
1	60.6	41.7	12.2
2	52.6	42.2	10.2
3	79.9	48.2	7.3
4	66.0	44.5	12.9
avg.	64.78	44.15	10.65

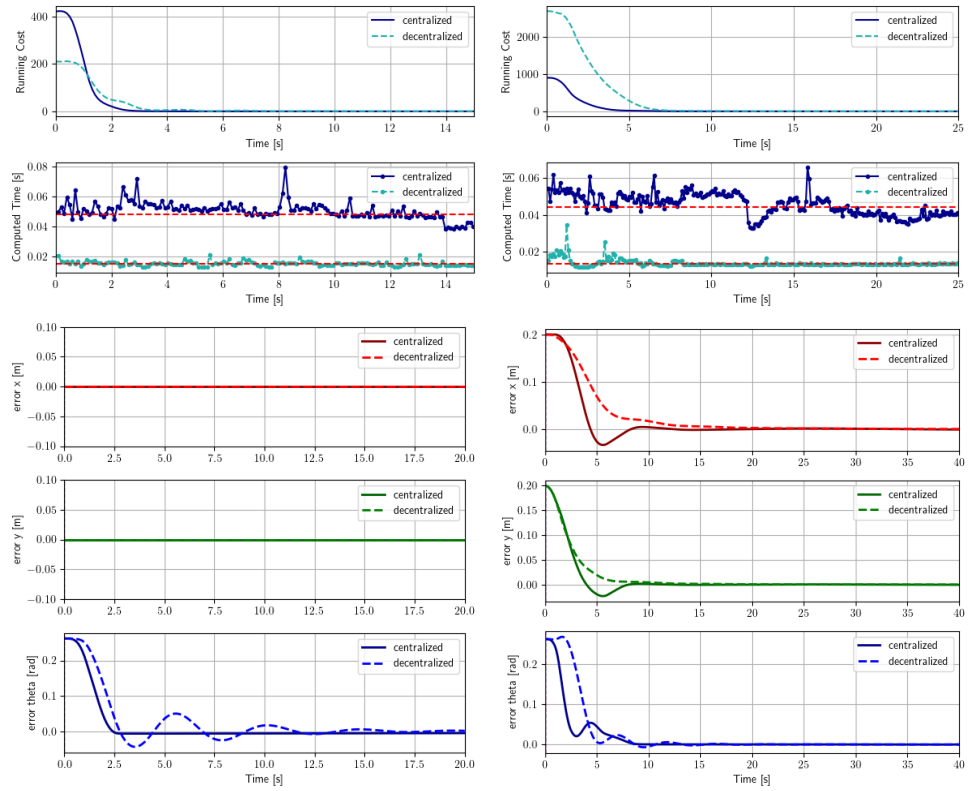
Task	**Deceentralized**		
	Max. computation time [ms]	**Avg. computation time [ms]**	**Convergence time [s]**
1	19.0	13.3	19.8
2	19.4	13.1	10.6
3	21.2	15.2	16.0
4	34.8	13.8	21.3
avg.	23.60	13.85	16.93

Table 5.5: Results from point tracking experiments.



(a) Translation in the x-axis direction for 0.2 m.

(b) Translation in the y-axis direction for 0.2 m.



(c) Rotation around z-axis for 15 degrees.

(d) Combined task.

Figure 5.5: Results from point tracking experiments.

5.2.2.2 Multiple points tracking experiment

To demonstrate the actual application of the controller, we will perform multiple-point tracking experiments. This experiment is conducted by generating the desired trajectory of the load for the system to track. Next, this desired trajectory is discretized into a sequence of desired setpoints. Then, the desired setpoints are sent to the controller to track. In other words, the same point-tracking controller is used to move the load following the whole setpoints consequently. There are two trajectories that are used in this experiment: circle and figure eight as shown in Fig. 5.6. The circle trajectory can be formulated as follows:

$$x = r\cos(\alpha) \tag{5.6}$$
$$y = r\sin(\alpha) \tag{5.7}$$

where r is the radius of the circle and α is the angle starting from 0 to 2π to traverse the circle. The velocity and the angular velocity are calculated from the derivative of the circle equation as follows:

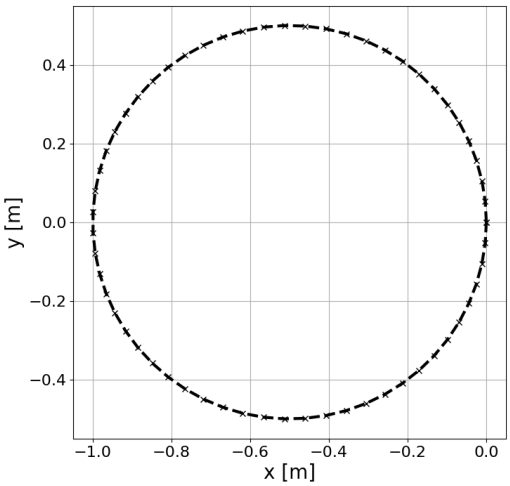
$$\dot{x} = -r\sin(\alpha)\dot{\alpha} \tag{5.8}$$
$$\dot{y} = r\cos(\alpha)\dot{\alpha} \tag{5.9}$$
$$\dot{\alpha} = \text{constant} \tag{5.10}$$

where $\dot{\alpha}$ is a constant value, which is calculated from the required time to finish the desired trajectory. The orientation θ_L and angular velocity of the load $\dot{\theta_L}$ are set to be similar to α and $\dot{\alpha}$, respectively. The figure eight trajectory can be formulated as follows:

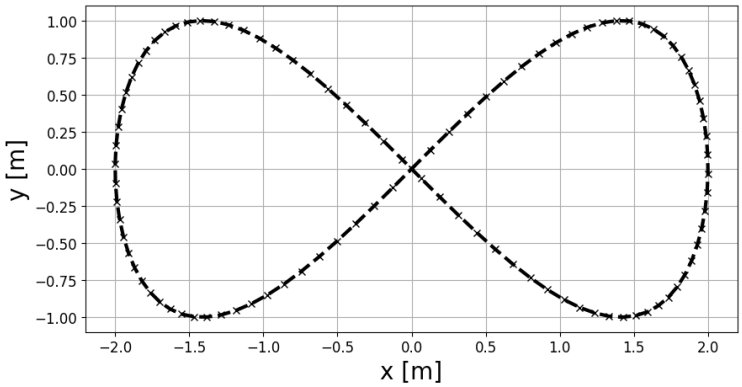
$$x = a\sin(\alpha) \tag{5.11}$$
$$y = a\sin(\alpha)\cos(\alpha) \tag{5.12}$$

where a is a curvature parameter and α is the angle starting from 0 to 2π. The derivation of the trajectory equation is as follows:

$$\dot{x} = a\cos(\alpha)\dot{\alpha} \tag{5.13}$$
$$\dot{y} = a\dot{\alpha}\left(\cos^2(\alpha) - \sin^2(\alpha)\right) \tag{5.14}$$



(a) Circle trajectory



(b) Figure eight trajectory

Figure 5.6: Trajectories for the experiments.

The desired pose of the robot is calculated by using the kinematics of the system at an equilibrium point. By assuming the whole system to be a single rigid body as shown in Fig. 5.7, the desired pose of the robot can be calculated from the states of the load as follows:

$$\boldsymbol{p}_i = \boldsymbol{p}_L + \boldsymbol{R}_i + \boldsymbol{l}_i + \boldsymbol{r}_i \qquad i = 1, 2 \tag{5.15}$$

$$\boldsymbol{v}_i = \boldsymbol{v}_L + \boldsymbol{\omega}_L \times \boldsymbol{r}_i \qquad i = 1, 2 \tag{5.16}$$

$$\boldsymbol{\omega}_i = \boldsymbol{\omega}_L \qquad i = 1, 2 \tag{5.17}$$

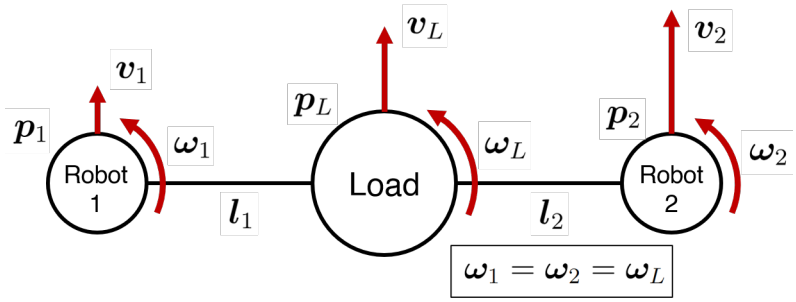
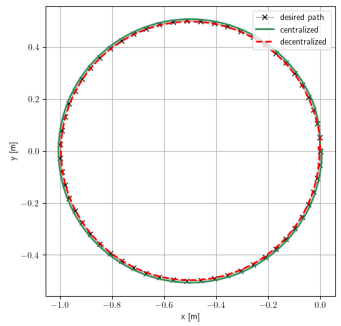
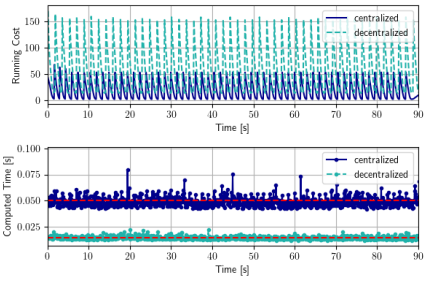


Figure 5.7: We assume the whole system to be a single rigid body, then states of the robots can be computed directly from states of the load.

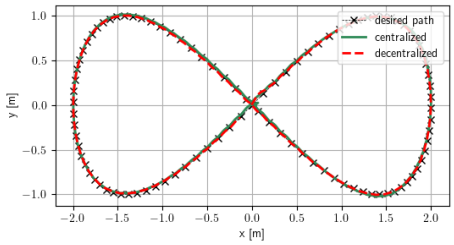
We generate two trajectories for the system to track. The first trajectory is a simple circle trajectory with a radius of 0.5 m and 60 reference points in total. The second trajectory is a figure eight curve with $a = 2$ and 100 reference points in total. As shown in Fig. 5.8, results show that both centralized and decentralized controllers can track the given trajectories. Centralized controller requires 97 ms and 86 ms average computation for circular and eight-figure, respectively. On the other hand, the decentralized controller uses less computational time with 14.3 ms and 14.4 ms.



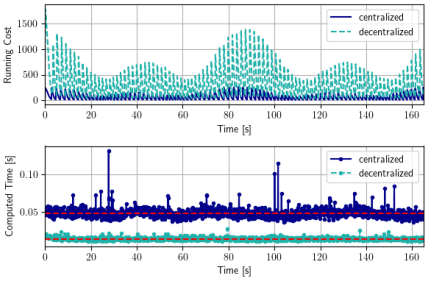
(a) Circle trajectory



(b) Circle trajectory



(c) Figure eight trajectory



(d) Figure eight trajectory

Figure 5.8: Results from the multiple points tracking experiments.

5.3 Physical Simulation

Since the numerical simulator has to wait until the controller finishes the calculation before executing the next simulation step, the drawback of the numerical simulator is that it is unable to capture the real-time behaviour of the system. Also, the rigid link model is used in this simulator, resulting in oversimplifying the actual behaviour of the tethered system. Thus, we develop a physical simulator that can simulate the system in real-time and more accurate behaviour. In this section, we will describe the physical simulator, which is developed in Gazebo, and also the communication protocol in Robot Operating System (ROS 2).

5.3.1 Simulator Description

To create a more realistic simulation, a physics-based simulator, Gazebo, is used. This simulator can not only simulate the dynamics of the system in real-time but also simulate the sensor and communication protocol. Therefore, we use Gazebo to simulate the actual behaviour of the system. The main challenge in our tethered system is to simulate the cable dynamics. One can simulate the cable dynamics by dividing the cable into small segments and then connecting them together with the ball joint, this joint allows the link to rotate in all axes but prevents translation. However, to make this method work, we need to use lots of segments to get an accurate result, which is computationally expensive for the simulator. Therefore, we propose another method to simulate the cable dynamics. We connect the robot to the load with three unactuated joints (i.e., passive joint): revolute, prismatic and revolute joint, respectively as shown in Fig. 5.9. By using these joints, the tension and non-tension behaviour can be achieved by the prismatic joint. In other words, when the distance between the robot and the load is equal or larger than the maximum joint limit, the prismatic joint will act like a rigid rod, resulting in tension force in the cable. On the other hand, when this distance is smaller than the limit, the joint will behave like a free joint, resulting in no tension force in the cable. Two revolute joints are used to allow the rotation of the cable on the z-axis. Therefore, we can simulate the cable dynamics by using only three joints and this method is more computationally efficient than the previous method.

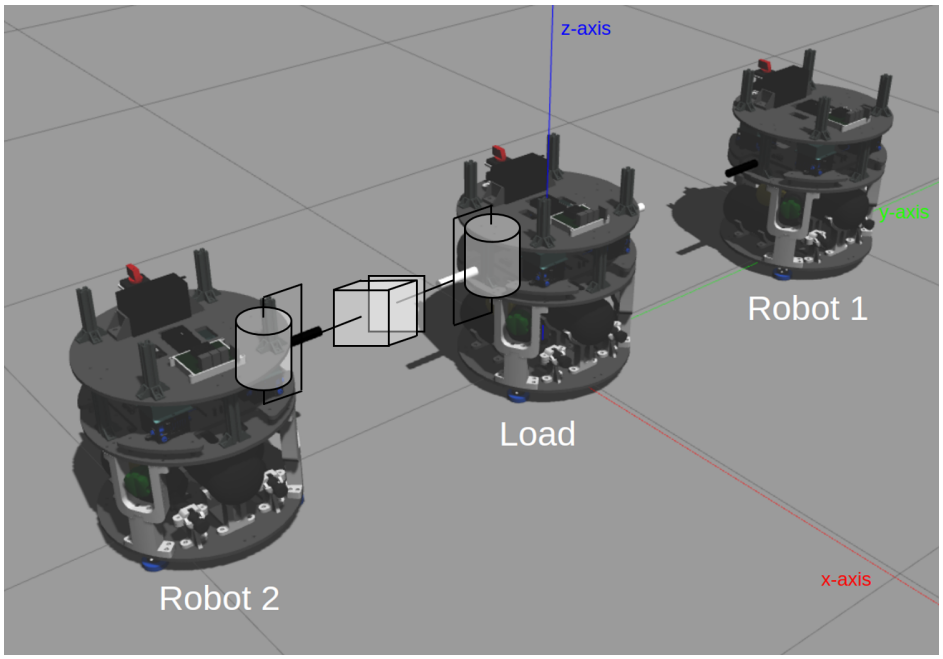


Figure 5.9: Physical simulator in Gazebo displays prismatic and revolute joints to imitate cable behaviour.

5.3.2 Communication Framework

The physical simulator is created along with the communication framework to communicate between controllers and the simulation. ROS 2 is used as a communication protocol to communicate between the controller and the simulator, which will also later be used for the actual robot. The Gazebo simulator will publish the states of the system to the controller. Then, the controller will calculate the control input and publish it back to the simulator at the specific publishing rate. This structure can also be used for the actual robot by replacing the simulator with a real robot. The whole control and communication structure for the simulator is shown in Fig. 5.10.

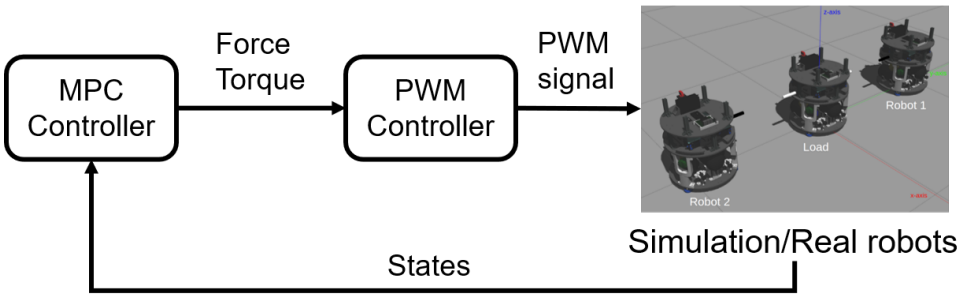


Figure 5.10: The controller framework shows the communication between controllers and robots. First, the MPC controller sends body force and torque signals to the PWM controller. Next, the PWM controller converts the signal and sends the proper open sequence to thrusters. The states of the robots are then measured and sent back to the MPC controller.

5.3.3 MPC Controller Node

The controller node is used to run the MPC controller to solve the optimization problem. This node subscribes to the states of the robot and publishes the desired body wrench, force and torque, to the PWM controller node. This node will run at a constant rate, which is different from the numerical simulator that always waits for the controller to finish the calculation. As a result, the controller node has to be able to run in real-time. In a centralized control scheme, there is only one controller node for all robots. This single node will receive the states of all robots and calculate the control input for all robots. On the other hand, in a decentralized control scheme, there is one controller node for each robot.

5.3.4 PWM Controller Node

To imitate the actual PWM controller in the simulator, PWM controller node is used to receive the force command from the MPC controller node and convert it to an on or off signal for each thruster. This controller node works by having a counter that starts when it receives the command from MPC controller node. This counter then checks the current step with the duration time. If the value is smaller, it will send the open command (i.e., full thrust) to the thruster. On the contrary, if the value is larger, then it will send the zero command to the thruster. This counter is reset every duty cycle. Then the thruster command is converted to the body force in the simulator at the center of mass of each robot using the thruster direction matrix.

5.3.5 Experimental Setup and Results

We conducted the experiment in a similar manner to the ones in numerical simulation by tracking the desired points. There are four consequence desired points to achieve, which represent four tasks: translating in the x-axis, translating in the y-axis, rotating around the z-axis and combining tasks. The points (x, y, θ) are designed to be $(0.2, 0.0, 0.0)$, $(0.2, 0.2, 0.0)$, $(0.2, 0.2, \pi/12)$ and $(0.4, 0.0, 0.0)$, respectively. θ is given in radian. The controller is implemented in a centralized and decentralized manner. The results shown in Fig. 5.11 and Fig. 5.13 display that both controllers can achieve given tasks. We can also observe that, during motion, the centralized controller scheme behaves like a single rigid body motion, resulting in less oscillation in the states of the robots as shown in Fig. 5.12. However, as can be seen in Fig. 5.14, each robot exhibits more oscillatory behaviour when moving under a decentralized controller scheme. The average computation time as displayed in Fig. 5.15 of the centralized controller is 44 ms, which is slower than the one in the decentralized controller with 28.3 ms. This is due to the fact that the decentralized controller has to solve the smaller optimization problem for only one robot.

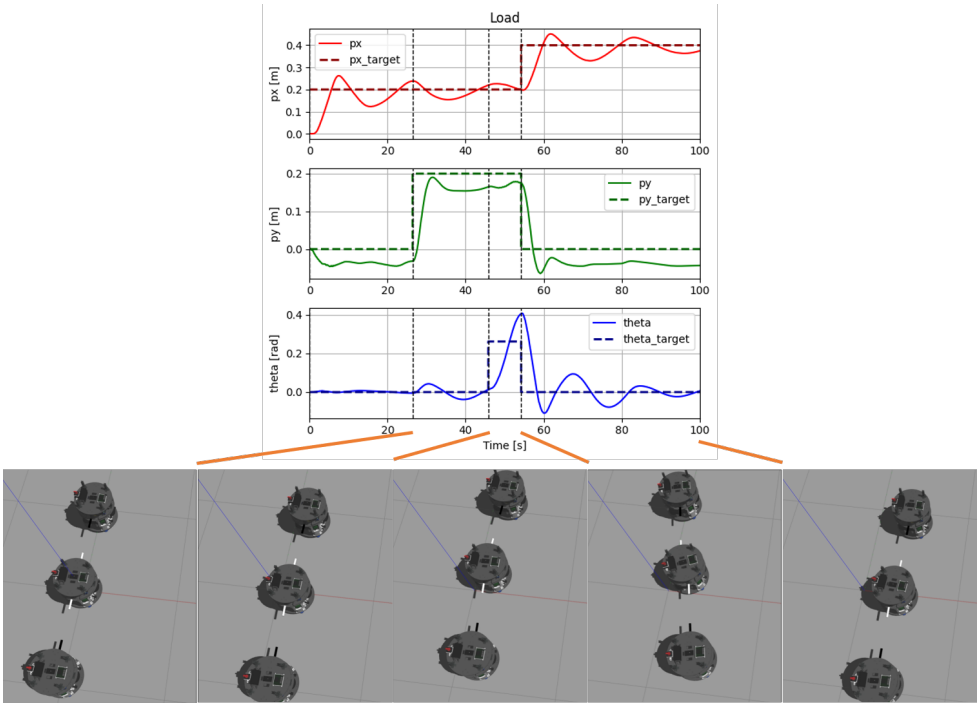


Figure 5.11: States of the load during the motion in the centralized scheme.

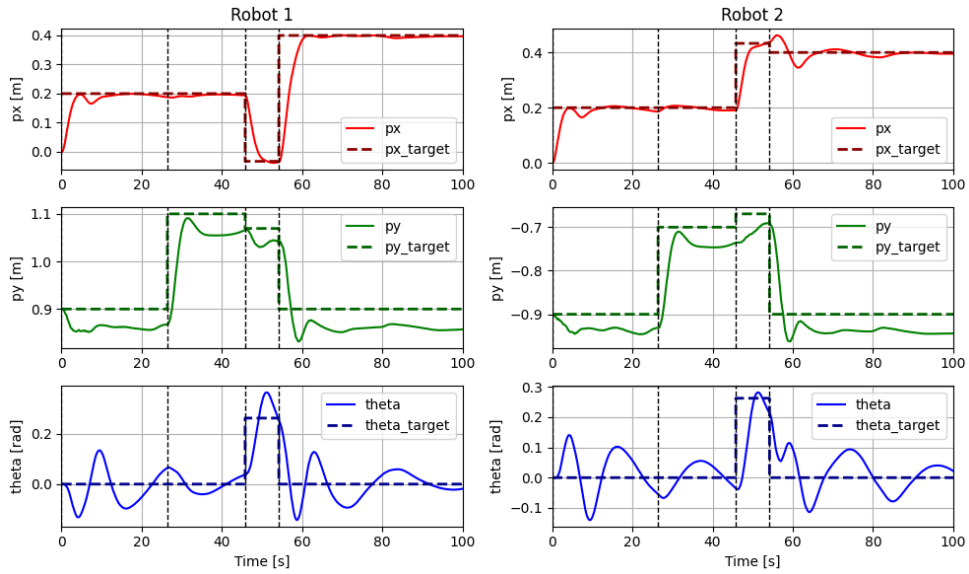


Figure 5.12: States of the robots during the motion in the centralized scheme.

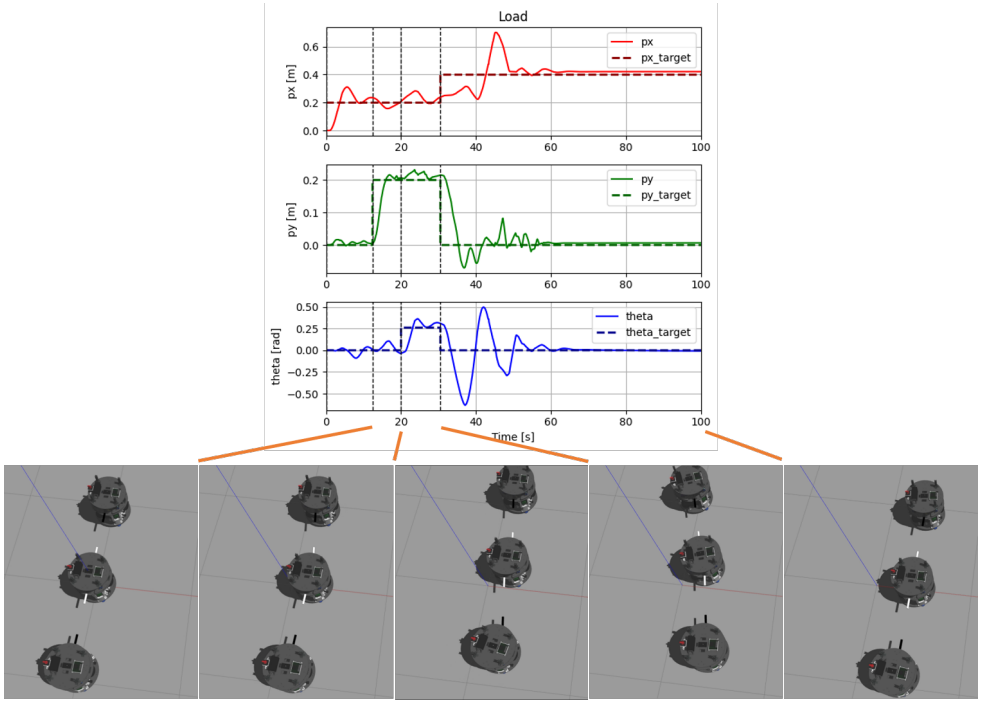


Figure 5.13: States of the load during the motion in the decentralized scheme.

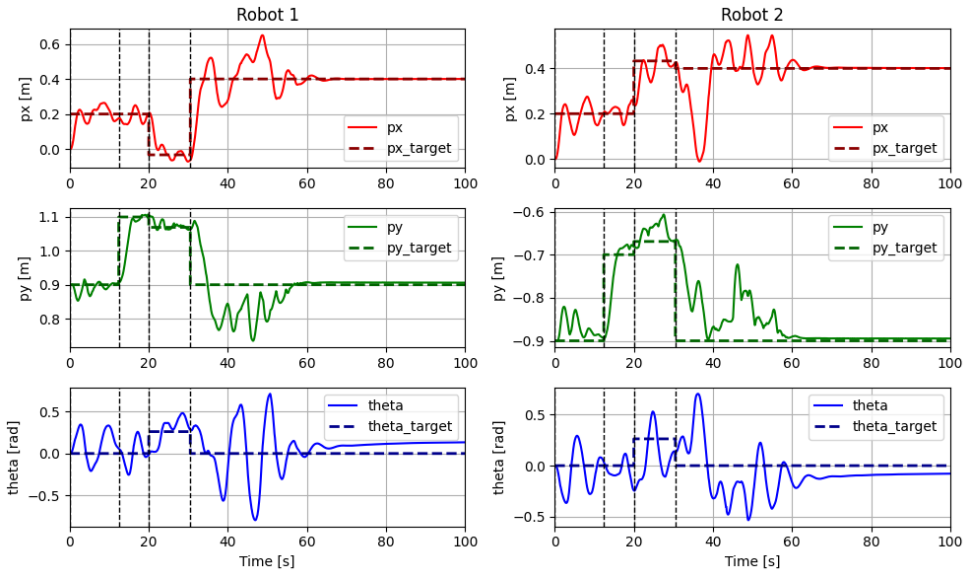


Figure 5.14: States of the robots during the motion in the decentralized scheme.

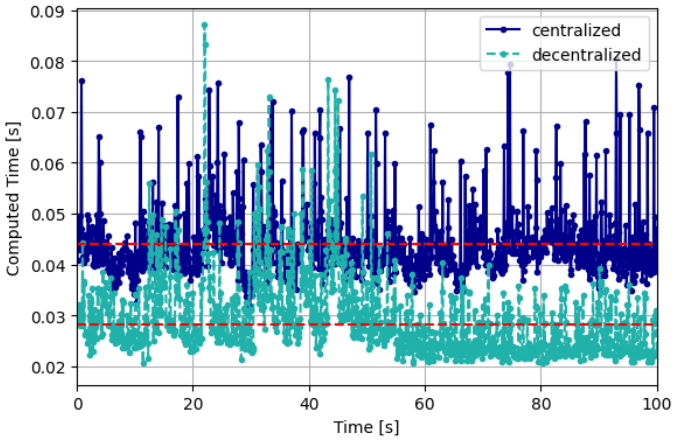


Figure 5.15: Computation time of centralized and decentralized controller during motion.

5.4 Laboratory Experimental Setup

In future work, we will implement the controller on the actual robot. The experimental setup is shown in Fig. 5.16. The test area is a 15 m^2 flat resin floor. The Motion Capture system is installed around the area to measure and

send the states of the robot to the controller. The controller will run on the onboard computer, which is NVIDIA Jetson ORIN NX. The NVIDIA Jetson ORIN NX will communicate with the PWM controller, which is Pixhawk 6X, through the Ethernet cable. All data will be communicated using the ROS 2 protocol.

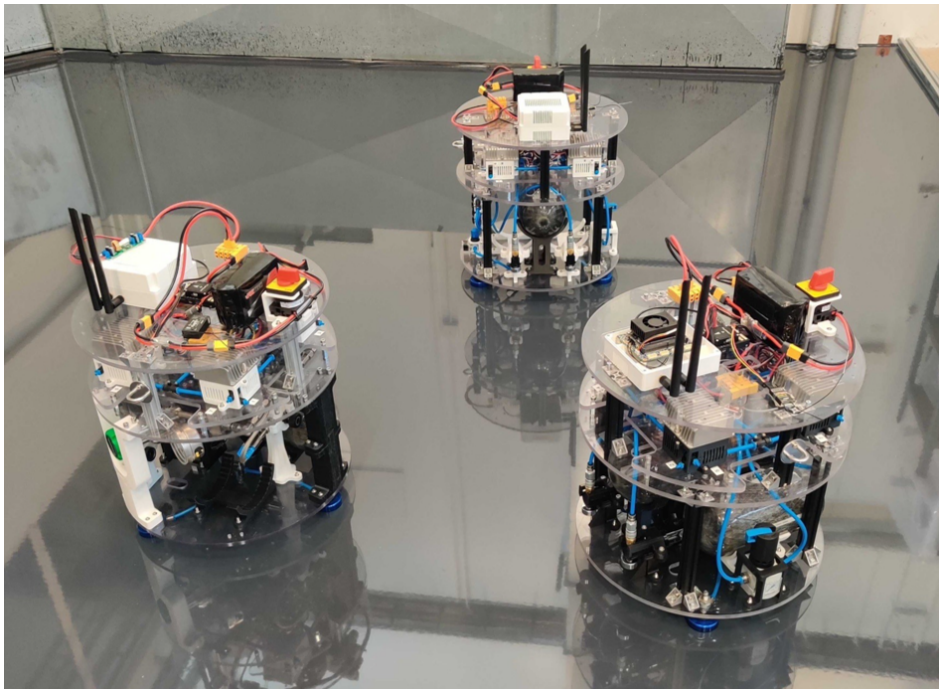
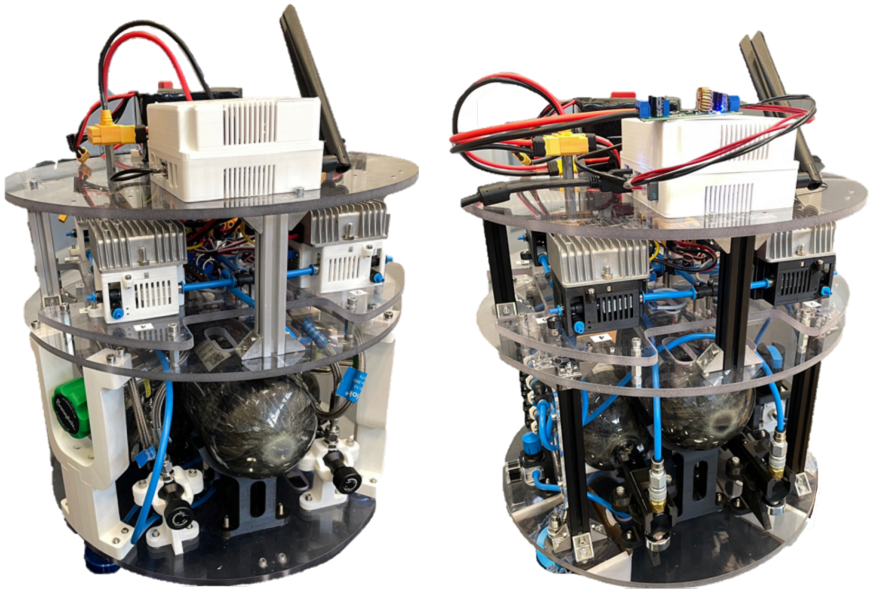


Figure 5.16: Laboratory setup in Space Robotics Lab.

Chapter 6

Platform description

In this chapter, we explain the detailed description of the air carriages used in the experiments. These air carriages are designed to imitate the dynamics of the object in a microgravity environment or space. Since it is difficult to create a full microgravity environment on Earth due to the existence of gravity, the air carriage is designed to imitate the dynamics of the objects only in planar motion. In this specific case, we can introduce a frictionless environment by using an air-bearing carriage and substantially smooth floor [8, 9]. The air-bearing carriage is a device that can float on a thin layer of air, which is generated by compressed air through the bearing. This air cushion will reduce the friction between the robot and the floor to be negligible. There are three main systems in the air carriage: mechanical system, pneumatic system, and electrical system. The mechanical system is the structure that connects the pneumatic and the electrical system. The pneumatic system is the system that stores and generates air for levitation and thruster systems. The electrical system is the system that controls the whole system. There are two design versions: Proto and Alpha. The first version is used as a load, while the second version is used for the robot or the transporter. The design of the Alpha version is based on the design of the Proto version as shown in Fig 6.1.



(a) Proto (b) Alpha

Figure 6.1: Air carriage platforms for the experiments.

6.1 Design Requirements

The goal of the air carriage is to imitate the dynamic of the free-flyer. In this thesis, we design based on Astrobee [31, 32], which is operated in the ISS. The design parameters are chosen to be larger than Astrobee in order to mimic the behaviour as shown in Tab. 6.1.

Table 6.1: Robot Specification

Paremeters	**Astrobee**	**Proto**	**Alpha**
Mass [kg]	10	18.3	14.5
Max. acceleration [m/s^2]	0.1	0.19	0.24
Max. angular acceleration [rad/s^2]	0.59	1.02	1.13
Max. thrust (x-axis)*[N]	0.425	1.75	1.75
Max. thrust (y-axis)*[N]	0.203	1.75	1.75
Max. torque [Nm]	0.085	0.42	0.42
Moment of inertia [kgm^2]	-	0.412	0.370

* Calculated per vent for Astrobee and per thruster for Proto and Alpha.

6.2 Mechanical Design

Our air carriages are designed to be compact and modular. The main structure is divided into three layers: the pneumatic layer, the actuation layer and the electronics layer. The pneumatic layer is the bottom layer that is used to mount the pneumatic system and air bearing. The actuation layer is designed to be replaceable. In this work, we use thrusters as the actuation system. However, one can replace it with other actuators such as propellers. The electrical layer on the top is used to mount the electrical system and battery. This layer also has an extension, which can rather connect to more sensors, including cameras, or other payloads. The dimension of the air carriage is shown in Fig. 6.2 and Fig.6.3. for Proto and Alpha, respectively.

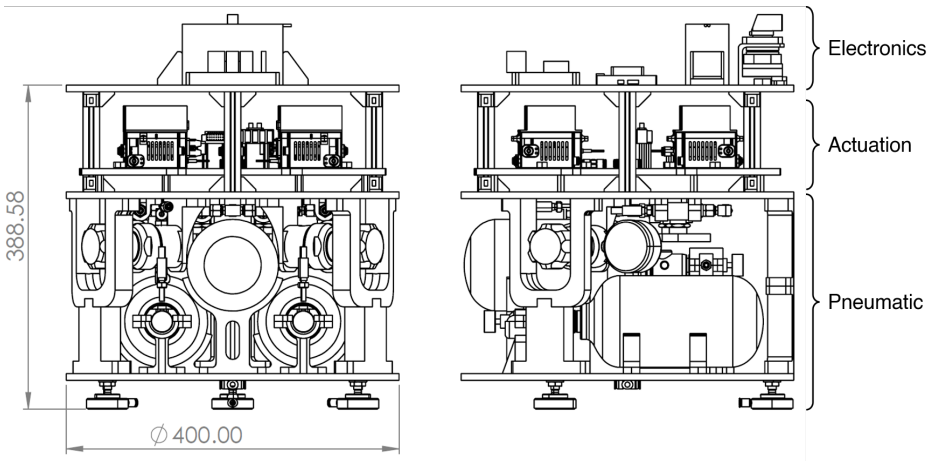


Figure 6.2: Proto version.

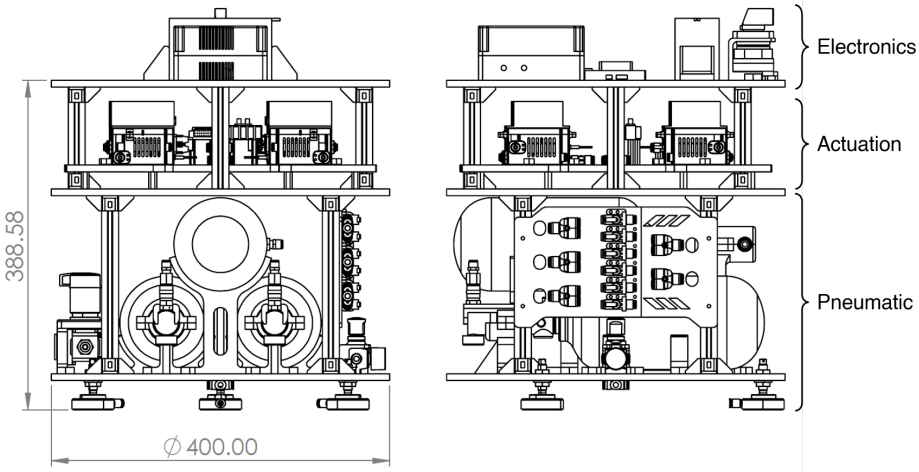


Figure 6.3: Alpha version.

6.3 Pneumatic System

In order to imitate the dynamics of the robots in the microgravity environment, there are two main systems: levitation and thruster system. The former system is used to demonstrate the frictionless scenario in planar motion or three-degree-of-freedom (3DOF) system. The latter system is designed to generate the force and torque to actuate the robot. To generate the air for both systems, we need to design the pneumatic system. This system consists of two subsystems: air storage and air regulation. The air storage is used to store the air from the air compressor. There are three air tanks for each robot. The air regulation is used to regulate and supply air at the proper pressure for both levitation and thruster systems. This system is designed in a customized way to fit the requirements of the air carriage. To illustrate, it can select the number of air tanks for each main system. For instance, one can choose one air tank for the levitation system and two air tanks for the thruster system for the active air carriage. Also, one can choose three air tanks for the levitation in order to make air carriage act as a passive load. Output air from the tanks is then regulated by the regulator to a desired operating pressure for each main system. The whole pneumatic system is displayed in Fig. 6.4.

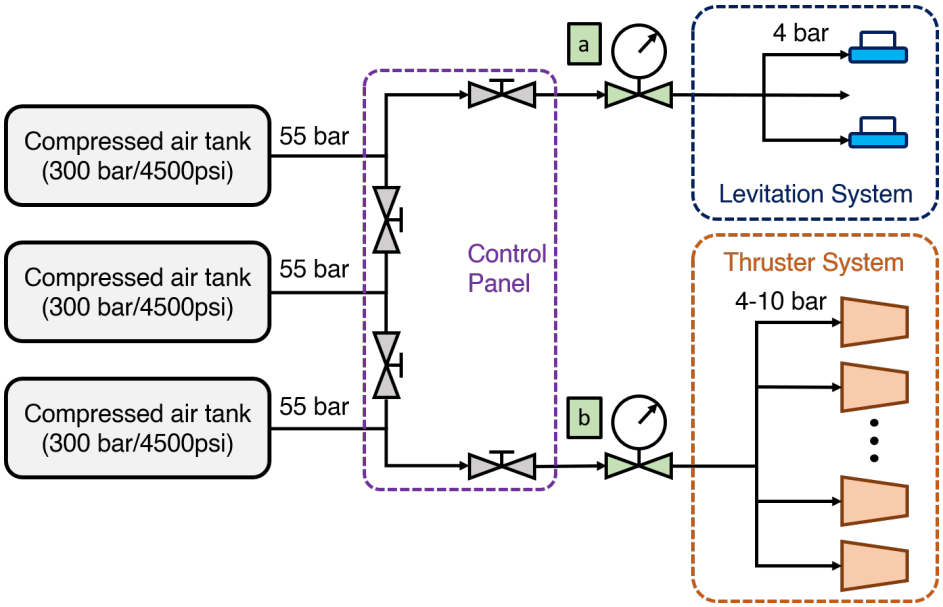


Figure 6.4: Schematic of the pneumatic system shows important components.

This system is also the main difference between the Proto and Alpha versions. The Proto version uses 1.3L air tanks and two industrial high-pressure regulators while the Alpha version uses 1.5L air tanks, three bottle regulators and two low-pressure regulators. The reason for this change is to reduce the weight and cost and increase the operating time of the air carriage. Additionally, with the new design, the control panel or the pneumatic manifold is also changed. The Alpha version uses smaller valves and a smaller manifold, which can attach to the side of the robot instead of the top of the robot. This modification is intended to make it simpler for the user to configure the air carriage. Moreover, there are additional components in the Alpha version, which are high-pressure regulators. These regulators are connected directly to the bottle regulator to reduce the pressure from 55 bar to 10 bar. With this configuration, the new air carriage does not require an industrial regular as in the old version. As a result, the smaller regulator can be used, which is lighter and cheaper. The new pneumatic system is shown in Fig. 6.5. Components for Proto and Alpha versions are summarized in Tab. 6.2.

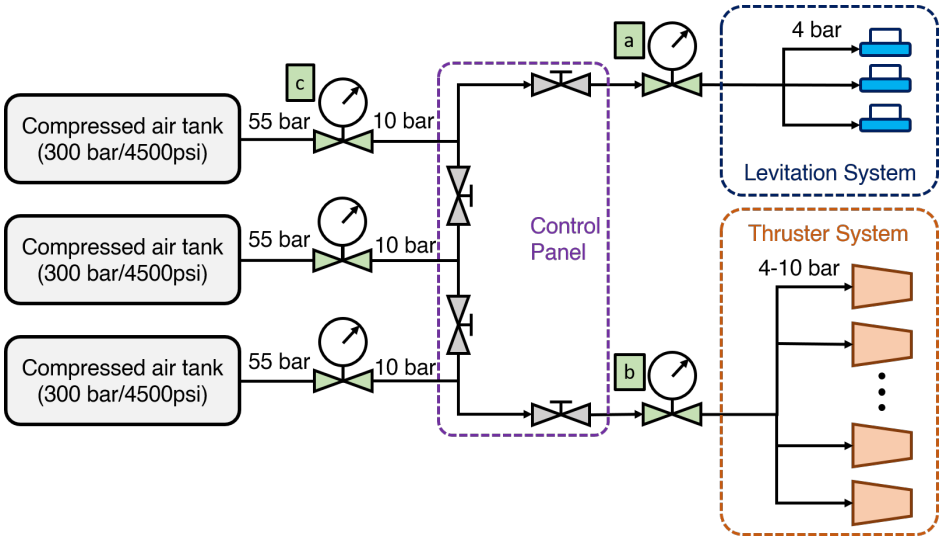


Figure 6.5: Schematic of the new pneumatic system includes additional high-pressure regulators.

Table 6.2: Pneumatic components

Items	Proto
Levitation regulator (a)	AS-STKHM-KPR1FLC411A200H0*
Thruster regulator (b)	AS-STKHM-KPR1GLB415A200H0*
Compressed air bottle	DYE CORE AIR TANK 1.3L 4500PSI
Bottle regulator	DYE LT THROTTLE REGULATOR 4500PSI

* Components from Swagelok

Items	Alpha
Levitation regulator (a)	MS4-LR-1/4-D7-AS+
Thruster regulator (b)	MS4-LR-1/4-D7-AS+
Compressed air bottle	DYE CORE AIR TANK 1.5L 4500PSI
Bottle regulator	DYE LT THROTTLE REGULATOR 4500PSI
High pressure regulator (c)	Polarstar micro MR GEN2 Regulator

+ Components from Festo

6.3.1 Levitation System

This system utilizes the characteristics of the air bearings to demonstrate frictionless behaviour. These specific bearings generate a thin film of pressurized air as an air cushion between the bearing and the contact area.

This film provides a zero-friction interface between the surfaces. To create this film, air needed to be supplied properly through the bearing surface into the gap. In this thesis, we use NEWWAY air bearing S105001 * as shown in Fig. 6.6, which requires an input pressure of 60 psi (4.1 bar) and it can hold the load up to 35 kg. There are 3 bearings on each robot.

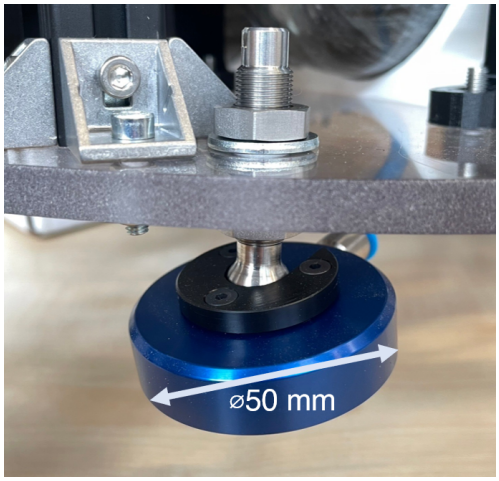


Figure 6.6: Air bearing at the base of the robot.

6.3.2 Thruster System

Another important system in the air carriage is the thruster or actuation system. This system is used to drive the robot in the desired direction. The thruster uses the same design principle as the rocket by pushing out the gas from the nozzle to generate thrust. Compressed air is used instead of the propellant in this carriage. There are two thrusters on each side of the robot, a total of eight as shown in Fig 3.3. This configuration allows the robot to move in all directions in a planar coordinate (3DOF) and rotate around the z-axis (the axis perpendicular to the plane). We use an MHJ10-S-2,5-QS-6-HF solenoid valve to open and close the airflow of the thruster. The valve is controlled by the PWM signal from a microcontroller. The solenoid valve is connected to the nozzle through the 6 mm tube (4 mm inner diameter). This nozzle is a simple 4 mm cylindrical tube (2 mm inner diameter). In order to make this system compact, we combine two solenoid valves, two nozzles, and one buck-boost converter into one module as shown in Fig 6.7. There are four modules on each robot. The schematic of the thruster system is shown in Fig. 6.8.

*https://www.newwayairbearings.com/catalog/product/50mm-flat-round-air-bearings

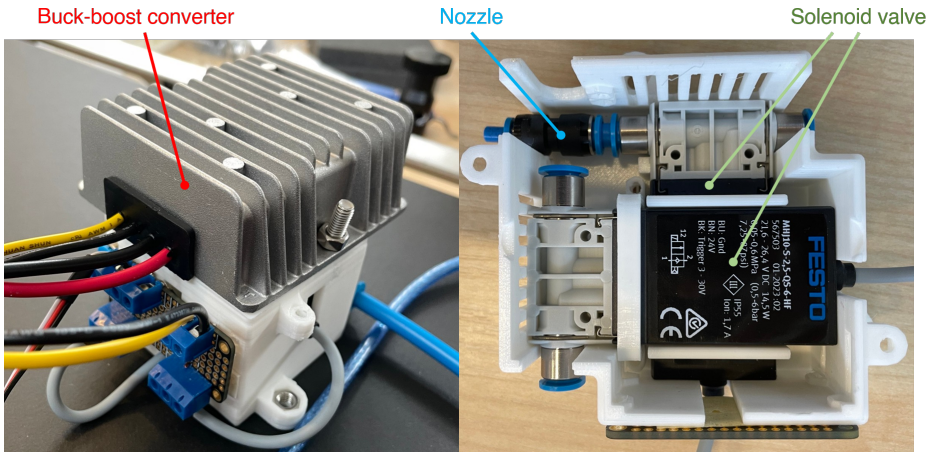


Figure 6.7: Thruster module consists of two solenoid valves, two nozzles and one buck-boost converter.

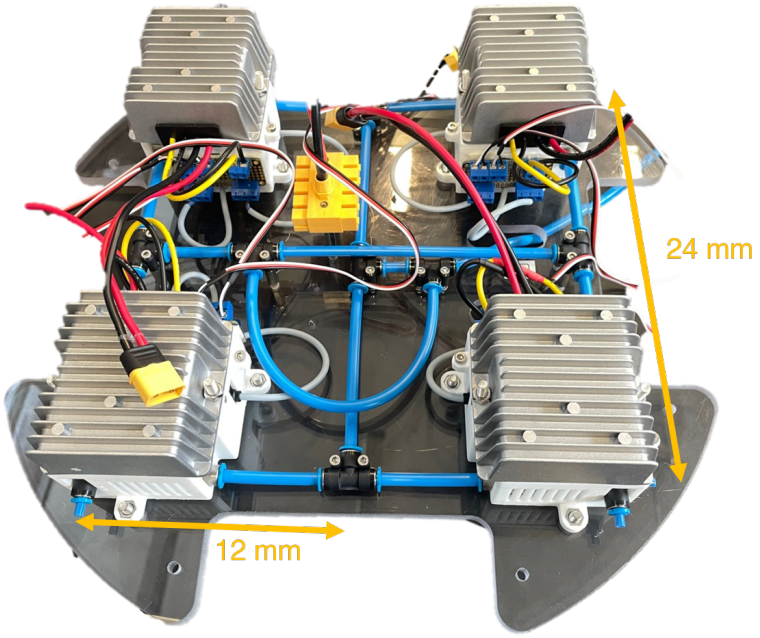


Figure 6.8: Thruster configuration of the robot on the actuation layer.

6.4 Electronics

This system is used to operate the robot. The main components are a computer, microcontroller, and battery. NVIDIA Jetson ORIN NX is used as the main

computer. This onboard computer is used to run the control algorithm (i.e., MPC) and communicate with the microcontroller. We choose Pixhawk 6X to be the microcontroller, which is used to send the PWM signal to control the solenoid valve. Foxtech 6S 9500mAh Li-ion Battery is used to supply power to the whole system. The schematic of the electrical system is shown in Fig. 6.9.

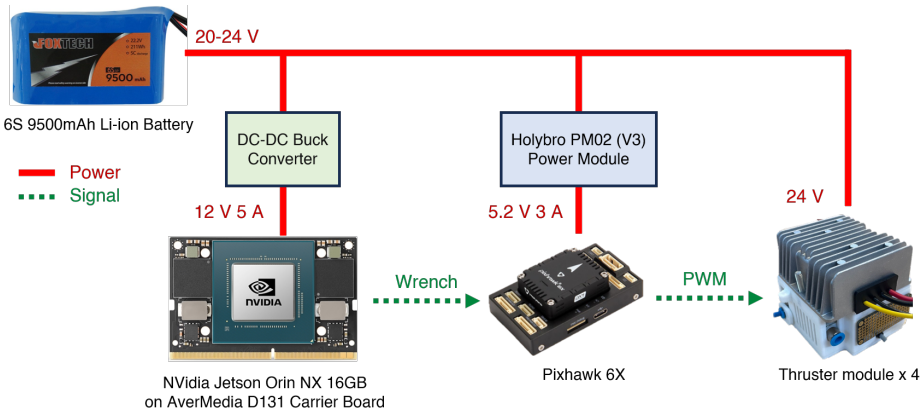


Figure 6.9: Electronics system of the platform.

Chapter 7

Conclusions and Future work

In this thesis, we have proposed a centralized and decentralized MPC to achieve the multi-agent load transportation task in a microgravity environment. Also, the dynamics model of the n-agent tethered system in three dimensions is derived. This model is then used to design the predictive controllers. We also propose a way to reduce the nonlinearity of the system by reformulating the optimization problem by introducing a new minimizer instead of a full explicit model. The key advantage of this approach is the faster computation time, which allows the controller to run at a higher frequency. Next, we design a centralized controller for the multi-agent system. The controller is then tested in the simulation environment. The simulation results show that our controller is capable of tracking the desired point and trajectory. To demonstrate the practicality of our controller on the real-world system, we develop a physical simulator based on the actual air carriage's behaviour in planar motion. this simulation is used to verify the performance of our controller in the real-time environment. Our centralized controller is then tested again in this environment and the results show that our controller is capable of tracking the desired points. However, the computation time is still a challenge for the centralized controller. Therefore, we propose a decentralized controller to solve this problem. By dividing the system into several subsystems, we can reduce the number of states, inputs and constraints in the optimization problem, resulting in faster computation time. The decentralized controller is also tested in the simulation and shows the capability of tracking the desired point and trajectory, even without communication.

Furthermore, we present the development of the modular design air carriage. The design requirement is based on NASA Astrobee in terms of

velocity and acceleration. The design is then optimized to reduce the weight and increase the usability of the system. The final design is then fabricated and tested in the lab. The carriage uses the air bearing to imitate the frictionless environment. It also uses the same pneumatic system to generate the thrust in order to manoeuvre around. This thruster system uses the same principle as the rocket in space, which is the conservation of momentum. PWM controller is then used to control each solenoid valve to open and close in a designed manner.

Through the outcomes of the experiments, although our centralized controller can move the load following the desired setpoints, they are still limited by the computation time, resulting in the challenge of scaling the controller up for more agents. The computation time is not only mainly affected by the number of states and constraints, but also the nonlinearity of the system. The proposed decentralized scheme is just one of several solutions to address this problem. Future work can address these problems by reformulating the optimization problem and also linearizing the system. The decentralized controller, on the other hand, can achieve the desired setpoints with a faster computation time. Additionally, this approach will suffer less from the curse of dimensionality. However, the system requires a longer time to converge to the desired point. This is due to the fact that the controller is not able to perceive the position of the other agents. As a result, the controller uses the virtual cable tension to estimate the actual cable tension. Therefore, the solution control input from the controller is not optimal, leading to a longer converging time and oscillating behaviour. In future work, we can address this problem by formulating a better assumption for the virtual cable tension. Also, one can also address this problem by introducing communication between the agents using distributed control framework. This will allow the controller to perceive the position of the other agents, resulting in the optimal control input. Finally, the more obvious benefits of the proposed controllers need also to be shown by applying this control algorithm to actual robots.

References

[1] "The Current State of Space Debris." [Online]. Available: https://www.esa.int/Space_Safety/Space_Debris/The_current_state_of_space_debris [Page 2.]

[2] N. Orbital Debris Program Office, "Orbital Debris Quarterly News 27-2," Tech. Rep. [Online]. Available: https://software.nasa.gov/software/ [Page 2.]

[3] Donald J. Kessler and Burton G. Cour-Palais, "Collision Frequency of Artificial Satellites: The Creation of a Debris Belt," 1978. [Page 2.]

[4] M. Shan, J. Guo, and E. Gill, "Review and comparison of active space debris capturing and removal methods," pp. 18–32, 1 2016. [Page 2.]

[5] A. Ledkov and V. Aslanov, "Review of contact and contactless active space debris removal approaches," 10 2022. [Page 2.]

[6] M. Ekal, K. Albee, B. Coltin, R. Ventura, R. Linares, and D. W. Miller, "Online Information-Aware Motion Planning with Inertial Parameter Learning for Robotic Free-Flyers," 12 2021. [Online]. Available: http://arxiv.org/abs/2112.05878 [Page 2.]

[7] NASA ODPO, "LEO-2019-4096." [Online]. Available: https://orbitaldebris.jsc.nasa.gov/photo-gallery/_images/highresolution/LEO-2019-4096.jpg [Pages vi and 3.]

[8] Y. Nakka, R. Foust, E. S. Lupu, and S.-J. Chung, "Six Degree-of-Freedom Spacecraft Dynamics Simulator for Formation Control Research Chance Constrained Nonlinear Stochastic Optimal Control For Motion Planning View project," Tech. Rep., 2018. [Online]. Available: https://www.researchgate.net/publication/327067426 [Pages 3 and 56.]

[9] A. Banerjee, J. Haluska, S. G. Satpute, D. Kominiak, and G. Nikolakopoulos, "Slider: On the Design and Modeling of a 2D Floating Satellite Platform," 1 2021. [Online]. Available: http://arxiv.org/abs/2101.06335 [Pages 3 and 56.]

[10] R. C. Sundin, P. Roque, and D. V. Dimarogonas, "Decentralized Model Predictive Control for Equilibrium-based Collaborative UAV Bar Transportation," Tech. Rep., 2022. [Pages 3 and 4.]

[11] W. S. Cortez, C. K. Verginis, and D. V. Dimarogonas, "A Distributed, Event-Triggered, Adaptive Controller for Cooperative Manipulation with Rolling Contacts," *IEEE Transactions on Robotics*, vol. 39, no. 4, pp. 3120–3133, 8 2023. doi: 10.1109/TRO.2023.3268595 [Page 3.]

[12] B. Wang, Z. Meng, and P. Huang, "Attitude control of towed space debris using only tether," *Acta Astronautica*, vol. 138, pp. 152–167, 9 2017. doi: 10.1016/j.actaastro.2017.05.012 [Page 4.]

[13] Mario L. Cosmo and Enrico C. Lorenzini, "Tethers In Space Handbook-Second Edition," Tech. Rep., 1997. [Page 4.]

[14] A. Tagliabue, M. Kamel, S. Verling, R. Siegwart, and J. Nieto, "Collaborative transportation using MAVs via passive force control," in *Proceedings - IEEE International Conference on Robotics and Automation*. Institute of Electrical and Electronics Engineers Inc., 7 2017. doi: 10.1109/ICRA.2017.7989678. ISBN 9781509046331. ISSN 10504729 pp. 5766–5773. [Page 4.]

[15] C. Yang, G. N. Sue, Z. Li, L. Yang, H. Shen, Y. Chi, A. Rai, J. Zeng, and K. Sreenath, "Collaborative Navigation and Manipulation of a Cable-Towed Load by Multiple Quadrupedal Robots," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 10 041–10 048, 10 2022. doi: 10.1109/LRA.2022.3191170 [Page 4.]

[16] M. Gassner, T. Cieslewski, and D. Scaramuzza, "Dynamic Collaboration without Communication: Vision-Based Cable-Suspended Load Transport with Two Quadrotors," Tech. Rep. [Online]. Available: http://www.ifi.uzh.ch/en/rpg.html [Page 4.]

[17] S. Erhart, D. Sieber, and S. Hirche, "An impedance-based control architecture for multi-robot cooperative dual-arm mobile manipulation," Tech. Rep. [Page 4.]

[18] P. Behruzi, D. Roascio, D. R. Kirk, G. Lapilli, and H. J. Zachrau, "SPHERES TETHER SLOSH free flyer experiment on ISS," in *2018 Joint Propulsion Conference*. American Institute of Aeronautics and Astronautics Inc, AIAA, 2018. doi: 10.2514/6.2018-4939. ISBN 9781624105708 [Page 4.]

[19] H. T. Linskens and E. Mooij, "Tether dynamics analysis and guidance and control design for active space-debris removal," *Journal of Guidance, Control, and Dynamics*, vol. 39, no. 6, pp. 1232–1243, 2016. doi: 10.2514/1.G001651 [Page 4.]

[20] T. Lee, "Geometric Control of Quadrotor UAVs Transporting a Cable-Suspended Rigid Body," *IEEE Transactions on Control Systems Technology*, vol. 26, no. 1, pp. 255–264, 1 2018. doi: 10.1109/TCST.2017.2656060 [Page 4.]

[21] F. De Vincenti and S. Coros, "Centralized Model Predictive Control for Collaborative Loco-Manipulation," Tech. Rep., 2023. [Page 4.]

[22] J. Kim, R. T. Fawcett, V. R. Kamidi, A. D. Ames, and K. A. Hamed, "Layered Control for Cooperative Locomotion of Two Quadrupedal Robots: Centralized and Distributed Approaches," 11 2022. [Online]. Available: http://arxiv.org/abs/2211.06913 [Pages 5 and 30.]

[23] F. Dunn and I. Parberry, "3D Math Primer for Graphics and Game Development Second Edition," Tech. Rep. [Page 7.]

[24] F. Landis Markley and J. L. Crassidis, "Fundamentals of Spacecraft Attitude Determination and Control," Tech. Rep., 2014. [Online]. Available: http://www.springer.com/series/6575 [Pages 7 and 8.]

[25] Y.-B. Jia, "Quaternions* (Com S 477/577 Notes)," 2022. [Pages 8 and 9.]

[26] D. Q. Huynh, "Metrics for 3D rotations: Comparison and analysis," *Journal of Mathematical Imaging and Vision*, vol. 35, no. 2, pp. 155–164, 10 2009. doi: 10.1007/s10851-009-0161-2 [Page 11.]

[27] J. B. Rawlings, D. Q. Mayne, and M. M. Diehl, "Model Predictive Control: Theory, Computation, and Design 2nd Edition," Tech. Rep. [Online]. Available: http://www.nobhillpublishing.com [Page 14.]

[28] J. Albersmeyer and M. Diehl, "The Lifted Newton Method and Its Application in Optimization," *SIAM Journal on Optimization*, vol. 20, no. 3, pp. 1655–1684, 1 2010. doi: 10.1137/080724885 [Page 15.]

[29] C. M. Pong, A. Saenz-Otero, and D. W. Miller, "Autonomous thruster failure recovery on underactuated spacecraft using model predictive control AUTONOMOUS THRUSTER FAILURE RECOVERY ON UNDERACTUATED SPACECRAFT USING MODEL PREDICTIVE CONTROL," Tech. Rep., 2011. [Page 17.]

[30] J. A. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi: a software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 3 2019. doi: 10.1007/s12532-018-0139-4 [Page 38.]

[31] K. Albee, M. Ekal, C. Oestreich, and P. Roque, "A Brief Guide to Astrobee's Flight Software Revision 1.1," Tech. Rep., 2021. [Online]. Available: https://github.com/albee/a-brief-guide-to-astrobee [Page 57.]

[32] "Astrobee Guest Science Guide," Tech. Rep., 2017. [Online]. Available: https://www.nasa.gov/astrobee. [Page 57.]

TRITA-EECS-EX- 2023:0000