MPC-GRAPH: NONLINEAR FEEDBACK MOTION PLANNING USING SPARSE SAMPLING BASED NEIGHBORHOOD GRAPH

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

SİMAY ATASOY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

JANUARY 2022

Approval of the thesis:

MPC-GRAPH: NONLINEAR FEEDBACK MOTION PLANNING USING SPARSE SAMPLING BASED NEIGHBORHOOD GRAPH

submitted by **SİMAY ATASOY** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. İlkay Ulusoy
Head of Department, **Electrical and Electronics Engineering** _____

Assist. Prof. Dr. Mustafa Mert Ankaralı
Supervisor, **Electrical and Electronics Engineering, METU** _____

Examining Committee Members:

Prof. Dr. Kemal Leblebicioğlu
Electrical and Electronics Engineering, METU _____

Assist. Prof. Dr. Mustafa Mert Ankaralı
Electrical and Electronics Engineering, METU _____

Prof. Dr. Ömer Morgül
Electrical and Electronics Engineering, Bilkent University _____

Assoc. Prof. Dr. Emre Özkan
Electrical and Electronics Engineering, METU _____

Assoc. Prof. Dr. Ahmet Buğra Koku
Mechanical Engineering, METU _____

Date: 26.01.2022

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Simay Atasoy

Signature :

ABSTRACT

MPC-GRAPH: NONLINEAR FEEDBACK MOTION PLANNING USING SPARSE SAMPLING BASED NEIGHBORHOOD GRAPH

Atasoy, Simay
M.S., Department of Electrical and Electronics Engineering
Supervisor: Assist. Prof. Dr. Mustafa Mert Ankaralı

January 2022, 65 pages

Robust and safe feedback motion planning and navigation is a critical task for autonomous mobile robotic systems considering the highly dynamic and uncertain nature scenarios of modern applications. For these reasons motion planning and navigation algorithms that have deep roots in feedback control theory has been at the center stage of this domain recently. However, the vast majority of such policies still rely on the idea that a motion planner first generates a set of open-loop possibly time-dependent trajectories, and then a set of feedback control policies track these trajectories in closed-loop while providing some error bounds and guarantees around these trajectories. In contrast to trajectory-based approaches, some researchers developed feedback motion planning strategies based on connected obstacle-free regions, where the task of the local control policies is to drive the robot(s) in between these particular connected regions. In this work, we propose a feedback motion planning algorithm based on sparse random neighborhood graphs and constrained nonlinear Model Predictive Control (MPC). The algorithm first generates a sparse neighborhood graph as a set of connected simple rectangular regions. After that, during navigation, an MPC based online feedback control policy funnels the robot with nonlinear dynamics from

one rectangle to the other in the network, ensuring no constraint violation on state and input variables occurs with guaranteed stability. In this framework, we can drive the robot to any goal location provided that the connected region network covers both the initial condition and the goal position.

In this thesis, we demonstrate the effectiveness and validity of the algorithm on simulation studies which include four different robot motion models. Our work mainly focuses on motion planning applications implemented on Unmanned Surface Vehicles (USV). In order to show the robustness of the proposed algorithm, we applied process noise to the system and report the results. We compare the sampling performance of the proposed algorithm with sampling-based neighborhood graph method. The results show that MPC-Graph algorithm generates a more sparse graph structure and can drive the robot to the goal location in the presence of process noise.

Keywords: Model Predictive Control, Optimal Control, Sampling-based Motion Planning, Unmanned Surface Vehicles

ÖZ

MPC-GRAPH: SEYREK ÖRNEKLEME BAZLI KOMŞU GRAFİĞİ KULLANARAK DOĞRUSAL OLMAYAN GERİ BESLEMELİ HAREKET PLANLAMA

Atasoy, Simay
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Dr. Öğr. Üyesi. Mustafa Mert Ankaralı

Ocak 2022 , 65 sayfa

Modern uygulamaların son derece dinamik ve belirsiz doğa senaryoları göz önüne alındığında, sağlam ve güvenli geri bildirim hareket planlaması ve navigasyonu, otonom mobil robot sistemleri için kritik bir görevdir. Bu nedenlerden dolayı, geri beslemeli kontrol teorisinde derin kökleri olan hareket planlama ve navigasyon algoritmaları, son zamanlarda bu alanın merkezinde yer almaktadır. Ancak bu tür poliçelerin çoğunluğu hala hareket planlayıcısının öncelikle zamana bağlı yörüngelerden oluşan açık döngü seti oluşturduğunu kabul etmektedir. Sonrasında geri bildirimli kontrol poliçesi seti oluşturulan bu yörüngeleri belli hata sınırları çerçevesinde takip eder. Yörünge bazlı yaklaşımların aksine, bazı araştırmacılar birbirine bağlı engelsiz alanlar yaklaşımını kullanarak geri beslemeli hareket planlaması yöntemi geliştirdiler. Bu yaklaşımda lokal kontrol poliçesinin amacı robotun özel olarak belirlenmiş bağlı engelsiz bölgeler arasında hareket planlamasını gerçekleştirmektir. Bu çalışmada seyrek rastgele komşuluk grafiklerini ve kısıtlı doğrusal olmayan Model Öngörülü Kontrolcü (MPC) kullanarak, bir hareket planlama algoritması öneriyoruz. Algoritma ilk olarak

birbirine bağlı dikdötgenlerden oluşan seyrek bir komşuluk grafiği oluşturur. Sonrasında, robotun hareket kontrolü için MPC bazlı geri beslemeli kontrol poliçesi doğrusal dinamikleri olmayan robotu bir dikdörtgensel bölgeden diğerine hareket ettirir. Bu hareket esnasında da sistemin durum ve girdi kısıtlarının ihlal edilmediğinden sistemin kararlılığını da garanti ederek emin olur. Bu çerçevede, birbirine bağlı bölgelerin başlangıç ve bitiş noktalarını kapsadığını varsayarsak, robotu herhangi bir hedef noktasına sürebiliriz.

Bu tez çalışmasında, algoritmanın geçerliliğini ve uygulanabilirliğini farklı robot hareket modellerini içeren simülasyon ortamında test ettik. Çalışmamız başlıca Suüstü İnsansız Araçlar üzerinde hareket planlama algoritması geliştirilmesi üzerinde yoğunlaşmaktadır. Algoritmamızın gürültü varlığında gürbüzlüğünü gösterebilmek için sisteme gürültü uyguladık. Algoritmamızın örnekleme performansını literatürde karşılatığımız benzer yöntemlerle karşılaştırdık. Sonuçlar MPC-Graph algoritmasının seyrek bir grafik oluşturduğunu ve gürültü varlığında bile sistemi hedef noktasına ulaştırabildiğini göstermektedir.

Anahtar Kelimeler: Model Öngörülü Kontrol, Optimal Kontrol, Örnekleme Tabanlı Hareket Planlama, İnsansız Suüstü Araçları

To my dearest grandfather İsmail Atasoy

ACKNOWLEDGMENTS

I would like to start by expressing my deepest gratitude and appreciation to my supervisor Assist. Prof. Dr. Mustafa Mert Ankaralı for his endless support, encouragement and guidance throughout this work. I am grateful that I had the opportunity to work with him.

I would like to thank Osman Kaan Karagöz for his patience and faith in me. He has always gave me the best advises and I am glad that I have the chance to have invaluable discussions with him.

My sincere thanks also goes to İzel Sever Gökmen for her unconditional support. Even in my stressful times, she has always been there for me to cheer me up. I am so lucky to have her as my close friend.

I would like to thank Ulaş Süreyya Bingöl, for his endless love and support. I am thankful that I have the chance to grasp his way of approaching to problems both in my engineering and daily life.

I would like to thank Turkish Scientific and Technological Research Council(TÜBİTAK) for their financial support through project 118E195.

And above all, this work wouldn't have been possible without the support, love and patience of my dearest family. I would like to thank my mother Serpil Atasoy and my father Vahit Atasoy for being there for me throughout this journey.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

2D	2 Dimensional
3D	3 Dimensional
LQR	Linear Quadratic Regulator
USV	Unmanned Surface Vehicle
UGV	Unmanned Ground Vehicle
MPC	Model Predictive Control
DOF	Degrees of Freedom
SNG	Sampling-based Neighborhood Graph
PRM	Probabilistic Roadmaps
RRT	Rapidly-exploring Random Trees
MIMO	Multiple-input Multiple-output
SNR	Signal-to-noise ratio

CHAPTER 1

INTRODUCTION

1.1 Motivation of the Study

Advancements in hardware and software technology pave the way for the common usage of autonomous mobile robots in every aspect of daily life. Today, autonomous robot platforms operate in numerous areas ranging from space and underwater exploration to agriculture, disinfection, mining, search and rescue, warehouse automation, military and many more. In the last decade, autonomous systems started to replace human operators for applications in which precision is required, the working environment is hazardous or threatens human life. In order to use autonomous systems for the aforementioned applications, human factor in the process should be eliminated or at least minimized. Configurations and algorithms implemented on these systems are tailored towards their specific utilization. In this work, we mainly focus on developing a path and motion planning algorithm which is a crucial aspect for mobile robotic systems.

The main concern of autonomous motion planning is to drive the agent from the start configuration to the goal configuration while obeying the constraints coming from the environment and agent itself. In motion planning applications the success of the algorithm mainly depends on its capability of handling obstacles. Today, safe and robust collision-free motion planning is still an active research area open to improvements.

In general, motion planning algorithms first generate a set of open loop piecewise-smooth trajectories and then follow these trajectories with feedback control policies. For the first phase, rather than using a trajectory-based approach some researchers implemented a trajectory-free motion planning concept. In this approach, the given map

of the environment is sampled with connected sub-regions and the aim of the planner is to drive the robot to the goal region while respecting the constraints coming from these sampled regions [4–8]. Instead of directly following a pre-defined trajectory, this approach relaxes the constraints by increasing the possible movement area for the robot.

For the second phase, feedback control, in order to effectively force the constraints we implemented Model Predictive Control (MPC). In the last decade Model Predictive Control (MPC) have attracted much attention for motion planning tasks ranging from applications in aerial vehicles [9,10], legged robots [11,12], spacecrafts [13,14], underwater vehicles [15, 16] etc. MPC offers a framework that can handle multiple-input multiple-output (MIMO) systems and force constraints for states and inputs, making it an effective instrument in collision-free motion planning applications. In this work in order to show the capabilities of MPC, we implemented it on four different robot motion models; fully-actuated holonomic linear model, fully-actuated holonomic model with damping, fully-actuated USV model with four thrusters, differential USV model with two thrusters.

In recent years there is an increasing trend in using autonomous systems in maritime industry. According to Annual Overview of Marine Casualties and Incidents (2020) between years 2014 and 2019 [17], 43% of the casualties with ships are due to navigational casualties which include accidents resulting from contact, collision and grounding/stranding. From 1499 accident events occurred between 2014 and 2019, 60.6% of them were attributed to human erroneous action. In order to decrease the accident rates that are attributed to human errors, studies regarding automation of surface vehicles is a currently attractive research area. This is one of the main reasons why this thesis work focuses on motion planning problem for surface vehicles.

1.2 Literature Review

The early applications of MPC emerged as a need for optimizing the multi-constraint processes in chemical industry, which mainly includes refining and petrochemical sectors [18, 19]. In order to meet the requirements and extend the usage area of

MPC, researchers implemented several improvements. Garcia et al. [19] formulated a constraint MPC framework for linear stable systems focusing on the applications of chemical engineering processes. Several researches [20–23] concentrated on developing a theoretical formulation for guaranteeing the closed loop stability for constrained linear systems.

Since real world problems are highly nonlinear, a considerable amount of research has been carried out to extend the usage of MPC to constrained nonlinear systems [24–28]. As opposed to linear case, the infinite horizon problem for nonlinear systems cannot be solved numerically and reducing the horizon may cause undesired system behaviors. In order to address this issue, Michalska and Mayne [26] proposed a hybrid MPC that replaces the terminal constraint with a terminal region. When the nonlinear system reaches this region, another controller is employed and as a result the system is asymptotically stabilized. However, to guarantee the stability of the system, a global optimization problem is required to be solved. Chen and Allgöwer [28] on the other hand, use an infinite horizon approach and calculate a penalty term for the final state to bound the infinite horizon cost. They establish the bound by controlling the nonlinear system with a fictitious linear state feedback in the predetermined terminal region. The work of Nicolao et al. [27] implements a similar approach.

With the extension of MPC to nonlinear systems, several robotics applications implemented this framework for path and motion planning operations. In recent years, numerous works have been published using nonlinear MPC for control of unmanned ground vehicles (UGV) [29–34] and unmanned aerial vehicles (UAV) [35–40]. In [29], time varying weights are implemented for trajectory tracking of a two-wheeled mobile robot with constraints. In works [30] and [33], stability of NMPC is guaranteed by inclusion of terminal state penalty cost and terminal region. In [35, 36], for the path planning of UAV platforms an NMPC framework is implemented in the presence of state and input constraints. Shimada et al. [39] and Tanveer et al. [40] used NMPC for disturbance rejection in UAV applications.

With an increasing demand towards the utilization of autonomous surface vehicles in the military, search and rescue, transportation and exploration, the literature regarding control of USVs started to flourish in the last few decades [41]. Several works in lit-

erature employ MPC for USV control [42–46]. Zhao et al. [42] propose an improved MPC framework with the inclusion of global course constraint and event-triggered mechanism. In [43], researchers propose a finite control set model predictive control for collision avoidance problem.

1.3 Contributions

A preliminary version of this work reporting early results was presented by Karagöz et al. [7]. This thesis work presents a significantly improved version with the following qualities: (i) performance of the algorithm in the presence of process noise is evaluated, (ii) algorithm is modified in order to recover from failures caused by noise (iii) motion models are generalized to include orientation and angular rate of the system , (iv) MPC and graph search costs are modified to increase the performance of the proposed algorithm.

The main contribution of our work is fusion of sampling-based motion planning with model predictive control. This thesis work proposes a new trajectory-free, sampling-based feedback motion planning algorithm that can handle arbitrary obstacle configurations for autonomous robots in 2D environments. In the proposed algorithm the obstacle free region is sampled with rectangular regions. In order to increase the sparsity of the of the obtained graph structure, the nodes are expanded. One of the similar approaches that Yang et al. [3] proposed, samples the obstacle free region with spherical balls. Compared to SNG algorithm, MPC-Graph generates a more sparse graph regarding the results presented in Chapter 4. Furthermore, the proposed algorithm returns a sequence of nodes rather than a predefined trajectory which makes the robot move in a smoother route. As long as it is guaranteed that the robot stays inside the sampled regions, collision with static obstacles is prevented. For that purpose, MPC forces the state and input constraints to the system.

One of the advantages MPC provides is that, it makes possible to use both linear and nonlinear system models, which highly increases the application areas of the proposed algorithm. In this thesis work, we implement the algorithm on both linear and nonlinear system models and obtained satisfactory results. In order to show the

robustness of the algorithm, we added process noise and report the behavior of the system in Chapter 4.

We implemented MPC-Graph algorithm using Matlab simulations and tested it with four different system models. To show the sampling performance of the algorithm, we generated maps with both polygonal and circular obstacles.

1.4 The Outline of the Thesis

The organization of the thesis is as follows. In Chapter 2, we give a brief introduction to sampling-based motion planning and MPC concepts. We also explain the dynamics of the unmanned surface vehicle model and mention some of the important parameters that have significant effect on its motion. After providing the fundamental background for the proposed algorithm, in Chapter 3 we thoroughly explain our novel MPC-Graph algorithm which fundamentally consists of three major phases. Chapter 4 reports the implementation and simulation results. We tested our algorithm on four different motion models and in the beginning of the chapter these models are briefly explained. In order to show the effectiveness and validity of the proposed algorithm we performed Monte Carlo experiments. Finally Chapter 5, outlines our work and discusses the future directions of the proposed study.

CHAPTER 2

PRELIMINARIES AND BACKGROUND

MPC-Graph algorithm uses a sampling-based motion planning method combined with the Model Predictive Control to impose state and input constraints. This chapter mainly focuses on giving a brief introduction to these concepts.

2.1 Sampling-based Motion Planning

The core idea behind sampling-based motion planning is randomly sampling the available configurations in a given map and returning these available configurations in a tree or a graph like structure. Representation of the configuration space with the aforementioned structures reduces the computational complexity coming from explicitly modeling the entire free space.

Probabilistic Roadmaps (PRM) and Rapidly-exploring Random Trees (RRT) are fundamental algorithms that provide a bedrock for sampling-based motion planning. PRM algorithm consists of two consecutive phases which are learning phase and query phase. Learning phase, randomly samples the collision-free configurations and connects these available configurations to obtain a graph structure. Then, the query phase connects the start and goal configurations to the obtained roadmap [47]. Although PRM is a widely used powerful algorithm in motion planning applications, it is unable to handle nonholonomic constraints. In order to address this flaw of PRM, RRT algorithm is proposed [6]. As opposed to PRM algorithm, RRT constructs a tree structure that is spawned from its root which is the start configuration. Then, continues sampling the free space until a connection between start and goal configurations is obtained. In literature, several works are available that compare the aspects of using

an RRT or PRM based algorithm and their variants [48, 49].

The aforementioned algorithms and their variants generally tend to sample the free space with 'point-based' nodes. In [3], a different approach is implemented by using spherical regions to indicate the collision-free areas that are safe for the robot to travel. The proposed algorithm, first samples the obstacle-free space with connected spherical regions until a predefined termination condition is satisfied. These regions construct a neighborhood graph for the navigation of the robot. Then, a global navigation function drives the robot from one node to another until it reaches the goal point. One of the advantages this kind of a framework provides is that compared to RRT and PRM algorithms fewer nodes are obtained and thus a sparse graph structure is attained. Fig.2.1 is an illustration based on the work [3], that visualizes the sampling of collision-free space with circular regions.

In literature, several works implemented sparse neighborhood trees for path planning applications. These works include different types of representation structures for the nodes such as ellipse, circle or square [4, 5, 50].

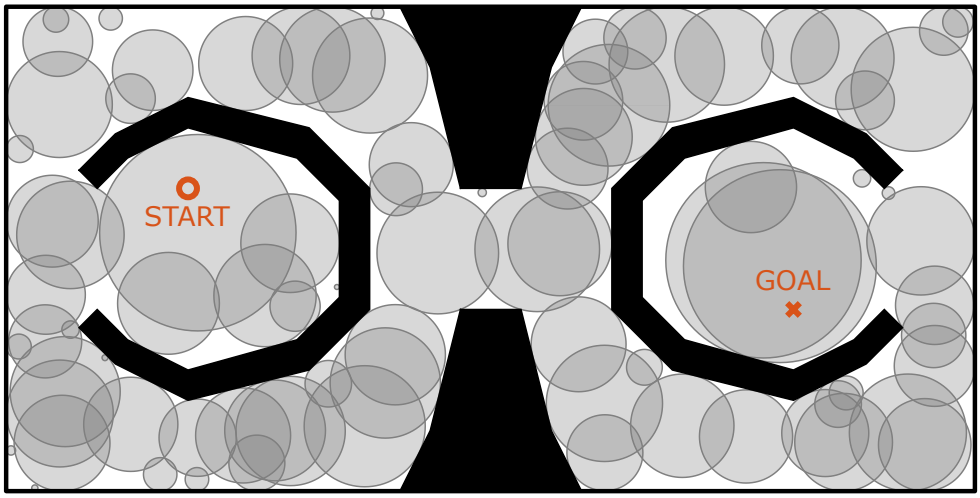


Figure 2.1: Sampling the obstacle-free space with circular regions as in [3]

2.2 Model Predictive Control

Model Predictive Control is a feedback control algorithm that generates an input sequence at each sampling instant for the indicated horizon to minimize the related

cost function while respecting the specified constraints. After the input sequence is obtained first element of this sequence is applied to the plant and the horizon is proceeded towards the future [51]. In this thesis work, we implement MPC on both linear and nonlinear systems.

2.2.1 Linear Model Predictive Control

For the implementation of MPC on linear systems, we used a so called dual mode approach which combines the finite horizon and infinite horizon control. The first mode computes the optimal inputs by applying finite horizon Linear Quadratic Regulator (LQR), whereas the second mode uses infinite horizon LQR to obtain the inputs. The problem formulation is as follows:

$$J = \sum_{j=0}^{N-1} (q_{j|k}^T Q q_{j|k} + u_{j|k}^T R u_{j|k}) + q_{N|j}^T P q_{N|j} \tag{2.1}$$

subject to

$$q_{j+1|k} = A q_{j|k} + B u_{j|k},\ j = 0, 1, ..., N-1 \tag{2.2a}$$

$$q_{min} \leq q_{j|k} \leq q_{max},\ j = 0, 1, ..., N \tag{2.2b}$$

$$u_{min} \leq u_{j|k} \leq u_{max},\ j = 0, 1, ..., N \tag{2.2c}$$

In equation (2.1), N is the prediction horizon, Q is the state cost matrix, R is the input cost matrix and P is the final state cost matrix. Equation (2.2b) indicates the state constraints. Likewise, equation (2.2c) indicates the input constraints.

In dual mode MPC, P matrix is calculated by solving the discrete-time Lyapunov equation,

$$(A - BK_\infty)^T P (A - BK_\infty) - P + K_\infty R K_\infty = 0 \tag{2.3}$$

and K_∞ is the constant state feedback gain of the infinite horizon LQR problem.

2.2.2 Nonlinear Model Predictive Control

For the control of the nonlinear systems, we used the approach presented in [28]. The work of Chen and Allgöwer is applicable to both stable and unstable systems and with

the inclusion of terminal region and terminal cost matrix the algorithm guarantees asymptotic closed-loop stability. The objective cost function comprises of mainly two parts: an integral square error calculated over a finite horizon and a quadratic terminal cost term as in the linear case.

The optimization problem of MPC can be formalized as follows:

$$J(\mathbf{q}(t), \mathbf{u}(\cdot)) = \int_{t}^{t+T_p} \Big(\mathbf{q}(\tau)^T Q \mathbf{q}(\tau) + \mathbf{u}(\tau)^T R \mathbf{u}(\tau)\Big) d\tau + \mathbf{q}(t+T_p)^T P \mathbf{q}(t+T_p) \tag{2.4}$$

subject to

$$\dot{\mathbf{q}} = \mathbf{f}(\mathbf{q}, \mathbf{u}) \tag{2.5a}$$

$$\mathbf{u}(\tau) \in U, \qquad \tau \in [t, t+T_p] \tag{2.5b}$$

$$\mathbf{q}(t+T_p) \in \mathbf{\Omega}. \tag{2.5c}$$

$\mathbf{q}(\cdot)$ is the system state vector, $\mathbf{u}(\cdot)$ is the input, the system is defined by a set of non-linear differential equations, $f(\mathbf{q}(\cdot), \mathbf{u}(\cdot))$ and T_p is a finite prediction horizon. Q and R are positive-definite and symmetric cost matrices for states and inputs, respectively. We obtained terminal state penalty matrix P and terminal region $\mathbf{\Omega}$ by applying the procedure presented in [28]. First, the Jacobian linearization of the system at the origin is calculated,

$$\dot{\mathbf{q}} = A\mathbf{q} + B\mathbf{u} \tag{2.6}$$

where $A = (\partial \mathbf{f}/\partial \mathbf{q})(0,0)$ and $B = (\partial \mathbf{f}/\partial \mathbf{u})(0,0)$. Provided that eqn. (2.6) is stabilizable, a linear state feedback $\mathbf{u} = K\mathbf{q}$ can be obtained and by substituting $\mathbf{u}$ in eqn. 2.6,

$$\dot{\mathbf{q}} = (A + BK)\mathbf{q} \tag{2.7}$$

the obtained matrix $A_K = A + BK$ is asymptotically stable. Provided that the Jacobian linearization (2.6) at the origin is stabilizable, the Lyapunov equation,

$$(A_K + \kappa I)^T P + P(A_K + \kappa I) = -Q^\star \tag{2.8}$$

where,

$$Q^\star = Q + K^T R K \in \mathbb{R}^{n \times n} \tag{2.9}$$

admits a unique positive-definite and symmetric solution P. In eqn. (2.8), κ is chosen such that it satisfies the following condition,

$$\kappa < -\lambda_{max}(A_K) \qquad \kappa \in [0, \infty). \tag{2.10}$$

After determining the terminal state penalty matrix P, the procedure continues with determination of the terminal region. Provided that, there exists a constant $\alpha \in (0, \infty)$, a region $\mathbf{\Omega}_\alpha$ in the neighborhood of α is defined as follows,

$$\mathbf{\Omega}_\alpha = \{\mathbf{q} \in \mathbb{R}^n | \mathbf{q}^T P \mathbf{q} \leq \alpha\}. \tag{2.11}$$

such that $\forall x_1 \in \Omega_\alpha$, infinite horizon cost J^∞,

$$J^\infty(\mathbf{q}_1, \mathbf{u}) = \int_{t_1}^{\infty} \Big(\mathbf{q}(t)^T Q \mathbf{q}(t) + \mathbf{u}(t)^T R \mathbf{u}(t)\Big) dt \tag{2.12}$$

starting from $\mathbf{q}(t_1) = \mathbf{q}_1$ and controlled by $\mathbf{u} = K\mathbf{q}$ is bounded from above as follows,

$$J^\infty(\mathbf{q}_1, \mathbf{u}) \leq \mathbf{q}_1^T P \mathbf{q}_1. \tag{2.13}$$

With the inclusion of an upper bound to the infinite horizon cost, the asymptotic stability of the system is guaranteed.

2.2.3 USV Dynamics

Compared to unmanned ground vehicles (UGV), unmanned surface vehicles (USV) are exposed to different environment dynamics due to their application medium. Since USVs operate in sea environment, on top of rigid body dynamics also added mass and damping terms should be considered in a USV model.

Let $\boldsymbol{\eta} = [x\ y\ \psi]^T \in \mathbb{R}^3$ denote the pose vector, where x and y are the earth-fixed reference frame coordinates and ψ is the heading angle, let $\boldsymbol{\nu} = [u\ v\ r]^T \in \mathbb{R}^3$ denote the velocity vector of the dynamic model where u and v are linear velocities, called surge and sway, and r is the angular velocity. Fig. 2.2 shows a schematic for the USV model.

In our simulations, we use the 3 DOF horizontal plane model presented in [52]. The formulation is as follows,

$$\dot{\boldsymbol{\eta}} = J(\psi)\boldsymbol{\nu} \tag{2.14a}$$

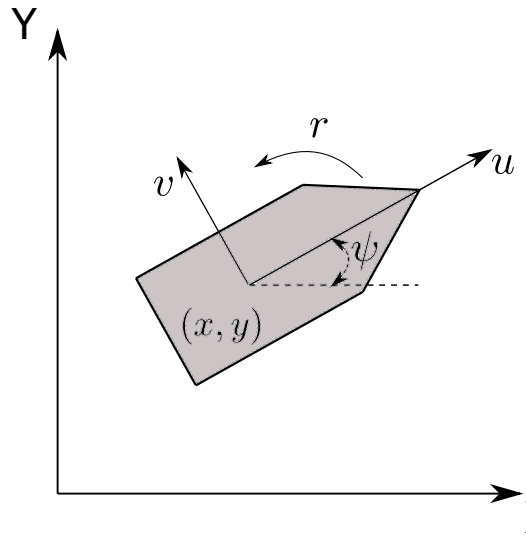


Figure 2.2: Schematic for the USV model. X-Y and u-v denote the earth-fixed and body fixed reference frames, respectively.

$$M\dot{\boldsymbol{\nu}} + C(\boldsymbol{\nu})\boldsymbol{\nu} + D(\boldsymbol{\nu})\boldsymbol{\nu} = \boldsymbol{\tau} \tag{2.14b}$$

where

$$M = M_A + M_{RB} \tag{2.14c}$$

$$C(\boldsymbol{\nu}) = C_A(\boldsymbol{\nu}) + C_{RB}(\boldsymbol{\nu}) \tag{2.14d}$$

$$D(\boldsymbol{\nu}) = D_l + D_n(\boldsymbol{\nu}). \tag{2.14e}$$

In (2.14a), $J(\psi)$ represents the rotation matrix from body reference frame to earth fixed reference frame. In (2.14b) the general formulation for the USV dynamics is given where M, $C(\boldsymbol{\nu})$, $D(\boldsymbol{\nu})$ are inertia, Coriolis/centripetal, damping matrices and $\boldsymbol{\tau}$ is the thruster force vector, respectively. A body moving in a liquid medium, transports some of the surrounding liquid by its motion. As a result, it is observed that the body weighs more compared to its original weight. In order to compensate this effect, added mass terms M_A and $C_A(\boldsymbol{\nu})$ are included in (2.14c) and (2.14d). The damping effect is given in (2.14e) where D_l and $D_n(\boldsymbol{\nu})$ denote the linear and nonlinear damping matrices, respectively. The subscript RB in (2.14c) and (2.14d) stands for rigid body terms. The matrices in (2.14a)-(2.14e) is represented as follows,

$$J = \begin{bmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}, M_{RB} = \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & I_z \end{bmatrix}, M_A = \begin{bmatrix} X_{\dot{u}} & 0 & 0 \\ 0 & Y_{\dot{v}} & Y_{\dot{r}} \\ 0 & Y_{\dot{r}} & N_{\dot{r}} \end{bmatrix}$$

$$C_A = \begin{bmatrix} 0 & 0 & Y_{\dot{v}}v + Y_{\dot{r}}r \\ 0 & 0 & -X_{\dot{u}}u \\ -Y_{\dot{v}}v - Y_{\dot{r}}r & X_{\dot{u}}u & 0 \end{bmatrix}, C_{RB} = \begin{bmatrix} 0 & 0 & -mv \\ 0 & 0 & mu \\ mv & -mu & 0 \end{bmatrix}$$

$$D_l = \begin{bmatrix} X_u & 0 & 0 \\ 0 & Y_v & Y_r \\ 0 & N_v & N_r \end{bmatrix}, D_n = \begin{bmatrix} X_{|u|u}|u| & 0 & 0 \\ 0 & Y_{|v|v}|v| & 0 \\ 0 & 0 & N_{|r|r}|r| \end{bmatrix} \tag{2.15}$$

where m is the mass and I_z is the moment of inertia of the vessel perpendicular to the horizontal plane, $\{X_{\dot{u}}, Y_{\dot{v}}, Y_{\dot{r}}, N_{\dot{r}}\}$, $\{X_u, Y_v, Y_r, N_v, N_r\}$ and $\{X_{|u|u}, Y_{|v|v}, N_{|r|r}\}$ are added mass, linear damping and nonlinear damping parameters, respectively. These predefined parameters are scalar constants that do not change over time.

CHAPTER 3

MPC-GRAPH: FEEDBACK MOTION PLANNING USING SPARSE SAMPLING BASED NEIGHBORHOOD GRAPH

Proposed MPC-Graph algorithm executes three successive phases to sample the arena and navigate the robot: graph generation, graph search and motion control. Graph generation phase takes the map as input and than samples the obstacle-free areas with overlapping rectangular regions until the predefined termination condition is satisfied. After the completion of the graph generation phase, Dijkstra's algorithm is executed to search the obtained neighborhood graph for the shortest available path from any node to the goal node. With the execution of the graph generation phase, the order of the nodes that robot should pass is determined. Motion control phase takes the determined nodes as input and navigates the robot to the goal configuration while respecting the constraints coming from the states and system inputs. In Fig. 3.1 block diagram representation of the proposed algorithm is presented.

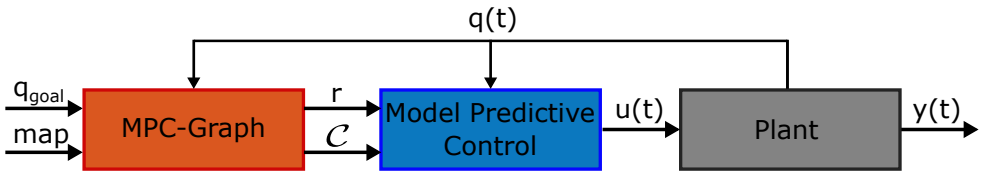


Figure 3.1: Block diagram of the algorithm. MPC-Graph generates a reference signal, r, to reach from current state, $q(t)$, to the goal location, q_{goal} considering the obstacles, goal and map limits. Model Predictive Control computes the optimal input, $u(t)$, to get $r(t)$ in accordance with the constraints on $q(t)$, $\mathcal{C}$.

3.1 Graph Generation

Graph generation phase starts with randomly sampling the obstacle-free space and then, nodes are generated around those samples until the defined termination condition is satisfied. As a similar approach in the work of Yang and Lavalle [3], they sampled the obstacle-free space with circular regions. Karagoz et al. [7] reports that using rectangular regions compared to circular ones highly reduces the number of nodes in the graph which results in a more sparse graph structure. In the work of Golbol et al. [5], they use square regions to cover the obstacle-free space for controller reusability. First, they create a tree-like structure and then use a reference governor based controller to navigate the robot. In this thesis work, due to implementation of MPC, we can directly reuse all convex constraint sets, so we construct a graph consisting of rectangular nodes with arbitrary aspect ratios which results in a more sparse graph structure.

The MPC-Graph framework starts its execution by sampling the free space. Let $\mathcal{B}$ be the collection of points encompassed by a node:

$$\mathcal{B} = \bigcup_{k} Node_k \tag{3.1}$$

After guaranteeing the sampled point q_{rand} is in the obstacle-free region, then a rectangular node $Node_k$ is generated and expanded around q_{rand}. Then, $Node_k$ is added to the graph and edges are created between this node and the neighboring nodes overlapping with it. Node generation and expansion process continues until the defined termination condition is satisfied. In the algorithm, we implemented the condition presented in [53] which mainly estimates the quality of the coverage of the sampled space. Let $\mathcal{C}_{free}$ denote the set of configurations in the obstacle free region and μ be de Lebesgue measure in $\mathcal{C}_{free}$, the implemented termination condition in (3.2), implicitly determines estimates for the expression $\mu(\mathcal{B})/\mu(\mathcal{C}_{free})$. For that purpose, statistics collected from randomly taking samples to find a new configuration in $\mu(\mathcal{C}_{free}) \setminus \mu(\mathcal{B})$ is analyzed. The implemented termination condition is formalized as follows,

$$m \geq \frac{\ln(1 - P_c)}{\ln\alpha} - 1 \tag{3.2}$$

where *m* is the number of successive failures followed by the first success, P_c and α

Algorithm 1 Graph Generation

1: **for** $k = 1$ to K **do**
2: **do**
3: $q_{rand} \leftarrow$ UniformRandConf()
4: **while** $q_{rand} \in WO$ and $q_{rand} \in \mathcal{B}$
5: $Node_k \leftarrow$ GenerateRectRegion(q_{rand})
6: $Node_k \leftarrow$ Expand($Node_k$)
7: G.InsertNode($Node_k$)
8: $\mathcal{B} \leftarrow \mathcal{B} \cup Node_k$
9: **if** $TerminationSatisfied(G, \alpha, P_c)$ **then**
10: Po = DijkstraAlgorithm(G)
11: return Po
12: **end if**
13: **end for**

are user determined parameters that effect the density of the coverage of the map. A more detailed information about the derivation of (3.2) is presented in [53]. After satisfying the termination condition, algorithm continues with graph search phase to find the optimal route. Algorithm 1 gives a detailed explanation of the graph generation phase.

3.1.1 Node Generation

In order to generate a node, first a random point q_{rand} is sampled in the obstacle free region. Then, from the given obstacle set the shortest distance between q_{rand} and the obstacles is calculated. Formally, let WO denote the obstacle set, WO_i be the i^{th} obstacle in the given map, q_{obs} be the point on the obstacle which is closest to the sampled point q_{rand},

$$WO = \bigcup_i WO_i \tag{3.3a}$$

$$q_{obs} = \arg\min_{q \in WO} \|q - q_{rand}\|. \tag{3.3b}$$

Let d_{min} denote the Euclidean distance between points q_{rand} and q_{obs}, $d_{min} = \|q_{rand} - q_{obs}\|$. By taking the calculated d_{min} value as the radius and q_{rand} as the center a

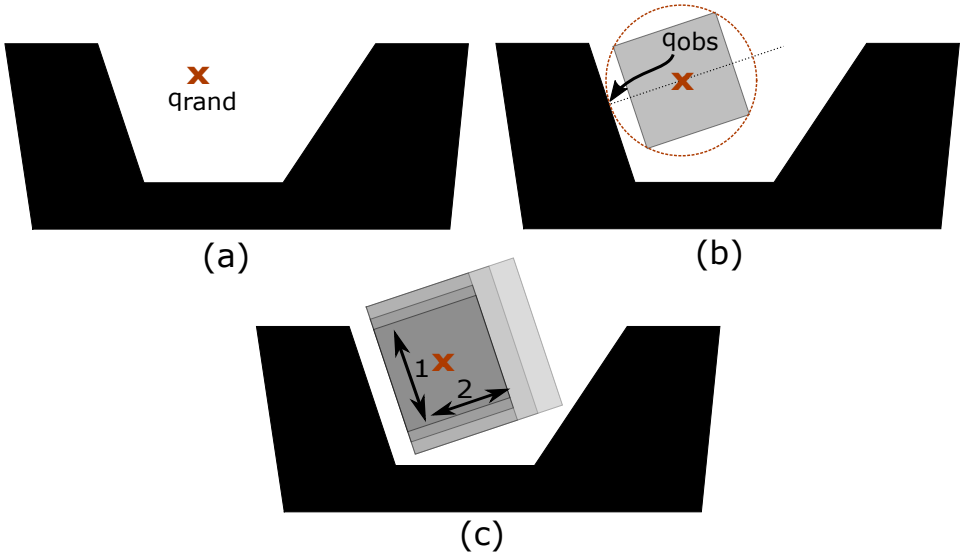


Figure 3.2: Generation of a node: (a) initial map of the arena, (b) a square node is generated, (c) square node is expanded in discrete steps along directions indicated as 1 and 2.

hypothetical circle is constructed. This procedure ensures that the generated circle is in the obstacle-free region, $\mathcal{C}_{circle}(q_{rand}, d_{min}) \subset \mathcal{C}_{free}$. After obtaining the circle, the largest square that can fit inside $\mathcal{C}_{circle}(q_{rand}, d_{min})$ with one edge perpendicular to the line segment connecting q_{rand} and q_{obs} is generated. Fig. 3.2a and 3.2b illustrates the whole node generation process.

3.1.2 Node Expansion

In order to cover larger areas and have a more sparse graph, we extend the previously generated square regions in directions 1 and 2 until they hit an obstacle or the limits of the arena as indicated in Fig. 3.2c. Another advantage provided by the larger nodes is that robot moves *faster* in larger regions. Since region boundaries are given as state constraints for the MPC, controller ensures that robot strictly stays inside the boundaries of the active node. In order to enforce this constraint, MPC slows down the robot as it approaches a region boundary. Therefore, in the presence of larger nodes robot stays longer in the same region; hence it moves at high speed for longer time.

In node expansion step, first the obtained square node extends in direction 1 in discrete time steps until it hits an obstacle or the limits of the arena as illustrated in Fig. 3.2c. If a collision occurs, the last expansion process is reverted. The same process is also repeated for direction 2. For the expansion process, the current edge length is multiplied with a constant factor γ which is set as γ =1.2 for this work.

3.2 Graph Search

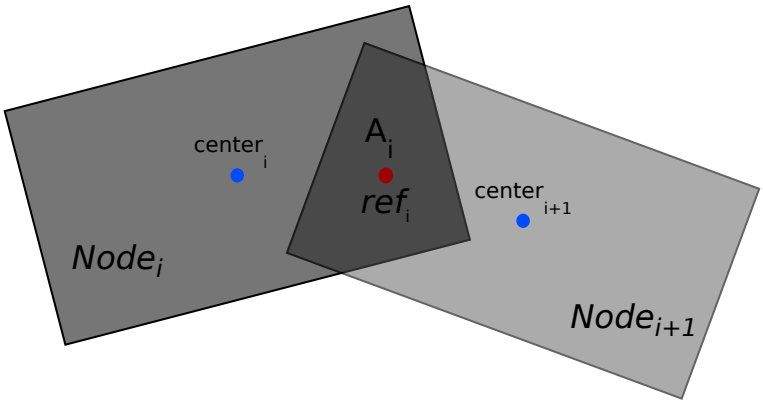


Figure 3.3: Visualization of edge cost parameters

After the termination condition given in eqn. (3.2) is satisfied, the algorithm continues with searching the obtained graph, *G*, with Dijkstra's search algorithm to find the optimal discrete planner. For each node, the policy returns the next node that robot should go.

The constructed graph has an edge between two nodes if they have an overlapping area. We calculate the edge cost as follows,

$$cost_{edge} = \|center_i - ref_i\|_2 + \|center_{i+1} - ref_i\|_2 + \frac{\alpha}{A_i} \tag{3.4}$$

where *center* is the intersection point of the diagonals of the corresponding node, *ref* is the center and *A* is the area of the intersection region. We include the reciprocal of the intersection area as a parameter for the edge cost since larger areas provide smoother behavior for the robot. In our simulations we used weight α for the reciprocal of the intersection area which is set as $\alpha = 1$. Thus, Dijkstra's search algorithm

chooses a route in favor of larger intersection areas. Fig. 3.3 visualizes the parameters given in 3.4.

3.3 Motion Control

The last phase of the proposed MPC-Graph algorithm is motion control. At this stage of the algorithm, with the previously determined policy *Po*, MPC navigates the robot from an arbitrary node to the *GoalNode* in discrete time steps. The aim of the motion control phase is to safely and smoothly navigating the robot from its *CurrentNode* to the *NextNode* with the established policy *Po* while forcing the state and input constraints. It is important to note that, as long as robot stays inside the sampled rectangular regions, it is guaranteed that collisions with obstacles are avoided. In order to simulate the physical limits for the robot, we implemented velocity and acceleration constraints. We adopted the quasi infinite MPC [28] approach to guarantee stability for systems whose Jacobian linearization is stabilizable.

The node that robot currently in is denoted as the $CurrentNode$ and the target node that robot is travelling to is called the $NextNode$. The reference point, *ref*, robot aiming to reach is chosen to be the centroid of intersection for that region, $[x_{ref}\ y_{ref}]^T = Centroid(CurrentNode \cap NextNode)$. After determining the centroid, by taking that point as the origin a target reference frame $\mathcal{T}$ is placed. In order to determine the orientation of that frame, a hypothetical vector ,$\overrightarrow{\mathbf{v}}$,starting from the previous reference point ref_{i-1} and ending at the target reference point ref_i is constructed. The angle, θ, between the world frame, $\mathcal{W}$, and $\overrightarrow{\mathbf{v}}$ is taken as the orientation for the target frame. Fig. 3.4 illustrates world $\mathcal{W}$, target $\mathcal{T}$ and robot frames $\mathcal{R}$.

After defining the necessary frames for the algorithm, the following transformation matrices are constructed,

$$T_t^w = \begin{bmatrix} cos\theta & -sin\theta & 0 & x_{ref} \\ sin\theta & cos\theta & 0 & y_{ref} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, T_r^w = \begin{bmatrix} cos\psi & -sin\psi & 0 & x \\ sin\psi & cos\psi & 0 & y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

where, $T_t^w \in SE(3)$ is the pose of the target frame with respect to world frame and

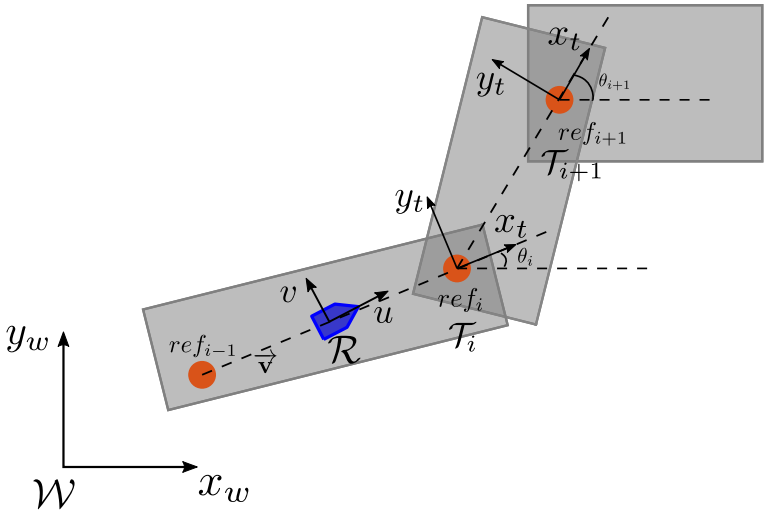


Figure 3.4: Representation of world $\mathcal{W}$, target $\mathcal{T}$ and robot frames $\mathcal{R}$

$T_r^w \in SE(3)$ is the pose of the robot frame with respect to world frame. By using the matrices given in (3.5), we calculate the pose of the robot frame with respect to target frame $T_r^t \in SE(3)$, as follows,

$$T_r^t = \left(T_t^w\right)^{-1} T_r^w. \tag{3.6}$$

By using the transformation matrix T_r^t, we calculate the position vector of the robot with respect to the target frame $\mathbf{q}_{tr} = [x_{tr}\ y_{tr}\ \psi_{tr}]^T$. In cost function (2.4), we use $\mathbf{q}_{tr}$ for robot states. This approach aims to overlap the target frame with body fixed robot frame. By using MPC, we calculate the optimal finite-horizon input sequence that satisfies the constraints and navigates the robot towards the origin of the target frame. The first element of this input sequence is applied to the robot.

When robot enters the intersection area, the $NextNode$ becomes the new $CurrentNode$ and the new $NextNode$ is determined by checking the next element in policy Po. It is important to note that for every new region robot enters, the state constraints coming from the boundaries of the rectangular nodes are re-calculated. This process executes recursively until the robot arrives at the node which includes the goal point q_{goal}. In the goal region, q_{goal} becomes the reference point and $NextNode$ is no longer applicable.

Due to unpredictable effects such as process noise, robot may end up in an unsampled region or in another node different than its $CurrentNode$. For the former case, the

last position of the robot is treated as the sampled random point, q_{rand}, and fed to the node generation function. This newly generated node is inserted to the previously obtained graph, G, and taken as the new $CurrentNode$ for the robot. Then, Dijsktra's search algorithm is executed in order to generate a new policy. These steps are given in lines 11-19 of Algorithm 2. Illustration of this resampling procedure is presented in Fig. 3.5.

For the latter case, the obtained graph is searched in order to determine the possible $CurrentNode$ which ensures the minimum cost route according to the equation (3.4). After determining the $CurrentNode$, by using the previously generated policy, Po, a new route is obtained. These steps are given in lines 21-23 of Algorithm 2. A complete procedure of the motion control phase is summarized in Algorithm 2.

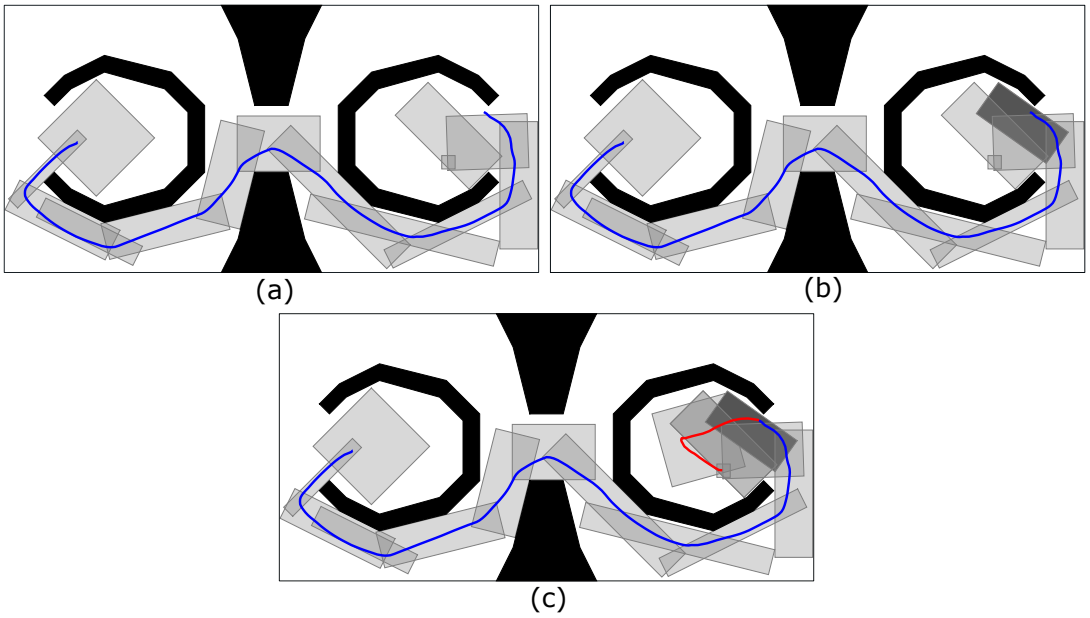


Figure 3.5: (a) Robot arrives to an unsampled region. (b) A new node is generated in the unsampled region which is indicated with dark grey color. (c) Robot follows the red route by using the newly created region.

Algorithm 2 Motion Control

$CurrentNode \leftarrow StartNode()$
$NextNode \leftarrow Po$.Next($CurrentNode$)
$ref \leftarrow$ Centroid($CurrentNode \cap NextNode$)
while q_{goal} **not** reached **do**
    if $q_t \in GoalNode$ **then**
        $ref \leftarrow q_{goal}$
    else if $q_t \in NextNode$ **then**
        $CurrentNode \leftarrow NextNode$
        $NextNode \leftarrow Po$.Next($CurrentNode$)
        $ref \leftarrow$ Centroid($CurrentNode \cap NextNode$)
    else if $q_t \in UnsampledRegion$ **then**
        $q_t \leftarrow$ UniformRandConf()
        $Node_i \leftarrow$ GenerateRectRegion(q_t)
        $Node_i \leftarrow$ Expand($Node_i$)
        G.InsertNode($Node_i$)
        Po = DijkstraAlgorithm(G)
        $CurrentNode \leftarrow Node_i$
        $NextNode \leftarrow Po$.Next($CurrentNode$)
        $ref \leftarrow$ Centroid($CurrentNode \cap NextNode$)
    else
        $CurrentNode \leftarrow$ SearchNodes(G, q_t)
        $NextNode \leftarrow Po$.Next($CurrentNode$)
        $ref \leftarrow$ Centroid($CurrentNode \cap NextNode$)
    end if
    $u_t \leftarrow$ MPC($q_t, ref, CurrentNode$)
end while

CHAPTER 4

IMPLEMENTATION

This section reports the simulation results obtained from the implementation of MPC-Graph algorithm. We implemented our algorithm on MATLAB and performed simulations on a laptop with Intel i7 2.4 GHz processor running Windows OS.

4.1 Robot Motion Models

We implemented MPC-Graph algorithm on four different robot motion models which include both linear and nonlinear systems: linear fully-actuated robot model, nonlinear fully-actuated robot model, fully actuated USV model and two thruster USV model.

4.1.1 Linear Fully-Actuated Robot Model

As a first approach, we implemented MPC-Graph algorithm to a fully actuated point model with double integrator plant dynamics for each axis. State vector of the robot is $\mathbf{q} = [x\ v_x\ y\ v_y]^T$ and the input vector is $\mathbf{u} = [u_x\ u_y]^T$. We assume that full state measurements are available in real-time. We discretize the continuous state-space dynamics under zero-older-hold operation and uniform synchronous sampling of measurements with a sampling frequency of $f_s = 20Hz$ (or $T_s = 0.05s$).

In our control policy, we used constrained MPC with dual mode approach. The finite horizon length is $N = 10$, which gives us enough degrees of freedom in enforcing the constraints while also satisfying the asymptotic stability.

The discrete time state-space model of the system has the following form:

$$\mathbf{q}_{t+1} = A\mathbf{q_t} + B\mathbf{u_t} \tag{4.1}$$

where A and B matrices are as follows :

$$A = \begin{bmatrix} 1 & 0.05 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.05 \\ 0 & 0 & 0 & 1 \end{bmatrix}, B = \begin{bmatrix} 0.0013 & 0 \\ 0.05 & 0 \\ 0 & 0.0013 \\ 0 & 0.05 \end{bmatrix} \tag{4.2}$$

$$f(x, y) \leq c \tag{4.3a}$$

$$-1 \leq v_x, v_y \leq 1 \tag{4.3b}$$

$$-1 \leq u_x, u_y \leq 1 \tag{4.3c}$$

The constraints are given in equation (4.3). Note that equation (4.3a) is calculated for each node on the path.

4.1.2 Non-linear Fully-Actuated Robot Model

We model the robot as a fully actuated acceleration driven holonomic model in the presence of non-linear quadratic friction force acting on each axis. The state vector and the input vector of the model takes the form $\mathbf{q} = [x\ y\ v_x\ v_y]^T$, and $\mathbf{u} = [u_x\ u_y]^T$ respectively. In this context, the equations of motion for the robot model is

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} -\lambda\ v_x^2 + u_x \\ -\lambda\ v_y^2 + u_y \end{bmatrix} \tag{4.4}$$

where $\lambda = 0.7$. We assume that full state measurements are available in real-time. We discretize the continuous nonlinear dynamics of the system and uniform synchronous sampling of measurements with a sampling frequency of $f_s = 10Hz$ (or $T_s = 0.1s$).

In our control policy, we used quasi-infinite horizon MPC. The finite horizon length is $T_p = 1.5s$, which gives us enough degrees of freedom in enforcing the state and input constraints. We choose the weighting matrices for the objective cost function

in (2.4) as an identity matrix, $Q = I_4$ and $R = I_2$. Then, by using the procedure presented in [28], Jacobian linearization given in (2.6) is calculated,

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}. \tag{4.5}$$

Then we obtain the following state feedback gain K, by using the Jacobian linearization and the cost matrices,

$$K = \begin{bmatrix} 1.00 & 0 & 1.73 & 0 \\ 0 & 1.00 & 0 & 1.73 \end{bmatrix}. \tag{4.6}$$

Then by using the obtained matrices, $A_K = A - BK$ is calculated as follows,

$$A_K = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & 0 & -1.7321 & 0 \\ 0 & -1 & 0 & -1.7321 \end{bmatrix}. \tag{4.7}$$

We choose κ as 0.816 by investigating the eigenvalues of A_K. After that we calculate $Q^\star$ by using (2.9),

$$Q^\star = \begin{bmatrix} 2 & 0 & 1.7321 & 0 \\ 0 & 2 & 0 & 1.7321 \\ 1.7321 & 0 & 4 & 0 \\ 0 & 1.7321 & 0 & 4 \end{bmatrix}. \tag{4.8}$$

Lastly, we obtain the following terminal penalty matrix P,

$$P = \begin{bmatrix} 185.28 & 0 & -152.19 & 0 \\ 0 & 185.28 & 0 & -152.19 \\ -152.19 & 0 & 168.32 & 0 \\ 0 & -152.19 & 0 & 168.32 \end{bmatrix}. \tag{4.9}$$

$$f(x, y) \leq c \tag{4.10a}$$

$$-1 \leq v_x, v_y \leq 1 \tag{4.10b}$$

$$-3 \leq u_x, u_y \leq 3 \tag{4.10c}$$

The constraints are given in equation (4.10). Note that equation (4.10a) is calculated for each node on the path.

4.1.3 Fully Actuated USV Model

For the fully actuated USV model, we implement the outlined model illustrated in Fig. 4.1 that consists of four thrusters which generate the indicated force vectors, F_1, F_2, F_3 and F_4. With this outlined model, the force vector $\boldsymbol{\tau}_{fa}$ takes the following form,

$$\boldsymbol{\tau}_{fa} = \begin{bmatrix} (F_1 + F_2 - F_3 - F_4)sin\alpha \\ (F_2 + F_4 - F_1 - F_3)cos\alpha \\ (F_2 + F_3 - F_1 - F_4)(sin\alpha + cos\alpha)b/2 \end{bmatrix}. \tag{4.11}$$

With the oriented placement of the thrusters with respect to the body, vessel can generate a force vector in direction v as opposed to the differential USV model. In the implementation, we adopted the parameters given in [1] for inertia, damping and added mass terms. The state vector and the input vector for the model takes the form $\mathbf{q} = [x\ y\ \theta\ u\ v\ r]^T$ and $\mathbf{u} = [F_1\ F_2\ F_3\ F_4]^T$, respectively.

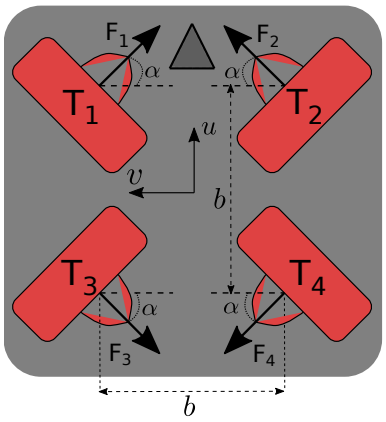


Figure 4.1: Placement of thrusters T_1, T_2, T_3 and T_4 for fully actuated USV model

In the simulations we discretize the continuous nonlinear dynamics of the system and uniform synchronous sampling of measurements with a sampling frequency of $f_s = 10\ Hz$ (or $T_s = 0.1s$). We navigate the vessel throughout the map with quasi-infinite horizon MPC. The finite horizon length is $T_p = 1.5s$, which gives us enough degrees of freedom in enforcing the state and input constraints. We choose the state

and input matrices, Q and R, for the objective cost function (2.4) as follows,

$$Q = \begin{bmatrix} 5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.2 \end{bmatrix}, R = \begin{bmatrix} 0.01 & 0 & 0 & 0 \\ 0 & 0.01 & 0 & 0 \\ 0 & 0 & 0.01 & 0 \\ 0 & 0 & 0 & 0.01 \end{bmatrix}. \tag{4.12}$$

After determining the cost matrices, by following the procedure presented in [28], Jacobian linearization given in (2.6) is calculated,

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -0.0776 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.1064 & 0.1274 \\ 0 & 0 & 0 & 0 & 0.2192 & -0.9809 \end{bmatrix}, \tag{4.13}$$

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.0637 & 0.0637 & -0.0637 & -0.0637 \\ -0.0405 & 0.0405 & -0.1342 & 0.1342 \\ -0.1805 & 0.1805 & 0.5408 & -0.5408 \end{bmatrix}. \tag{4.14}$$

Then we obtain the following state feedback gain K, by using the Jacobian linearization and the cost matrices,

$$K = \begin{bmatrix} 11.1803 & -13.1061 & -8.8448 & 11.9553 & -15.2518 & -4.4919 \\ 11.1803 & 13.1061 & 8.8448 & 11.9553 & 15.2518 & 4.4919 \\ -11.1803 & -8.8448 & 13.1061 & -11.9553 & -8.1447 & 3.9470 \\ -11.1803 & 8.8448 & -13.1061 & -11.9553 & 8.1447 & -3.9470 \end{bmatrix}. \tag{4.15}$$

Then by using the obtained matrices, $A_K = A - BK$ is calculated as follows,

$$A_K = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ -2.8503 & 0 & 0 & -3.1255 & 0 & 0 \\ 0 & -3.4361 & 2.8006 & 0 & -3.5285 & 0.8226 \\ 0 & 4.8341 & -17.3696 & 0 & 3.5213 & -6.8720 \end{bmatrix}. \quad (4.16)$$

We choose κ as 3.2529 by investigating the eigenvalues of A_K. After that we calculate $Q^\star$ by using 2.9,

$$Q^\star = \begin{bmatrix} 10 & 0 & 0 & 5.3466 & 0 & 0 \\ 0 & 10 & 0 & 0 & 5.4386 & 0.4792 \\ 0 & 0 & 10. & 0 & 0.5631 & 1.8292 \\ 5.3466 & 0 & 0 & 8.2172 & 0 & 0 \\ 0 & 5.4386 & 0.5631 & 0 & 8.4791 & 0.7272 \\ 0 & 0.4792 & 1.8292 & 0 & 0.7272 & 0.9151 \end{bmatrix}. \quad (4.17)$$

Lastly, we obtain the following terminal penalty matrix P,

$$P = \begin{bmatrix} -1.80 & 0 & 0 & 0.85 & 0 & 0 \\ 0 & -1.57 & -3.89 & 0 & 0.09 & 10.92 \\ 0 & -3.89 & 33.45 & 0 & 14.37 & -113.82 \\ 0.85 & 0 & 0 & -13.33 & 0 & 0 \\ 0 & 0.09 & 14.37 & 0 & -9.19 & -56.78 \\ 0 & 10.92 & -113.82 & 0 & -56.78 & 505.74 \end{bmatrix}. \quad (4.18)$$

We implemented the following state and input constraints to the system,

$$f(x, y) \leq c \quad (4.19a)$$

$$-1\ rad/s \leq r \leq 1\ rad/s \quad (4.19b)$$

$$-1.5\ m/s \leq u \leq 1.5\ m/s \quad (4.19c)$$

$$-20\ N \leq F_1, F_2, F_3, F_4 \leq 20\ N \quad (4.19d)$$

Equation (4.19a) is calculated for each node on the path. Table 4.1 presents the parameters used for the fully actuated USV model.

Table 4.1: Fully Actuated USV Parameters [1]

m	$10\ kg$	Y_v	-3.5059	$X_{\dot{u}}$	-1.0946	$N_{\|r\|r}$	-0.2605
I_z	0.6487	$Y_{\|v\|v}$	-3.9310	$Y_{\dot{v}}$	-1.0536	α	$45°$
X_u	-5.6128	Y_r	-0.0001	$Y_{\dot{r}}$	-1.4353	b	$0.26\ m$
$X_{\|u\|u}$	-2.3136	N_v	-0.0001	$N_{\dot{v}}$	-1.4353	$N_{\dot{r}}$	-0.1039

4.1.4 Under Actuated USV Model

For the differential USV model with two thrusters, we implement the outlined model illustrated in Fig. 4.2. With this outlined model, the force vector $\boldsymbol{\tau}_{ua}$ takes the following form,

$$\boldsymbol{\tau}_{ua} = \begin{bmatrix} F_1 + F_2 \\ 0 \\ b(F_2 - F_1) \end{bmatrix}. \tag{4.20}$$

It is important to note that the placement of thrusters prevents the generation of a force vector in v direction. In the implementation, we adopted the parameters given in [2] for inertia, damping and added mass terms. The state vector and the input vector for the model takes the form $\mathbf{q} = [x\ y\ \theta\ u\ v\ r]^T$ and $\mathbf{u} = [F_1\ F_2]^T$, respectively.

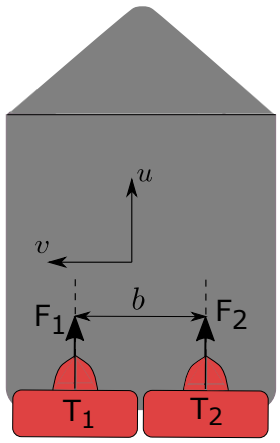


Figure 4.2: Placement of thrusters T_1 and T_2 for underactuated USV model

In the simulations we discretize the continuous nonlinear dynamics of the system and uniform synchronous sampling of measurements with a sampling frequency of

Table 4.2: Differential USV Parameters [2]

m	$23.8\ kg$	Y_v	-0.8612	$X_{\dot{u}}$	-2.0	$N_{\|r\|r}$	0.2605
I_z	1.76	$Y_{\|v\|v}$	-36.2810	$Y_{\dot{v}}$	-10	$N_{\dot{r}}$	-1.0
X_u	-0.7225	Y_r	0.1079	$Y_{\dot{r}}$	0		
$X_{\|u\|u}$	-1.3274	N_v	0.1052	$N_{\dot{v}}$	0		

$f_s = 10Hz$ (or $T_s = 0.1s$). For this system it is important to note that the linearized system at the origin is not stabilizable so terminal region and terminal cost matrix cannot be calculated. In order to control the system and predict its future behavior, we selected the finite horizon length as $T_p = 9s$. Note that it is 6 times larger than the horizon length used for fully actuated USV model. We choose the state and input matrices, Q and R, for the objective cost function 2.4 as follows,

$$Q = \begin{bmatrix} 5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.2 \end{bmatrix}, R = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.01 \end{bmatrix}. \tag{4.21}$$

We implemented the following state and input constraints to the system,

$$f(x, y) \leq c \tag{4.22a}$$

$$-2\ rad/s \leq r \leq 2\ rad/s \tag{4.22b}$$

$$-0.5\ m/s \leq u \leq 3\ m/s \tag{4.22c}$$

$$-10\ N \leq F_1, F_2 \leq 20\ N \tag{4.22d}$$

Equation 4.22a is calculated for each node on the path. Table 4.2 presents the parameters used for the differential USV model.

4.2 CASE 1: Sampling-based Neighborhood Graph and MPC-Graph Comparison

We performed a comparative analysis for the node generation stage of SNG and MPC-Graph algorithms. We implemented the SNG algorithm with the enhancement presented in [53].To provide a fair comparison, we performed Monte-Carlo experiments on 4 different maps (# simulations = 1000, for each map), as illustrated in Fig. 4.3. We compared the algorithms in terms of sparsity and computational efficiency. Table 4.3 reports the Monte-Carlo simulation results.

In our simulations, we used the termination condition given in (3.2) and conducted the experiments with $\alpha = 0.95$ and $P_c = 0.95$. These values give satisfactory results in terms of map coverage and computational time. We adopted Map 1 from Yang and LaValle [53], which is a simple map composed of three polygonal obstacles. In this map MPC-Graph algorithm generates a more sparse graph compared to SNG algorithm ($\sim \%55$ reduction in $\#$ nodes), whereas the SNG creates its random map faster than the MPC-Graph algorithm ($\sim \%25$ faster). In addition, we tested both approaches on a more complicated map, Map 2 Fig. 4.3, composed of four polygonal obstacles with a relatively narrow path. In this scenario, while computation times are comparable to each other (SNG is $\%10$ faster than the MPC-Grapha), MPC-Graph illustrates a remarkable sparsity performance ($\sim \%63$ reduction in $\#$ nodes). Furthermore, in the sampling of the narrow path, MPC-Graph algorithm obtains connected samples, whereas with the same α and P_c values SNG algorithm is unable to obtain a connected structure.

In order to provide a fair comparison, we also tested two other maps, Map 3 and Map 4, that are composed of curved (circular, elliptic etc.) obstacles and boundary (in Map 4). Qualitatively, the results are similar to the Maps 1 $\&$ 2, in the sense that the MPC-Graph algorithm generates more sparse graphs with the added cost of computational efficiency. We believe that offline computational performances of both methods are comparable, and both techniques provide possible times for real applications. Thus, due to the sparsity performance of our approach, the MPC-Graph method could be beneficial and powerful in different robotic applications.

Table 4.3: Comparative Results of Node Generation

Map	Algorithm	# Nodes	CPU Time
Map 1	SNG	90.85	0.14
	MPC-Graph	41.70	0.19
Map 2	SNG	125.60	0.17
	MPC-Graph	46.56	0.19
Map 3	SNG	199.51	0.82
	MPC-Graph	133.53	1.05
Map 4	SNG	138.63	1.12
	MPC-Graph	117.35	1.35

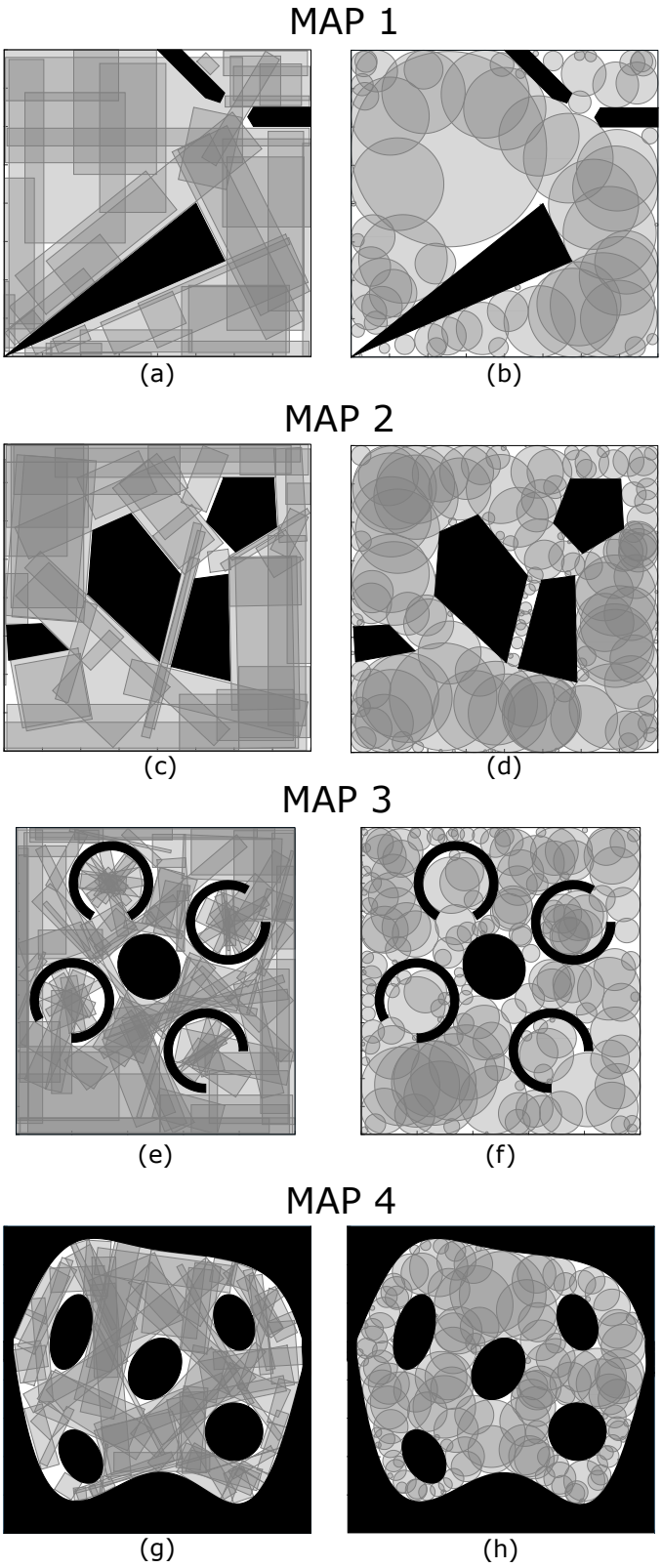


Figure 4.3: On the different maps two different sampling based algorithms are tested. The proposed MPC-Graph algorithm generates 49, 51, 129 and 102 rectangular nodes in (a), (c), (e) and (g) respectively. SNG algorithm resulted in generating 78, 155, 192 and 148 circular nodes in (b), (d), (f) and (h) respectively.

4.3 CASE 2: Simulation Results for Linear Fully-Actuated Robot Model

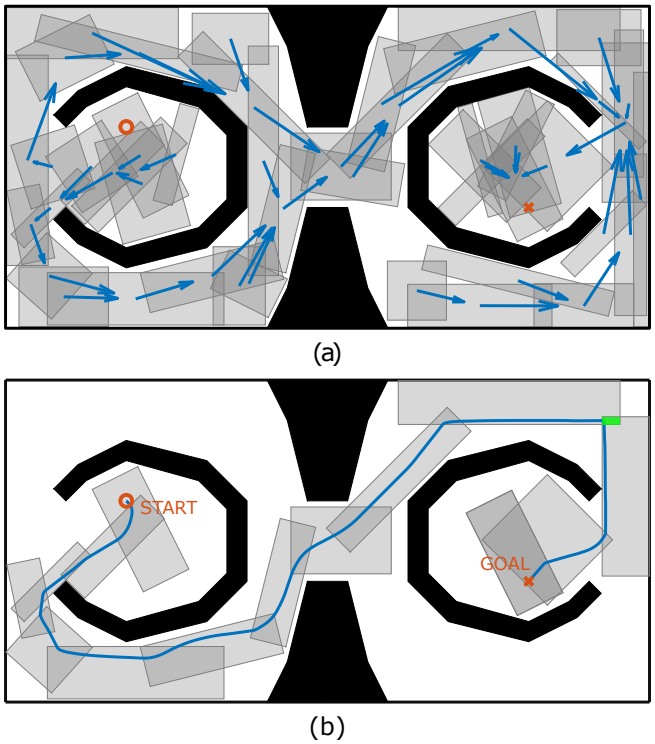
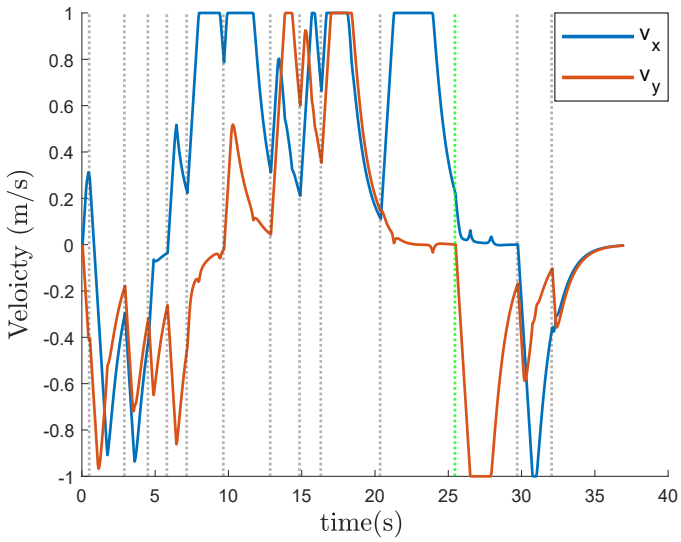


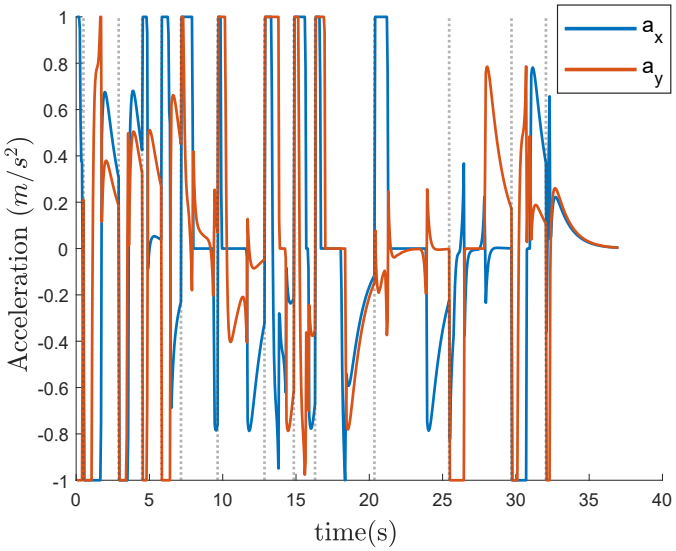
Figure 4.4: (a) Representation of how rectangular regions are connected. Blue arrow indicates the node in the policy. (b) Execution of the algorithm. Blue line shows the trajectory that followed by the robot. Note that figure shows only the nodes that are in the path.

The results presented in this section uses the linear fully actuated robot motion model given in Section 4.1.1. For illustration, we used a map which has a rectangular shape with $16m \times 8m$ dimensions and U-shaped obstacles. The graph structure and optimal policy generated by graph search stage can be seen on Fig 4.4(a). Trajectory followed by the robot is given in Fig. 4.4(b). Velocity and acceleration plots of the robot during motion control stage are given in Fig 4.5. These figures show that MPC successfully enforces the constraints on states, q and inputs, u. CPU time of MPC for each iteration for this model is at average $t_{CPU} = 0.01s$. Fig. 4.5 also illustrates that if the intersection of two consecutive regions is small, the robot gets close to ref, and hence slows down before entering the next region. This case is illustrated in Fig. 4.4(b) as a green region on top-right corner, and correspondingly in Fig. 4.5(a) just before the

green dotted line.



(a)



(b)

Figure 4.5: Velocity (a) and acceleration (b) of the robot for the simulation represented in Fig. 4.4. Grey vertical dotted lines show where reference, ref, is changed. Green vertical dotted line corresponds to green region in the upper right corner in Fig. 4.4(b)

Since MPC-Graph algorithm calculates the policy from all nodes to the goal, if the robot finds itself in another node other than it is supposed to be in, instead of re-planning, it can continue by following the policy of this new node. To illustrate this

situation while robot is passing through the green dot in Fig. 4.6, we instantaneously moved it to the purple dot. As seen from the figure, the robot could reach the goal configuration using the same policy but following another path.

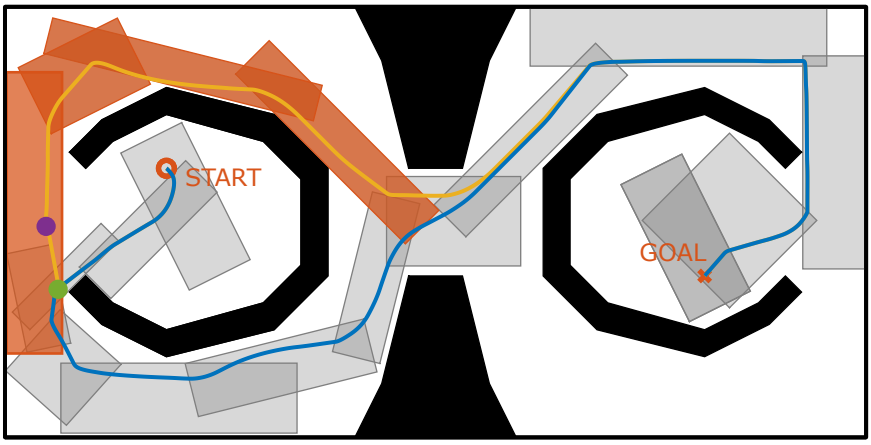
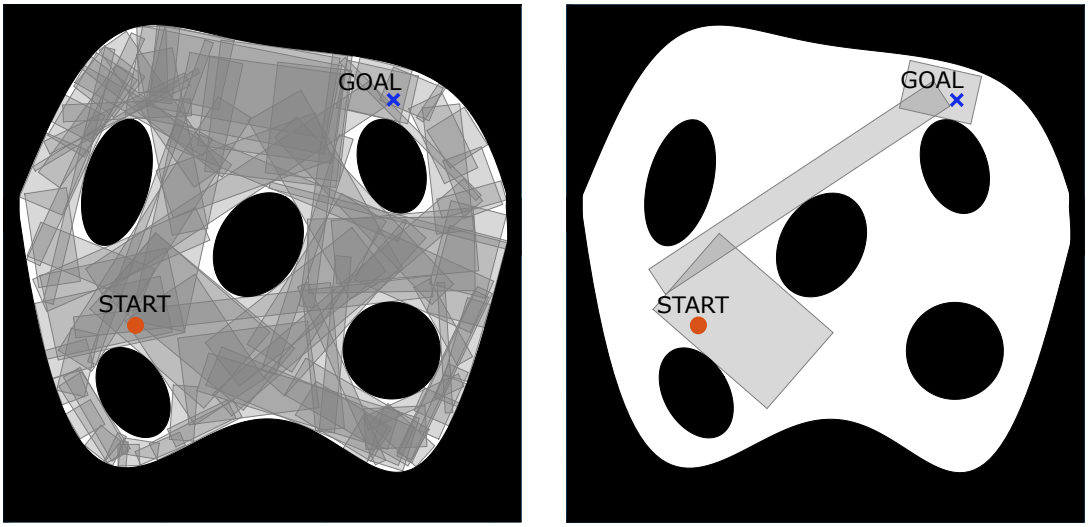


Figure 4.6: Instantaneous disturbance is applied when robot arrives the location represented by a green dot. As a result new robot location becomes the purple dot. New trajectory followed by the robot is indicated with the yellow curve.

4.4 CASE 3: Simulation Results for Nonlinear Fully Actuated Robot Model

The results presented in this section uses the robot motion model given in Section 4.1.2. For the following results, we used MAP 4 provided in Fig. 4.3. First the obstacle-free region is sampled with rectangular nodes until the termination condition is satisfied as given in Fig. 4.7(a). As a result of the sampling phase, algorithm generates 99 nodes. Lines 1-8 in Algorithm 1 states this procedure. Then, Dijkstra's search algorithm executes with previously determined edge cost parameters and calculates an 'optimal' policy *Po* as illustrated in Fig. 4.7(b). Lines 9-13 in Algorithm 1 stands for this procedure.

After the completion of graph generation phase, motion control phase executes. MPC navigates the robot from the start location to the goal location while forcing the constraints given in equation (4.10). The trajectory followed by the robot and the calculated reference points are visualized in Fig. 4.8. As can be seen from Fig. 4.8, MPC successfully forces the constraints coming from the boundaries of the rect-



(a) (b)

Figure 4.7: (a) Node generation phase. As a result 114 nodes are generated. (b) Graph search phase determines the shortest route consisting of 3 nodes.

angular nodes. CPU time of MPC for each iteration for this model is at average $t_{CPU} = 0.08s$.

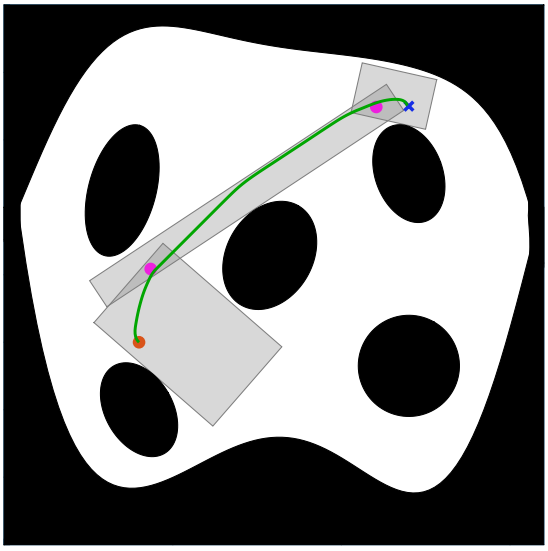


Figure 4.8: Green curve indicates the trajectory followed by the robot. Pink points represent the calculated reference points in Algorithm 2.

Fig. 4.9 shows the velocity and acceleration inputs of the robot throughout the simulation scenario presented in Fig. 4.8. As stated previously, the robot has velocity

and input limits of $1m/s$ and $3m/s^2$, respectively which are indicated as dark dashed lines in the figure. Plots presented in Fig.4.9 conclude that MPC can force state and input constraints imposed on the system.

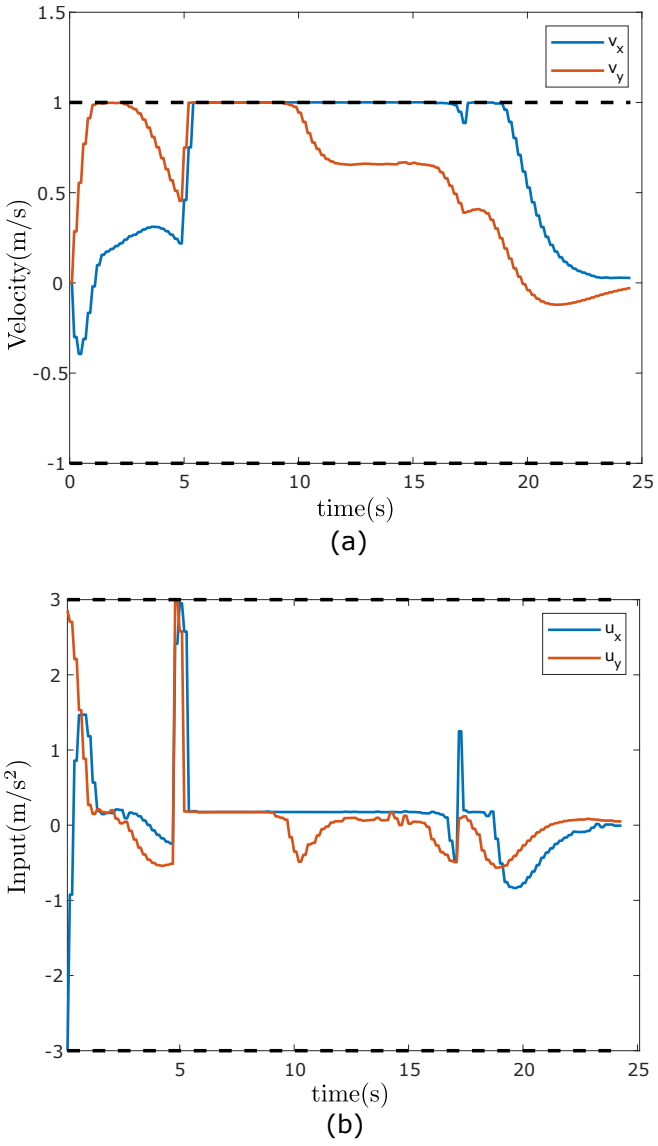


Figure 4.9: Velocity (a) and acceleration input (b) of the robot for the simulation presented in Fig. 4.8. Dark dashed lines represent the constraints implemented on velocity and input values.

4.5 CASE 4: Simulation Results for Fully Actuated USV Model

This section reports the results of the proposed MPC-Graph algorithm applied on the robot motion model given in Section 4.1.3.

4.5.1 Performance Without Process Noise

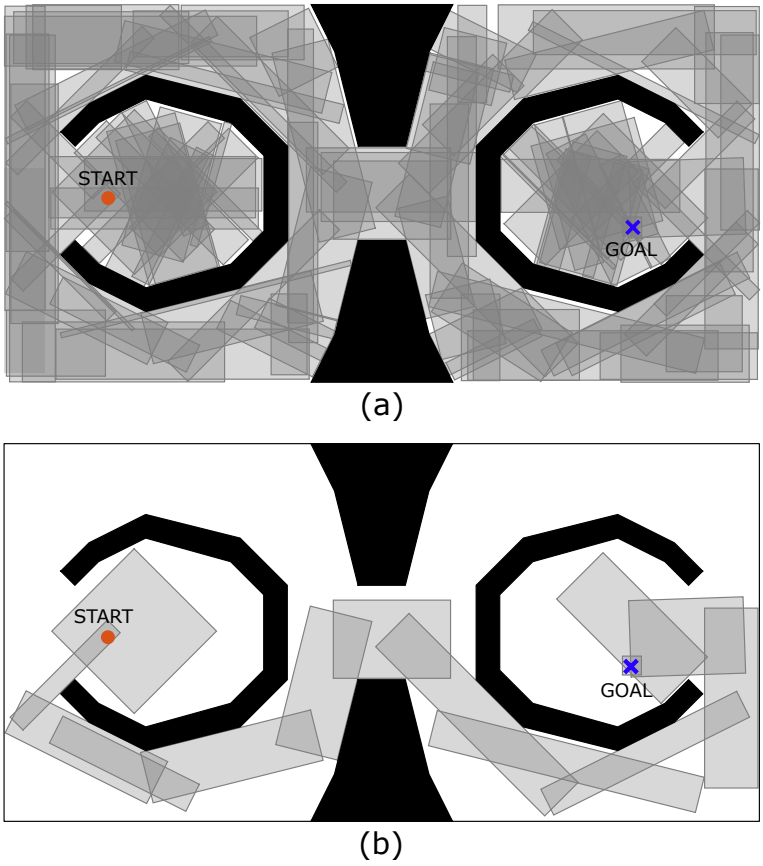


Figure 4.10: (a) Node generation phase. As a result 114 nodes are generated. (b) The calculated optimal route consists of 14 nodes.

In this simulation scenario, we used a map that consists of 4 obstacles. In the graph generation phase a total of 114 nodes are generated. In order to show the ability of the controller to force the constraints coming from boundaries of the rectangular nodes, we replaced the goal node with a smaller square region. With this approach we also aim to show the docking performance of the algorithm. After the execution of Dijkstra's search algorithm the optimal route consists of 14 nodes. Fig. 4.10(a)

and (b) visualizes the obtained rectangular regions and the node set that constitutes the followed route by the robot, respectively.

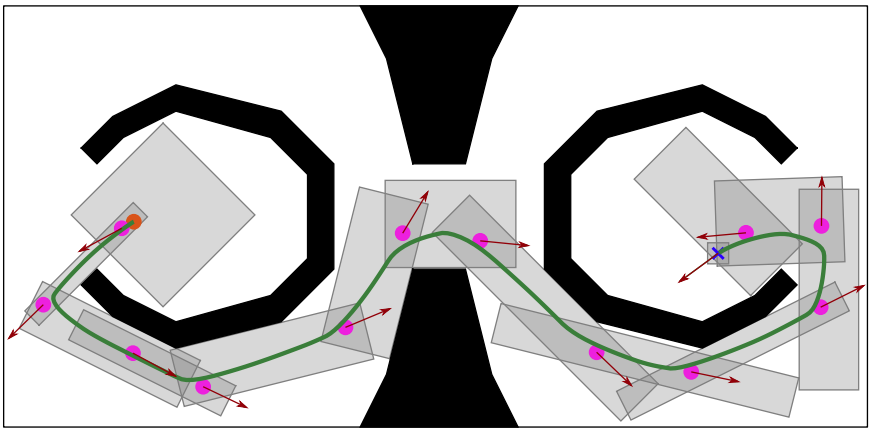


Figure 4.11: Green curve indicates the trajectory followed by the robot. Pink points represent the calculated reference points. Red arrows show the calculated target frame $\mathcal{T}$ orientations in positive x_t direction.

Fig. 4.11 shows the route followed by the robot and the calculated reference points with target orientations. Red arrows are the positive x_t directions for the target frames $\mathcal{T}$. From the figure it can be concluded that robot obeys the constraints coming from the boundaries of the sampled rectangular regions. Even in the smaller goal region ($\sim$ 24 times smaller with respect to the average size of the nodes in *Po*) robot approaches to goal point smoothly by considering the dynamics and the imposed constraints on the system.

Plot presented in Fig. 4.12(a) shows the forces F_1, F_2, F_3, F_4 calculated in the motion control phase. In order to simulate a realistic system, we set upper and lower bounds $20N$ and $-20N$ for the input forces, respectively. Dashed black lines in the figure indicate these constraints. Furthermore, we also added velocity constraints on the system. For the surge speed u, we set upper and lower constraints as $1.5m/s$ and $-1.5m/s$, respectively. For the angular rate r, we set upper and lower constraints as $1rad/s$ and $-1rad/s$, respectively. In Fig. 4.12(b) the imposed constraints on surge speed and angular rate are indicated with black and green dashed lines, respectively. It can be inferred from the figures that MPC can force both state and input constraints successfully.

CPU time of MPC for each iteration for this model is at average $t_{CPU} = 0.32s$. In Fig. 4.11 it can be seen that start location is close to the first reference point, thus initial cost value is approximately zero. Since robot starts its motion from the intersection area of the first two nodes, the second reference point becomes the next target and as a result cost value increases instantaneously at the start of the simulation.

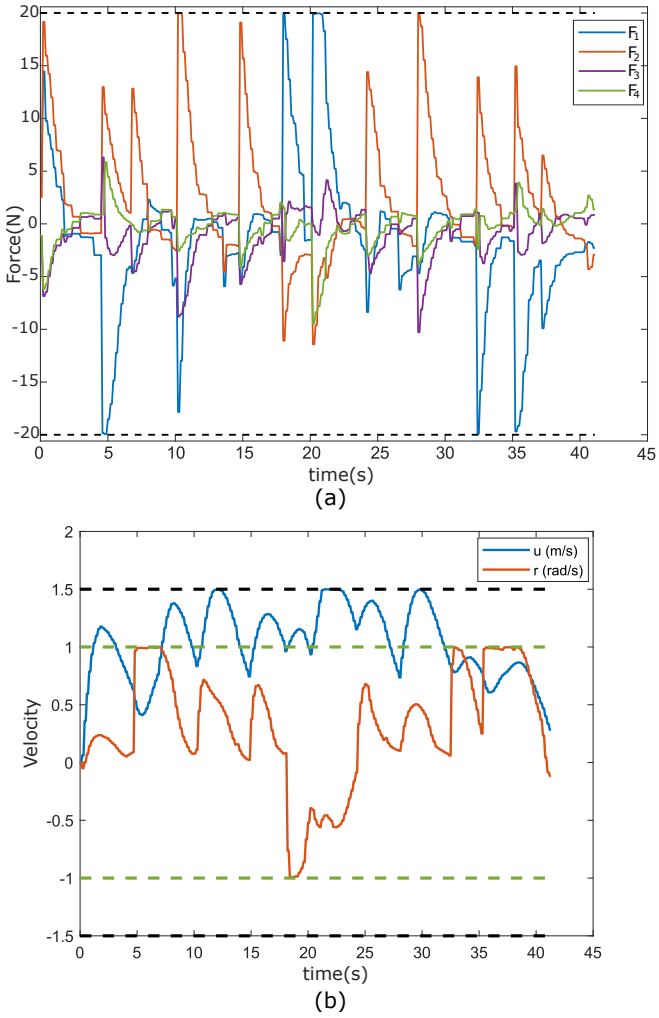


Figure 4.12: (a) Applied thruster input forces F_1, F_2, F_3, F_4 to the system. Dark dashed lines indicate upper and lower constraints for the input. (b) Surge speed u and angular rate r of the robot for the simulation presented in Fig. 4.11. Black and green dashed lines correspond to the constraints for surge speed and angular rate, respectively.

4.5.2 Performance in the Presence of Process Noise

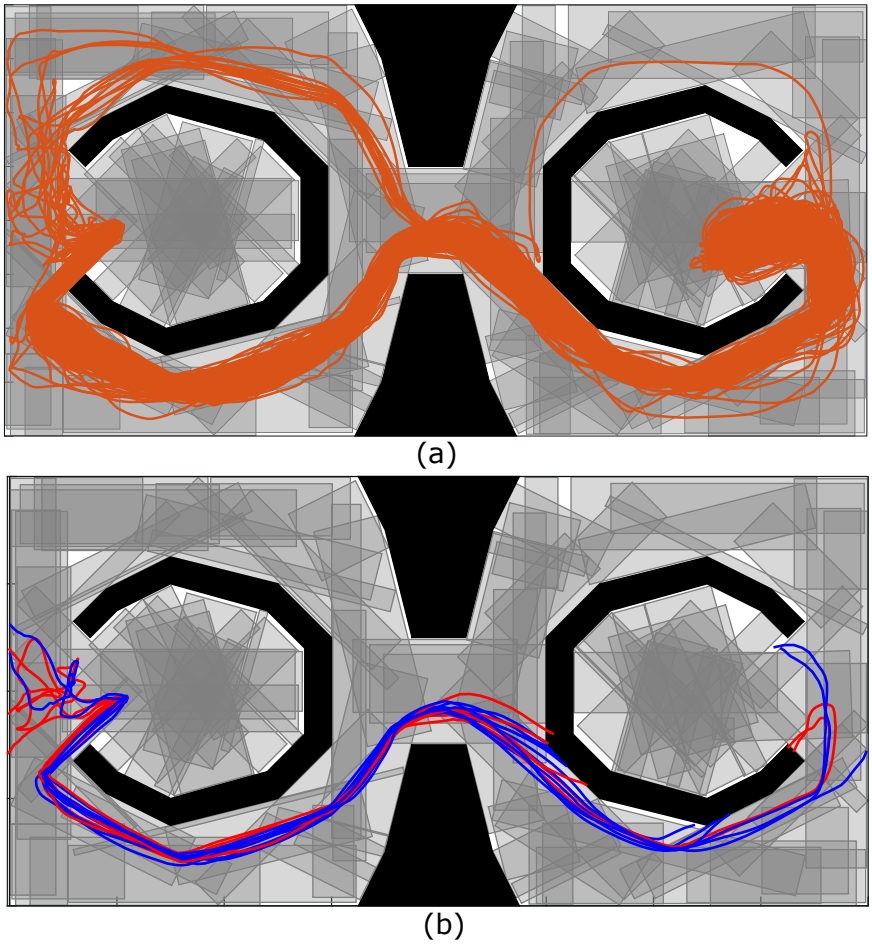


Figure 4.13: Obtained trajectories from Monte-Carlo experiments in the presence of process noise. (a) Successful trajectories ended up in goal point. (b) Red trajectories resulted in a collision with the obstacles or arrival to the outside the limits of the arena. Blue trajectories resulted in arrival of the robot to unsampled regions.

In order to analyze the performance of MPC-Graph algorithm in the presence of process noise, we added random noise (SNR=1) to the calculated thruster forces, $u = [F_1\ F_2\ F_3\ F_4]^T$. We performed Monte-Carlo experiments (# simulations = 1000) on the same map with the same nodes and plot the trajectories followed by the robot. We determine an input saturation limit of $F = 25N$ for the thrusters. From 1000 Monte-Carlo experiments, in 20 of them ($= \%\ 2$ of the simulations) robot failed to reach goal location due to process noise. Failures include the cases in which robot ends up in an unsampled region, outside the limits of the arena or inside the obsta-

cles. From 1000 Monte-Carlo experiments in 12 of them (= % 1.2 of the simulations) robot ended up in an unsampled region and in 8 of them (= % 0.8 of the simulations) robot had a collision with an obstacle or ended up outside the limits of the arena. Fig. 4.13(a) and (b) visualizes the successful and failed attempts for reaching the goal point, respectively.

If robot ends up in a different node than the $CurrentNode$ due to process noise, algorithm generates another route taking into account the previously determined policy Po. Note that policy Po includes all possible 'optimal' paths starting from any arbitrary node to the goal node. To have a fair comparison with the case that does not include process noise, we used the same map with the same rectangular nodes.

From 12 experiments in which robot ended up in an unsampled region, with the implementation of resampling procedure robot can recover from 4 of the failure cases and reach the goal location. In Fig. 4.14 red routes show the trajectories followed by the robot and darker nodes represent the newly generated nodes after resampling procedure.

In Fig. 4.15(a) red and dashed blue trajectories indicate the routes followed by the robot in the presence of process noise and without noise, respectively. Fig. 4.15(b) shows the thruster forces and process noise applied on each thruster. Even in the presence of high level noise, in $\%98.4$ of the experiments fully actuated USV model reaches the goal location. Obtained results indicate that MPC is an effective controller for handling the high noise scenarios.

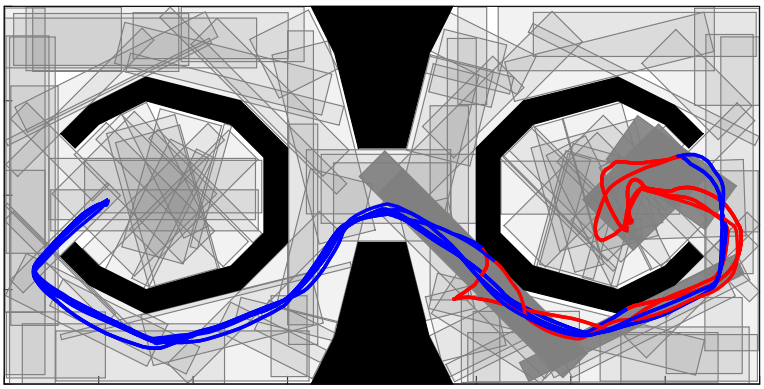
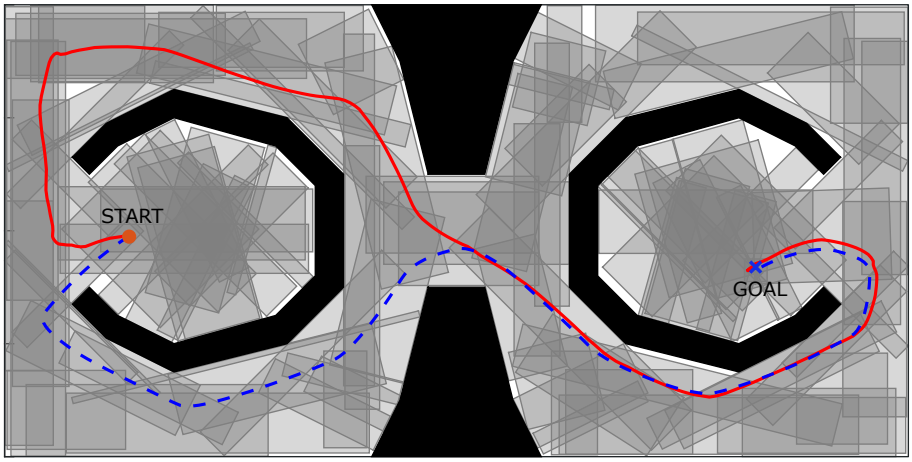
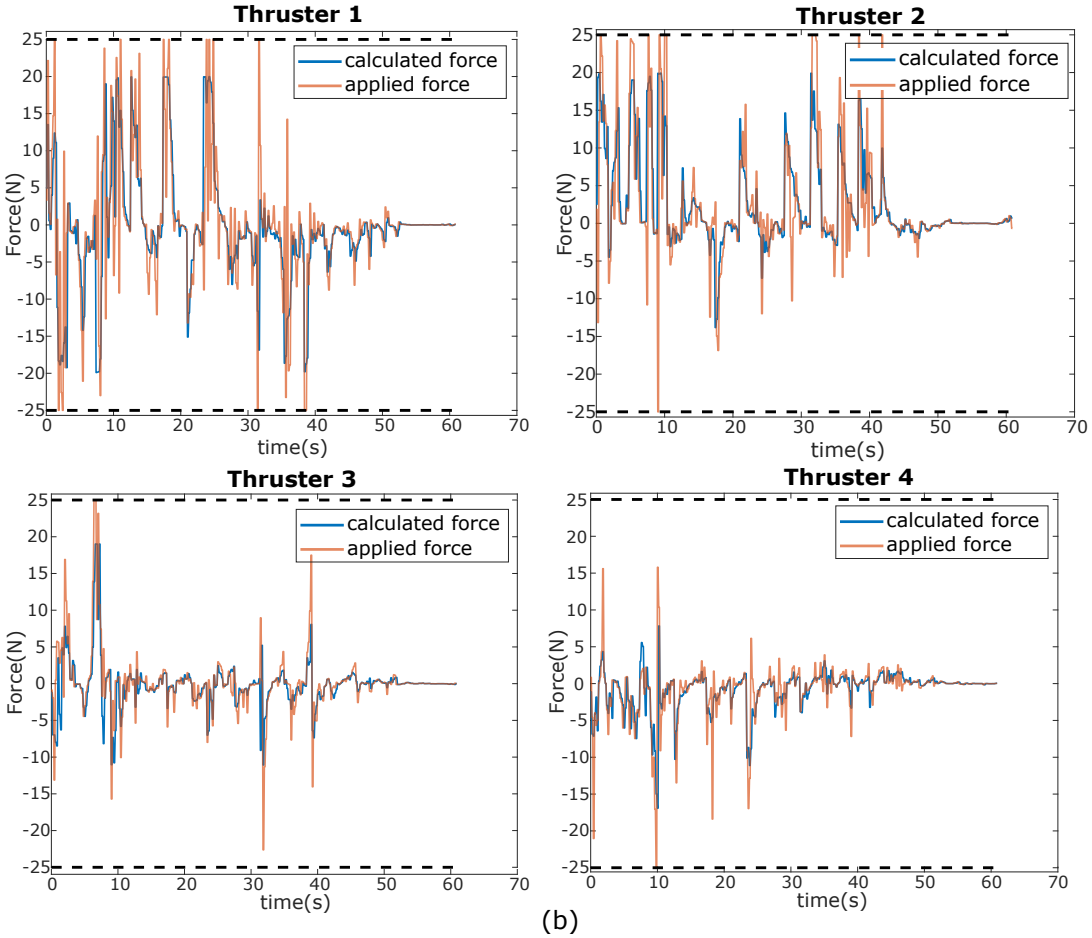


Figure 4.14: Red curves indicate the routes followed by the robot and darker nodes represent the newly generated nodes after resampling procedure.



(a)



(b)

Figure 4.15: (a) Red trajectory indicates the followed route in the presence of process noise. Dashed blue trajectory indicates the followed trajectory without the process noise. (b) Plots show the calculated forces and applied forces on each thruster. Black dashed lines indicate the saturation limits for the thrusters.

4.6 CASE 5: Simulation Results for Differential USV Model

This section reports the results of the proposed MPC-Graph algorithm applied on the robot motion model given in Section 4.1.4.

4.6.1 Performance Without Process Noise

In this simulation scenario, we used a map that resembles a parking area. In the graph generation phase a total of 75 nodes are generated. After the execution of Dijkstra's search algorithm the 'optimal' route consists of 8 nodes. Fig. 4.16(a) and (b) visualizes the obtained rectangular regions and the node set that robot navigates in, respectively.

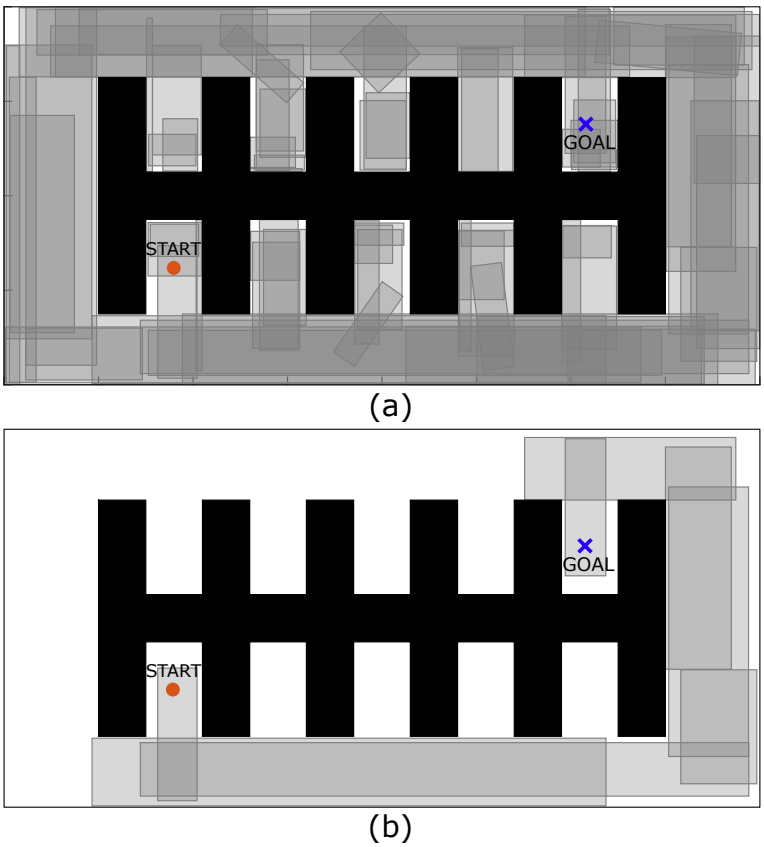


Figure 4.16: (a) Node generation phase. As a result 75 nodes are generated. (b) Graph search phase determines the shortest route consisting of 8 nodes.

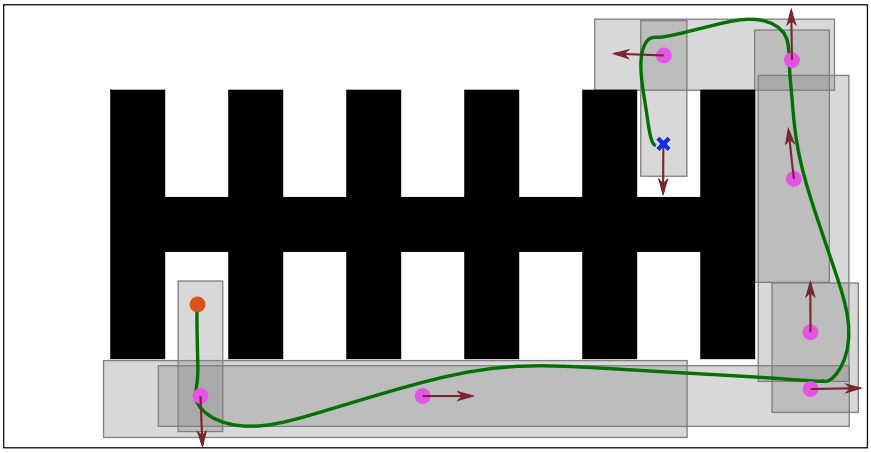


Figure 4.17: Green curve indicates the trajectory followed by the robot. Pink points represent the calculated reference points. Red arrows show the calculated target frame $\mathcal{T}$ orientations in positive x_t direction.

Fig. 4.17 illustrates the path followed by the robot and calculated reference points with target orientations. Red arrows are the positive x_t directions for the target frames $\mathcal{T}$. From the figure it can be inferred that robot obeys the constraints coming from the boundaries of the rectangular areas.

Plot presented in Fig. 4.18(a) shows the forces F_1, F_2 calculated in the motion control phase. In order to simulate a realistic system, we set upper and lower bounds $20N$ and $-10N$ for the input forces, respectively. In order to restrict the backward motion of the vehicle, we set the lower limit for the thruster forces to be $-10N$. Dashed black lines in the figure indicate these constraints. Furthermore, we also added velocity constraints on the system. For the surge speed u, we set upper and lower constraints as $3m/s$ and $-0.5m/s$, respectively. For the angular rate r, we set upper and lower constraints as $2rad/s$ and $-2rad/s$, respectively. In Fig. 4.18(b) the imposed constraints on surge speed and angular rate are indicated with black and green dashed lines, respectively. It is important to note that, linearized differential USV model is not stabilizable thus it is not possible to determine a terminal region and a terminal cost matrix for this system. Although the stability is not guaranteed for differential USV model, with the adjusted MPC parameters, controller can drive the system to goal location while obeying the imposed constraints. CPU time of MPC for each iteration for this model is at average $t_{CPU} = 2.04s$.

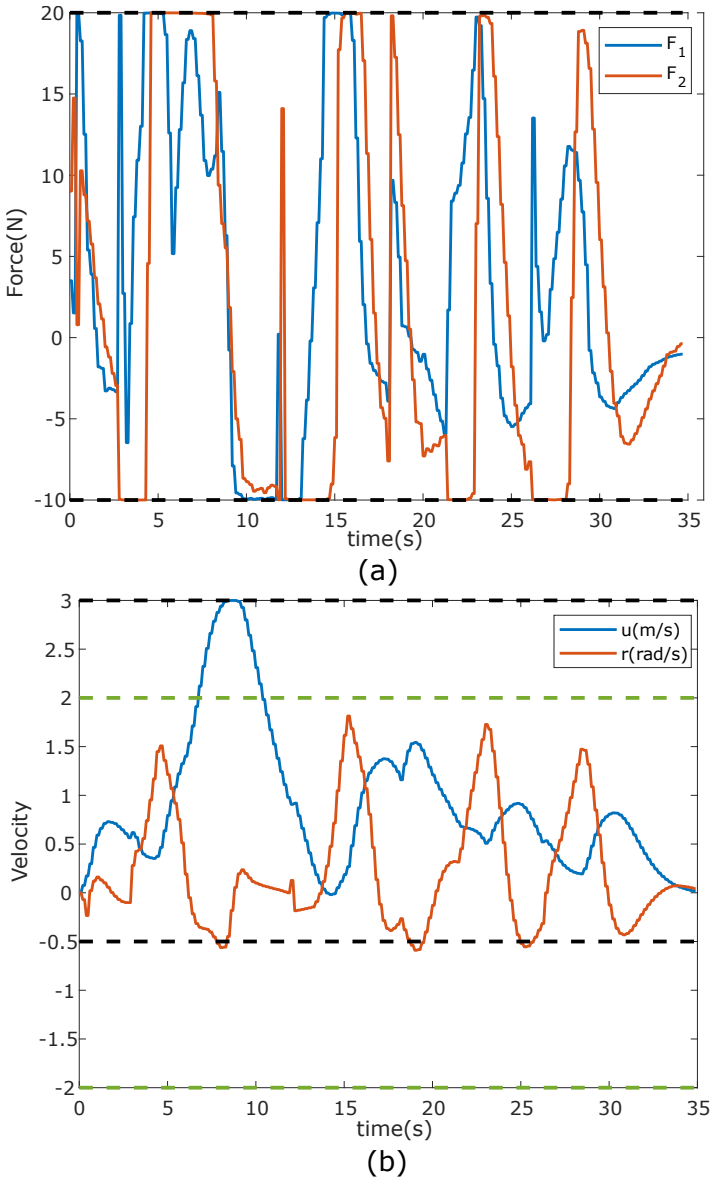


Figure 4.18: (a) Applied thruster input forces F_1, F_2 to the system. Dark dashed lines indicate upper and lower constraints for the input. (b) Surge speed u and angular rate r of the robot for the simulation presented in Fig. 4.17. Black and green dashed lines correspond to the constraints for surge speed and angular rate, respectively.

4.6.2 Performance in the Presence Process Noise

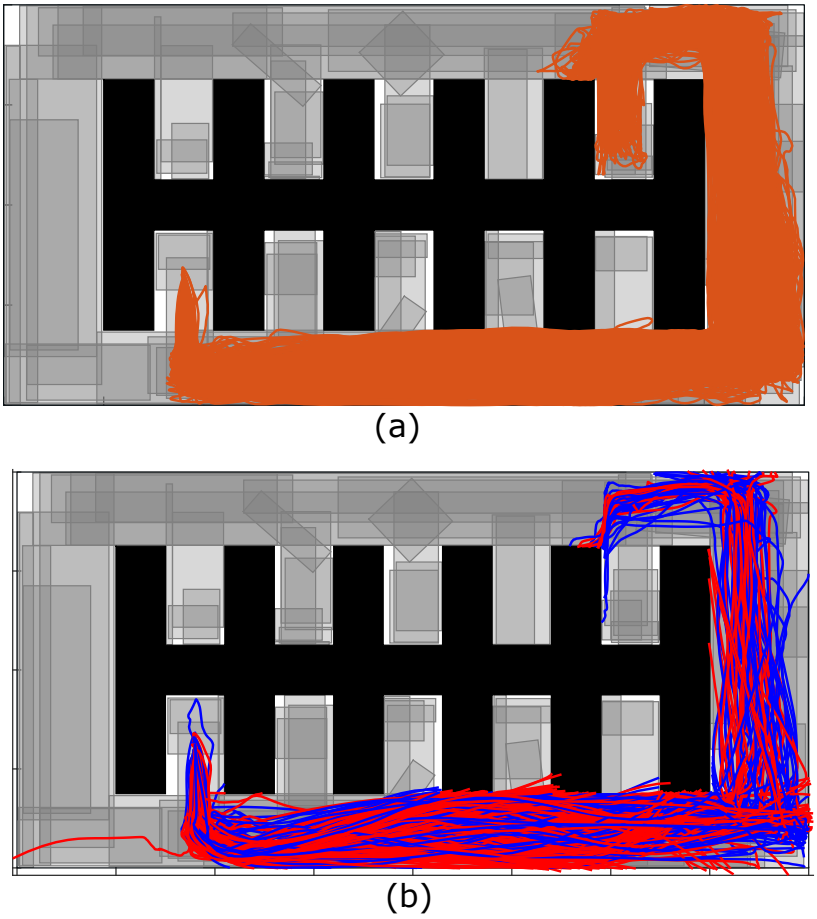


Figure 4.19: Obtained trajectories from Monte-Carlo experiments in the presence of process noise. (a) Successful trajectories ended up in goal point. (b) Red trajectories resulted in a collision with the obstacles or arrival to the outside the limits of the arena. Blue trajectories resulted in arrival of the robot to unsampled regions.

In order to analyze the performance of MPC-Graph algorithm in the presence of process noise, we added random noise (SNR=1) to the calculated thruster forces, $u = [F_1\ F_2]^T$. We performed Monte-Carlo experiments (# simulations = 1000) on the same map with the same nodes and plot the trajectories followed by the robot. We determine an input saturation limit of $F = 25N$ for the thrusters. From 1000 Monte-Carlo experiments, in 332 of them (= % 33.2 of the simulations) robot failed to reach goal location due to process noise. Failures include the cases in which robot ends up in an unsampled region, outside the limits of the arena or inside the obstacles. From

1000 Monte-Carlo experiments in 108 of them (= % 10.8 of the simulations) robot ended up in an unsampled region and in 224 of them (= % 22.4 of the simulations) robot had a collision with an obstacle or ended up outside the limits of the arena. Fig. 4.19(a) and (b) visualizes the successful and failed attempts for reaching the goal point, respectively.

If robot ends up in a different node than the $CurrentNode$ due to process noise, algorithm generates another route taking into account the previously determined policy Po. To have a fair comparison with the case that does not include process noise, we used the same map with the same rectangular nodes.

From 108 experiments in which robot ended up in an unsampled region, with the implementation of resampling procedure robot can recover from 44 of the failure cases and reach the goal location. In Fig. 4.20 red routes show the trajectories followed by the robot and darker nodes represent the newly generated nodes after resampling procedure.

In Fig. 4.21(a) red and dashed blue trajectories indicate the routes followed by the robot in the presence of process noise and without noise, respectively. Fig. 4.21(b) shows the thruster forces and process noise applied on each thruster. Even in the presence of high level noise, in $\%71.2$ of the experiments differential USV model reaches the goal location.

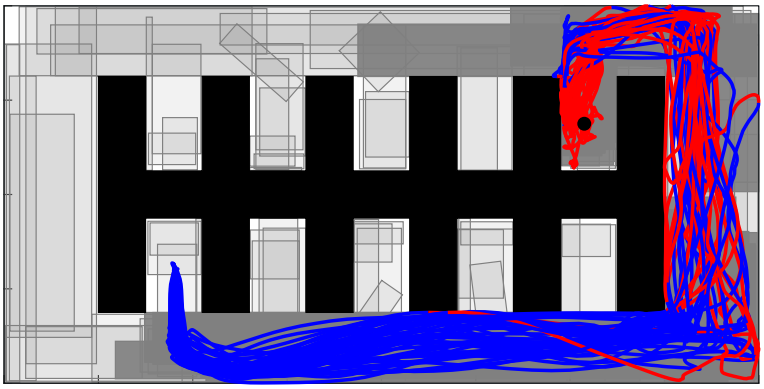
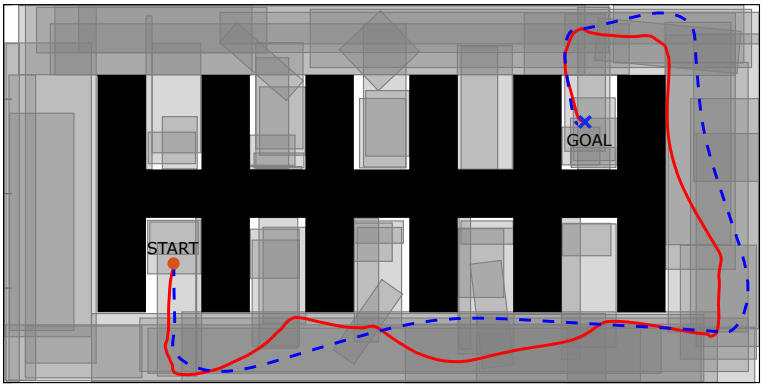
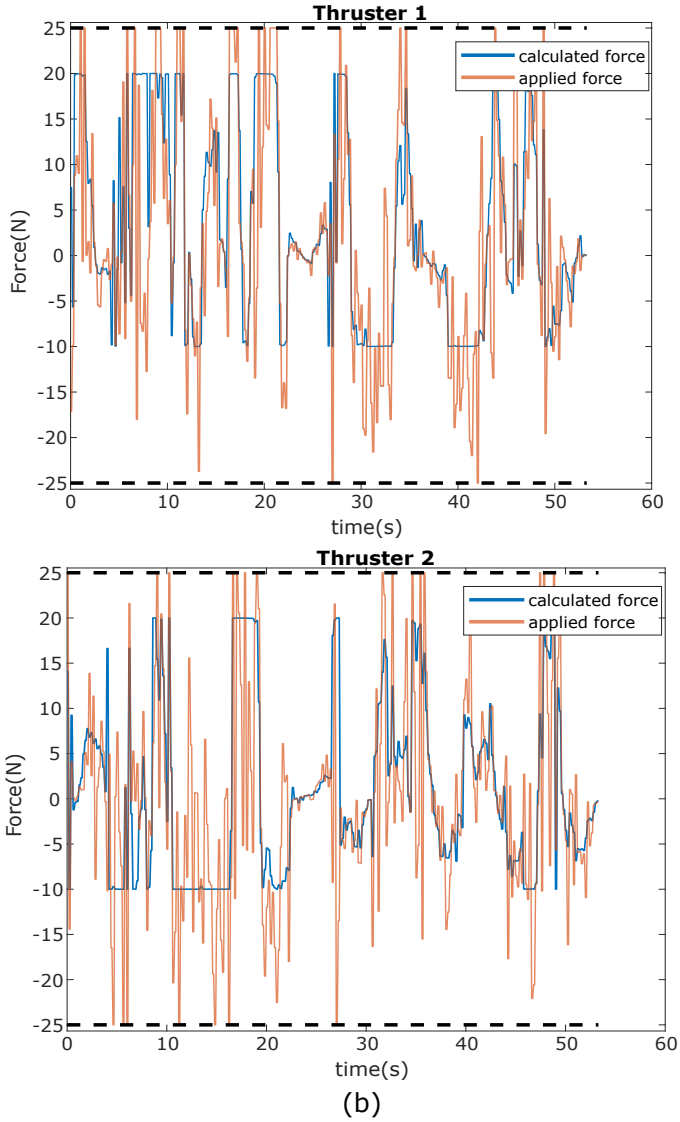


Figure 4.20: Red curves indicate the routes followed by the robot and darker nodes represent the newly generated nodes after resampling procedure.



(a)



(b)

Figure 4.21: (a) Red trajectory indicates the followed route in the presence of process noise. Dashed blue trajectory indicates the followed trajectory without the process noise. (b) Plots show the calculated forces and applied forces on each thruster. Black dashed lines indicate the saturation limits.

4.7 Time Complexity and Comparison of Results

We implemented MPC-Graph algorithm on four different motion models. Table 4.4 shows the CPU times for each iteration of MPC with respect to the implemented motion models. The CPU time for the graph search phase is independent of the motion model and is at average $t_{CPU} = 0.3s$.

Table 4.4: Time Complexity of MPC-Graph Algorithm

Motion Models	CPU Time
Linear Model	0.01s
Nonlinear Fully Actuated	0.08s
Fully Actuated USV	0.32s
Differential USV	2.04s

In our simulations we mainly focus on the capabilities of MPC-Graph algorithm applied on realistic robot motion models. For this reason we constructed simulation scenarios concerning the behaviors of a fully actuated and differential USV motion models. Results show that fully actuated USV has a failure rate of $\%1.6$ whereas differential USV has a failure rate of $\%28.8$ if process noise is applied on the system. One of the main reasons for this difference is that since differential USV cannot generate a force vector in order to control its motion in v direction, it cannot recover from its faulty behavior after a certain point. One of the solutions to decrease the failure rates can be increasing the horizon length T_p. Also at certain times throughout the simulation significant process noise is applied on the system as can be seen from Fig. 4.21 and Fig. 4.15. In real world applications the noise level is not expected to reach these values. In order to eliminate the failures coming from robots arrival at unsampled regions, we added an improvement to the MPC-Graph algorithm. With this improvement failures coming from robots ending up in an unsampled regions can

be decreased. However, robot usually ends up in small unsampled regions close to obstacles which makes it difficult to generate appropriate inputs to avoid collisions with obstacles.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

In this thesis, we proposed a new sampling-based trajectory free motion planning algorithm, MPC-Graph, that can be applied to systems that have linear or nonlinear dynamics with guaranteed asymptotic stability as long as it is provided that the Jacobian linearization of the dynamic system at origin is stabilizable. The algorithm first generates a sparse graph structure by randomly sampling the obstacle-free space with overlapping rectangular nodes. The sampling process continues until the defined termination condition is satisfied. Then, Dijkstra's search algorithm executes to find the 'optimal' route connecting start and goal locations. Dijkstra's search algorithm also returns policy *Po* that gives a route from any node in the graph to the goal node. After determination of the optimal path, MPC guides the robot to the goal location. In this work, we adopted quasi-infinite MPC framework of [28]. The implemented MPC approach ensures that robot stays inside the boundaries of the sampled nodes and satisfies the state and input constraints forced on the system.

During the implementation of the algorithm we observed that, if the intersection areas of the two consecutive nodes drops below a certain value in order to guarantee that robot stays inside the intersection area, MPC generates small inputs close to zero and thus robot gets stuck in this area. To avoid this problem we generated the edge cost given in (3.4). With this approach Dijkstra's search algorithm determines a route not only the shortest but has the largest intersection areas.

We implemented and tested our algorithm with Monte-Carlo experiments in Matlab. We generated 2D maps with both convex and concave obstacles. As a first approach

we compared the sampling performance our algorithm with the work of Yang et al. [3] and observed that our method generates a more sparse graph with a comparable CPU time. We tested our algorithm on robot motion models with gradually increasing nonlinearity. As a first approach we used a linear system model and observed that MPC satisfies the constraints forced on the system and drives the robot to the goal location. As a second approach we used a simple nonlinear system model with drag friction. In order to guarantee asymptotic stability we calculated terminal region and terminal cost matrix. Finally, we implemented a fully actuated and differential USV models which include rigid body dynamics, added mass effects, linear and nonlinear damping by using the dynamic model presented in [52].

Towards showing the robustness of the algorithm, we run Monte-Carlo simulations in the presence of process noise for USV models. Due to noise, in some cases robot ended up in a different node than its current node. Since *Po* determines the connection between any other node to the goal node, another route for the robot can be easily established. This is one of the benefits that using a graph structure provides. Also, in the presence of noise in several experiments robot ended up in unsampled region. For those situations we generated new nodes and added them to the previously obtained graph structure. Our results show that even in the presence of high level noise MPC can drive the robot to the goal location in most of the simulations.

5.2 Future Work

In this work, we tested our algorithm on a differential USV model without guaranteed asymptotic stability. In the future, we are planning to extend our approach to guarantee the stability of the under-actuated systems. Results presented in this work is currently limited to simulation results. As a future direction, we want to test our algorithm on real world applications.

MPC-Graph algorithm is currently implemented for 2D navigation but it can be generalized to include higher dimensional spaces. For example, in 3D environments, prismatic zones can be generated to cover the obstacle-free space. With this improvement algorithm can be used by robotic systems that operate in 3D spaces such as

quadrotors.

One of the drawbacks of using MPC is the high computational cost. We should further decrease the CPU time for the MPC-Graph algorithm to apply the algorithm in less powerful embedded platforms. Several researchers [54,55] proposed different solvers that can reduce the computation time for MPC based control policies. We are planning to adopt a similar approach to reduce the CPU time of our algorithm, and hence improve the applicability of the algorithm in a wide range of robotic applications.

REFERENCES

[1] M. Kumru, "Navigation and control of an unmanned sea surface vehicle," Master's thesis, Middle East Technical University, 2015.

[2] R. Skjetne, T. I. Fossen, and P. V. Kokotović, "Adaptive maneuvering, with experiments, for a model ship in a marine control laboratory," *Automatica*, vol. 41, no. 2, pp. 289–298, 2005.

[3] L. Yang and S. M. LaValle, "A framework for planning feedback motion strategies based on a random neighborhood graph," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 1, pp. 544–549, IEEE, 2000.

[4] E. Ege and M. M. Ankarali, "Feedback motion planning of unmanned surface vehicles via random sequential composition," *Transactions of the Institute of Measurement and Control*, vol. 41, no. 12, pp. 3321–3330, 2019.

[5] F. Golbol, M. M. Ankarali, and A. Saranli, "Rg-trees: Trajectory-free feedback motion planning using sparse random reference governor trees," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6506–6511, IEEE, 2018.

[6] S. M. LaValle *et al.*, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[7] O. K. Karagoz, S. Atasoy, and M. M. Ankarali, "Mpc-graph: Feedback motion planning using sparse sampling based neighborhood graph," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6797–6802, IEEE, 2020.

[8] M. Özcan and M. M. Ankarali, "Feedback motion planning for a dynamic car

model via random sequential composition," in *2019 IEEE international conference on systems, man and cybernetics (SMC)*, pp. 4239–4244, IEEE, 2019.

[9] M. Saska, Z. Kasl, and L. Přeucil, "Motion planning and control of formations of micro aerial vehicles," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 1228–1233, 2014.

[10] T. J. Stastny, G. A. Garcia, and S. S. Keshmiri, "Collision and obstacle avoidance in unmanned aerial systems using morphing potential field navigation and nonlinear model predictive control," *Journal of Dynamic Systems, Measurement, and Control*, vol. 137, no. 1, 2015.

[11] F. Farshidian, E. Jelavic, A. Satapathy, M. Giftthaler, and J. Buchli, "Real-time motion planning of legged robots: A model predictive control approach," in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pp. 577–584, IEEE, 2017.

[12] K. Yuan and Z. Li, "An improved formulation for model predictive control of legged robots for gait planning and feedback control," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1–9, IEEE, 2018.

[13] H. Li, W. Yan, and Y. Shi, "Continuous-time model predictive control of under-actuated spacecraft with bounded control torques," *Automatica*, vol. 75, pp. 144–153, 2017.

[14] A. Weiss, I. Kolmanovsky, M. Baldwin, and R. S. Erwin, "Model predictive control of three dimensional spacecraft relative motion," in *2012 American Control Conference (ACC)*, pp. 173–178, IEEE, 2012.

[15] C. Shen, Y. Shi, and B. Buckham, "Trajectory tracking control of an autonomous underwater vehicle using lyapunov-based model predictive control," *IEEE Transactions on Industrial Electronics*, vol. 65, no. 7, pp. 5796–5805, 2017.

[16] D. C. Fernández and G. A. Hollinger, "Model predictive control for underwater robots in ocean waves," *IEEE Robotics and Automation letters*, vol. 2, no. 1, pp. 88–95, 2016.

[17] "Annual overview of marine casualties and incidents 2020," Technical report, European Maritime Safety Agency, 2020.

[18] J. H. Lee, "Model predictive control: Review of the three decades of development," *International Journal of Control, Automation and Systems*, vol. 9, no. 3, pp. 415–424, 2011.

[19] C. E. Garcia, D. M. Prett, and M. Morari, "Model predictive control: Theory and practice—a survey," *Automatica*, vol. 25, no. 3, pp. 335–348, 1989.

[20] E. Polak and T. Yang, "Moving horizon control of linear systems with input saturation and plant uncertainty part 1. robustness," *International Journal of Control*, vol. 58, no. 3, pp. 613–638, 1993.

[21] E. Polak and T. Yang, "Moving horizon control of linear systems with input saturation and plant uncertainty part 2. disturbance rejection and tracking," *International Journal of Control*, vol. 58, no. 3, pp. 639–663, 1993.

[22] S. Keerthi and E. G. Gilbert, "Optimal infinite-horizon feedback laws for a general class of constrained discrete-time systems: Stability and moving-horizon approximations," *Journal of optimization theory and applications*, vol. 57, no. 2, pp. 265–293, 1988.

[23] V. Nevistić and J. A. Primbs, "Finite receding horizon linear quadratic control: A unifying theory for stability and performance analysis," 1997.

[24] W. C. Li and L. T. Biegler, "Process control strategies for constrained nonlinear systems," *Industrial & engineering chemistry research*, vol. 27, no. 8, pp. 1421–1433, 1988.

[25] R. Findeisen and F. Allgöwer, "An introduction to nonlinear model predictive control," in *21st Benelux meeting on systems and control*, vol. 11, pp. 119–141, Citeseer, 2002.

[26] H. Michalska and D. Q. Mayne, "Robust receding horizon control of constrained nonlinear systems," *IEEE transactions on automatic control*, vol. 38, no. 11, pp. 1623–1633, 1993.

[27] G. De Nicolao, L. Magni, and R. Scattolini, "Stabilizing receding-horizon control of nonlinear time-varying systems," *IEEE Transactions on Automatic Control*, vol. 43, no. 7, pp. 1030–1036, 1998.

[28] H. Chen and F. Allgöwer, "A quasi-infinite horizon nonlinear model predictive control scheme with guaranteed stability," *Automatica*, vol. 34, no. 10, pp. 1205–1217, 1998.

[29] H. Van Essen and H. Nijmeijer, "Non-linear model predictive control for constrained mobile robots," in *2001 European Control Conference (ECC)*, pp. 1157–1162, IEEE, 2001.

[30] D. Gu and H. Hu, "Model predictive control for simultaneous robot tracking and regulation," in *2004 International Conference on Intelligent Mechatronics and Automation, 2004. Proceedings.*, pp. 212–217, IEEE, 2004.

[31] A. S. Conceição, H. P. Oliveira, A. S. e Silva, D. Oliveira, and A. P. Moreira, "A nonlinear model predictive control of an omni-directional mobile robot," in *2007 IEEE international symposium on industrial electronics*, pp. 2161–2166, IEEE, 2007.

[32] H. Lim, Y. Kang, C. Kim, J. Kim, and B.-J. You, "Nonlinear model predictive controller design with obstacle avoidance for a mobile robot," in *2008 IEEE/ASME International Conference on Mechtronic and Embedded Systems and Applications*, pp. 494–499, IEEE, 2008.

[33] M.-M. Ma, S. Li, and X.-J. Liu, "Tracking control and stabilization of wheeled mobile robots by nonlinear model predictive control," in *Proceedings of the 31st Chinese Control Conference*, pp. 4056–4061, IEEE, 2012.

[34] X. Du, K. K. K. Htet, and K. K. Tan, "Development of a genetic-algorithm-based nonlinear model predictive control scheme on velocity and steering of autonomous vehicles," *IEEE Transactions on Industrial Electronics*, vol. 63, no. 11, pp. 6970–6977, 2016.

[35] H. J. Kim, D. H. Shim, and S. Sastry, "Nonlinear model predictive tracking control for rotorcraft-based unmanned aerial vehicles," in *Proceedings of the*

2002 American control conference (IEEE Cat. No. CH37301), vol. 5, pp. 3576–3581, IEEE, 2002.

[36] Y. Kuwata and J. How, "Three dimensional receding horizon control for uavs," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, p. 5144, 2004.

[37] Y. Kang and J. Hedrick, "Design of nonlinear model predictive controller for a small fixed-wing unmanned aerial vehicle," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, p. 6685, 2006.

[38] Y. Kang and J. K. Hedrick, "Linear tracking for a fixed-wing uav using nonlinear model predictive control," *IEEE Transactions on Control Systems Technology*, vol. 17, no. 5, pp. 1202–1210, 2009.

[39] K. Shimada and T. Nishida, "Particle filter-model predictive control of quadcopters," in *Proceedings of the 2014 International Conference on Advanced Mechatronic Systems*, pp. 421–424, IEEE, 2014.

[40] M. H. Tanveer, D. Hazry, S. F. Ahmed, M. K. Joyo, F. A. Warsi, H. Kamaruddin, Z. M. Razlan, K. Wan, and A. Shahriman, "Nmpc-pid based control structure design for avoiding uncertainties in attitude and altitude tracking control of quad-rotor (uav)," in *2014 IEEE 10th International Colloquium on Signal Processing and its Applications*, pp. 117–122, IEEE, 2014.

[41] A. Vagale, R. Oucheikh, R. T. Bye, O. L. Osen, and T. I. Fossen, "Path planning and collision avoidance for autonomous surface vehicles i: a review," *Journal of Marine Science and Technology*, pp. 1–15, 2021.

[42] B. Zhao, X. Zhang, C. Liang, and X. Han, "An improved model predictive control for path-following of usv based on global course constraint and event-triggered mechanism," *IEEE Access*, 2021.

[43] X. Sun, G. Wang, Y. Fan, D. Mu, and B. Qiu, "Collision avoidance using finite control set model predictive control for unmanned surface vehicle," *Applied Sciences*, vol. 8, no. 6, p. 926, 2018.

[44] X. Zhou, Y. Wu, and J. Huang, "Mpc-based path tracking control method for usv," in *2020 Chinese Automation Congress (CAC)*, pp. 1669–1673, IEEE, 2020.

[45] Z. Liu, C. Geng, and J. Zhang, "Model predictive controller design with disturbance observer for path following of unmanned surface vessel," in *2017 IEEE International Conference on Mechatronics and Automation (ICMA)*, pp. 1827–1832, IEEE, 2017.

[46] W. Wang, T. Shan, P. Leoni, D. Fernández-Gutiérrez, D. Meyers, C. Ratti, and D. Rus, "Roboat ii: A novel autonomous surface vessel for urban environments," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1740–1747, IEEE, 2020.

[47] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[48] K. I. Tsianos, I. A. Sucan, and L. E. Kavraki, "Sampling-based robot motion planning: Towards realistic applications," *Computer Science Review*, vol. 1, no. 1, pp. 2–11, 2007.

[49] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review," *Ieee access*, vol. 2, pp. 56–77, 2014.

[50] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts, "Lqr-trees: Feedback motion planning via sums-of-squares verification," *The International Journal of Robotics Research*, vol. 29, no. 8, pp. 1038–1052, 2010.

[51] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, no. 6, pp. 789–814, 2000.

[52] T. I. Fossen, *Handbook of marine craft hydrodynamics and motion control*. John Wiley & Sons, 2011.

[53] L. Yang and S. M. Lavalle, "The sampling-based neighborhood graph: An approach to computing and executing feedback motion strategies," *IEEE Transactions on Robotics and Automation*, vol. 20, no. 3, pp. 419–432, 2004.

[54] B. Stellato, V. V. Naik, A. Bemporad, P. Goulart, and S. Boyd, "Embedded mixed-integer quadratic optimization using the osqp solver," in *2018 European Control Conference (ECC)*, pp. 1536–1541, IEEE, 2018.

[55] S. Richter, *Computational complexity certification of gradient methods for real-time model predictive control*. ETH Zurich, 2012.