University of Tartu

Faculty of Science and Technology

Institute of Technology

Houman Masnavi

MULTI-ROBOT MOTION PLANNING FOR SHARED PAYLOAD TRANSPORTATION

Master's thesis (30 EAP)
Robotics and Computer Engineering

Supervisors:

Karl Kruusamäe
Arun Kumar Singh
Robert Valner

Tartu 2020

Abstract

MULTI-ROBOT MOTION PLANNING FOR SHARED PAYLOAD TRANSPORTATION

Shared payload transportation has emerged as one of the key real-world applications that warrants the deployment of multiple robots. The key motivation stems from the fact that actuation and sensing abilities of multiple robots can be pooled together to transport objects that are either too big or heavy to be handled by a single robot. This thesis proposes algorithmic and software frameworks to achieve precise multi-robot coordination for object transportation. On the algorithmic side, a trajectory optimization formulation is developed which generates collision-free and smooth trajectories for the robots transporting the object. State-of-the art Gradient Descent variants are utilized for obtaining the solution. On the software side, a trajectory planner (local planner) is developed and integrated to Robot Operating System (ROS). The local planner is responsible for calculating individual velocities for any number of robots forming a rigid geometric in-plane constellation. Extensive simulation as well as real-world experiments are performed to demonstrate the validity of the developed solutions. It is demonstrated that how the proposed trajectory optimization approach outperforms off-the-shelf planners with respect to metrics like smoothness and collision avoidance.
Keywords: Object transportation, motion planning, fleet management, ROS

CERCS: T125 Automation, robotics, control engineering

Abstract in Estonian

RAJAPLANEERIMINE MULTI-ROBOT SÜSTEEMILE JAGATUD LASTI TRANSPORTIMISEL

Ühise lasti transportimine mitme roboti poolt on kujunenud üheks rakendusvaldkonnaks, kus mitme roboti samaaegne kasutamine on õigustatud. Mitme roboti andureid ja ajameid on eriti kasulik kasutada transportimaks objekte, mis on ühe roboti jaoks kas liiga suured ja/või rasked. Käesolev lõputöö pakub välja algoritmilise ja tarkvaralise raamistiku, mis võimaldab täpselt koordineerida mitme roboti koostööd ühise lasti liigutamisel. Välja on töötatud trajektooride optimeerimise algoritm, mis genereerib kokkupõrkevabad ja sujuvad ühist objekti kandvate robotite trajektoorid. Selleks on kasutatud nüüdisaegset gradientlaskumise (ingl Gradient Descent) meetodit. Tarkvara poolelt on loodud trajektoori planeerija (lokaalne planeerija) ja see on integreeritud arendusplatvormil ROS (Robot Operating System). Lokaalne planeerija arvutab individuaalsed kiirused igale robotile, mis moodustavad ühise jäiga tasapinnalise kujundi, kusjuures robotite arv kujundis ei ole piiratud. Väljatöötatud lahenduse toimimist on kontrollitud ulatuslike simulatsioonide abil aga ka viies läbi praktilisi katseid. Väljapakutud trajektoori optimeerimise lahendus ületab olemasolevaid planeerijaidd nii trajektoori sujuvuse kui ka kokkupõrgete vältimise võime osas.

võtmesõnad: objektide transport, rajaplaneerimine, parvehaldus, ROS

CERCS: T125 Automatiseerimine, robootika, juhtimistehnika

Contents

List of Figures

List of Tables

Table 1: Mathematical definitions

Num	Symbol	Definition
1	x_t	x position of the base footprint's origin at time instant t
2	y_t	y position of the base footprint's origin at time instant t
3	ϕ_t	orientation of the base footprint's origin at time instant t
4	x_0 and y_0	coordinates of the base footprint's origin
5	${}^{i}\phi$	angle of the i^{th} robot with respect to base footprint's origin
6	superscript i	i^{th} robot
7	d^{oj}	radius of j^{th} obstacle
8	r_i	distance between i^{th} circle on the rod and the center point of the rod
9	x^{oj} and y^{oj}	center point coordinates of the j^{th} obstacle
10	${}^{i}e$	distance of the i^{th} robot from the origin (0,0) of base footprint
11	${}^{i}dt_n$	time difference between each two consecutive waypoints in i^{th} robot trajectory
12	${}^{i}vx_t$	linear x velocity of i^{th} robot at time instant t
13	${}^{i}vy_t$	linear y velocity of i^{th} robot at time instant t
14	${}^{i}v\phi_t$	angular z velocity of i^{th} robot at time instant t

1 Introduction

1.1 Background

Multi-robot systems, i.e, a group of robots working together to accomplish a given task have been rapidly evolving and gaining popularity in recent years [1]. The motivation stems from the fact that as compared to using single robots, multi-robot systems provide increased efficiency and robustness to failures in applications like search and rescue, object transportation, mapping and localization, inspection etc [2]. For example, a group of robots can map a given area quicker than a single robot. Furthermore, even if one robot breaks down, the task can still be completed, albeit with some reduced efficiency (e.g increase in mapping time). Another motivation for using multiple robots in a task is that one can potentially pool the sensing and computation ability of individual robots together to accomplish a task which cannot be performed by a single robot. A very apt example of this observation is the object transportation application. A group of robots can be used to carry objects that are too heavy for the actuation ability of individual robots.

1.2 Motivation

There have been algorithms and solutions to multi-robot motion planning problems for the purpose of object transportation, however, there hardly exist practical implementations of them, in order for everyone to easily leverage those solutions. ROS (Robot Operating System) is vastly adopted by the robotics community and a growing number of robots either support it or are based on it, hence making it a great choice to serve the purpose of bringing practical implementations. Different libraries and tools are incorporated into ROS such as Navigation - a set of software capabilities for controlling mobile robots. ROS Navigation leverages two types of planners, namely local and global, to plan for a robot and make it navigate towards the goal [3]. Although algorithms and planners exist in ROS to plan for a team of stationary manipulators, ROS Navigation is designed only for commanding a single mobile robot system.

1.3 Objectives and Contributions

The objective of this thesis is to integrate multi-robot motion planning into ROS Navigation. In this work, a specific case of multi-robot application, namely shared payload transportation is considered. In this task, several robots are deployed to carry a shared object from a

given point to a goal point with associated performance requirements such as speed, obstacle avoidance, and damage reduction (see Figure 1.1). In light of the aforementioned objective, the contribution of this thesis is twofold:

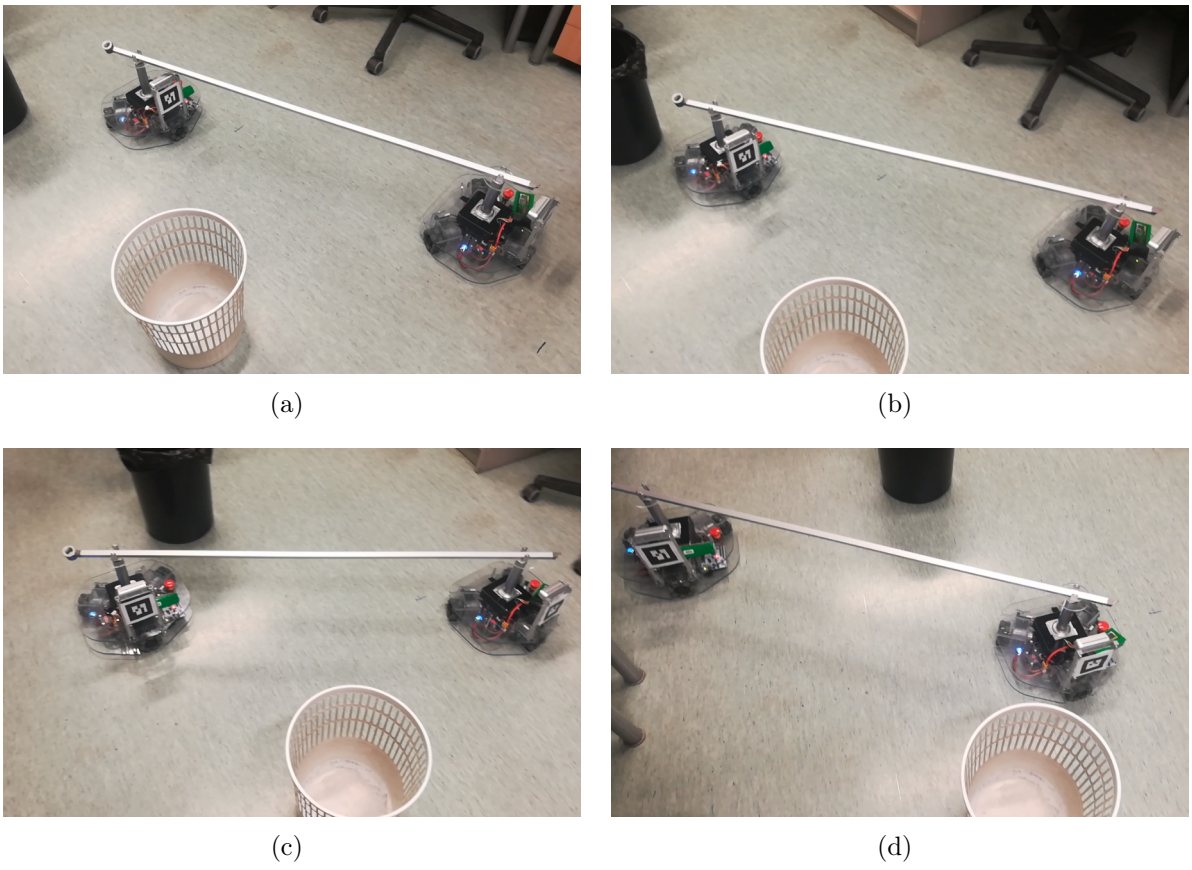
- **C1:** A motion planning algorithm is developed which can generate smooth, collision-free trajectories for a pair of robots transporting an object. The motion planning problem is formulated as an optimization problem where the desired behaviors of the robots are modelled through appropriately designed cost and constraint functions. Several problem-specific customizations are introduced to simplify the mathematical formulation. An analysis is presented for different first order gradient based optimization algorithms that can be employed to solve the motion planning problem. In addition, it is shown how the proposed optimization-based approach outperforms an existing off-the-shelf ROS motion planner in terms of metrics like trajectory smoothness and collision avoidance.
- **C2:** Multi-robot motion planning capability is added to ROS Navigation. In particular, a local planner is developed which takes as an input the desired trajectory for a given payload and generates corresponding individual velocities for a group of robots. This specific contribution plugs the existing gap in ROS Navigation which allows for simultaneous actuation command of only single robot. To the best of the author's knowledge, there is no implementation of a motion planner able to compute trajectories and velocities for multiple mobile bases in ROS Navigation.

It is worth pointing out that the developed local planner can be used together with the developed or any of the ROS Navigation's existing global planners to compute individual velocities for every robot carrying the shared payload and it can handle arbitrary number of robots carrying an object of any shape or size. However, the global motion planner developed in **C1** is specific to the case of a pair of robots transporting an object.

1.4 Organization of Thesis

The thesis is organized as follows:

- **In the current chapter:** Introduction, motivation, contributions and objectives of the thesis are discussed.
- **Background:** This chapter explains motion planning as well as the preliminary concepts. The process flow of ROS Navigation is also presented.
- **Design:** Detailed explanation of the work, the proposed global planning algorithm as well as the local planner are presented.
- **Results:** The findings of the work and the comparisons are shown.
- **Future work:** Feasible future directions and improvements are stated.
- **Conclusion**: The outcomes of the work are discussed.



(a) (b)

(c) (d)

Figure 1.1: How a rod-like object is transported using two holonomic robots. Since the rod's length is longer than the distance between the obstacles, the robots move and change their relative pose in such way that results in the rod going in between the obstacles and reaching the goal without having collisions.

2 Background

This chapter explains related concepts used in motion planning and robot navigation. First, motion planning and its various techniques are discussed. Then, the concept of trajectory optimization and its pertinent definitions are introduced. Finally, overall architecture and process flow of ROS Navigation is explained.

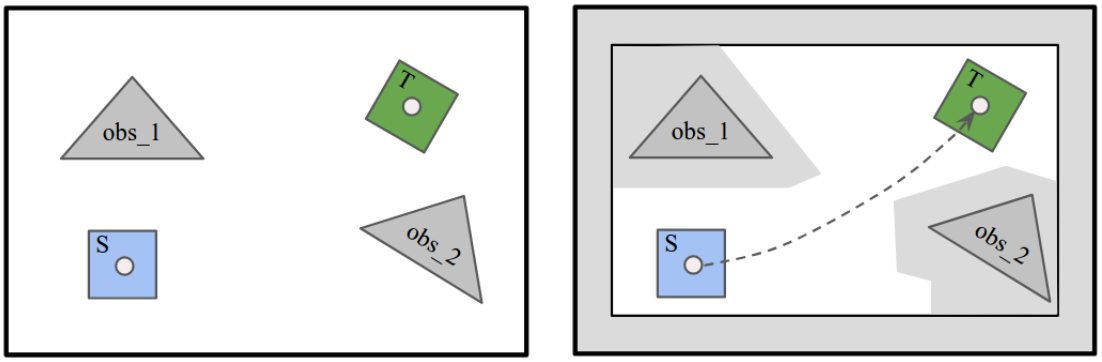
2.1 Motion planning

Motion planning, in its most basic form, is the process of connecting a given start and end pose of a robot in an environment with obstacles with a collision-free trajectory [4]. The spatial pose of a robot, is defined by its degrees of freedom (DoF). The set of all kinematically-valid poses that a robot can obtain C, is termed as the robot's configuration space (i.e. the $C - space$)[5]. Within the $C - space$ exist free and obstacle subsets. Free space is denoted as C_{free} and it is the set of configurations avoiding collision with obstacles. Obstacle or forbidden space, C_{obs}, is the complement of C_{free} in C. That is, C_{obs} is where the obstacles are and the robot should not be in this space at any time instant. Configuration spaces are determined using the robot's geometry and its degrees of freedom along with the map provided to the planner algorithm. Figure 2.2 depicts C_{free} and C_{obs} configurations.

There are two broad classes of motion planning algorithms, namely sampling based and optimization based. In the following, the basic concepts related to them are discussed.

2.1.1 Sampling Based Planner

Sampling based planners have four main steps [6]. Firstly, they need to generate a sequence of robot configurations by sampling from some known probability distributions (e.g. a uniform distribution). Subsequently, the generated samples are tested for collision by a collision checker that given a robot configuration outputs whether it is in collision or not. The third step is computing the proximity of the sampled robot configurations from a given set of configurations. Finally, a function generates a trajectory between two sampled configurations. There are two different approaches for implementing the above four steps, namely Probabilistic Road Maps (PRM) [7] and Rapidly Exploring Random Trees (RRT) [8]. There are different variants of PRM and RRT, designed to remove some of their key shortcomings. One of the key problems with PRM and RRT is that it often produces collision-free trajectories that are highly sub-optimal. For example, the generated trajectory may include large detours



(a) (b)

Figure 2.1: *Motion planning for a 3-dof planar square-shaped robot. The starting and the target points are shown by S and T, respectively. Shown in white in (a) is the configuration space. The white area and light-gray highlights brought in (b) represent the free and forbidden configuration spaces, respectively. Notice that the curve connecting start and target configurations is entirely collision free.*

around the obstacles. Thus, algorithms like PRM^* and RRT^* [9] introduce notion of shortest path within the PRM and RRT algorithm, respectively. Collision checking is another key bottleneck in sampling based planners as most of the computation time is spent in this step. Algorithms like Lazy PRM [10] solve this problem by performing collision checks on only a subset of sampled configurations. Some works like [11] accelerate the collision checking process itself by using supervised learning.

2.1.2 Trajectory Optimization

An alternate approach for motion planning is to formulate it as an optimization problem. This approach, popularly known as trajectory optimization, has become the preferred motion planning approach for several robotic applications ranging from manipulation [12], [13], [14] to autonomous driving [15]. Succinctly, a trajectory optimization problem has the following generic mathematical form.

$$\begin{aligned} &\text{minimize} && f_0(\mathbf{x}), && && (2.1)\\ &\text{subject to} && f_i(\mathbf{x}) \leq b_i, && i = 1, ..., m, && (2.2)\\ &\text{subject to} && g_j(\mathbf{x}) = \mathbf{0}, && j = 1, ..., n, && (2.3) \end{aligned}$$

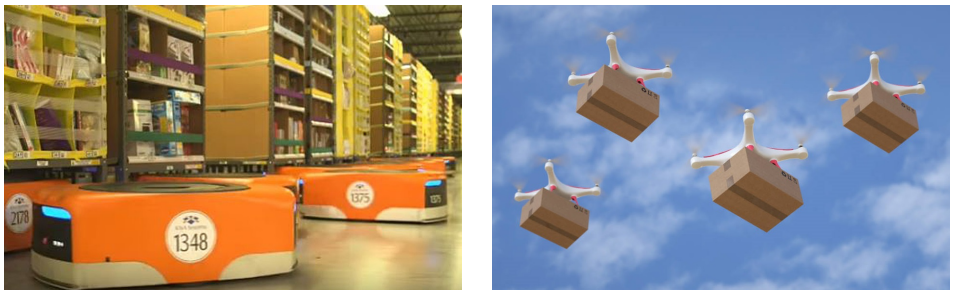
where the variable $\mathbf{x}$ can be used to represent the various poses that the robot can take. Its definition can also be expanded to incorporate the various velocities, accelerations as well as the control inputs required to move the robot. The f_0 in (2.1) is called the cost function and models the notion of optimality in the motion planning. For example, one can construct a suitable f_o to generate trajectories that connect the start and end pose of the robot with the shortest possible path. The function f_i in inequality constraints (2.2) can be used to model constraints such as collision avoidance. Along similar lines, the function g_j in (2.3) can be

used to model constraints imposed by the dynamics of the robot or the task it is performing [14].

Trajectory optimization problems are computationally simple (i.e the global minimum can be computed in polynomial time) when they have a property called convexity wherein the functions f_o, f_i are shaped like a bowl and g_j has the shape of a hyper-plane [16]. Mathematically, the bowl shape of f_o, f_i results from their Hessian (or second derivative) being positive semi-definite. The function g_j in turn should have an affine form to have the hyper-plane shape. Unfortunately, most robot motion planning problems are non-convex wherein the cost and constraint functions have an arbitrary non-linear form (e.g product of trigonometric functions). Such optimization problems have several local minimums besides the single global minimum. Obtaining the global minimum in such problems is computationally intractable. Thus, existing works focus on obtaining just the locally optimal solution. To this end, algorithms like Gradient Descent [12] and sequential quadratic programming [14] are often used. Gradient descent is an approach to minimize a cost function parameterized by a model's parameters through updating the parameters in the opposite direction of the cost function's gradient with respect to its parameters.

2.1.3 Multi-robot Motion Planning

In recent years multi-robot systems have been gaining ground due to their various applications, including object transportation, package delivery, and mobile manipulation [17]. One of the most famous instances of these systems is Amazon Kiva robots that deploys hundreds of robots to facilitate order assembly tasks [18, 19]. Moreover, Google, Amazon, and DHL have shown working prototypes of aerial vehicles capable of automated package delivery. Since the vehicles are intended to operate in an autonomous, swarm-like setting, we would foresee the demand in the future for efficient path-planning algorithms designed for such systems [19].



(a) Amazon Kiva robot fleet (b) Multi-drone package delivery

Figure 2.2: *Examples of multi-robot systems*

One of the key challenges in this field is multi-robot motion planning, which is an extension of motion planning and includes collision-free coordination of a robot fleet in a shared environment. Motion planning for multiple robots is considered difficult as it often involves many degrees of freedom, and consequently a vast search space [20].

Planning for a fleet of robots can either be coupled or decoupled. In the former, the planner computes the path in a combined configuration space, which treats with all the robots as one combined robot. While in the latter, the motion planner generates the path for each robot separately. Planning for a robot fleet can also be categorized as either *labelled* or *unlabelled*. In the labelled setting, each robot in a crowd needs to move from a specific start configuration to another specific target configuration. It means each robot has its own individual task. In some practical cases, robots are identical in terms of functionality and thus the tasks for different robots can be interchanged. However, in the unlabelled scenario, robots are given a set of target positions and the goal is to move the robots in a collision-free manner so that each robot ends up at some target, without initially knowing the target [21, 20]. In both of these scenarios, the planner should be robust enough to plan for a fleet of robots and accommodate for any disturbances or deviations.

Multi-robot path planning in complex, yet static, environments can be achieved by computing global paths from the initial configurations to the goal configurations and a set of intermediate collision-free configurations for a team of robots. Kushleyev et al. [22] coined the problem as a mixed-integer quadratic optimization and Saha et al. [23] relied on discretized linear temporal logic. Both of these methods provide guarantees in terms of furnishing a collision-free trajectory, however, they scale poorly in terms of the number of robots and only rely on squared formations of the robot fleet.

2.1.4 Motion Planning and Control for Object Transportation

Object transportation is an application where multi-robot motion planning becomes essential. The aim of motion planning here is to compute trajectories for each robot carrying the object such that the following conditions are met [24]: (i) the relative motions between the robots is appropriate to allow each robot to hold onto the object. (ii) the trajectories are collision-free for the robots and for the transported object. The "relative motion" requirement can be better understood with the example shown in Figure 1.1. In the scenario shown in Figure 1.1, the trajectories for each of the robots should be such that at every time instant, the relative distance between the robots is equal to the length of the rod.

A large number of existing works like [25], [26] focus primarily on precise force-feedback control between the robots neglecting the collision avoidance requirement. The collision avoidance in other works is often handled in a reactive manner with techniques like potential field [27] or velocity obstacle [28], [29], [30]. As discussed in [30], the object transportation problem is also tightly coupled with formation control as the robots need to constantly adapt their relative positions with respect to each other for avoiding collisions (e.g, see Fig. 1.1).

A key novelty of this thesis is going beyond the reactive one step collision avoidance approach prevalent in existing works by formulating a global trajectory optimization problem. As a result, various relative re-positionings are obtained that the robots need to perform throughout the length of the trajectory.

2.2 Robot Operating System

Robot Operating System (ROS) is an open source, Linux-based framework to operate and program various kinds of robots. It leverages P2P communication, in which executable programs, called nodes, communicate with each other at runtime [3]. The nodes are connected to a ROS master, which is in charge of coordinating the communication among them and through which they can exchange information. The nodes communicate through publishing or subscribing to topics containing messages. Therefore, in case a node needs some data, it subscribes to the relevant topic and, likewise, if it generates some data, it publishes them in form of messages on a topic. ROS makes the robot system decoupled, so that different parts of robots can perform different functions without hindering the others, thus if one of the nodes crashes, it does not necessarily halt the system entirely. Another advantage of ROS is reusability of the code on various robots and avoiding the reinvention of the wheel when working with different robots [31]. For example, if someone has already developed navigation features for a robot using ROS, it could be used on a different robot running ROS.

2.2.1 ROS Navigation

ROS Navigation is a powerful tool for integrating collision-free path-planning to obtain velocity commands required for navigating robots from one point to another. Velocity commands (cmd_vel) are a stream of velocity values meant to be executed by mobile bases and are defined in terms of Twist messages in ROS, consisting separate components for linear and angular velocities in different axes. ROS Navigation provides a standard plug-in based interface for motion planners, supporting both differential drive and holonomic platforms [32]. Figure 2.3 shows the structure of ROS Navigation, which integrates localization, mapping, global and local planners. Using the map and localization components the robot is accurately localized. Navigating a robot starts with sending a goal to ROS Navigation. The goal is PoseStamped and it consists of the goal pose, timestamp, and its reference frame. After the goal is received, the global planner generates a collision-free path for the robot to follow and, subsequently, the local planner generates corresponding velocity commands which are then published to the robot's driver to move towards the goal.

2.2.2 Global and local planners

ROS Navigation uses two-level planning - global and local - for calculating the series of velocity commands from the robot's current pose to its goal state [33]. A global planner takes into account the base footprint, a projection of the robot's outline on the ground plane, and a global costmap to generate a collision-free trajectory consisting of series of wayposes to reach the goal state. However, for the sake of simplicity and efficiency, global planners consider the base footprint as a single point in the process of generating the global plan, if the base footprint is defined as a polygon geometry [34]. Since mobile platforms are commonly navigated by velocity commands, local planners in ROS Navigation are in charge of translating the wayposes provided by the global planner into corresponding velocity commands to be sent to the robot [33]. Although global planners publish the whole trajectory to be followed by the robot, local planners, too, plan for some variable steps ahead at each

instance of time and update the global trajectory. This way, local planners would be able to fix any errors due to slipping and driving precision while trying to follow the global planner's trajectory. Thus, this two-level planning increases the planning accuracy and helps avoiding any new appearing or moving obstacles. However, this also means the trajectory generated by local planners is not necessarily following the initial global plan.

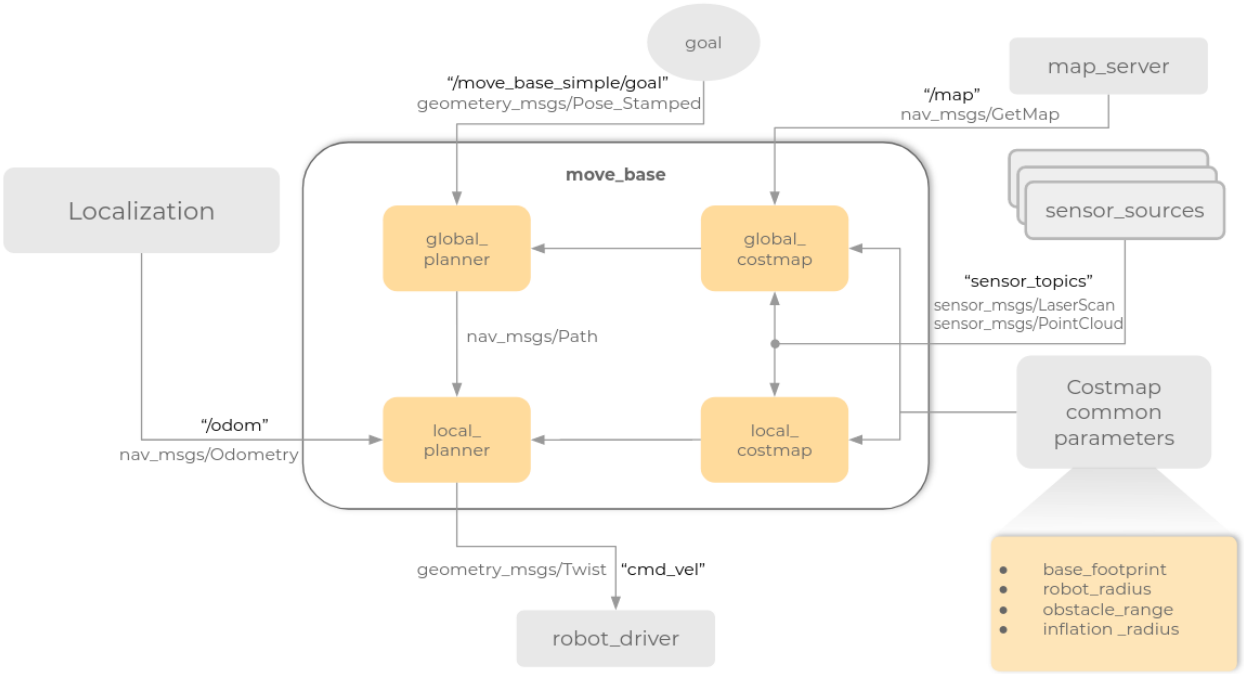


Figure 2.3: *ROS Navigation*

3 Requirements

3.1 Objective

In this work, a coupled system of robots would carry a shared payload from a start configuration to an end configuration. An optimization-based global planner is developed that finds collision-free motions for the robot fleet using gradient descent. In addition, a local planner is developed that can be used together with the developed or any of the ROS Navigation's existing global planners to compute individual velocity commands for every robot carrying the shared payload.

3.2 Functional Requirements

- Generation of a collision-free global plan for a robot fleet carrying a shared payload.
- Generating velocity commands for individual ROS-Navigation-enabled mobile robots carrying a shared payload.
- Being scalable in terms of the shape of the shared payload as well as the number of robots carrying it.

3.3 Software Requirements

- Ubuntu Linux 16.04 or higher for running ROS Kinetic or later distributions.
- Python for leveraging its libraries, including CVXOPT, Scipy, Sympy, Numpy, and Quadprog, which are involved in developing the global planning algorithm.
- C++ language for developing a local planner as a plugin to ROS Navigation.

3.4 Hardware Requirements

- ROS-enabled mobile platforms.

4 Design

In this chapter, first, the proposed global motion planning algorithm for object transportation is presented and subsequently, a ROS local planner plugin is introduced that can take the output of the proposed motion planner or any of the ROS Navigation's existing global planners and generate velocities for individual robots in a robot fleet carrying a shared payload.

4.1 Global path planning algorithm for multiple robots

In this work, multiple robots are utilized for the purpose of transporting a rigid object and for that, a global motion planning algorithm is designed considering a robot fleet as a coupled system.

4.1.1 Problem Formulation

The global motion planner developed in this work is specific to the case of a pair of robots carrying a rod-like (or more generally a rectangular) object. A demonstration of the planner is presented in Fig 1.1. In this example, each end of the rod is placed on a robot using a revolute joint. Thus, as shown in Figure 1.1, the position and orientation of the rod depends on the relative position of the robots, i.e the position that one robot has with respect to another. The crux of the motion planning problem in this scenario is to compute the position that each robot should have at each time instance in order to ensure that neither the robots nor the rod collide with the obstacles.

The above problem description can be mathematically formulated as the following optimization problem:

$$\min_{{}^1x_t, {}^1y_t, {}^2x_t, {}^2y_t} \sum_i f({}^ix_t) + f({}^iy_t), \forall i = 1, 2 \tag{4.1}$$

$$({}^2x_t - {}^1x_t)^2 + ({}^2y_t - {}^1y_t)^2 = l^2, \forall t \tag{4.2}$$

$${}^1x_t, {}^1y_t, {}^2x_t, {}^2y_t \in C_{free} \tag{4.3}$$

$$f(\xi) = \sum_{t=0}^{t=n} (\xi_{t+1} - \xi_t)^2 + (\xi_{t+2} - 2\xi_{t+1} + \xi_t)^2, \forall \xi = {}^ix_t, {}^iy_t \tag{4.4}$$

$({}^1x_t, {}^1y_t)$ and $({}^2x_t, {}^2y_t)$ are the positions of the pair of robots at time t. The solution of the optimization problem is the time stamped sequence of waypoints for the pair of robots in the

time interval $[0, n]$, in other words, the complete trajectory for the robots pair. The cost (4.1) is designed to ensure smoothness in the resulting trajectory. As shown in (4.4), the function $f(\xi)$, models the smoothness by decomposing it into two terms [12], [13]. The first term in (4.4) is a first order finite difference of the time stamped positions and acts as a surrogate for minimizing the magnitude of the velocity at each time instant. Similarly, the second term is a second-order finite difference of the time stamped position values and acts as a surrogate for minimizing the magnitude of acceleration values at each time instant. The smaller the summation is, the smoother is the trajectory. Such model for smoothness in terms of velocity and acceleration has been extensively used in existing literature [12], [13].

The equality constraint (4.2) enforces the condition that the distance between the robots should be equal to the length of the rod. For a more general case of transporting a rectangular object, the term length can be used to denote the euclidean distance between the points where the object is connected to the robots. The constraint (4.3) enforces the condition that the position of the robots always lie on the collision free space C_{free}. The equations for implicitly modeling C_{free} would be derived later on this chapter.

4.1.2 Interpretation as motion planning of an unconstrained rod-like object

One of the key challenges of optimization (4.1)-(4.3) stems from the equality constraint (4.2). This is because (4.2) defines a manifold (or a surface) on which $({}^1x_t, {}^1y_t)$ and $({}^2x_t, {}^2y_t)$ must lie at all times. To see how, note that for a given $({}^1x_t, {}^1y_t)$, the set of all feasible $({}^2x_t, {}^2y_t)$ which satisfy (4.2) constitute a sphere centered at $({}^1x_t, {}^1y_t)$ with radius l. Constraints of the form (4.2) are called manifold constraints and are commonly encountered in task-planning of manipulators [35]. Typically, these type of constraints are difficult to incorporate within an optimization problem. A common work-around in existing literature has been to approximately satisfy constraints of the form (4.2) by reformulating it as a cost [36]. However, such an approach is not suitable for our application as it would mean that the distance between $({}^1x_t, {}^1y_t)$ and $({}^2x_t, {}^2y_t)$ would be close to but not exactly equal to l.

To counter this intractability, in this work, it is proposed to reinterpret the problem of motion planning for a pair of robots carrying a rod as the problem of motion planning for a single robot, where the single robot is the rod itself. The rod-shaped robot would have 3 DoFs pertaining to its position and orientation. Since, the rod is attached through a revolute joint on top of the robots, each robot's position or even velocity can be extracted from the position and orientation of the rod (see Figure 4.1). Specifically, let x_t, y_t, ϕ_t be the position and orientation of the rod at time t. Then the positions of robots are given as follows (Figure 4.1):

$$ {}^1x_t = x_t + \frac{l}{2}\cos\phi_t, {}^1y_t = y_t + \frac{l}{2}\sin\phi_t \tag{4.5} $$

$$ {}^2x_t = x_t - \frac{l}{2}\cos\phi_t, {}^2y_t = y_t - \frac{l}{2}\sin\phi_t \tag{4.6} $$

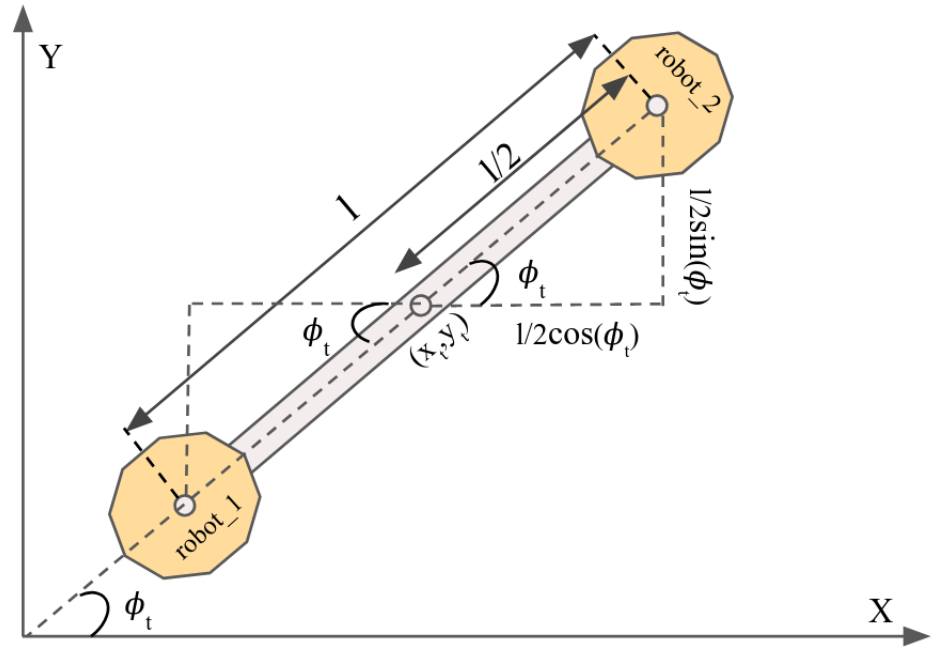


Figure 4.1: *Relative positions of the robots from the center point of the rectangular object*

Differentiating (4.5)-(4.6), the velocities of the individual robots can be expressed according to the velocity of the rod.

$$^{1}\dot{x}_t = \dot{x}_t - \frac{l}{2}\sin\phi_t\dot{\phi}_t, {}^{1}\dot{y}_t = \dot{y}_t + \frac{l}{2}\cos\phi_t\dot{\phi}_t \tag{4.7}$$

$$^{2}\dot{x}_t = \dot{x}_t + \frac{l}{2}\sin\phi_t\dot{\phi}_t, {}^{2}\dot{y}_t = \dot{y}_t - \frac{l}{2}\cos\phi_t\dot{\phi}_t \tag{4.8}$$

The motion planning problem for the purpose of this work can be formally defined as motion planning for a rod-like object.

Problem Definition: Given start and end positions and orientations of a rod-like object, compute a smooth, collision-free trajectory for the rod.

The above problem description can be converted to the following trajectory optimization problem:

$$\min_{x_t,y_t,\phi_t} f(x_t) + f(y_t) + f(\phi_t) \tag{4.9}$$

$$x_t, y_t, \phi_t \in \mathcal{C}_{free} \tag{4.10}$$

The solution to the above optimization problem is the time stamped sequence of x_t, y_t, ϕ_t in time interval $[0, n]$. It should be noted that the initial and final values $(t = 0, n)$ are known and provided as an input to the optimization problem. The cost shown in (4.9) has the same

definition as that given in (4.1) but now acts over the orientation angle ϕ_t as well. Like before, it ensures smoothness in time stamped sequence of position and orientation values. The constraint (4.10) enforces the rod position and orientation to satisfy the collision avoidance condition.

Remark: The advantage of (4.9)-(4.10) over (4.1)-(4.3) is that the former does not have any equality (or manifold) constraints.

4.1.3 Modeling $\mathcal{C}_{free}$

Checking collisions between non-convex or irregular shaped objects (such as our rod shaped robot) are considered to be a difficult problem [37]. Thus, it is common in robotics literature to approximate an irregular shaped robots and obstacles through primitives like circles [12], [15]. In this section, the same approach is followed to derive the equations which implicitly define the collision-free space $\mathcal{C}_{free}$.

As shown in Figure 4.2, the rod is covered with multiple circles. The number of circles and their radii are chosen in way that they cover the whole length of the rod as well as the physical space that would be occupied by the robots. A simple trial and error approach is adopted for this. Using the circle approximation, the collision avoidance condition between the rod (and the robots) with a circular obstacle is given by the following inequality:

$$c(x_t, y_t, x^{o_j}, y^{o_j}) = -(x_t + r_i\cos(\phi_i) - x^{o_j})^2 - (y_t + r_i\sin(\phi_t) - y^{o_j})^2 + (d_i + d^{o_j})^2 \leq 0, \forall t, i, j \tag{4.11}$$

where d_i is the radius of the i^{th} circle of the rod and d^{oj} is the radius of the j^{th} obstacle. The variable r_i locates the center of the i^{th} circle from the center of the rod (see Figure 4.2). Inequality (4.11) implicitly defines the set of collision free positions and orientations of the rod, in other words, $\mathcal{C}_{free}$.

4.1.4 Reformulation as an Un-constrained Optimization Problem

Using (4.11), the optimization problem (4.9)-(4.10) can be further simplified in the following manner:

$$\min \mathcal{L}(x_t, y_t, \phi_t) = w_1(f(x_t) + f(y_t) + f(\phi_t)) + w_2 \sum_{t=0}^{t=n} \max(0, c(x_t, y_t, x^{o_j}, y^{o_j})) \tag{4.12}$$

Optimization problem (4.12) does not have any explicit constraints as the collision avoidance, which is now accommodated into the cost function through the max penalty. The key concept employed here is that any constraint $h \leq 0$ can be interpreted as a cost in the form $\max(0, h)$ [14]. When $h \leq 0$, the output of the max term is zero, i.e no additional cost is incurred beyond the primary smoothness cost (first three terms in (4.12)). On the other hand, if $h > 0$, then output of the max penalty is the value of h itself and it adds on to the primary smoothness cost.

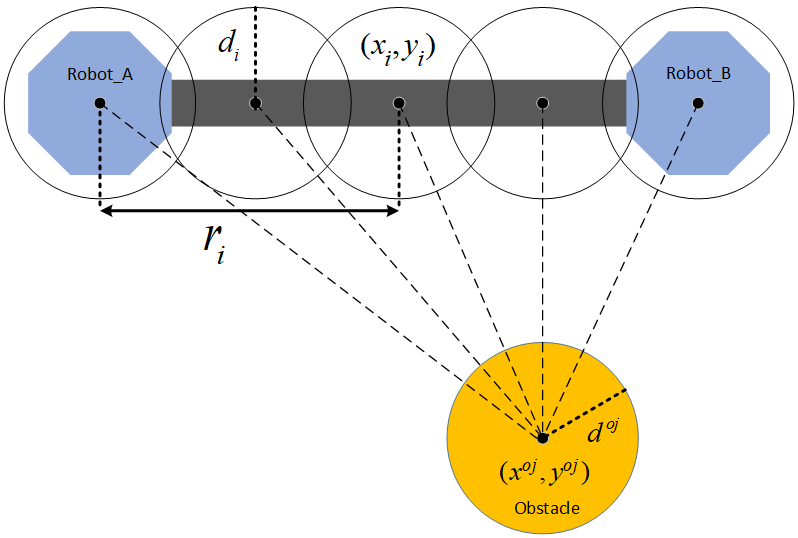


Figure 4.2: *Collision Avoidance Model*

4.1.5 Gradient Based Approaches

Cost function (4.12) has the same form as the frequently encountered loss functions in deep learning [38]. Thus, state of the art variants of gradient descent like ADAM and Momentum common in deep learning are chosen to solve the optimization problem (4.12). Each of the optimizers resolves the problem of choosing an appropriate learning rate (or step size) in gradient descent. The main steps of these optimizer are summarized in the following, wherein, the variable $\mathbf{x}$ is formed by stacking the various x_t at different instants. Similar process follows for $\mathbf{y}$ and $\boldsymbol{\phi}$ as well.

4.1.6 Momentum

Gradient descent with momentum considers the past gradients to smooth out the update. It computes an exponentially weighted average of the gradients, and then uses that gradient to update the solution vector. It converges faster than the standard gradient descent algorithm [39].

The main steps of gradient descent with momentum are brought in (4.13) and (4.14) [40]. The the superscript k refers to the iteration number [40].

$$\mathbf{v}^k = \gamma \mathbf{v}^{k-1} + \eta \nabla_{x_t, y_t, \phi_t} \mathcal{L}(x_t, y_t, \phi_t), \tag{4.13}$$

$$(\mathbf{x}, \mathbf{y}, \boldsymbol{\phi})^k = (\mathbf{x}, \mathbf{y}, \boldsymbol{\phi})^{k-1} - \mathbf{v}^k, \tag{4.14}$$

The vector $\mathbf{v}^k$ computes the average of the past gradients and the current gradient. The term γ is often set to 0.9. The term η is the learning rate in gradient descent.

4.1.7 Adaptive Moment Estimation

Adaptive Moment Estimation (ADAM) is a method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients v_k, it too keeps an exponentially decaying average of past gradients m_k, similar to momentum [40]. The main steps of ADAM optimizer are summarized in (4.15)-(4.19) based on [40].

$$\mathbf{m}^k = \beta_1 \mathbf{m}^{k-1} + (1 - \beta_1)\nabla\mathcal{L} \tag{4.15}$$

$$\mathbf{v}^{=}\beta_2 \mathbf{v}^{k-1} + (1 - \beta_2)\nabla\mathcal{L}^2 \tag{4.16}$$

$$\widetilde{\mathbf{m}}^k = \frac{\mathbf{m}^k}{1 - \beta_1^k} \tag{4.17}$$

$$\widetilde{\mathbf{v}}^k = \frac{\mathbf{v}^k}{1 - \beta_2^k} \tag{4.18}$$

$$(\mathbf{x}, \mathbf{y}, \boldsymbol{\phi})^k = (\mathbf{x}, \mathbf{y}, \boldsymbol{\phi})^{k-1} - \eta \frac{\widetilde{\mathbf{m}}^k}{\sqrt{\widetilde{\mathbf{v}}^k + \epsilon}} \tag{4.19}$$

4.2 ROS Compatible Local Planner for Multiple Robots

ROS Navigation uses local and global planners for motion planning. Global planners generate the whole nominal trajectory to be followed by the robot, while local planners are mainly responsible for making the robot follow the path generated by the global planners through generating corresponding velocity commands for the robot.

Since ROS Navigation is meant for single-robot applications, to the best of author's knowledge, there exists no local planner which can be used for the purpose of this work being multi-robot motion planning. In order to achieve it, a local planner is developed to take in a global motion plan for a group of robots and output velocities for each individual robot. This implementation extends the original ROS Navigation (Figure 2.3) to what is depicted in Figure 4.3, where the local planner factors in a multi-robot configuration to generate individual velocities for every robot carrying the shared payload.

In this setup, a variable number of robots work together to carry an object while avoiding obstacles. As depicted in Figure 4.4, after the global plan is received, the new local planner segregates it to the number of trajectories equal to the number of robots defined in the *robots_ configuration* YAML file (Figure 4.5(b)), after which it generates the sequence of velocity commands published in a robot-specific namespace. The number of robots to carry an object is variable and can be determined by the user in the *robots_ configuration* YAML. A user might choose to carry an object using either one or n robots.

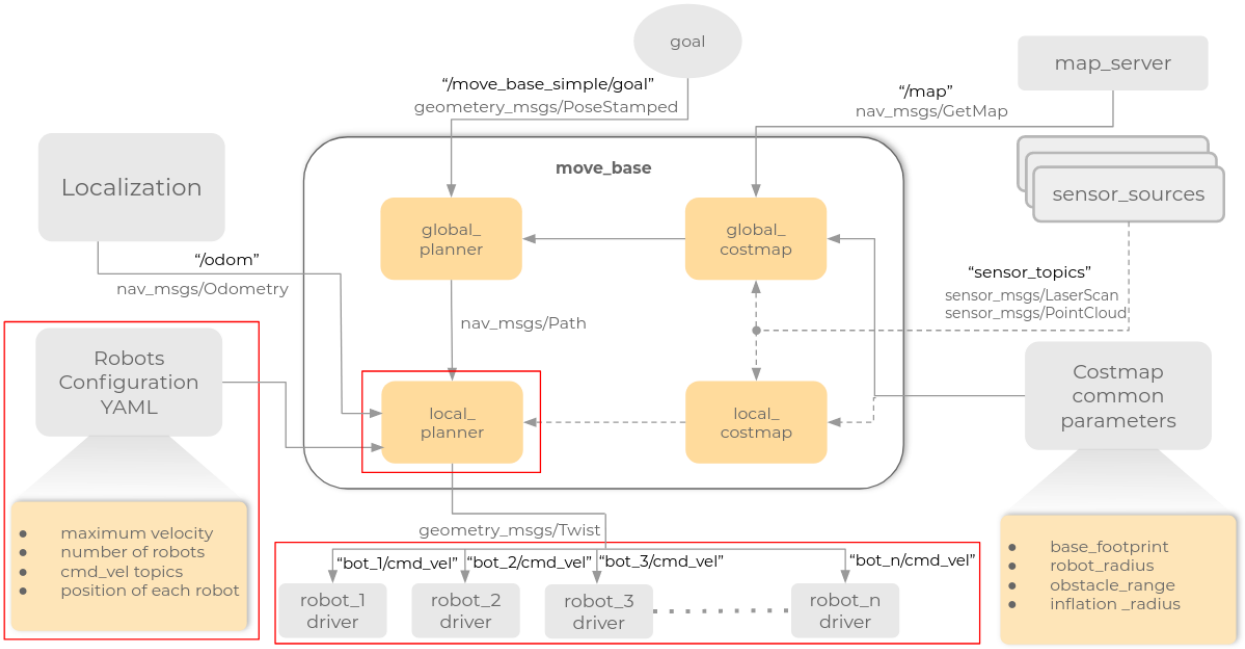


Figure 4.3: *Navigation with the proposed local planner.*
The red bounding boxes indicate the new features added to ROS Navigation using the proposed local planner. The dashed lines in the figure indicate that those features should be there for the implementation to fully comply with ROS Navigation, but they are yet to be there in the current implementation.

4.2.1 Implementation

In this section, the implementation of the local planner is presented by providing an insight to how the local planner gets a global trajectory and moves a robot fleet.

As shown in Figure 4.4, the input of the local planner is a global trajectory in the form of *nav_msg/Path*. The global plan contains the trajectory for origin (zero point) of the base footprint. Base footprints are user-definable and generated through a sequence of consecutive points in the *costmap_common_parameters* YAML as shown in Figure 4.5(a). In this work, the base footprint is defined so that it accommodates both the geometry of the to-be-transported object as well as the robots (an example is brought in section 4.3).

After the global trajectory is received, the trajectory segregator (Num. 1 in Figure 4.4) divides the trajectory to individual trajectories corresponding to the number of robots and their positions in the base footprint based on parameters in *robots_configuration* YAML as shown in Figure 4.5(b) (Algorithm 1 line 2). In order to segregate the global trajectory to different trajectories for each robot, the wayposes in the global trajectory would be transferred to different poses for each robot using Equations 4.20 - 4.22 as shown in Figure 4.6.

First, the distance of each robot from the origin of the base footprint, ${}^{i}e$, is calculated using Equation 4.20.

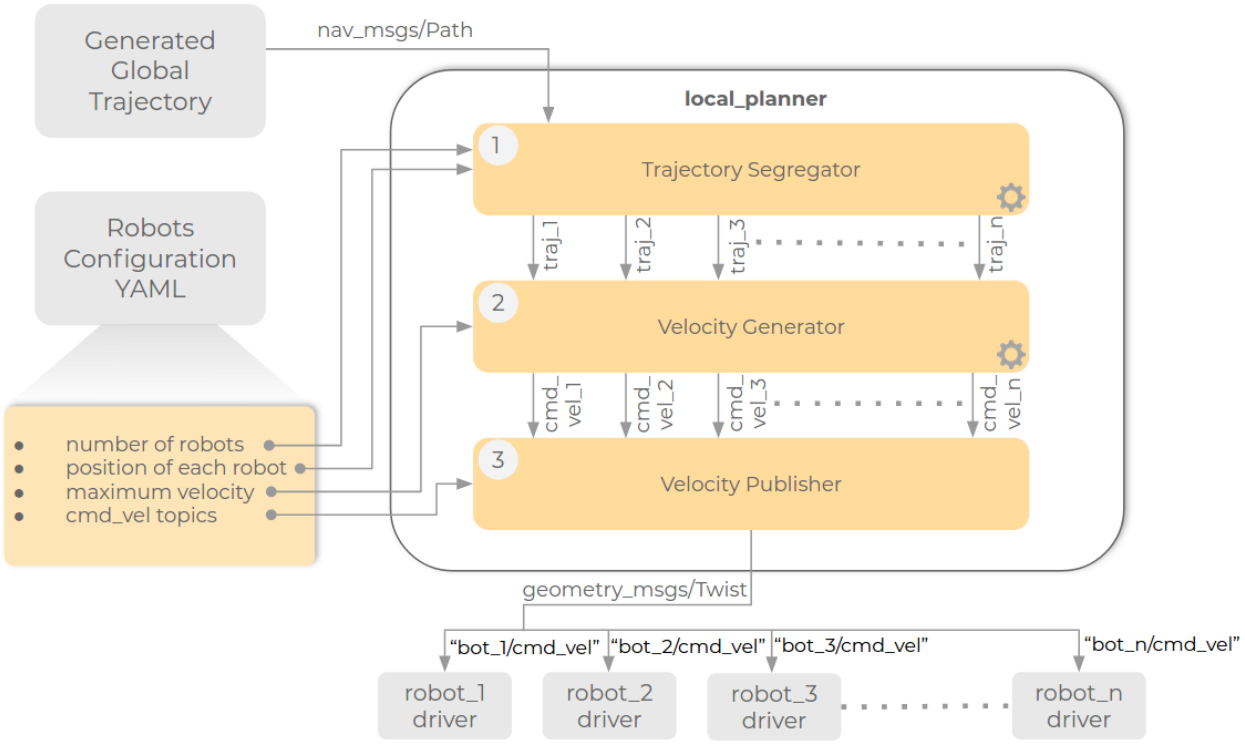


Figure 4.4: Flowchart of the local planner.

$$^{i}e = \sqrt{(^{i}x - x_0)^2 + (^{i}y - y_0)^2} \tag{4.20}$$

The ^{i}x, ^{i}y are the i^{th} robot's position coordinates with respect to the base footprint's origin, x_0 and y_0. These values are fixed and defined in *robots_ configuration* YAML (Figure 4.5(b)).

Then, new waypoints are generated for each robot as follows:

$$^{i}x_t = x_t + {}^{i}e\cos(\phi_t + {}^{i}\phi) \tag{4.21}$$

$$^{i}y_t = y_t + {}^{i}e\sin(\phi_t + {}^{i}\phi) \tag{4.22}$$

The $^{i}x_t$ and $^{i}y_t$ denote the waypoint coordinates for i^{th} robot at time t, x_t and y_t are the position coordinates for the origin of base footprint at time t in the global trajectory, $^{i}\phi$ is the angle of each robot with respect to base footprint's origin, and ϕ_t is the orientation of the base footprint's origin with respect to the z axis at time t.

In the next stage, the trajectory segregator (Num. 1 in Figure 4.4) transforms the newly-generated trajectories from the map frame to the each robot's local frame using rotation matrix (4.23) through Equations 4.24 and 4.25 (Line 3 in Algorithm 1).

```
footprint: [[0.24,0.45],[0.24,0.61], [0.56,
0.61],[0.56, 0.24],[0.4, 0.24],[0.4,
-0.24],[0.56, -0.24],[0.56, -0.61],[0.24,
-0.61],[0.24, -0.45], [-0.24, -0.45],[-0.24,
-0.61],[-0.56, -0.61],[-0.56, -0.24],[-0.4,
-0.24],[-0.4, 0.24],[-0.56, 0.24],[-0.56,
0.61],[-0.24, 0.61],[-0.24, 0.45]]
obstacle_range: 2.0
raytrace_range: 2.5
inflation_radius: 1.0
cost_scaling_factor: 3.0
map_type: costmap
```

(a) common_costmap_parameters

```
Global:
- center_position: [-1.5, -1.0, 0.0]
- max_velocity: 1.0
- rotation: true

Robots:
- name: "bot_1"
  position: [-1.08, -0.55, 0.0]
  orientation: [0.0, 0.0, 0.75]
  cmd_vel_topic: "bot_1/cmd_vel"

- name: "bot_2"
  position: [-1.08, -1.45, 0.0]
  orientation: [0.0, 0.0, 1.0]
  cmd_vel_topic: "bot_2/cmd_vel"

- name: "bot_3"
  position: [-1.90, -1.45, 0.0]
  orientation: [0.0, 0.0, 1.57]
  cmd_vel_topic: "bot_3/cmd_vel"

- name: "bot_4"
  position: [-1.90, -0.55, 0.0]
  orientation: [0.0, 0.0, 1.57]
  cmd_vel_topic: "bot_4/cmd_vel"
```

(b) robots_configuration

Figure 4.5: *A sample of robots_configuration and common_costmap_parameters YAML files*

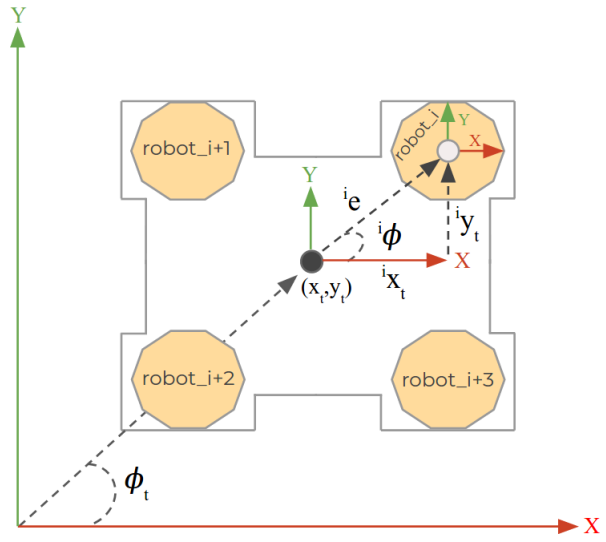


Figure 4.6: *Segregation of the global trajectory to different trajectories for each robot by generating new trajectories according to the waypoints for the base footprint's origin and each robot's pose relative to base footprint's origin pose.*

Algorithm 1: How the local planner works

Input:

- A set of waypoints for the (0,0) point of the base footprint.
- Number of the robots and their pose with respect to base footprint (0,0).
- Maximum linear and angular velocity for the robots.

Output: Velocities for each robot in the robot fleet.

```
1 begin
2 |  new_trajectories ← TrajSegregation(global_traj, num_robots, robots_pose)
3 |  mapped_trajectories ← TrajMap(new_trajectories)
4 |  generated_velocities ← VelocityGenerator(mapped_trajectories, max_velocity)
5 |  VelocityPulisher(generated_velocities, robots_cmd_vel_topics)
6 end
```

$$R(-\phi) = \begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{bmatrix} \tag{4.23}$$

$$^{i}x'_t = {}^{i}x_t \cos(\phi_t + {}^{i}\phi) + {}^{i}y_t \sin(\phi_t + {}^{i}\phi) \tag{4.24}$$

$$^{i}y'_t = {}^{i}y_t \cos(\phi_t + {}^{i}\phi) - {}^{i}x_t \sin(\phi_t + {}^{i}\phi) \tag{4.25}$$

The $^{i}x'_t$ and $^{i}y'_t$ are the position coordinates of each waypoint in the trajectories transformed into each robot's local frame.

The trajectories in robots' local frames of reference are input to the velocity generator (Num. 2 in Figure 4.4). The velocity generator calculates the time which takes for moving the i^{th} robot between two consecutive waypoints, $^{i}dt_n$, using *max_ velocity* parameter already loaded to parameter server from the *robots_ configuration* YAML through Equation 4.26 (Line 4 in Algorithm 1).

$$^{i}dt_n = \frac{\sqrt{(^{i}x'_{t+1} - {}^{i}x'_t)^2 + (^{i}y'_{t+1} - {}^{i}y'_t)^2}}{max\_velocity}, \tag{4.26}$$

wherein subscript n indicates n^{th} two consecutive waypoints in i^{th} robot trajectory.

The velocity generator then calculates the velocities for robots for each two consecutive waypoints through Equations 4.27 - 4.29.

$$^{i}v_{x_t} = \frac{^{i}x'_{t+1} - {}^{i}x'_t}{^{i}dt_n}, \tag{4.27}$$

$$^{i}v_{y_t} = \frac{y'_{t+1} - y'_t}{^{i}dt_n}, \tag{4.28}$$

$$^{i}v_{\phi_t} = \frac{\phi_{t+1} - \phi_t}{^{i}dt_n}, \tag{4.29}$$

wherein $^{i}v_{x_t}$, $^{i}v_{y_t}$, and $^{i}v_{\phi_t}$ are the linear and angular velocities between each two consecutive waypoints for each robot, respectively.

In the last stage, the velocities generated in the previous stage are sent to the velocity publisher (Num. 3 in Figure 4.4), where it gets the cmd_vel topic for each robot from the parameter server and publishes the velocities as Twist messages to the corresponding robots (Line 5 in Algorithm 1).

4.3 Setting up ROS Navigation for multi-robot use

This section will walk you through how to setup ROS Navigation for multi-robot use by bringing an example of the scenario described in the global planning section, i.e, a robot fleet carrying a rod.

There are three steps to get this setup running:

1. Defining the base footprint in *common_ costmap_ parameters* YAML.
2. Setting up the right paremeters in *robots_ configuration* YAML according to the geomtery of the shared payload as well as the number of the robots carrying it.
3. Setting ROS Navigation local planner to the developed local planner, *MRPFPlannerROS* (short for multi-robot pose follower), and sending goals.

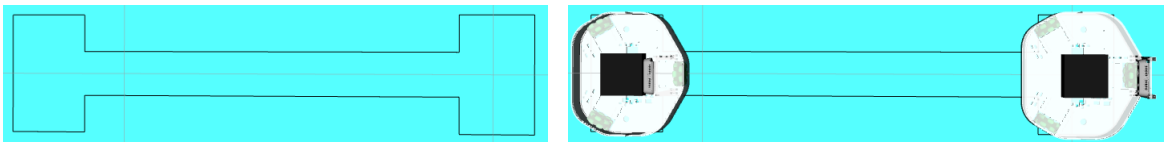
Step 1:

For a multi-robot system to carry a rod-like shared payload while avoiding obstacles using the design introduced in this work, the base footprint should be defined so that it encompasses the geometry of the shared payload as well as the geometry of the robots. For example, if the rod itself is 1.0 m in length and 0.2 m in width, and the robots carrying it are 0.3 m in length and 0.2 m in width, the base footprint could be defined as follows:

footprint: *[[-0.7, 0.15], [-0.5, 0.15],[-0.5, 0.05],[0.5, 0.05] ,[0.5, 0.15],[0.7, 0.15],[0.7,-0.15], [0.5, -0.15],[0.5, -0.05],[-0.5, -0.05],[-0.5, -0.15],[-0.7, -0.15]]*

This definition for the base footprint would make it look like Figure 4.7, where the two robots are at each end of the rod and the base footprint's origin is located in its middle.

After this, *common_ costmap _parameters* YAML would look like Figure 4.8(a).
All the parameters in *common_ costmap_ parameters* can be modified according to the user's preferences.



(a) base footprint (b) base footprint with robots in place

Figure 4.7: *Base footprint for a rod-like object carried by two robots*

```
footprint: [[-0.7, 0.15], [-0.5,
0.15],[-0.5, 0.05],[0.5, 0.05],[0.5,
0.15],[0.7, 0.15],[0.7,-0.15],[0.5,
-0.15],[0.5, -0.05],[-0.5,
-0.05],[-0.5, -0.15],[-0.7, -0.15]]
obstacle_range: 3.0
raytrace_range: 3.5
inflation_radius: 1.0
cost_scaling_factor: 3.0
map_type: costmap
```

(a) common_costmap_parameters

```
Global:
- center_position: [0.0, 0.0, 0.0]
- max_velocity: 1.0
- rotation: true

Robots:
- name: "bot_1"
  position: [0.0, -0.6, 0.0]
  orientation: [0.0, 0.0, 0.0]
  cmd_vel_topic: "bot_1/cmd_vel"

- name: "bot_2"
  position: [0.0, 0.6, 0.0]
  orientation: [0.0, 0.0, 0.0]
  cmd_vel_topic: "bot_2/cmd_vel"
```

(b) robots_configuration

Figure 4.8: *common_ costmap_ parameters and robots_ configuration YAML files*

Step 2:

In this step, the *robots_ configuration* YAML should be set up.
Considering the example scenario, the YAML file would include the position and orientation of the each robot in the base footprint, coordinates of the base footprint's origin, *cmd_ vel* topics as well as the maximum velocity. The parameter *rotation* determines whether the robots should keep their orientation as the base footprint's origin point orientation when moving along the trajectory.
At this point, the *robots_ configuration* YAML would look like Figure 4.8(b).

Step 3:

In this step, the local planner for ROS Navigation should be chosen as *MRPFPlannerROS*. To do that, *base_ local_ planner* parameter for *move_ base* should be set to *mrpf_ local_ planner/MRPFPlannerROS*.

After setting the local planner to *MRPFPlannerROS*, a goal point can be sent, upon which different trajectories for each robot would be generated and velocities would be published

on the corresponding topics. An example of how collision-free trajectories are generated for two robots to go from a start configuration to a goal configuration and how the robots have moved the shared payload is illustrated in Figure 5.13 in the Results section.

5 Results

5.1 Comparisons

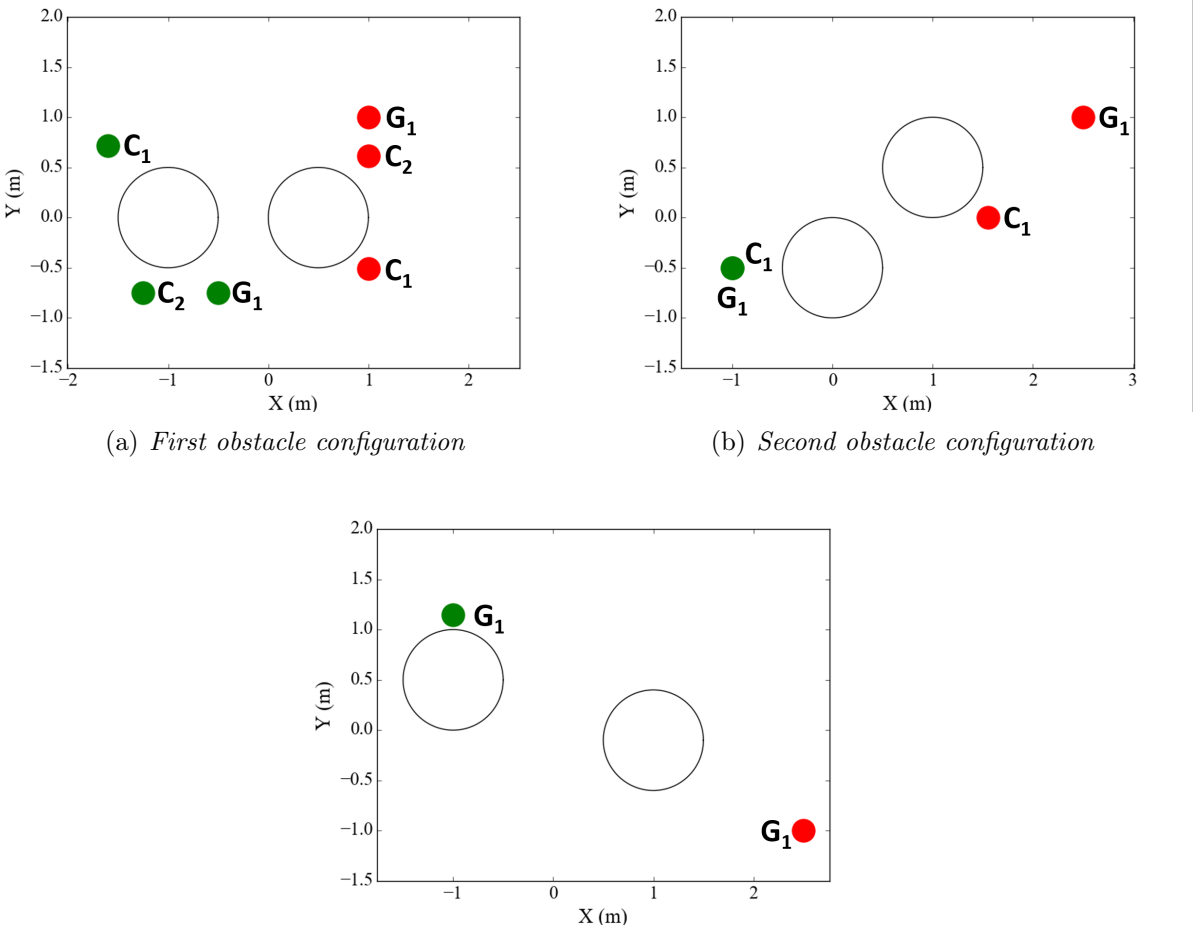
This section, first, presents the comparison of the gradient descent variants for the developed optimization-based global planner. Then, the developed global planner with the optimum gradient descent variant is compared to the "Global Planner", the only ROS Navigation planner providing orientation of waypoints in the global trajectory and suitable for multi-robot shared payload transportation. The quality of collision avoidance and the smoothness of generated trajectories as well as the execution times are compared.

5.1.1 Setup

In order to compare different gradient descent algorithms for solving the motion planning problem of a rod-like object, three obstacle configurations with various start and goal configurations are considered (Figure 5.1). Collision avoidance, average of total cost, execution time, and number of iterations to reach 75 percent of the average total cost are recorded and the optimal gradient descent method is chosen. The algorithms are run for 1600 iterations. The total cost is composed of two parts, collision-avoidance and smoothness. In the proposed optimization-based motion planner, the trajectory is collision-free and the smoothest when the cost is zero. Thus, the gradient descent method which reaches a smaller cost after certain number of iterations (1600) is considered better.

To compare the developed global planning algorithm with ROS Navigation existing global planners, a global planner that could provide the orientation of the waypoints in the global trajectory was needed. Several experiments were performed by sending different goal poses to ROS Navigation while choosing various global planners, including NavfnROS, CarrotPlanner, and GlobalPlanner. It was revealed throughout the experiments that only "GlobalPlanner" provides the orientations along with positions of waypoints in the generated trajectory. Thus, the global planning algorithm developed in this work is compared to "GlobalPlanner". The parameters for "GlobalPlanner" are set to optimal values according to a comprehensive study conducted in [41]. For the comparison, two obstacle configurations with three different goal and start configurations are considered (Figures 5.1(a) and 5.1(b)). The qualities of collision avoidance and smoothness as well as the execution times are recorded.

In order to compare the smoothness in trajectories, Equation 4.4 is used. Using this equation, the smoothness can be calculated for each degree of freedom.



(a) *First obstacle configuration*

(b) *Second obstacle configuration*

(c) *Third obstacle configuration*

Figure 5.1: *Three different obstacle configurations used in the experiments. The start and goal configurations are drawn as green and red points, respectively. G indicates the configuration used when comparing gradient descent algorithms. C indicates the configuration used when comparing ROS global planner to the developed global planning algorithm.*

All the test are carried out on a 2.60 GHz laptop with 8 GB of DDR3 RAM and Intel i7-4720HQ CPU.

5.1.2 Gradient Descent Variants

The global planning algorithm developed in this work is optimization-based and solves the motion planning problem through state-of-the-art gradient descent methods.

Presented in this section are the plots of the trajectories and their total costs generated for a rod-like object when the motion planning problem is solved through two variants of gradient descent, namely Momentum and ADAM. Three obstacle configurations with three different start and goal configurations are considered (Figure 5.1). As it can be seen in Figures 5.2 -

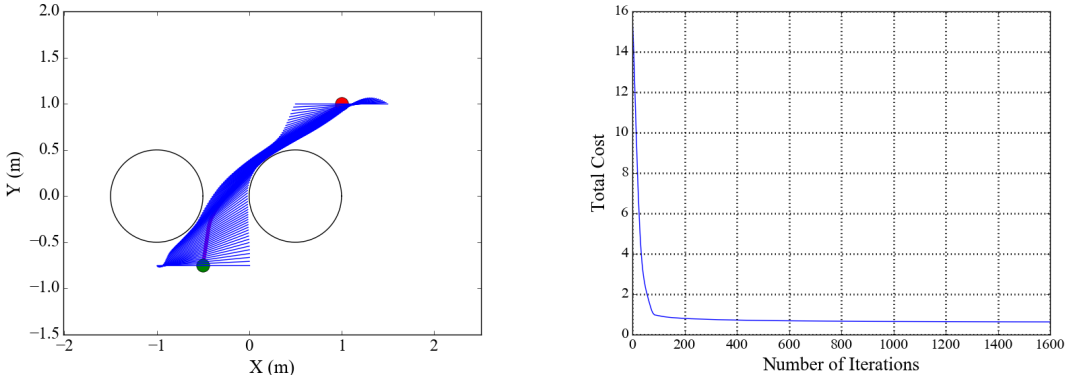
5.7, Momentum generated collision-free trajectories for all the obstacle configurations, while ADAM failed to generate a collision-free trajectory for the third obstacle configuration (Figure 5.7(a)). Red arrows in Figure 5.7(a) show where in the ADAM-generated trajectory for the rod collides with one of the obstacles.

Comparing the average total costs for Momentum and ADAM to generate trajectories for the three obstacle configurations reveals that Momentum solves the optimization problem at a lower average cost of 1.375 after 1600 iterations, shown in Figure 5.8(a). As mentioned in section 5.1.1, the total cost consists of both collision-avoidance and smoothness costs which should be minimized. Therefore, the smaller total cost is considered better.

Another measure of comparison can be the average number of iterations it takes the algorithms to reach 75 percent of the cost when generating the trajectories for the three obstacle configurations. This comparison criterion shows which algorithm converges faster towards zero in terms of the total cost. As it can be seen in Figure 5.8(b), Momentum reaches 75 percent of the cost with a smaller number of iterations by 561 compared to that of 732 for ADAM.

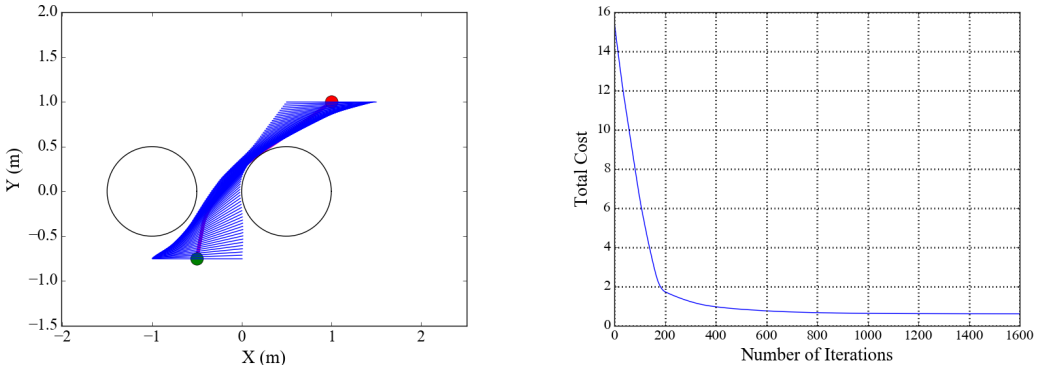
In terms of the execution time, both algorithms took roughly the same amount time when generating trajectories for the three obstacle configurations by the average of 18.13 s and 18.10 s for Momentum and ADAM, respectively (Figure 5.8(c)).

Considering the comparison measures above, it is conclusive to say that Momentum works better for the purpose of this work, being motion planning for a rod-like object. Thus, gradient descent with Momentum was chosen as the method of solving the optimization problem in the global planning algorithm developed in this work.



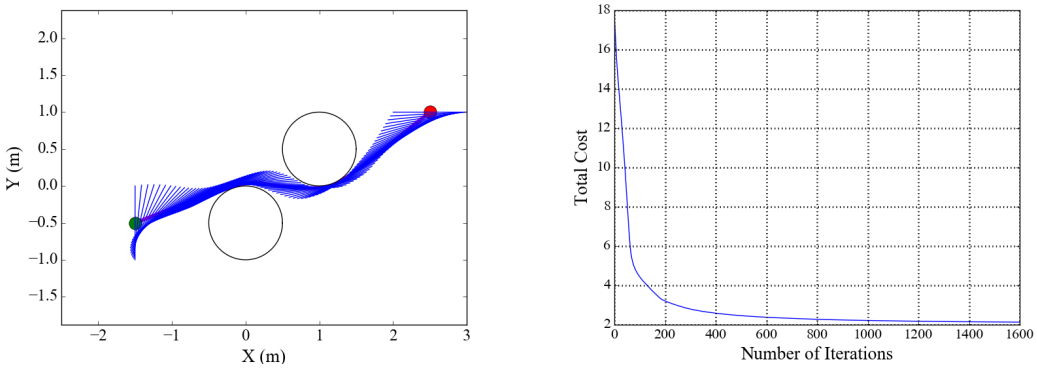
(a) *Trajectory* (b) *Total cost*

Figure 5.2: *Momentum first obstacle configuration*



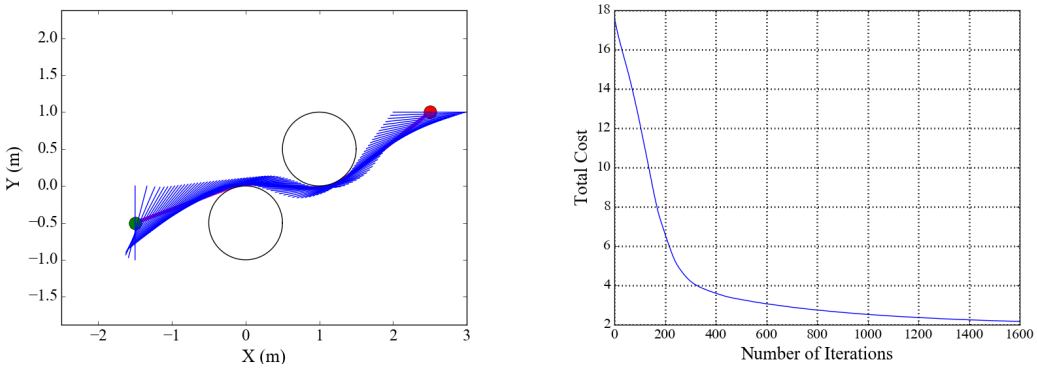
(a) *Trajectory* (b) *Total cost*

Figure 5.3: *ADAM first obstacle configuration*



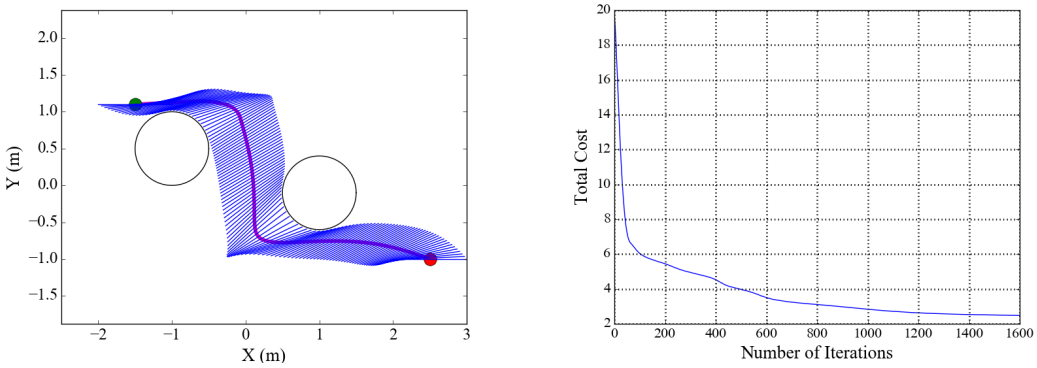
(a) *Trajectory* (b) *Total cost*

Figure 5.4: *Momentum second obstacle configuration*



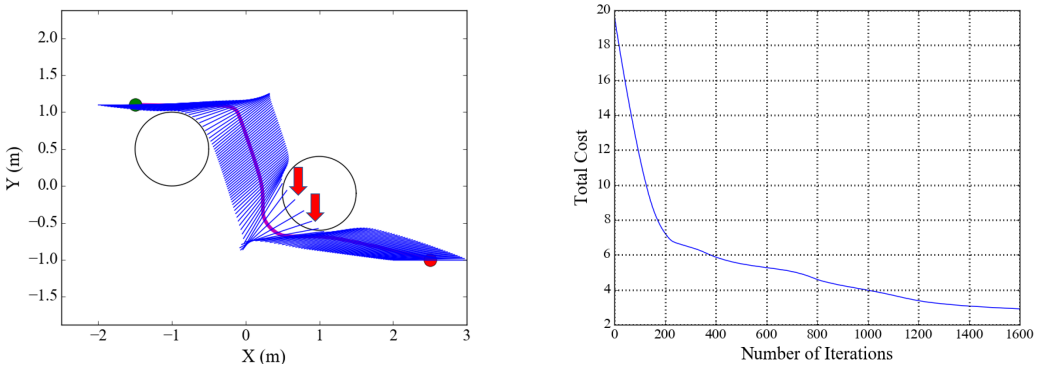
(a) *Trajectory* (b) *Total cost*

Figure 5.5: *ADAM second obstacle configuration*



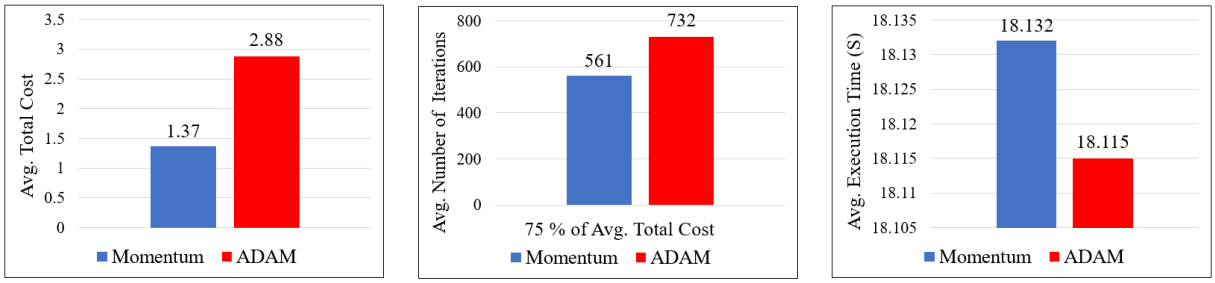
(a) *Trajectory* (b) *Total cost*

Figure 5.6: *Momentum third obstacle configuration*



(a) *Trajectory* (b) *Total cost*

Figure 5.7: *ADAM third obstacle configuration*



(a) *Avg. Total Cost* (b) *Avg. Number of Iterations* (c) *Avg. Execution Time*

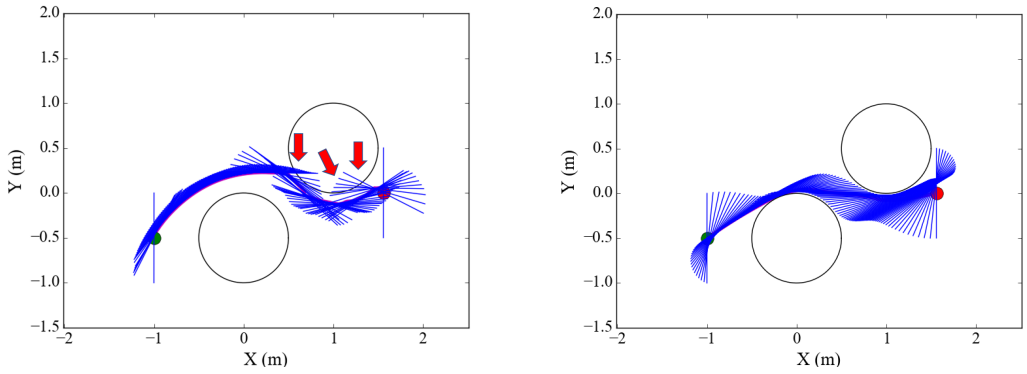
Figure 5.8: *Gradient descent comparisons*

5.1.3 ROS Planner and Developed Global Planning Algorithm

In this section, collision avoidance and smoothness qualities of ROS Navigation "GlobalPlanner" and the developed global planning algorithm as well as their execution times are

compared.

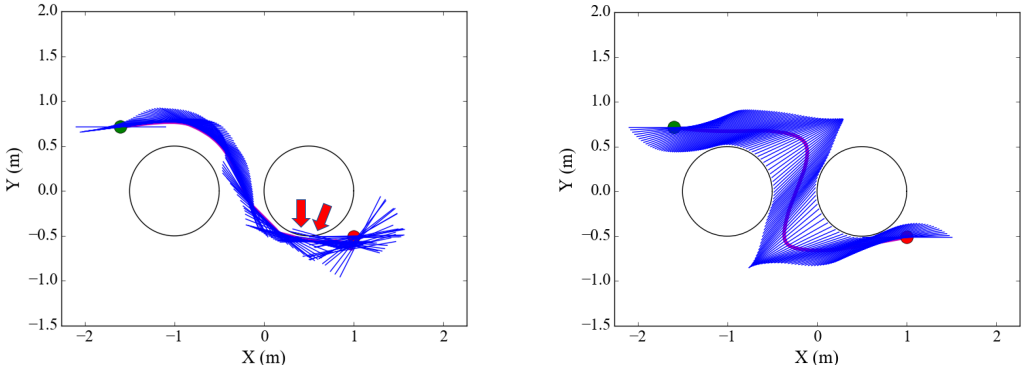
Presented in Figures 5.9(a) - 5.10(b) are the trajectories generated for a rod-like base footprint for two different obstacle configurations using ROS "GlobalPLanner" and the proposed planning algorithm. As indicated by the red arrow markers in Figures 5.9(a) and 5.10(a), the trajectories generated by ROS "GlobalPlanner" do not take into account the rod-like base footprint, hence colliding with the obstacles. Whereas, the proposed planner algorithm factors in the base footprint and generates collision-free trajectories for both of the configurations as shown in Figures 5.9(b) and 5.10(b). This is, as mentioned in section 2.2.2, because ROS Navigation global planners consider the base footprint as a single point in the process of generating the global plan, when the base footprint is defined as a polygon geometry (in this case, the rod and the robots carrying it). Furthermore, ROS Navigation global planners would generate a path only if it is collision-free [41], however, it can be seen in Figures 5.9(a) and 5.10(a) that the plans have been generated despite having collisions. This again proves that ROS global planner has considered the base footprint as a single point and if we follow the point trajectories for the middle of the rod (i.e. the red curve) in Figures 5.9(a) and 5.10(a), it is clear that they are collision-free. That being mentioned, throughout the experiments, ROS "GlobalPlanner" could fortuitously generate a collision-free trajectory for the rod-shaped base footprint for a specific start and goal configurations in the second obstacle configuration as shown in Figure 5.11(a). The same configurations were used for the proposed global planner and the trajectory generated is presented in Figure 5.11(b).



(a) *ROS planner* (b) *Proposed planner*

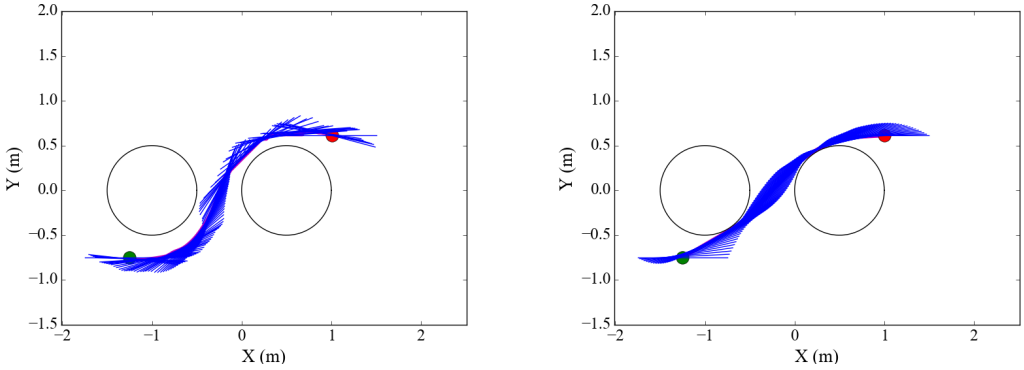
Figure 5.9: *Trajectories generated for the second obstacle configuration using ROS "GlobalPlanner" and the proposed planner*

As mentioned in section 5.1.1, smoothness is calculated using Equation 4.4. The smaller is the smoothness value, the smoother is the trajectory. Figure 5.12 shows the smoothness in position (x and y) and orientation (ϕ) for scenarios in Figures 5.9 - 5.11. As it can been seen, the proposed planner generates smoother trajectories for all the configurations, as it has smaller smoothness values in all of DoFs. Although the proposed planner has smaller smoothness value in all DoFs, the difference is significantly higher in ϕ. This is again due to



(a) *ROS planner* (b) *Proposed planner*

Figure 5.10: *Trajectories generated for the first obstacle configurations using ROS "GlobalPlanner" and the proposed planner*

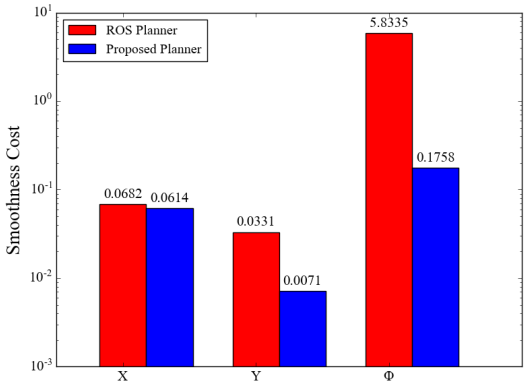


(a) *ROS planner* (b) *Proposed planner*

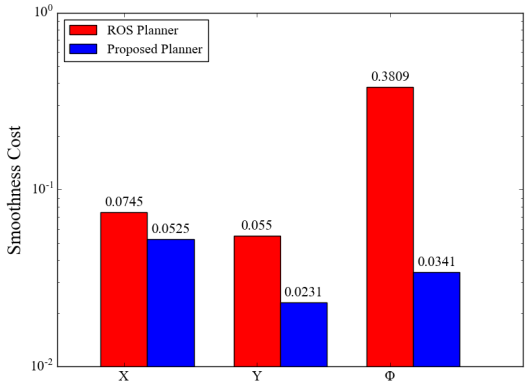
Figure 5.11: *Trajectories generated for the first obstacle configurations (different goal pose) using ROS "GlobalPlanner" and the proposed planner*

the fact that ROS planner takes into account the polygon-shaped base footprint as a point and generates trajectories regardless of the base footprint's orientation, hence the orientation arbitrarily varies throughout the trajectories.

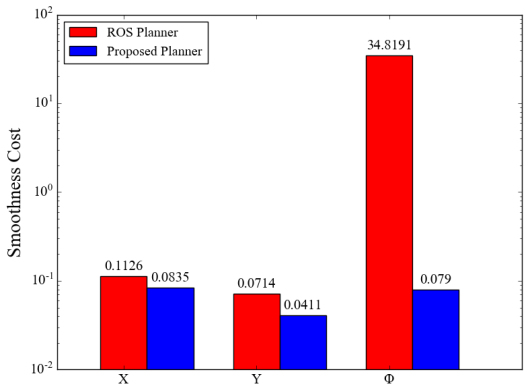
The trajectories for each of the configurations were generated in roughly 18 s using the proposed global planner, while the same trajectories were generated almost instantly (as soon as the goal was sent) using ROS "GlobalPlanner". The execution time for ROS Global planner is reported knowing that it does not take into account the rod-shaped base footprint. Therefore, it is not feasible to mention how long it would have taken ROS GlobalPlanner to generate a collision-free trajectory if it had factored in the rod-like base footprint.



(a) *First obstacle configuration*



(b) *Second obstacle configuration*



(c) *Second obstacle configuration (different goal pose)*

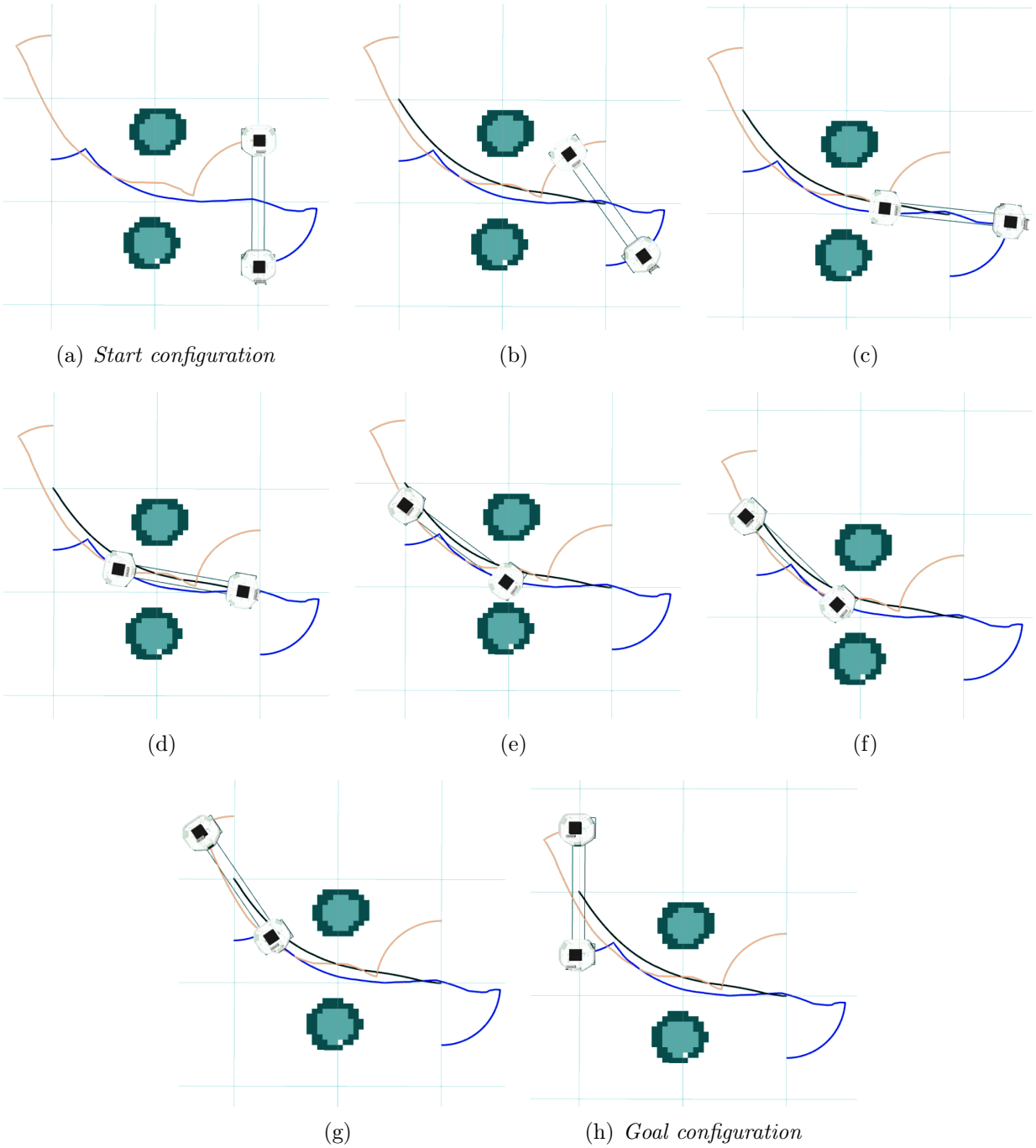
Figure 5.12: *Smoothness comparison for three different start and goal poses in two various obstacle configurations*

5.2 Developed global planning algorithm and multi-robot local planner

In this section, it is shown that using the proposed global planning algorithm, a global collision-free trajectory is generated (black line in Figure 5.13) for a robot fleet carrying a rectangular object in an environment with two obstacles. It is depicted how the robot fleet would move along the trajectory and carry the shared payload using the developed multi-robot local planner.

In the scenario illustrated in Figure 5.13, a random start and goal configurations for the shared payload; i.e, the rod-like object, is sent to the developed global planning algorithm and a collision-free trajectory has been furnished. The global trajectory is then sent to the developed local planner, where it has been segregated into two different trajectories (blue and orange lines in Figure 5.13) for the robots carrying the shared payload. Consequently,

corresponding velocities are generated through the developed local planner for the robots to move along the trajectories.



(a) *Start configuration* (b) (c)

(d) (e) (f)

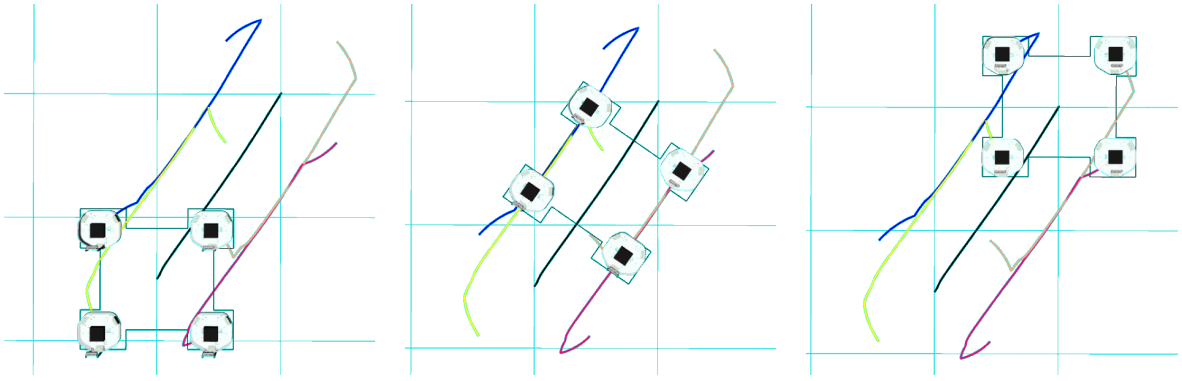
(g) (h) *Goal configuration*

Figure 5.13: *Trajectory generation for a rod-like shared payload carried by two robots. The blue, orange, and black lines are the generated trajectories for the first robot, second robot, and the origin point of the base footprint, respectively.*

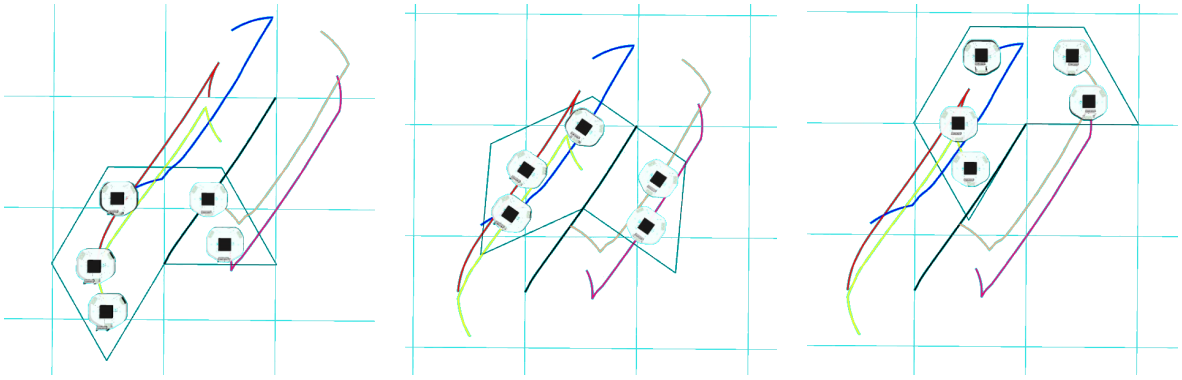
5.3 Carrying Objects of Various Geometries

The local planner developed in this work for the purpose of object transportation using multiple robots is capable of generating right trajectories and command velocities for each individual robot in a robot fleet carrying an object of any shape according to the global trajectory.

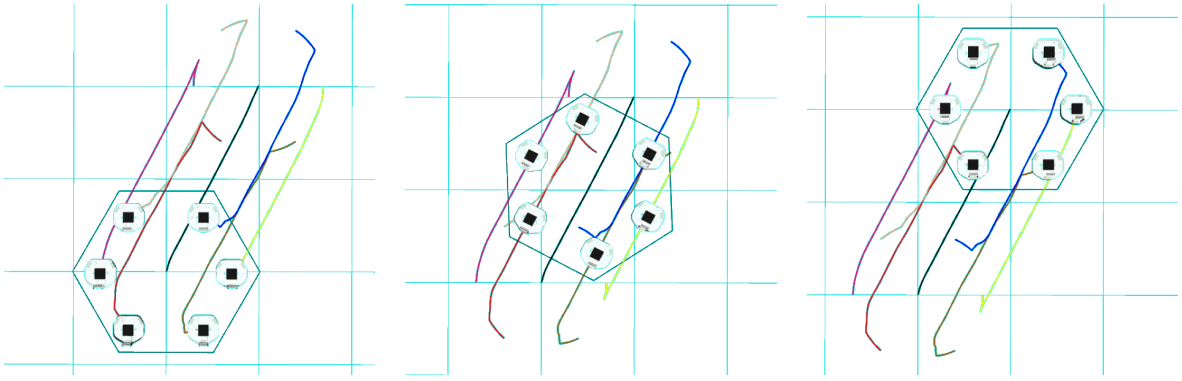
Presented in Figures 5.14 are three examples of how the base footprint of a coupled system of robots would look like if a robot fleet would carry a squared, concave, or hexagonal object. The base footprint can be defined however the user desires and the carrier robots could be placed anywhere in the base footprint depending on the user's preferences. The developed local planner only needs to know how the base footprint is defined and where in the base footprint the robots are located. According to the number of robots carrying the shared payload, the developed local planner generates new trajectories as well as corresponding velocities for individual robots to carry the shared payload. The base footprint definitions for the three objects brought in Figure 5.14 can be found in mrpf_local_planner GitHub.



(a) *Square-Start Configuration* (b) *Square-Middle Configuration* (c) *Square-Goal Configuration*



(d) *Concave-Start Configuration* (e) *Concave-Middle Configuration* (f) *Concave-Goal Configuration*



(g) *Hexagon-Start Configuration* (h) *Hexagon-Middle Configuration* (i) *Hexagon-Goal Configuration*

Figure 5.14: *Trajectory segregation for different shared payloads and their movements along the generated trajectories*

6 Future work

The MRPFPlannerROS local planner developed in this thesis can be used in ROS Navigation as-is but many of the existing local planners support both differential-drive and holonomic platforms. However, the local planner developed in this work only supports holonomic platforms. Thus, an extension to the developed local planner would be adding the feature for both the platforms in order to make the planner usable for different mobile platforms with various limitations.

In addition, the local planner currently does not support collision avoidance in environments with dynamic obstacles since it does not receive the data of the local_costmap. To address this issue, the LaserScan data from all the robots in the fleet should be combined together so that the coupled system of robots carrying the object be able to perceive its surrounding environment. Therefore, this issue should also be taken into consideration and tackled in the future.

Moreover, the global planning algorithm proposed in this work is yet to be integrated as a standard global planner to ROS Navigation. For that, the algorithm should be written in C++ and specific methods for a standard global planner be added to it. Thus, one may take this work forward and adhere the global planning algorithm as a standard global planner to ROS Navigation.

7 Conclusion

This thesis focused on proposing a method for solving the problem of object transportation in multi-robot systems. It is shown that how the problem can be tackled using ROS by developing a global planning algorithm as well as a local planner integrated to ROS Navigation. In the current state of the work, the local planner is already integrated into ROS Navigation and can be easily used by setting ROS Navigation local planner to MRPFPlannerROS.

A global planning algorithm was written to plan for a fleet of robots and generate collision-free trajectories for them. As one of the novelties of this thesis, it went beyond the reactive one step collision-avoidance approach prevalent in existing works by formulating a global trajectory optimization problem. As a result, the various relative re-positioning that the robots need to perform throughout the length of the trajectory were obtained. As the formulated optimization problem was much alike the problems in deep learning, state-of-the-art gradient descent methods such as Momentum and ADAM were utilized to solve the problem. It was revealed that Momentum works better for the purpose of this work. The developed global motion planning algorithm was compared to an off-the-shelf ROS Navigation existing global planner and the qualities of smoothness and collision avoidance were compared. The results showed that the existing ROS Navigation planner failed in generating a collision-free trajectory in most of the cases. It was also shown that the trajectories generated by the developed global planner were smoother than those of the existing ROS planner.

A local planner was developed and integrated to ROS Navigation that could take in a global trajectory for a robot fleet carrying a shared payload and output right velocities for individual robots to move along the trajectory and carry the object.

The global planning algorithm developed is specific to the case of carrying a rectangular object using two robots. However, the developed local planner is general enough to output corresponding velocities for each robot in a rigid constellation moving along a global trajectory (e.g., for transporting an object).

Acknowledgements

I would like to thank

my supervisors Karl Kruusamäe, Arun Kumar Singh, and Robert Valner for their guidance and support not only in this thesis but also in the whole journey of my Master's,

my parents who always support me and give me solace whenever I am distressed or down,

and the other members of IMS Lab, including Fatemeh, Iman, Morteza, and Priit who helped me getting this work done.

Bibliography

[1] E. J. Rodríguez-Seda et al. "Bilateral Teleoperation of Multiple Mobile Agents: Coordinated Motion and Collision Avoidance". In: *IEEE Transactions on Control Systems Technology* 18.4 (July 2010), pp. 984–992. ISSN: 2374-0159. DOI: 10.1109/TCST.2009.2030176.

[2] Wang Guanghua et al. "Study on formation control of multi-robot systems". In: *2013 Third International Conference on Intelligent System Design and Engineering Applications*. IEEE. 2013, pp. 1335–1339.

[3] Morgan Quigley et al. "ROS: an open-source Robot Operating System". In:

[4] Gordon Wilfong. "Motion planning in the presence of movable obstacles". In: *Annals of Mathematics and Artificial Intelligence* 3 (Jan. 1991), pp. 131–150. DOI: 10.1007/BF01530890.

[5] Anybot. *iRobot*. URL: https://en.wikipedia.org/wiki/Motion_planning (visited on 04/25/2020).

[6] Joshua Bialkowski et al. "Efficient collision checking in sampling-based motion planning via safety certificates". In: *The International Journal of Robotics Research* 35.7 (2016), pp. 767–796.

[7] Lydia E Kavraki et al. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces". In: *IEEE transactions on Robotics and Automation* 12.4 (1996), pp. 566–580.

[8] James J Kuffner and Steven M LaValle. "RRT-connect: An efficient approach to single-query path planning". In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*. Vol. 2. IEEE. 2000, pp. 995–1001.

[9] Sertac Karaman and Emilio Frazzoli. "Sampling-based algorithms for optimal motion planning". In: *The international journal of robotics research* 30.7 (2011), pp. 846–894.

[10] Robert Bohlin and Lydia E Kavraki. "Path planning using lazy PRM". In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*. Vol. 1. IEEE. 2000, pp. 521–528.

[11] Jinwook Huh and Daniel D Lee. "Learning high-dimensional mixture models for fast collision detection in rapidly-exploring random trees". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 63–69.

[12] Nathan Ratliff et al. "CHOMP: Gradient optimization techniques for efficient motion planning". In: *2009 IEEE International Conference on Robotics and Automation*. IEEE. 2009, pp. 489–494.

[13] Mrinal Kalakrishnan et al. "STOMP: Stochastic trajectory optimization for motion planning". In: *2011 IEEE international conference on robotics and automation*. IEEE. 2011, pp. 4569–4574.

[14] Wikipedia. *Trajectory Optimization*. URL: `https://en.wikipedia.org/wiki/Trajectory_optimization` (visited on 03/31/2020).

[15] Wilko Schwarting et al. "Parallel autonomy in automated vehicles: Safe motion generation with minimal intervention". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 1928–1935.

[16] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[17] C. W. de Silva. "Some issues and applications of multi-robot cooperation". In: *2016 IEEE 20th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. May 2016, pp. 2–2. DOI: 10.1109/CSCWD.2016.7565951.

[18] Peter R Wurman, Raffaello D'Andrea, and Mick Mountz. "Coordinating hundreds of cooperative, autonomous vehicles in warehouses". In: *AI magazine* 29.1 (2008), pp. 9–9.

[19] Kiril Solovey and Dan Halperin. "On the hardness of unlabeled multi-robot motion planning". In: *The International Journal of Robotics Research* 35.14 (2016), pp. 1750–1759.

[20] Kevin M Lynch and Frank C Park. *Modern Robotics*. Cambridge University Press, 2017.

[21] Stephen Kloder and Seth Hutchinson. "Path planning for permutation-invariant multirobot formations". In: *IEEE Transactions on Robotics* 22.4 (2006), pp. 650–665.

[22] Alex Kushleyev et al. "Towards a swarm of agile micro quadrotors". In: *Autonomous Robots* 35.4 (2013), pp. 287–300.

[23] Indranil Saha et al. "Automated composition of motion primitives for multi-robot systems from safe LTL specifications". In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2014, pp. 1525–1532.

[24] Atsushi Yamashita et al. "Motion planning of multiple mobile robots for cooperative manipulation and transportation". In: *IEEE Transactions on Robotics and Automation* 19.2 (2003), pp. 223–237.

[25] Gyuho Eoh et al. "Multi-robot cooperative formation for overweight object transportation". In: *2011 IEEE/SICE International Symposium on System Integration (SII)*. IEEE. 2011, pp. 726–731.

[26] Golnaz Habibi et al. "Distributed centroid estimation and motion controllers for collective transport by multi-robot systems". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 1282–1288.

[27] Jonathan Fink, M Ani Hsieh, and Vijay Kumar. "Multi-robot manipulation via caging in environments with obstacles". In: *2008 IEEE International Conference on Robotics and Automation*. IEEE. 2008, pp. 1471–1476.

[28] Paolo Fiorini and Zvi Shiller. "Motion planning in dynamic environments using velocity obstacles". In: *The International Journal of Robotics Research* 17.7 (1998), pp. 760–772.

[29] Javier Alonso-Mora et al. "Local motion planning for collaborative multi-robot manipulation of deformable objects". In: *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2015, pp. 5495–5502.

[30] Javier Alonso-Mora, Stuart Baker, and Daniela Rus. "Multi-robot formation control and object transport in dynamic environments via constrained optimization". In: *The International Journal of Robotics Research* 36.9 (2017), pp. 1000–1021.

[31] R. Mishra and A. Javed. "ROS based service robot platform". In: *2018 4th International Conference on Control, Automation and Robotics (ICCAR)*. Apr. 2018, pp. 55–59. DOI: 10.1109/ICCAR.2018.8384644.

[32] *navigation/Tutorials/RobotSetup*. http://library.isr.ist.utl.pt/docs/roswiki/navigation(2f)Tutorials(2f)RobotSetup.html. [Online; 16-Jan-2020].

[33] *Navigation - ROS Wiki*. http://wiki.ros.org/navigation. [Online; 16-Jan-2020].

[34] ROS Wiki. *Factoring in A Polygon Base Footprint*. URL: https://answers.ros.org/question/296406/global-planner-with-taking-into-account-the-robot-footprint/ (visited on 03/31/2020).

[35] Dmitry Berenson et al. "Manipulation planning on constraint manifolds". In: *2009 IEEE International Conference on Robotics and Automation*. IEEE. 2009, pp. 625–632.

[36] Arun Kumar Singh et al. "Combining Method of Alternating Projections and Augmented Lagrangian for Task Constrained Trajectory Optimization". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 7568–7575.

[37] Xinyu Zhang, Minkyoung Lee, and Young J Kim. "Interactive continuous collision detection for non-convex polyhedra". In: *The Visual Computer* 22.9-11 (2006), pp. 749–760.

[38] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016).

[39] engmrk. *Gradient descent with momentum*. URL: https://engmrk.com/gradient-descent-with-momentum/ (visited on 04/20/2020).

[40] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *CoRR* abs/1609.04747 (2016). arXiv: 1609.04747. URL: http://arxiv.org/abs/1609.04747.

[41] Kaiyuzh. *ROS Navigation Tuning Guide*. URL: http://kaiyuzh.me/documents/navguide.pdf (visited on 03/31/2020).

Non-exclusive licence to reproduce thesis and make thesis public

I, Houman Masnavi

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   "MULTI-ROBOT MOTION PLANNING FOR SHARED PAYLOAD TRANSPORTATION"

   supervised by Karl Kruusamäe, Arun Kumar Singh, Robert Valner

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Houman Masnavi
20.05.2020