MOTION PLANNING AND CONTROL OF UNDERACTUATED SYSTEMS OVER OPTIMIZED TRAJECTORIES

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

BY

EMİNALP KOYUNCU

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

SEPTEMBER 2024

Approval of the thesis:

MOTION PLANNING AND CONTROL OF UNDERACTUATED SYSTEMS OVER OPTIMIZED TRAJECTORIES

submitted by **EMİNALP KOYUNCU** 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. Naci Emre Altun
Dean, Graduate School of **Natural and Applied Sciences** _____

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

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

Examining Committee Members:

Assist. Prof. Dr. Serkan Sarıtaş
Electrical And Electronics Engineering, METU _____

Assoc. Prof. Dr. Mustafa Mert Ankaralı
Electrical And Electronics Engineering, METU _____

Assoc. Prof. Dr. İsmail Uyanık
Electrical And Electronics Engineering, Hacettepe University _____

Date:05.09.2024

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: Eminalp Koyuncu

Signature :

ABSTRACT

MOTION PLANNING AND CONTROL OF UNDERACTUATED SYSTEMS OVER OPTIMIZED TRAJECTORIES

Koyuncu, Eminalp
M.S., Department of Electrical and Electronics Engineering
Supervisor: Assoc. Prof. Dr. Mustafa Mert Ankaralı

September 2024, 98 pages

In this work, we propose an optimal control strategy that is robust and capable of running real-time for nonlinear underactuated systems. Our method combines an optimization-based trajectory planner with the feedback motion planning methods. We combine the local controllers created by the feedback motion planning algorithms to generate a global trajectory with trajectory optimization, taking the underactuation into consideration. We follow the generated trajectory using a global controller.

We first generate a Sparse Neighborhood Graph (SNG) in the obstacle-free region of the configuration space. We generate waypoints at each node intersection on the graph, and hierarchical waypoints are identified along the shortest path from start to goal. We then run an optimization algorithm, taking the system dynamics and constraints into account to minimize input effort and generate trajectories between waypoints using a receding horizon optimization strategy. Finally, we use a linear time-varying (LTV) model predictive control (MPC) policy to track the generated trajectory, ensuring constraints are satisfied during system operation.

We tested our algorithm on underactuated unmanned surface vehicles (USVs) to

drive them in the presence of workspace obstacles, and input and speed constraints. We used 2 different USV models, one implemented in MATLAB and the other is Clearpath Robotics Heron USV on a ROS-Gazebo simulation. We compared our results with previous works, considering real-time performance and robustness. Our work showed superior results regarding all the criterion. However, one drawback of our method is the computational time and power required for the offline planning action.

Keywords: Trajectory Optimization, Underactuated Systems, Model Predictive Control

ÖZ

KISITLI TAHRİKLİ SİSTEMLERİN OPTİMİZE YÖRÜNGELER ÜZERİNDE HAREKET PLANLAMASI VE KONTROLÜ

Koyuncu, Eminalp
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. Mustafa Mert Ankaralı

Eylül 2024 , 98 sayfa

Bu çalışmada, doğrusal olmayan kısıtlı tahrikli sistemler için gürbüz ve gerçek zamanlı bir optimal kontrol stratejisi öneriyoruz. Metodumuz, geri beslemeli hareket planlama yöntemleri üzerine kurulu bir optimizasyon tabanlı yörünge planlayıcısı kullanıyor. Geri beslemeli planlayıcılar, konfigürasyon uzayını engelsiz bölgelere ayırarak, her bölgede yerel kontrolcü ile sistemin sürülmesini sağlar. Biz ise bu yerel kontrolcüleri, giriş kısıtlarını dikkate alarak, global bir yörünge oluşturmak için optimizasyon yöntemleri ile kullanıyoruz. Ardından, oluşturulan global yörünge bir kontrolcü ile takip ediliyor.

Çalışmamızda ilk olarak sistemin konfigürasyon uzayında Seyrek Komşuluk Grafiği oluşturarak, başlangıçtan hedefe en kısa yoldan ulaşan düşümlerin kesişim noktasında bulunan hiyerarşik ara noktaları belirliyoruz. Bu noktalar arasında, sistem dinamikleri ve kısıtlarını dikkate alarak, girdi eforunu minimize eden bir optimizasyon algoritması ile global yörüngeleri hesaplıyoruz. Son aşamada, sistemin bu yörüngeyi kapalı döngü bir kontrolcü ile takip etmesini sağlıyoruz.

Algoritmamızı, kısıtlı tahrikli insansız su üstü araçlarını engelli alanlarda sürerek sınadık. Çalışmamızı MATLAB üzerinde hazır bir model ve Clearpath Robotics Heron insansız su üstü aracının ROS-Gazebo verileriyle oluşturduğumuz bir modeli kullanarak denedik. Sonuçlar metodun çevrimdışı planlama aşamasında yüksek işlem gücü gerektirmesine rağmen önceki çalışmalarla karşılaştırıldığında sistemin çevrimiçi olarak gerçek zamanlı performans ve gürbüzlük açısından üstünlük gösterdiğini ispatladı.

Anahtar Kelimeler: Yörünge Optimizasyonu, Kısıtlı Tahrikli Sistemler, Model Öngörülü Kontrolcü

In the memory of Emin Koyuncu, my dear grandfather

ACKNOWLEDGMENTS

First of all, I would like to thank and express my gratitude to my supervisor Mustafa Mert Ankaralı for his valuable support and guidance throughout my thesis studies. His valuable feedback guided me for my professional life and gave me motivation to enhance my knowledge.

I would like to thank Mustafa Yıldırım for his efforts on bringing up Heron USV, examining it and helping me with the system identification of Heron USV. I would also thank him for the insightful conversations about the methods developed in this work.

I am grateful to Center for Robotics and Artificial Intelligence (ROMER) of Middle East Technical University to let me use their infrastructure during my thesis works. I am also thankful to them for employing me before I obtain my research assistant title.

I would like to thank TUBITAK, The National Scientific and Technological Research Council of Turkey, for supporting me through M.S Studies scholarship under 1001 program. This thesis is partially supported by TUBITAK 1001 program under project 122E249.

I am so grateful to Kübra Toral, for her affection, emotional support and caring during this journey. I wouldn't succeed in these studies without her support.

Finally, and most importantly, I would like to express my deepest appreciation to my family, my mother Zübeyde Koyuncu, my father Oğuzhan Koyuncu, my little brother Başaralp Koyuncu and my grandmothers Sevin Koyuncu and Emine Susuz for their sacrifices and effort during my education. I wouldn't be able to succeed in my whole education without them. I dedicate this work to Emin Koyuncu, my dear grandfather who passed away recently. May he rests in peace.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

2D	2 Dimensional
3D	3 Dimensional
USV	Unmanned Surface Vehicles
MPC	Model Predictive Control
SNG	Sparse Neighborhood Graphs
NARMAX	Nonlinear Auto Regressive Moving Average With Exogeneous Inputs
ROS	Robot Operating System
PRM	Probabilistic Roadmaps
RRT	Rapidly Exploring Random Trees
SNR	Signal-to-noise Ratio
SQP	Sequential Quadratic Programming

CHAPTER 1

INTRODUCTION

1.1 Motivation and Problem Definition

In the field of control theory, the control of underactuated systems has been subject to great interest due to the challenges introduced by underactuation. With such systems, any control action taking place at some point of time greatly affects the future trajectories of the system. That is because some modes of the system are not directly reachable at that point of time, and they evolve respecting the past control actions that took place. Another issue with the underactuation is that, it is proven that the classical linear, nonlinear and continuous state feedback policies cannot stabilize the underactuated systems [7]. To tackle such challenges, control theorists came up with the idea of using optimal control methods on underactuated systems [8] to generate a time-varying control policy. Optimal control methods simulate the system forward in a time window to see how underactuated modes evolve over time. Meanwhile, they decide an optimal control policy that will lead the system to reach a desired state eventually, minimizing a specified cost function and regarding constraints imposed on the system [9]. However, using such methods on highly dynamic and nonlinear systems requires a high amount of computational power and might not be feasible for real-time systems. In order to overcome this problem, the optimal control methods are reinforced using offline motion planning techniques [10], to decrease the computational burden on the optimal controller. In particular, with the emergence of feedback motion planning methods, the configuration space of the system is divided into the obstacle-free regions and in each region, a local feedback controller, including optimal controllers, is utilized to drive the system between these regions hierarchically, with the goal of reaching a desired configuration [6]. These

methods show a great performance dealing with the nonlinearities of highly dynamic systems, however, still fall short for underactuated systems, regarding the computational burden. In this work, we overcome these challenges with the receding horizon trajectory optimization method applied on the Sparse Neighborhood Graphs (SNG). With the trajectory optimization, we generate optimized trajectories over the regions generated using SNG, taking the underactuation into consideration. By doing so, we minimize the computational burden on the controller. We use a Model Predictive Control (MPC) policy to create a time-varying control policy that drives the system over these optimized trajectories in real-time.

1.2 Literature Review

Optimal control methods have been widely used in every aspect of science and robotics, from aquaculture modelling [11], to cancer chemotherapy [12], power electronics [13], spacecraft attitude control [14], control of underwater vehicles [15], and control of quadrotors [16] since the development of the dynamic programming concept by Richard Bellman [17]. The dynamic programming is a search method originally developed to solve optimal control problems, however, it suffers from the Bellman's curse of dimensionality [18] which states that the number of decision parameters for the search problem increases exponentially with the number of states (dimensions) of the dynamical system. As the interest for the optimal control increase, parameterizations other than the dynamic programming emerged to be able to parameterize the dynamics of the system and solve the optimal control problem effectively. Most popular of these parameterizations are the so-called direct transcription, direct single shooting and direct collocation methods [19, 20, 21]. There are also relatively new parameterizations called pseudospectral methods subject to current research [22, 23]. For the nonlinear optimal control, together with these parameterizations, a nonlinear optimizer such as sequential quadratic programming [24] or an interior-point based optimizer [25] is utilized to solve the nonlinear optimal control problem.

One of the most popular optimal control methods is the Model Predictive Control (MPC). This optimal control method is firstly developed to meet the need of optimizing multi-input multi-output constrained processes of petro-chemical industry with

the name of Dynamic Matrix Control [26, 27]. As the MPC gains popularity due to its superior performance and ability to handle complex multi-input multi-output systems, robotics researchers embraced the MPC and used it for various tasks, including the motion control of unmanned surface vehicles (USVs) [28, 29, 30, 31]. In [28, 30, 31], the authors utilize MPC for the path following problem of the USVs and in [29], they use a finite control set MPC for the obstacle avoidance of the USVs.

For the robots that operate in environments with obstacles, motion planning is a crucial part of the robot design. The motion planning problem for robotics can be summarized as finding the collision-free trajectories that drive the robot from an initial configuration to a goal state which obey the constraints imposed by the environment or the structure of the system [32]. In general, these trajectories can be generated offline and as the robot starts it operation, a motion controller drives the robot along these trajectories. This way, the most of the computational burden of the real-time controller is undertaken during the offline planning phase. Sampling-based motion planning algorithms gained popularity in the past years due to its relatively low computational complexity even for robots with higher dimensional configuration spaces [33]. Most notable sampling based methods are the Probabilistic Roadmaps (PRM) based planners [34] or the Rapidly Exploring Random Trees (RRT) based planners [35]. One disadvantage of the sampling based methods is that they generate open loop trajectories for the system that can be tracked using a motion controller later on. This approach can create infeasible trajectories for the system, especially if the system has complex and highly constrained dynamics, such as the underactuated systems. These problems are solved with the emergence of feedback motion planning methods, especially sequential composition of feedback controllers, that divides the complex configuration space of the system into simpler regions, along with a local feedback controller that drives the system between those regions [36]. The sequential composition method has been successfully applied to several problems in the literature [37, 38, 39, 40]. One challenge of this method is to find an effective way to divide the configuration space into simple regions. To solve this problem, sampling based sequential composition methods have emerged. These methods use the sampling tools of the sampling based methods and create those regions around the sampled configurations [41, 42, 43, 6]. Especially, in this study, we focus on increas-

ing the performance of MPC-Graph Algorithm [6], that uses Sparse Neighborhood Graphs to generate these regions and uses MPC as the local feedback controller for the underactuated USVs.

1.3 Contributions and Novelties

The main contribution of our work is to enhance the performance of the sequential composition of feedback controllers method by implementing the receding horizon trajectory optimization inside each funnel for underactuated systems. In our method, we first generate a Sparse Neighborhood Graph (SNG) on the obstacle-free region of the configuration space. Then, we generate possible waypoints for our system at the intersections of each node of the graph, and we find hierarchical waypoints along the shortest path, connecting the starting configuration to the desired configuration. After that, we run an optimization algorithm that takes the system dynamics and constraints into account and minimizes the input effort between each waypoint. This optimization procedure generates optimized open-loop state and input trajectories, that connect the starting configuration to the goal over time. SİNce most of the predictions of the system trajectories carried out offline, the computational burden for the controller decreases significantly. Then, we drive our system along these trajectories using a linear time varying (LTV) Model Predictive Control (MPC) policy to make sure that the constraints are still not violated during the operation of the system.

We developed and tested our algorithm on underactuated unmanned surface vehicles (USVs) to drive them in the presence of workspace obstacles, and input and speed constraints. We used 2 different nonlinear underactuated USV models, one implemented in MATLAB with a predefined model and other obtained by conducting system identification on Clearpath Robotics Heron USV on a ROS-Gazebo simulation. For the nonlinear system identification of Heron USV, we utilised the so called NARMAX methods and identified the model parameters of the USV based on Fossen's equations of motion. Our method showed superior results in terms of real-time performance, robustness and USV power consumption for the underactuated USV.

1.4 The Outline of the Thesis

In Chapter 2, we begin with explaining the key concepts used to develop and experiment with our method, namely, the sequential composition algorithm, Sparse Neighborhood Graphs, nonlinear optimization, parameterizations of optimal control, MPC, USV dynamics and NARMAX methods. After explaining these key concepts, in Chapter 3, we give the complete description of our algorithm, the receding horizon trajectory optimization on Sparse Neighborhood Graphs and the motion control over the optimized trajectories. In Chapter 4, we first explain the implementation of our method on two different underactuated USV models, one simulated in MATLAB environment using a pre-defined model and other is the Clearpath Robotics Heron USV on a Gazebo simulation. After we explain the implementation, we give the key results of our method. Finally, we compare and contrast the results of our algorithm with previous work and show its superior real-time performance, robustness and energy efficiency. In Chapter 5, we outline our work and discuss the possible future research directions.

CHAPTER 2

PRELIMINARIES AND BACKGROUND

2.1 Sequential Composition of Feedback Controllers

Sequential composition of feedback controllers is a feedback motion planning technique that divides the complex configuration space of a robot into simple regions called "funnels" [36]. At each funnel, a stabilizing feedback control policy drives the robot from an initial state to a local goal, that connects the active funnel to the next funnel. The funnel shapes are in general selected as the region of attraction of the local feedback controller. Following each funnel hierarchically, a robot can be driven from an initial state to a global goal state. Figure 2.1 illustrates the funnel analogy used in [36].

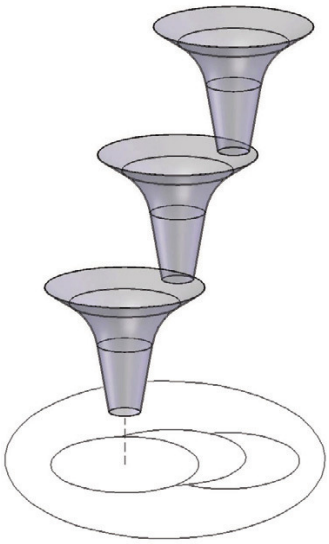


Figure 2.1: Illustration of the funnel analogy.

There are many studies based on sequential composition of feedback controllers differing in the funnel shape and the local control policy. In the study presented in [33], the authors use a circular funnel shape in combination with a Lyapunov based controller. Also, in [41], a square and in [43] an elliptic funnel shape are used. In [42] the local controller is selected as a Linear Quadratic Regulator (LQR) and the funnel shape is determined as the region of attraction of the LQR controller which is computed using a sums-of-squares verification of Lyapunov functions. Another study is the MPC-Graph [6], which creates Sparse Neighborhood Graphs using rectangular funnels that utilize Model Predictive Control (MPC) policy to constrain the system inside these funnels with box type of constraints. Our study aims to improve the performance of MPC-Graph in the presence of underactuation. The details of the Sparse Neighborhood Graph generation are explained in the next section.

2.2 Sparse Neighborhood Graphs

In our method, we utilize Sparse Neighborhood Graphs to generate collision-free waypoints for the trajectory optimization in a 2D workspace. The graph generation process is explained in detail in [44, 2, 1].

In the graph generation, the set of points covered by the nodes at any time is denoted by $\mathcal{B}$. During the process, first a sample q_{rand} is drawn and checked whether it collides with any of the obstacles or the previously sampled regions. If there is no collusion, a square node $Node_k$ is expanded around q_{rand} and if there is collision q_{rand} is discarded. So,

$$\mathcal{B} = \bigcup_k Node_k \tag{2.1}$$

After drawing q_{rand}, the shortest distance to any of the obstacles is calculated. If we denote the i^{th} obstacle by WO_i and the shortest distance as q_{obs},

$$WO = \bigcup_i WO_i \tag{2.2a}$$

$$q_{obs} = \arg\min_{q \in WO} ||q - q_{rand}|| \tag{2.2b}$$

If we let the minimum distance between the q_{obs} and q_{rand} by l_{min}, then we form the circle shown in the Figure 2.2b, with center q_{rand} and radius l_{min}. After forming this circle the largest square node is placed on the circle with edge length $l_{min}\sqrt{2}$, and its one edge perpendicular to the line segment connecting q_{obs} and q_{rand}. After placing the square node, the node expands into a rectangular node as shown in the 2.2c, in 1 and 2 directions incrementally with growth rate γ. It means, at every iteration, the current edge length of the rectangle is multiplied with γ in direction 1. If any collision occurs or the node expands outside the map, the last expansion is discarded and the same process is applied on the direction 2. The overall node generation process is illustrated in Figure 2.2.

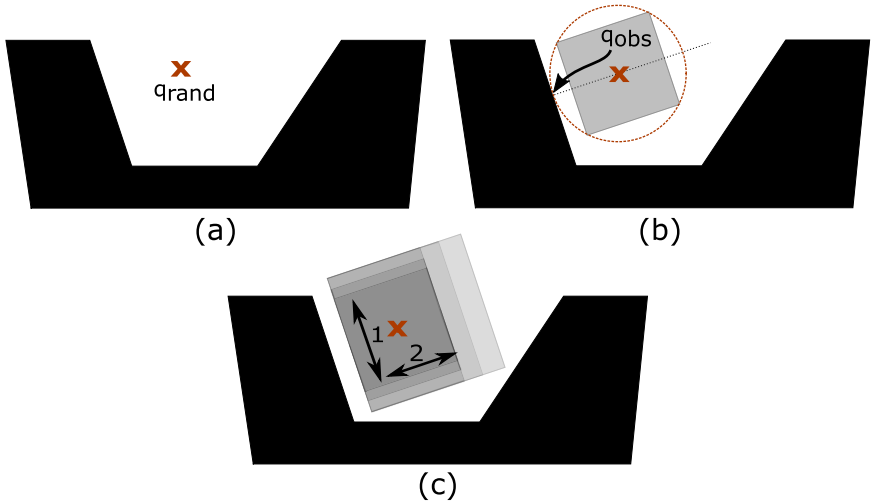


Figure 2.2: Node generation [1]. (a) A collision free random sample is drawn. (b) A square node is expanded until the encapsulating circle collides with an obstacle. (c) The node is continued to be expanded with discrete steps in 1 and 2 directions until it collides with an obstacle.

The node generation continues until $\mathcal{B}$ covers a sufficient region in the map by sampling and expanding nodes. The graph generation phase terminates when the following inequality holds,

$$m \geq \frac{\ln(1 - P_c)}{\ln\alpha} - 1. \tag{2.3}$$

In (2.3), m is the number of discarded samples after a successful node generation, and P_c and α are the user defined parameters that determines the quality of the generated graph.

After the graph is generated, for each $Node_i$, the center of the node $center_i$ is determined. Note that $center_i$ differs from q_{rand} corresponding to the node due to the rectangular expansion. After that, for each node, the overlapping area with each overlapping $Node_j$ is calculated as A_i and the centeroid q_i of A_i is determined. This process is illustrated in Figure 2.3. Finally, the edge cost of the edge connecting $Node_i$ to $Node_j$ for all $(Node_i, Node_j)$ pair is calculated as the (2.4).

$$cost_{i,j} = ||center_i - q_{i,j}||_2 + ||center_j - q_{i,j}||_2 + \frac{\gamma}{A_i} \tag{2.4}$$

After all the edge costs are calculated, the shortest path from given q_{start} to q_{goal} is calculated using the Dijkstra's algorithm, from the node containing q_{start} to the node containing q_{goal}. Figure 2.4 shows the generated nodes and the shortest path for a sample run of the algorithm.

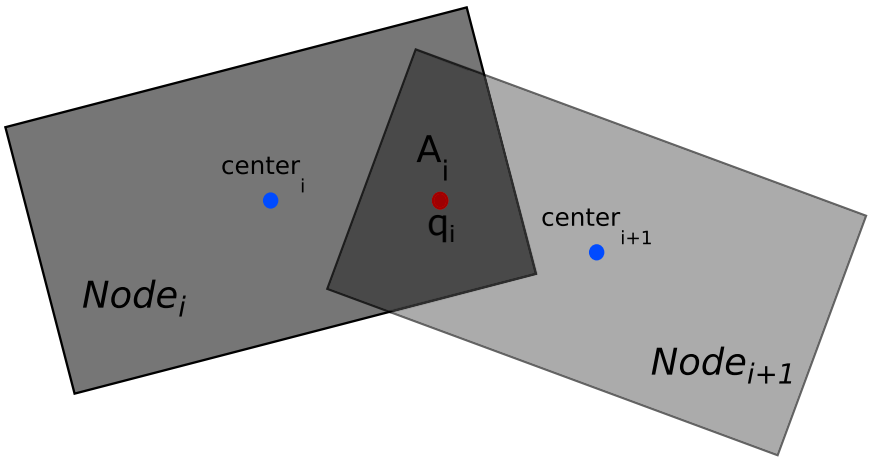


Figure 2.3: Visualization of the connected nodes to construct the graph [2]. The distances between the centers and q_i, and the overlapping arena A_i are used to calculate the edge cost. q_i serves as waypoint to the trajectory optimization.

In this study, we used the rectangular nodes with the same mentality presented in the [2], for its compatibility with linear MPC framework. In the control part, we constrain the position of the USV to be inside the rectangular nodes, which makes four linear inequality constraints per (x, y) Cartesian coordinate pair of the USV. There are also studies with square [41], and circle [45] nodes, however, rectangular nodes results in sparser graph structure than the square nodes and the position constraints emerging from circular nodes happens to be nonlinear. Finally, we keep track of the number of nodes in the output of the Dijkstra's algorithm to use in the trajectory optimization. Let $\mathcal{T}$ be the path found by using Dijkstra's algorithm. So, $\mathcal{N} = size(\mathcal{T})$ is the number of nodes in the shortest path with $size(.)$ function returning the number of elements in the argument. We index the nodes in the shortest path starting from the node containing the starting point to the node containing the goal node consecutively. So, following the nodes from $Node_i$ for $i = 1, \ldots, \mathcal{N}$, one can reach the goal node from the starting node.

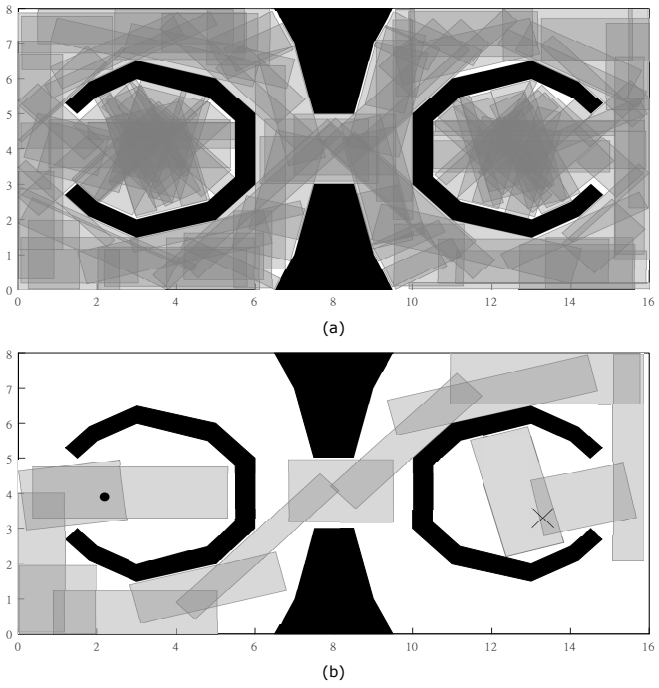


Figure 2.4: Graph generation and the shortest path after the graph search. (a) shows the overall graph structure after the graph generation is complete. (b) shows the shortest path. Hollow black circle is the starting point. Black cross is the goal.

2.3 Nonlinear Optimization

In this work, nonlinear optimization methods are heavily used for the trajectory optimization. In most general form, a nonlinear optimization problem can be formulated as (2.5). Note that the methods presented in this section applies for bot the convex and non-convex problems. However, for the non-convex problems, it is likely that the solver converges to a local minimum instead of a global one.

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{h}(\mathbf{x}) = \mathbf{0} \\ & \mathbf{g}(\mathbf{x}) \leq \mathbf{0} \end{aligned} \tag{2.5}$$

In this formulation, $\mathbf{x}$ is the vector of decision variables, $f(\mathbf{x})$ is the scalar valued nonlinear cost function, $\mathbf{h}(\mathbf{x})$ and $\mathbf{g}(\mathbf{x})$ are the vector of general nonlinear equality and inequality constraints in the form of $\mathbf{h}(\mathbf{x}) = \begin{bmatrix} h_1(\mathbf{x}) & h_2(\mathbf{x}) & \dots & h_m(\mathbf{x}) \end{bmatrix}^{\mathbf{T}}$ and $\mathbf{g}(\mathbf{x}) = \begin{bmatrix} g_1(\mathbf{x}) & g_2(\mathbf{x}) & \dots & g_n(\mathbf{x}) \end{bmatrix}^{\mathbf{T}}$, respectively. Here, each $h_i(\mathbf{x})$ represents a nonlinear equality constraint and $g_i(\mathbf{x})$ represents a nonlinear inequality constraint. An optimal feasible solution to the optimization problem is denoted by $\mathbf{x}^*$. The Lagrangian of this problem is defined as,

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\mathbf{x}) + \boldsymbol{\lambda}^{\mathbf{T}}\mathbf{g}(\mathbf{x}) + \boldsymbol{\mu}^{\mathbf{T}}\mathbf{h}(\mathbf{x}), \tag{2.6}$$

where $\boldsymbol{\lambda} = \begin{bmatrix} \lambda_1 & \lambda_2 & \dots \lambda_n \end{bmatrix}^T$ and $\boldsymbol{\mu} = \begin{bmatrix} \mu_1 & \mu_2 & \dots \mu_m \end{bmatrix}^T$ are called the Lagrange Multipliers for inequality and equality constraints, respectively. Let $\mathcal{D}$ be the set of admissible values of $\mathbf{x}$, and,

$$d(\boldsymbol{\lambda}, \boldsymbol{\mu}) = \inf_{\mathbf{x} \in \mathcal{D}} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \tag{2.7}$$

So that,

$$d(\boldsymbol{\lambda}, \boldsymbol{\mu}) \leq f(\mathbf{x}^*) \tag{2.8}$$

for all $\boldsymbol{\lambda} \geq 0$. (2.8) implies that if the equality holds, which is called strong duality, the following optimization problem in (2.9) has the same optimal value as the problem in (2.5), which is called the primal problem whereas (2.9) is called the dual problem.

$$\begin{aligned} \max_{\boldsymbol{\lambda},\boldsymbol{\mu}} \quad & d(\boldsymbol{\lambda},\boldsymbol{\mu}) \\ \text{s.t.} \quad & \boldsymbol{\lambda} \geq 0 \end{aligned} \tag{2.9}$$

A pair of primal optimal solution $\mathbf{x}^*$ to the problem (2.5) and dual optimal solution $(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ to the problem (2.9) is called primal-dual optimal feasible solution, and together they satisfy so called Karush-Kuhn-Tucker (KKT) conditions presented in (2.10).

$$\nabla f(\mathbf{x}^*) + \sum_{\mathbf{i=1}}^{\mathbf{n}} \lambda_{\mathbf{i}}^* \nabla g_i(\mathbf{x}^*) + \sum_{\mathbf{j=1}}^{\mathbf{m}} \mu_{\mathbf{j}}^* \nabla h_j(\mathbf{x}^*) = 0 \tag{2.10a}$$

$$\mathbf{h}(\mathbf{x}^*) = 0 \tag{2.10b}$$

$$\mathbf{g}(\mathbf{x}^*) \leq 0 \tag{2.10c}$$

$$\boldsymbol{\lambda}^* \geq 0 \tag{2.10d}$$

$$\lambda_i^* g_i(\mathbf{x}^*) = 0 \tag{2.10e}$$

In the given conditions, (2.10a) implies that, at the primal-dual optimal solution, the Lagrangian is stationary, meaning, the gradients of the Lagrangian vanishes. (2.10b) and (2.10c) represents the primal feasibility, meaning, the primal optimal solution is feasible, obeying the constraints. Similarly, (2.10d) implies the dual feasibility, meaning, the dual optimal solution obeys the constraints of the dual problem. (2.10e) is called the "Complementary Slackness" and it means that, at the primal-dual optimal point, either the constraint $g_i(\mathbf{x}^*) \leq \mathbf{0}$ is active with $g_i(\mathbf{x}^*) = \mathbf{0}$, or it is inactive with $g_i(\mathbf{x}^*) < \mathbf{0}$ and $\lambda_i = 0$.

In the case of strong duality, KKT conditions are necessary and sufficient for the solution to be globally optimal. However, if the duality is weak, KKT conditions are only necessary and not sufficient for the global optimality and the solution might be sub-optimal due to a local minimum or a stationary point.

In most of the cases, there is no analytical solution to the KKT system. However, there are iterative algorithms to find a solution to the KKT system. Two of these algorithms, namely, Sequential Quadratic Programming and Interior Point Algorithm will be discussed further in this section.

2.3.1 Sequential Quadratic Programming (SQP)

With the Sequential Quadratic Programming (SQP) algorithm, instead of directly solving the KKT system for the given nonlinear optimization problem, we iteratively solve it using a Newton step approach with the step direction determined by an approximate Quadratic Programming (QP) subproblem to the original problem. The Newton step is defined as,

$$\mathbf{x_{k+1}} = \mathbf{x_k} + \alpha_\mathbf{k} \mathbf{d_x^{k^*}} \tag{2.11a}$$

$$\boldsymbol{\lambda}_{k+1} = \boldsymbol{\lambda}_k + \alpha_k \mathbf{d}_\lambda^{\mathbf{k}^*} \tag{2.11b}$$

$$\boldsymbol{\mu}_{k+1} = \boldsymbol{\mu}_k + \alpha_k \mathbf{d}_\mu^{\mathbf{k}^*} \tag{2.11c}$$

where $\mathbf{d_x^{k^*}}$, $\mathbf{d}_\lambda^{\mathbf{k}^*}$ and $\mathbf{d}_\mu^{\mathbf{k}^*}$ are called the search or step direction for the decision variables, Lagrange multiplier for inequality constraint and Lagrange multiplier for equality constraint, respectively. To calculate the step direction, we first calculate the second order Taylor Series approximation of the Lagrangian and the linear approximations of the constraints in the vicinity of $\mathbf{x_k}$. Defining $\mathbf{d_x^k} = \mathbf{x_{k+1}} - \mathbf{x_k}$ and $H_k = \nabla^2_{xx} L(\mathbf{x_k}, \boldsymbol{\lambda_k}, \boldsymbol{\mu_k})$,

$$L(\mathbf{x_{k+1}}, \boldsymbol{\lambda_k}, \boldsymbol{\mu_k}) \approx L(\mathbf{x_k}, \boldsymbol{\lambda_k}, \boldsymbol{\mu_k}) + \nabla_\mathbf{x} L(\mathbf{x_k}, \boldsymbol{\lambda_k}, \boldsymbol{\mu_k})^\mathbf{T} \mathbf{d_x^k} + \mathbf{d_x^{k^T}} H_k \mathbf{d_x^k} \tag{2.12a}$$

$$h_i(\mathbf{x_{k+1}}) \approx h_i(\mathbf{x_k}) + \nabla h_i(\mathbf{x_k}) \mathbf{d_x^k} \tag{2.12b}$$

$$g_i(\mathbf{x_{k+1}}) \approx g_i(\mathbf{x_k}) + \nabla g_i(\mathbf{x_k}) \mathbf{d_x^k} \tag{2.12c}$$

$$\mathcal{D}\mathbf{h}(\mathbf{x_k}) = \begin{bmatrix} \nabla h_1(\mathbf{x_k})^\mathbf{T} \\ \vdots \\ \nabla h_m(\mathbf{x_k})^\mathbf{T} \end{bmatrix} \tag{2.12d}$$

$$\mathcal{D}\mathbf{g}(\mathbf{x_k}) = \begin{bmatrix} \nabla g_1(\mathbf{x_k})^\mathbf{T} \\ \dots \\ \nabla g_n(\mathbf{x_k})^\mathbf{T} \end{bmatrix} \tag{2.12e}$$

In (2.12), (2.12a) is the approximate Lagrangian. Also, (2.12d) and (2.12e) are the Jacobian matrix of the equality and inequality constraints, respectively. Then, we form the QP subproblem to find the step directions using the approximate Lagrangian

and constraints.

$$\begin{aligned} \min_{\mathbf{d_x^k}} \quad & L(\mathbf{x_k}, \boldsymbol{\lambda_k}, \boldsymbol{\mu_k}) + \nabla_{\mathbf{x}} L(\mathbf{x_k}, \boldsymbol{\lambda_k}, \boldsymbol{\mu_k})^{\mathbf{T}} \mathbf{d_x^k} + \mathbf{d_x^{k^T}} H_k \mathbf{d_x^k} \\ \text{s.t.} \quad & \mathbf{h}(\mathbf{x_k}) + \mathcal{D}\mathbf{h}(\mathbf{x_k}) \mathbf{d_x^k} = \mathbf{0} \\ & \mathbf{g}(\mathbf{x_k}) + \mathcal{D}\mathbf{g}(\mathbf{x_k}) \mathbf{d_x^k} \leq \mathbf{0} \end{aligned} \tag{2.13}$$

Since $L(\mathbf{x_k}, \boldsymbol{\lambda_k}, \boldsymbol{\mu_k})$ is a constant term, we drop it from the QP problem. Also, as long as the QP subproblem is feasible, $\nabla_x L(\mathbf{x_k}, \boldsymbol{\lambda_k}, \boldsymbol{\mu_k})^{\mathbf{T}} \mathbf{d_x^k}$ term becomes equal to $f(\mathbf{x_k}) \mathbf{d_x^k}$, which can be proven by calculating the expression of the gradient of the Lagrangian, substituting the constraints in the expression and incorporating with the KKT conditions. So, the QP subproblem can be written as,

$$\begin{aligned} \min_{\mathbf{d_x^k}} \quad & \nabla f(\mathbf{x_k})^{\mathbf{T}} \mathbf{d_x^k} + \mathbf{d_x^{k^T}} H_k \mathbf{d_x^k} \\ \text{s.t.} \quad & \mathbf{h}(\mathbf{x_k}) + \mathcal{D}\mathbf{h}(\mathbf{x_k}) \mathbf{d_x^k} = \mathbf{0} \\ & \mathbf{g}(\mathbf{x_k}) + \mathcal{D}\mathbf{g}(\mathbf{x_k}) \mathbf{d_x^k} \leq \mathbf{0} \end{aligned} \tag{2.14}$$

Let $\mathbf{d_x^{k^*}} = \mathbf{d_{x,qp}^k}$, $\mathbf{d_\lambda^{k^*}} = \mathbf{d_{\lambda,qp}^k}$ and $\mathbf{d_\mu^{k^*}} = \mathbf{d_{\mu,qp}^k}$ be the solution to the QP subproblem and the corresponding Lagrange multipliers. The SQP algorithm is iterated by updating (2.11) with the new step directions as a result of the QP subproblem. At each step, α_k is also updated with a line search, to maximize the step size. The iterations are terminated when the step size is dropped below a certain threshold.

The SQP algorithm is a very fast algorithm, especially when the cost function is already a quadratic cost function, which is a typical case in the optimal control. The SQP steps are not necessarily feasible due to the QP approximations, but, it guarantees that the optimal solution is feasible. However, since the intermediate steps might be non-feasible, it can diverge quickly especially when the initial guess is non-feasible and for the large scale optimization problems, where the number of decision variables is large. Figure 2.5 shows the general procedure for the SQP algorithm.

2.3.2 Interior Point Algorithm

Another algorithm introduced to solve the nonlinear optimization problems is the Interior Point algorithm. The essence of the Interior Point methods is to use a so called "Logarithmic Barrier Function" to incorporate the inequality constraints to the

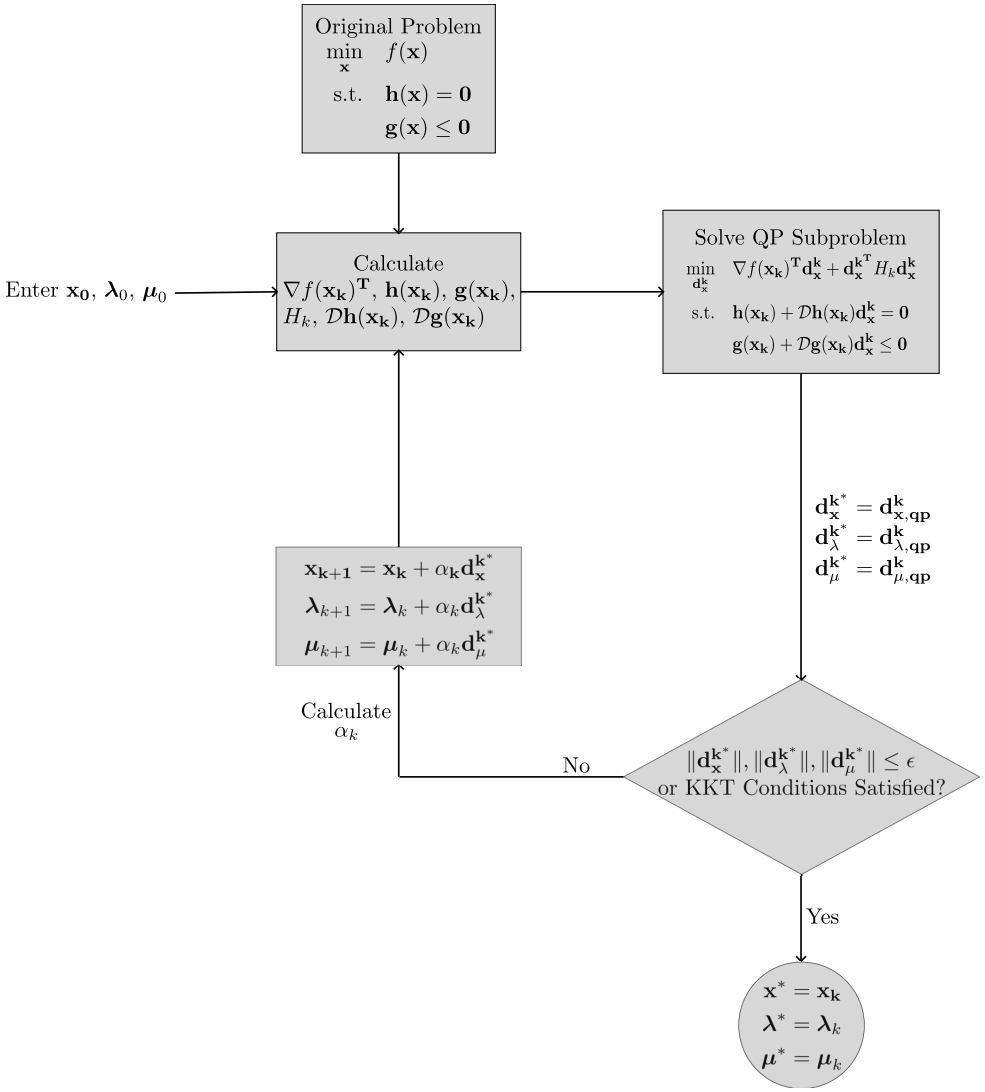


Figure 2.5: Overview of the SQP Algorithm.

cost function. The Logarithmic Barrier Function is given in (2.15).

$$\phi(\mathbf{x}) = -(1/t)\sum_{i=1}^{n} log(-g_i(\mathbf{x})) \tag{2.15}$$

In the barrier function, t is a parameter that adjusts the accuracy, or the sharpness of the function. As the value of $g_i(\mathbf{x})$ approaches to 0, the value of the barrier function goes to infinity, which means, as the constraint reaches the boundary, the value of the barrier function increases rapidly. Also, it is a convex and differentiable function, so it can be added to the cost function. Figure 2.6 shows the logarithmic barrier function for different values of t.

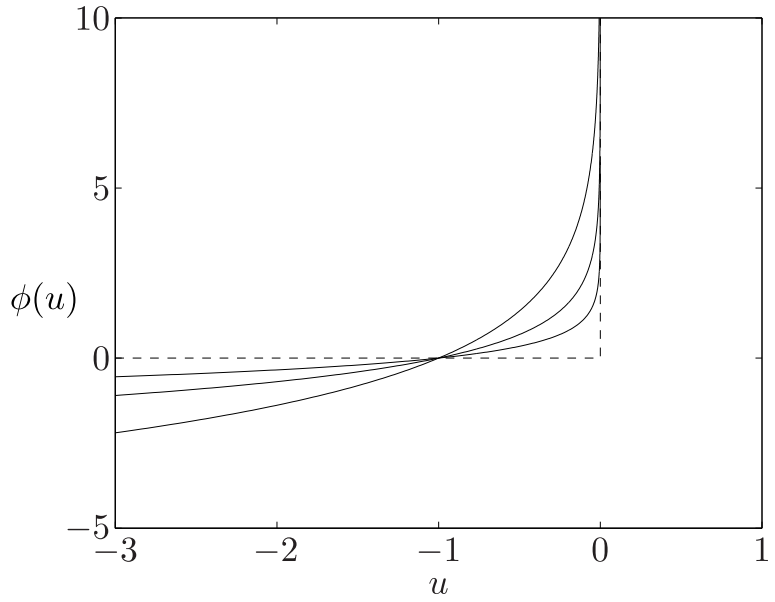


Figure 2.6: The logarithmic barrier function for different values of t [3]. Dashed line shows the ideal case, where $\phi(u)$ is equal to zero as long as $u < 0$ and it is equal to ∞ at $u = 0$. As the value of t increases, the logarithmic barrier function converges to the ideal case.

Adding the barrier function in the original optimization problem in (2.5), the following problem is obtained.

$$\begin{aligned} \min_{\mathbf{x}} \quad & t.f(\mathbf{x}) - \sum_{\mathbf{i=1}}^{\mathbf{n}} log(-g_i(\mathbf{x})) \\ \text{s.t.} \quad & \mathbf{h}(\mathbf{x}) = \mathbf{0} \end{aligned} \tag{2.16}$$

So the corresponding Lagrangian is,

$$\begin{aligned} L(\mathbf{x}, \boldsymbol{\mu}) &= t.f(\mathbf{x}) - \sum_{\mathbf{i=1}}^{\mathbf{n}} log(-g_i(\mathbf{x})) + \boldsymbol{\mu}^{\mathbf{T}}\mathbf{h}(\mathbf{x}) \\ &= t.f(\mathbf{x}) + t.\phi(\mathbf{x}) + \boldsymbol{\mu}^{\mathbf{T}}\mathbf{h}(\mathbf{x}) \end{aligned} \tag{2.17}$$

In this case, since there is no inequality constraint, the KKT conditions reduce to,

$$\nabla L(\mathbf{x}, \boldsymbol{\mu}) = t\nabla f(\mathbf{x}) + t\nabla\phi(\mathbf{x}) + \sum_{\mathbf{i=1}}^{\mathbf{m}} \mu_i \nabla h_i(\mathbf{x}) = \mathrm{o} \tag{2.18a}$$

$$\mathbf{h}(\mathbf{x}) = 0 \tag{2.18b}$$

Again, taking the first order approximations of the KKT conditions around $\mathbf{x_k}$ and

letting $\mathbf{d_x^k} = \mathbf{x_{k+1}} - \mathbf{x_k}$,

$$\nabla L(\mathbf{x_{k+1}}, \boldsymbol{\mu_k}) = \nabla L(\mathbf{x_k}, \boldsymbol{\mu_k}) + \nabla^{\mathbf{2}} L(\mathbf{x_k}, \boldsymbol{\mu_k})\mathbf{d_x^k} = \mathbf{0} \tag{2.19a}$$

$$\mathbf{h}(\mathbf{x_{k+1}}) = \mathbf{h}(\mathbf{x_k}) + \nabla \mathbf{h}(\mathbf{x_k})\mathbf{d_x^k} = \mathbf{0}. \tag{2.19b}$$

Rearranging the approximate KKT conditions given in (2.19), the following system of equations are formed to determine the direction of the Newton step.

$$\begin{aligned} &\begin{bmatrix} t\nabla^2 f(\mathbf{x_k}) + t\nabla^2\phi(\mathbf{x}) & \sum_{i=1}^{m} \mu_i \nabla^2 h(\mathbf{x_k}) \\ \mathcal{D}h(\mathbf{x_k}) & 0 \end{bmatrix} \begin{bmatrix} \mathbf{d_x^k} \\ \mathbf{d_\mu^k} \end{bmatrix} \\ &= - \begin{bmatrix} t\nabla f(\mathbf{x_k}) + \mathbf{t}\nabla\phi(\mathbf{x_k}) + \sum_{\mathbf{i=1}}^{\mathbf{m}} \mu_\mathbf{i} \nabla h(\mathbf{x_k}) \\ \mathbf{h}(\mathbf{x_k}) \end{bmatrix} \end{aligned} \tag{2.20}$$

After the step directions are determined by solving the system of equations in (2.20), the decision variables and Lagrange multipliers are updated in the direction of Newton step.

$$\mathbf{x_{k+1}} = \mathbf{x_k} + \alpha_\mathbf{k}\mathbf{d_x^k} \tag{2.21a}$$

$$\boldsymbol{\mu}_{k+1} = \boldsymbol{\mu}_k + \alpha_k \mathbf{d}_\mu^\mathbf{k} \tag{2.21b}$$

Again, the step size α_k is determined by line search to maximize the step size and the iterations can be terminated when the step size decreases below a specified threshold. Interior point algorithms require higher computational power than the SQP method, but it is way more robust. With the interior point algorithms, each iteration is guaranteed to be feasible even initializing from a non-feasible point. Due to its robustness, it can be utilized to solve large scale optimization problems with a large number of decision variables.

2.4 Parameterizations of System Dynamics for Optimization Problems

When an optimization problem is formulated over a time window, subject to system dynamics, the dynamics of the system is included in the problem as dynamic constraints. In general, when the optimization problem is formulated over nonlinear continuous time dynamics, the problem has infinite number of decision variables at each time instant. However, effective algorithms of nonlinear optimization presented

in Section 2.3 are not formulated for such cases and it is impossible to use them for infinite dimensional optimization problems. In order to use the standard formulations for the optimization problem, we need to parameterize the system dynamics. For continuous time systems, the parameterizations enable the solver to evaluate the dynamic and other constraints at specific time instants. In other words, it acts like a discretization scheme for the continuous time dynamics. For the systems that are already discrete time, these parameterizations create a framework that converts these discrete time dynamic constraints to the standard form. In this section, three of these parameterizations, namely, direct transcription, direct shooting and direct collocation methods, will be investigated. Figure 2.7 shows the overall function of these parameterizations.

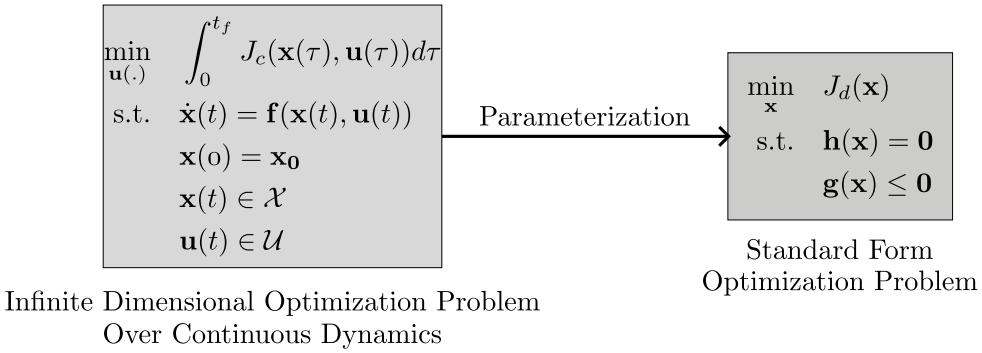


Figure 2.7: Parameterization of infinite dimensional optimization problem to form a standard form nonlinear optimization problem. $J(\mathbf{x}(t), \mathbf{u}(t))$ is the cost function being minimized over the continuous time nonlinear dynamics $\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))$ with initial conditions $\mathbf{x}(0) = \mathbf{x_0}$. $\mathcal{X}$ and $\mathcal{U}$ represents the admissible sets of the state and input variables. After the parameterization, the original problem is converted to the standard form optimization problem with finite number of decision variables.

2.4.1 Direct Transcription Method

With the direct transcription method, the state variables at each time instant is also included in the optimization problem as decision variables. First, we discretize the system dynamics with a sampling period of T. At each sampling period, we also keep

the input constant such that $\mathbf{u}(t) = \mathbf{u}[k], \quad t \in [kT,\ (k+1)T]$.

$$\mathbf{x[k+1]} = \mathbf{x[k]} + \int_{t_o+kT}^{t_o+(k+1)T} \mathbf{f}(\mathbf{x}(\tau), \mathbf{u[k]})d\tau \tag{2.22}$$

Then, we recursively construct the dynamics starting from $\mathbf{x(0) = x_0}$. Assuming that the time window that we are trying to conduct the optimization is $t \in [t_0,\ nT]$.

$$\mathbf{x[0]} = \mathbf{x}(t_o) \tag{2.23a}$$

$$\mathbf{x[1]} = \mathbf{x[0]} + \int_{t_o}^{t_o+T} \mathbf{f}(\mathbf{x}(\tau), \mathbf{u[0]})d\tau \tag{2.23b}$$

$$\mathbf{x[2]} = \mathbf{x[1]} + \int_{t_o+T}^{t_o+2T} \mathbf{f}(\mathbf{x}(\tau), \mathbf{u[1]})d\tau \tag{2.23c}$$

$$\vdots \tag{2.23d}$$

$$\mathbf{x[n]} = \mathbf{x[n-1]} + \int_{t_o+(n-1)T}^{t_o+nT} \mathbf{f}(\mathbf{x}(\tau), \mathbf{u[n-1]})d\tau \tag{2.23e}$$

Here nT is called the prediction horizon for the continuous time systems and n is the prediction horizon for the discrete time systems. Assuming we have m state and p input variables, the vector of decision variables take the following form for a single sampling period.

$$\mathbf{x_k} = \begin{bmatrix} \mathbf{x[k]} \\ \mathbf{u[k]} \end{bmatrix} = \begin{bmatrix} x_1[k] \\ \vdots \\ x_m[k] \\ u_1[k] \\ \vdots \\ u_p[k] \end{bmatrix} \tag{2.24}$$

In (2.24), $x_i[k]$ and $u_j[k]$ are the value of i^{th} and j^{th} state and input variables at k^{th} sample, respectively. So, the overall vector of decision variables become,

$$\mathbf{x} = \begin{bmatrix} \mathbf{x_0} \\ \mathbf{x_1} \\ \vdots \\ \mathbf{x_n} \end{bmatrix} \tag{2.25}$$

with $\mathbf{x_n} = \left[x_1[n] \dots x_m[n]\right]^{\mathbf{T}}$, not including the input variables. After all, the equality constraint for the nonlinear optimization problem can be written as,

$$\mathbf{h}(\mathbf{x}) = \begin{bmatrix} x_1[0] - x_1(0) \\ \vdots \\ x_m[0] - x_m(0) \\ x_1[1] - x_1[0] - \int_{t_0}^{t_0+T} f_1(\mathbf{x}(\tau), \mathbf{u}[\mathbf{0}])d\tau \\ \vdots \\ x_m[1] - x_m[0] - \int_{t_0}^{t_0+T} f_m(\mathbf{x}(\tau), \mathbf{u}[\mathbf{0}])d\tau \\ \vdots \\ x_1[n] - x_1[n-1] - \int_{t_0+(n-1)T}^{t_0+nT} f_1(\mathbf{x}(\tau), \mathbf{u}[\mathbf{n-1}])d\tau \\ \vdots \\ x_m[n] - x_m[n-1] - \int_{t_0+(n-1)T}^{t_0+nT} f_m(\mathbf{x}(\tau), \mathbf{u}[\mathbf{n-1}])d\tau \end{bmatrix} = \mathbf{0} \tag{2.26}$$

for a system with continuous time nonlinear dynamics. For the systems that is already discrete time, the expression for the equality constraint is straightforward.

$$\mathbf{h}(\mathbf{x}) = \begin{bmatrix} x_1[0] - x_1(0) \\ \vdots \\ x_m[0] - x_m(0) \\ x_1[1] - f_1(\mathbf{x}[\mathbf{0}], \mathbf{u}[\mathbf{0}]) \\ \vdots \\ x_m[1] - f_m(\mathbf{x}[\mathbf{0}], \mathbf{u}[\mathbf{0}]) \\ \vdots \\ x_1[n] - f_1(\mathbf{x}[\mathbf{n-1}], \mathbf{u}[\mathbf{n-1}]) \\ \vdots \\ x_m[n] - f_m(\mathbf{x}[\mathbf{n-1}], \mathbf{u}[\mathbf{n-1}]) \end{bmatrix} = \mathbf{0} \tag{2.27}$$

Another issue with the parameterizations is the parameterizaton of the cost function. For the integral cost, we define the accumulating cost for the parameterization.

$$J_d(\mathbf{x}) = \sum_{\mathbf{k=0}}^{\mathbf{n}} J_c(\mathbf{x_k}) \tag{2.28}$$

Finally, for the admissible values of the state and input variables, we generate the inequality constraints as follows.

$$\mathbf{g}(\mathbf{x}) = \begin{bmatrix} \mathbf{g_{x_0,max}}(\mathbf{x[0]}) \\ \mathbf{g_{u_0,max}}(\mathbf{u[0]}) \\ \mathbf{g_{x_0,min}}(\mathbf{x[0]}) \\ \mathbf{g_{u_0,min}}(\mathbf{u[0]}) \\ \vdots \\ \mathbf{g_{x_n,max}}(\mathbf{x[n]}) \\ \mathbf{g_{u_n,max}}(\mathbf{u[n]}) \\ \mathbf{g_{x_n,min}}(\mathbf{x[n]}) \\ \mathbf{g_{u_n,min}}(\mathbf{u[n]}) \end{bmatrix} \leq \mathbf{0} \tag{2.29}$$

2.4.2 Direct Single Shooting Method

For a given dynamic system, we can determine the state trajectories using only the initial conditions, input trajectory and the dynamic model. So, we don't need to impose the state trajectory as a constraint and add all the state variables to the decision variables of the nonlinear optimizer. With the Direct Single Shooting method, we simulate forwards the system using the initial conditions and initial guess of the input trajectory to get rid of the dynamic constraints. To utilize this method, we first partition the time horizon into time-steps $t_0 < t_1 < \cdots < t_k < \cdots < t_n$. At each partition interval, we assume that the input is constant such that $\mathbf{u}(t) = \mathbf{u}[k], \quad t \in [kT,\ (k+1)T]$. Let $\mathcal{U}^k = [u_1(t_0), \ldots, u_p(t_0), u_1(t_1), \ldots, u_p(t_1), \ldots, u_1(t_k), \ldots, u_p(t_k)]^T$ be the vector of input trajectory between t_0 and t_k for $k \in [0\ n]$. We can forward simulate the system to find the state values at the end of each time partition. Let $\mathcal{X}^n$ be

the vector of values of the state variables at the end of each time partition until t_n.

$$\mathcal{X}^n = \begin{bmatrix} \mathbf{x}(t_0) \\ \mathbf{x}(t_1) \\ \vdots \\ \mathbf{x}(t_n) \end{bmatrix} = \begin{bmatrix} x_1(t_0) \\ \vdots \\ x_m(t_0) \\ \int_{t_0}^{t_1} f_1(\mathbf{x}(\tau), \mathcal{U}^0) d\tau \\ \vdots \\ \int_{t_0}^{t_1} f_m(\mathbf{x}(\tau), \mathcal{U}^0) d\tau \\ \vdots \\ \int_{t_0}^{t_n} f_1(\mathbf{x}(\tau), \mathcal{U}^{n-1}) d\tau \\ \vdots \\ \int_{t_0}^{t_n} f_m(\mathbf{x}(\tau), \mathcal{U}^{n-1}) d\tau \end{bmatrix} \tag{2.30}$$

So, the cost function of the nonlinear optimization problem becomes,

$$J_d(\mathbf{x}(t_0), \mathcal{U}^n) = \sum_{k=0}^{n} J_c(\mathcal{X}_k^n, \mathcal{U}_k^n) \tag{2.31}$$

Since we formulate our cost function only in terms of initial condition $\mathbf{x_0}$ and past inputs $\mathcal{U}^k$, we don't have any equality constraint in this formulation. For the admissible values of the state and input variables,

$$\mathbf{g}(\mathbf{x}) = \mathbf{g}(\mathbf{x}(t_0), \mathcal{U}^{n-1}) = \begin{bmatrix} \mathbf{g_x}(\mathcal{X}^n) \\ \mathbf{g_u}(\mathcal{U}^{n-1}) \end{bmatrix} \leq \mathbf{0} \tag{2.32}$$

We can derive a similar expression for the discrete time systems by changing integrals to the sums. Assuming each partition interval has the same length $t_k - t_{k-1} = T$, let $\mathbf{u}[\mathbf{k}] = \mathbf{u}(\mathbf{kT})$ for $k \in [0,\ n]$, and

$$\hat{\mathcal{X}}^n = \begin{bmatrix} \mathbf{x}[0] \\ \mathbf{x}[1] \\ \vdots \\ \mathbf{x}[n] \end{bmatrix} \qquad \hat{\mathcal{U}}^{n-1} = \begin{bmatrix} \mathbf{u}[0] \\ \mathbf{u}[1] \\ \vdots \\ \mathbf{u}[n-1] \end{bmatrix} \tag{2.33}$$

From the forward simulation, we obtain 2.34.

$$\begin{aligned}
&\mathbf{x}[\mathbf{0}] = \mathbf{x_0} \\
&\mathbf{x}[\mathbf{1}] = \mathbf{f}(\mathbf{x}[\mathbf{0}], \mathbf{u}[\mathbf{0}]) \\
&\mathbf{x}[\mathbf{2}] = \mathbf{f}(\mathbf{f}(\mathbf{x}[\mathbf{0}], \mathbf{u}[\mathbf{0}]), \mathbf{u}[\mathbf{1}]) \\
&\quad \vdots \\
&\mathbf{x}[\mathbf{n}] = \mathbf{f}(\mathbf{x}[\mathbf{n-1}], \mathcal{U}^{n-1}) = \mathbf{f}(\mathbf{f}(\dots \mathbf{f}(\mathbf{x}[\mathbf{0}], \mathbf{u}[\mathbf{0}]), \mathbf{u}[\mathbf{1}]) \dots \mathbf{u}[\mathbf{n-1}])
\end{aligned} \tag{2.34}$$

Again, the inequality constraints and the cost function can be calculated as 2.35 and 2.36, respectively.

$$\mathbf{g}(\mathbf{x}) = \mathbf{g}(\mathbf{x}[\mathbf{0}], \hat{\mathcal{U}}^{n-1}) = \begin{bmatrix} \mathbf{g_x}(\hat{\mathcal{X}}^n) \\ \mathbf{g_u}(\hat{\mathcal{U}}^{n-1}) \end{bmatrix} \leq \mathbf{0} \tag{2.35}$$

$$J_d(\mathbf{x}[\mathbf{0}], \mathcal{U}^{n-1}) = \sum_{k=0}^{n} J_c(\hat{\mathcal{X}}_k^n, \hat{\mathcal{U}}_k^{n-1}) \tag{2.36}$$

One disadvantage of this parameterization is that the first input $\mathbf{u}[\mathbf{0}]$ or $\mathbf{u}(t_0)$ enters all the terms in the formulation. Hence, the optimizer focuses more on the first input and gives less attention to the preceding inputs, causing a "vanishing gradients" situation. So, it is not suggested to use this method for very long prediction horizons. On the other hand, the time horizon and the time partition can be dynamically adjusted in the optimizer using this parameterization. Thus, it can be used for the systems where the time horizon is not known a priori.

2.4.3 Direct Collocation Method

In the direct collocation method, instead of integrating the nonlinear differential equations during the optimization process, the solution of the differential equations are approximated using 3rd order splines for the state trajectories and 1st order splines for the input trajectories. After that, the state equations are evaluated at so-called "collocation" points. After the dynamics are evaluated, they are imposed as an equality constraint to the optimizer. Let, $x[k]$ and $u[k]$ be the knot points for the state trajectory and input trajectory splines, respectively. Here, k is the discretization index such that $t_k = t_0 + kT$ with T being the time step. So, the splines that approximate state

and input trajectories can be expressed as (2.37). Figure 2.8 shows the state trajectory spline, collocation and knot (sample) points. Finally, the dynamics satisfy (2.38).

$$u_i(t_{c,k}) = 1/2(u_i[k] + u_i[k+1]), \quad i \in [1,\ p] \tag{2.37a}$$

$$x_j(t_{c,k}) = (1/2)(x_j[k] + x_j[k+1]) + (h/8)(\dot{x}_j[k] - \dot{x}_j[k+1]), \quad j \in [1,\ m] \tag{2.37b}$$

$$\dot{x}_j(t_{c,k}) = (-2/3h)(x_j[k] - x_j[k+1]) - (1/4)(\dot{x}_j[k] + \dot{x}_j[k+1]), \quad j \in [1,\ m] \tag{2.37c}$$

$$\dot{x}_j(t_{c,k}) = f_j(\mathbf{x}(t_{c,k}), \mathbf{u}(t_{c,k})) \tag{2.38}$$

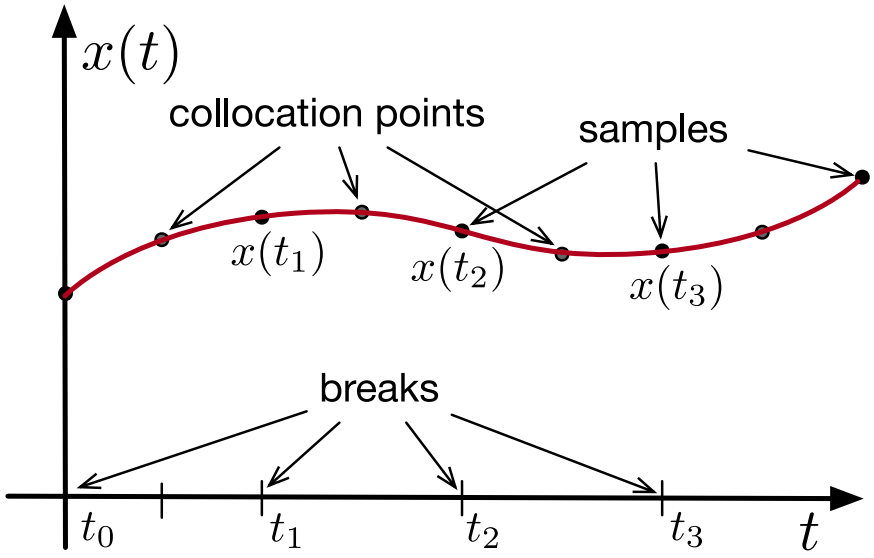


Figure 2.8: Overview of the spline fitting to the state trajectories [4]. Samples $x(t_1) = x[1]$, $x(t_2) = x[2]$ and $x(t_3) = x[3]$ are calculated at the knot points of the spline and used to evaluate the cost function and any additional inequality constraint relating the admissible values of the state and input variables. The dynamics are evaluated at the collocation points $x(t_{c,1})$, $x(t_{c,2})$ and $x(t_{c,3})$ etc.

With this parameterization, we can define our decision parameters as given in (2.39).

$$\mathbf{x} = \begin{bmatrix} x_1[0] \\ \vdots \\ x_m[0] \\ \vdots \\ x_1[n] \\ \vdots \\ x_m[n] \\ u_1[0] \\ \vdots \\ u_p[0] \\ \vdots \\ u_1[n] \\ \vdots \\ u_p[n] \end{bmatrix} \tag{2.39}$$

Thus, we can write equality constraints for our optimization problem as given in (2.40).

$$\mathbf{h}(\mathbf{x}) = \begin{bmatrix} x_1[0] - x_1(t_0) \\ \vdots \\ x_m[0] - x_m(t_0) \\ \dot{x}_1(t_{c,0}) - f_1(\mathbf{x}(t_{c,0}), \mathbf{u}(t_{c,0})) \\ \vdots \\ \dot{x}_m(t_{c,0}) - f_m(\mathbf{x}(t_{c,0}), \mathbf{u}(t_{c,0})) \\ \vdots \\ \dot{x}_1(t_{c,k}) - f_1(\mathbf{x}(t_{c,k}), \mathbf{u}(t_{c,k})) \\ \vdots \\ \dot{x}_m(t_{c,k}) - f_m(\mathbf{x}(t_{c,k}), \mathbf{u}(t_{c,k})) \\ \vdots \\ \dot{x}_1(t_{c,n}) - f_1(\mathbf{x}(t_{c,n}), \mathbf{u}(t_{c,n})) \\ \vdots \\ \dot{x}_m(t_{c,n}) - f_m(\mathbf{x}(t_{c,n}), \mathbf{u}(t_{c,n})) \end{bmatrix} = \mathbf{0} \tag{2.40}$$

From the expressions, we can see that the evaluation of the state dynamics at the

collocation points is no longer a differential equation, it is just a nonlinear function of several variables and can be computed just in terms of $x[k]$, $u[k]$, $x[k+1]$ and $u[k+1]$. Also, we can evaluate any other inequality constraint $\mathbf{g}(\mathbf{x}) \leq \mathbf{0}$ at the knot points $\mathbf{x}[\mathbf{k}]$ and $\mathbf{u}[\mathbf{k}]$. Finally, we can compute the cost function as given in (2.41).

$$J_d = \sum_{n=0}^{n} T J_c(x[n], u[n]) \tag{2.41}$$

The direct collocation method has the same number of decision variables as the direct transcription method. However, it does not involve any numerical integration step, which significantly decreases the computational cost for solving the optimization problem. As the knot points get closer to each other (decreasing time step T), the approximation of the collocation method converges to the direct transcription method. The method is developed solely to solve continuous time problems, so, there is no discrete time counterpart for this problem.

2.5 Linear Time Varying Model Predictive Control

In our method, after we find the nominal trajectories using the trajectory optimization, we linearize the nonlinear dynamics of the USV around the nominal trajectories, resulting in the time varying dynamics. Then, we control the system over the nominal trajectories using Model Predictive Control (MPC). For the linear time-varying (LTV) MPC problem imposed over the continuous time dynamics, the following optimization problem is solved at each prediction horizon.

$$\min_{u(.)} \quad \int_0^{T_P} [x^T(t+\tau|t)Qx(t+\tau|t) + u^T(t+\tau|t)Ru(t+\tau|t)]\, d\tau \tag{2.42a}$$

$$\text{s.t.} \quad \dot{x}(k|t) = A(k|t)x(k|t) + B(k|t)u(k|t), \quad k = [t, t+T_P], \tag{2.42b}$$

$$u(k|t) \in \mathcal{U}, \quad k = [t, t+T_P], \tag{2.42c}$$

$$x(k|t) \in \mathcal{X}, \quad k = [t, t+T_P], \tag{2.42d}$$

$$x(t|t) = x(t). \tag{2.42e}$$

Here, $x(k|t)$ and $u(k|t)$ denotes the predicted state and input values at prediction time k and the starting time for the prediction t, respectively. $A(k|t)$ and $B(k|t)$ represents

the linear time varying dynamics of the system starting from time t along the prediction horizon. Also, $\mathcal{X}$ and $\mathcal{U}$ are the set of admissible state and input values, i.e., the state and input constraints, respectively. Finally, Q and R are the state and input penalty matrices and T_P is the prediction horizon. For the continuous time systems, the MPC problem can be solved with linear counterparts of the parameterizations described in the previous section, without the need of any discretization, and these formulations will be explained further in the preceeding sections. For the systems already represented in discrete time, the following optimization problem can be solved, again, by the parameterizations represented previously.

$$\min_{u[.]} \quad \sum_{k_p=0}^{N_P} x^T[k+k_p|k]\ Q\ x[k+k_p|k] + u^T[k+k_p|k]\ R\ u[k+k_p|k] \tag{2.43a}$$

$$\text{s.t.} \quad x[k_p+1|k] = A[k_p|k]\ x[k_p|k] + B[k_p|k]\ u[k_p|k], \quad k_p = [k, k+N_P], \tag{2.43b}$$

$$u[k_p|k] \in \mathcal{U}, \quad k_p = [k, k+N_P], \tag{2.43c}$$

$$x[k_p|k] \in \mathcal{X}, \quad k_p = [k, k+N_P], \tag{2.43d}$$

$$x[k|k] = x[k]. \tag{2.43e}$$

At time t for the continuous time system and k for the discrete time system, the optimization problems (2.42) and (2.43) are solved. After solving each problem, for the continuous time system, first T_c seconds of the optimal input trajectory is applied to the system. Similarly, for the discrete time system, first N_c steps of optimal input sequence is applied to the system. Here, T_c and N_c are called the control horizon. After applying these inputs, the MPC problem is solved again and again for each $t + T_c$ and $k + N_c$.

2.6 USV Dynamics

To model the USV dynamics, Fossen's equations of motion for surface vehicles will be used [46]. Let $\boldsymbol{v} = \begin{bmatrix} u & v & r \end{bmatrix}^T$ denote the velocity vector of the USV with respect to the body frame. So, the equations of motion for the rigid body can be

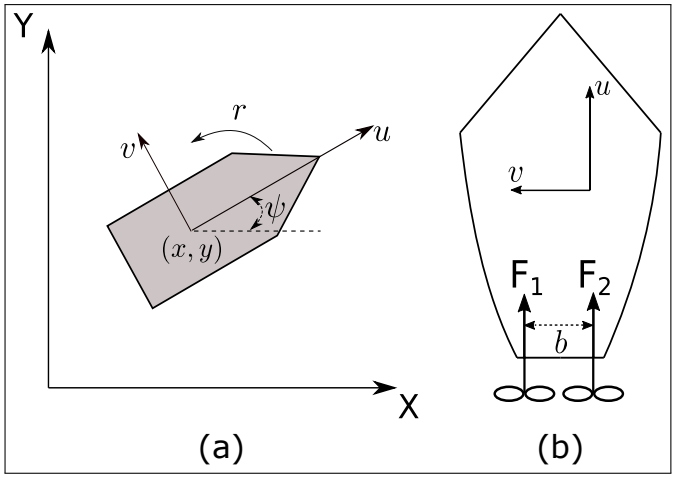


Figure 2.9: The illustration of the USV with respect to the body and global frames.

expressed as (2.44).

$$\mathbf{M}\dot{\mathbf{v}} + \mathbf{C}(\mathbf{v})\mathbf{v} + \mathbf{D}(\mathbf{v})\mathbf{v} = \tau \tag{2.44}$$

with,

$$\boldsymbol{M} = \boldsymbol{M}_A + \boldsymbol{M}_{RB} \tag{2.45a}$$

$$\boldsymbol{C}(\boldsymbol{v}) = \boldsymbol{C}_A(\boldsymbol{v}) + \boldsymbol{C}_{RB}(\boldsymbol{v}) \tag{2.45b}$$

$$\boldsymbol{D}(\boldsymbol{v}) = \boldsymbol{D}_l + \boldsymbol{D}_n(\boldsymbol{v}) \tag{2.45c}$$

and,

$$\boldsymbol{M}_{RB} = \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & I_z \end{bmatrix} \tag{2.46}$$

$$\boldsymbol{M}_A = \begin{bmatrix} X_{\dot{u}} & 0 & 0 \\ 0 & Y_{\dot{v}} & Y_{\dot{r}} \\ 0 & Y_{\dot{r}} & N_{\dot{r}} \end{bmatrix} \tag{2.47}$$

$$\boldsymbol{C}_A = \begin{bmatrix} 0 & 0 & Y_{\dot{v}}v + Y_{\dot{r}}r \\ 0 & 0 & -X_{\dot{u}}u \\ -Y_{\dot{v}}v - Y_{\dot{r}}r & X_{\dot{u}}u & 0 \end{bmatrix} \tag{2.48}$$

$$\boldsymbol{C}_{RB} = \begin{bmatrix} 0 & 0 & -mv \\ 0 & 0 & mu \\ mv & -mu & 0 \end{bmatrix} \tag{2.49}$$

$$\boldsymbol{D}_l = \begin{bmatrix} X_u & 0 & 0 \\ 0 & Y_v & Y_r \\ 0 & N_v & N_r \end{bmatrix} \tag{2.50}$$

$$\boldsymbol{D}_n = \begin{bmatrix} X_{|u|u}|u| & 0 & 0 \\ 0 & Y_{|v|v}|v| & 0 \\ 0 & 0 & N_{|r|r}|r|\ | \end{bmatrix} \tag{2.51}$$

In this formulation, $\mathbf{M}_{RB}$ and $\mathbf{C}_{RB}$ represent the mass and inertia of the rigid body, and the Coriolis forces due to the rigid body motion, respectively. On the other hand, $\mathbf{M}_A$ and $\mathbf{C}_A$ represent the added mass and inertia, and the Coriolis forces due to the added mass of the USV due to the hydrodynamics of the USV, respectively. Finally, $\mathbf{D}_l$ and $\mathbf{D}_n$ represent the linear and nonlinear damping forces acting on the USV, respectively. In these expressions, m is the mass and I_z is the moment of inertia of the vehicle perpendicular to the horizontal plane, $\{X_{\dot{u}}, Y_{\dot{v}}, Y_{\dot{r}}, N_{\dot{r}}, N_{\dot{v}}\}$, $\{X_u, Y_v, Y_r, N_v, N_r\}$ and $\left\{X_{|u|u}, Y_{|v|v}, N_{|r|r}\right\}$ are added mass, linear damping and nonlinear damping parameters, respectively. These parameters are scalar constants that do not change over time. In order to control the position of the USV with respect to the global reference frame, a simple coordinate transformation can be applied to the system. Let $\boldsymbol{\eta} = \begin{bmatrix} x & y & \psi \end{bmatrix}^T$ denote the position and orientation of the USV with respect to the global frame as shown in Figure 2.9. A simple rotational transformation dictates (2.52) with the transformation matrix represented in (2.53).

$$\dot{\boldsymbol{\eta}} = \boldsymbol{J}(\psi)\boldsymbol{v} \tag{2.52}$$

$$\boldsymbol{J}(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.53}$$

So, the overall state space becomes as (2.54).

$$\dot{\boldsymbol{\eta}} = \boldsymbol{J}(\psi)\boldsymbol{v} \tag{2.54a}$$

$$\dot{\mathbf{v}} = \mathbf{M}^{-1}\boldsymbol{\tau} - \mathbf{M}^{-1}(\mathbf{C}(\mathbf{v})\mathbf{v} + \mathbf{D}(\mathbf{v}))\mathbf{v} \tag{2.54b}$$

Here, τ represents the input thrust vector to the system and it is defined as 2.55.

$$\boldsymbol{\tau} = \begin{bmatrix} F_1 + F_2 \\ 0 \\ b(F_1 - F_2) \end{bmatrix} \tag{2.55}$$

Notice that there is no input thrust in the direction of v due to the underactuation.

2.7 NARMAX Methods

The method developed in this work uses the model of the system heavily, during both the motion planning and control phases. For the implementation of the method, we selected the domain of USVs. We implemented our method for two USV models. One of the USVs is the Clearpath Robotics Heron USV and there is no dynamic model presented in the literature for it. So, in order to implement our method to the Heron USV, we needed to conduct system identification to the nonlinear dynamics of the USV. To model the nonlinear dynamics, we selected the Nonlinear Auto Regressive Moving Average with Exogenous Inputs (NARMAX) model [47]. NARMAX models generate the system dynamics dependent on the nonlinear combination of previous state, input and error (noise) terms as presented in (2.56).

$$\begin{aligned} \mathbf{x[k]} = \mathbf{F}(&\mathbf{x[k-1]}, \mathbf{x[k-2]}, \ldots, \mathbf{x[k-n_x]}, \\ &\mathbf{u[k-1]}, \mathbf{u[k-2]}, \ldots, \mathbf{u[k-n_u]}, \\ &\mathbf{e[k-1]}, \mathbf{e[k-2]}, \ldots, \mathbf{e[k-n_e]}) \end{aligned} \tag{2.56}$$

In (2.56), $\mathbf{x[k]}$, $\mathbf{u[k]}$ and $\mathbf{e[k]}$ represents the vectors of state, input and error variables, respectively. Also, $\mathbf{n_x}$, $\mathbf{n_u}$ and $\mathbf{n_e}$ are the maximum lags for the state, input and error, and they determine how many past terms will influence the next value of the state. For convenience, we can define the NARMAX model as in (2.57).

$$\mathbf{x[k]} = \mathbf{F}(\mathbf{X^{n_x}}, \mathbf{U^{n_u}}, \mathbf{E^{n_e}}) \tag{2.57}$$

In this formulation, $\mathbf{X^{n_x}} = \begin{bmatrix} \mathbf{x[k-1]} & \mathbf{x[k-2]} & \ldots & \mathbf{x[k-n_x]} \end{bmatrix}^{\mathbf{T}}$ is the delayed state sequence up to $\mathbf{n_x}$, $\mathbf{U^{n_u}} = \begin{bmatrix} \mathbf{u[k-1]} & \mathbf{u[k-2]} & \ldots & \mathbf{u[k-n_u]} \end{bmatrix}^{\mathbf{T}}$ is the delayed input sequence up to $\mathbf{n_u}$ and $\mathbf{E^{n_e}} = \begin{bmatrix} \mathbf{e[k-1]} & \mathbf{e[k-2]} & \ldots & \mathbf{e[k-n_e]} \end{bmatrix}^{\mathbf{T}}$ is

the delayed error sequence up to $\mathbf{n_e}$. One particularly useful form of the NARMAX models is the so called "Linear in the Parameters" representation. In this form, the model parameters enter the nonlinear dynamics linearly as shown in (2.58).

$$x_i[k] = \sum_{i=1}^{P} \theta_i F_i(\mathbf{X^{n_x}}, \mathbf{U^{n_u}}, \mathbf{E^{n_e}}) \tag{2.58}$$

Since the parameters enter linearly to the dynamics in this representation, we can use the powerful tools of the linear system identification to identify the nonlinear dynamics [48]. We used this representation to identify the Fossen dynamics of the Heron USV. The details of the identification process will be presented in Chapter 4.

CHAPTER 3

METHODOLOGY

For the complex nonlinear underactuted systems, it is hard to implement the classical sequential composition method for the motion planning and control since these systems are not controllable or even stabilizable around a single operation point. So, it is hard to find a local feedback controller that will drive the system to a local operation point, i.e., the outlet of the funnel, or the resulting controllers are infeasible to use due to their computational complexity. In [42], the outlet of the funnels that LQR-Tree algorithm generates is a time varying operating point, that serves as a preplanned trajectory, not a single stationary operation point. So, applying the LQR control policy to the linearized system around these trajectories, the system is stabilized and the system is successfully driven to the goal over these trajectories. In our method, we again generate a complete trajectory on the configuration space of the underactuated system, using the receding horizon trajectory optimization explained in the next section. Our method first generates the simple rectangular obstacle free regions that serves as the funnels in the 2D workspace of the system using the Sparse Neighborhood Graphs and generates the waypoints for the algorithm. Next, we conduct a trajectory optimization between these waypoints employing a receding horizon strategy to generate the state and input trajectories for the system. With the receding horizon strategy, we look ahead one node further to predict the further trajectories of the system while generating trajectories inside the active node. By generating these trajectories inside each node and combining these trajectories to a single trajectory that connects the starting configuration to the goal, we are decreasing the computational burden for the online controller. The concept of receding horizon trajectory optimization is illustrated in Figure 3.1.

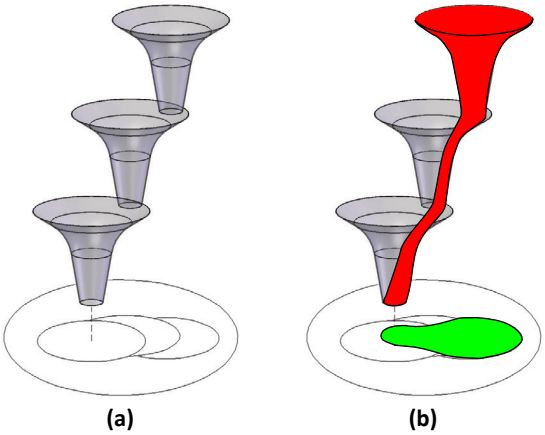


Figure 3.1: Illustration of the trajectory optimization. (a) shows the classical sequential composition of the funnels. (b) shows the resulting trajectories as a result of trajectory optimization inside the funnels.

The generated trajectory serves as a stabilizable trajectory for the underactuated system. We generate these trajectories considering the system dynamics and the configuration space constraints, in an optimal way. Thus, it is easier to find a stabilizer feedback control policy for the system around these trajectories. We implement a linear time-varying Model Predictive Control policy over the linearized system dynamics to stabilize the system around generated trajectories. In the MPC formulation, we impose the requirement of staying inside the funnels to the system, to successfully reach to goal configuration. We know that, as long as the MPC problem is feasible, it is guaranteed that the system stays within the obstacle free region of the workspace.

3.1 Receding Horizon Trajectory Optimization on Sparse Neighborhood Graphs

In this study, after generating the waypoints using the SNG algorithm, we used a receding horizon trajectory optimization method using the nonlinear dynamics to find the nominal speed and input trajectories inside the nodes along the shortest path.

Using a receding horizon approach in the trajectory optimization, we can look ahead the next node while planning for the current node, resulting in smoother trajectories and decreasing the computational burden on the controller. Note that, in this study, we only considered the next node during the receding horizon planning, but it can be

generalized for more than one consecutive node. Using such a method, we are able to find single and smooth state and input trajectories that connects the starting configuration to the goal configuration inside the obstacle free region of the workspace. In this study, we selected the underactuated system for this method as USV, so, we formulated the trajectory optimization in the domain of USV's with the convention given in Section 2.6. For this purpose, at each node, we iteratively solved the optimization problem given in (3.1) that minimizes the input effort for the nodes along the shortest path, i.e., for $Node_i$ starting from $i = 1$ to $i = \mathcal{N}$.

$$\min_{u(.)} \quad \int_{t_{0,i}}^{t_{0,i}+T_{pr,i}} u^T(\tau) R u(\tau)\, d\tau \tag{3.1a}$$

$$\text{s.t.} \quad \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}), \tag{3.1b}$$

$$\boldsymbol{\eta}(t) \in Node_i, \quad t \in [t_{0,i},\ t_{0,i} + T_{pl,i}], \tag{3.1c}$$

$$\boldsymbol{\eta}(t) \in Node_{i+1}, \tag{3.1d}$$
$$t \in [t_{0,i} + T_{pl,i},\ t_{0,i} + T_{pr,i}],$$

$$\boldsymbol{v}_{min} \leq \boldsymbol{v}(t) \leq \boldsymbol{v}_{max}, \tag{3.1e}$$
$$t \in [t_{0,i},\ t_{0,i} + T_{pr,i}],$$

$$\boldsymbol{u}_{min} \leq \boldsymbol{u}(t) \leq \boldsymbol{u}_{max}, \tag{3.1f}$$
$$t \in [t_{0,i},\ t_{0,i} + T_{pr,i}],$$

$$\boldsymbol{x}(t_{0,i}) = \boldsymbol{x}_{0,i}, \tag{3.1g}$$

$$\begin{bmatrix} x(t_{0,i} + T_{pl,i}) \\ y(t_{0,i} + T_{pl,i}) \end{bmatrix} = q_i, \tag{3.1h}$$

$$\begin{bmatrix} x(t_{0,i} + T_{pr,i}) \\ y(t_{0,i} + T_{pr,i}) \end{bmatrix} = q_{i+1}. \tag{3.1i}$$

In this formulation, (3.1b) corresponds to the nonlinear system dynamics of the underactuated system. Also, T_{pr} is the prediction horizon and T_{pl} is the planning horizon which are calculated as follows in the domain of USVs.

$$T_{pl,i} = \frac{||q_i - q_{i-1}||_2}{v_{des}}, \tag{3.2a}$$

$$T_{pr,i} = \frac{||q_{i+1} - q_{i-1}||_2 + ||q_{i+1,i+2} - q_{i,i+1}||_2}{v_{des}}, \tag{3.2b}$$

where v_{des} is the desired surge speed of the USV. We multiply these horizons with a coefficient $\mu \in [2, 3]$ at the starting and goal nodes to give enough time for ac-

celeration and deceleration. Let $\boldsymbol{\eta}_{i,i+1}(t)$ and $\boldsymbol{v}_{i,i+1}(t)$ denote the planned position and velocity trajectories of the USV in the domain $[t_{0,i},\ t_{0,i} + T_{pl,i}]$ and $t_{0,i}$ is the initial time for the planned trajectories inside $Node_i$ of the shortest path. Then, the following recursion occurs.

$$t_{0,i} = t_{0,i-1} + T_{pl,i-1}, \tag{3.3a}$$

$$\boldsymbol{x}(t_{0,i}) = \boldsymbol{x}_{0,i} = \boldsymbol{x}(t_{0,i-1} + T_{pl,i-1}) \tag{3.3b}$$

$$= \begin{bmatrix} \boldsymbol{\eta}_{i-1,i}(t_{0,i-1} + T_{pl,i-1}) \\ \boldsymbol{v}_{i-1,i}(t_{0,i-1} + T_{pl,i-1}) \end{bmatrix}. \tag{3.3c}$$

For $i = 1$ (starting node), and $i = \mathcal{N}$ (goal node), we have the following equalities.

$$t_{0,1} = 0, \tag{3.4a}$$

$$\boldsymbol{x}(t_{0,1}) = \begin{bmatrix} q_{start} \\ \psi_{start} \\ 0 \end{bmatrix}, \tag{3.4b}$$

$$\boldsymbol{x}(t_{0,\mathcal{N}} + T_{pl,\mathcal{N}}) = \boldsymbol{x}(t_{0,\mathcal{N}} + T_{pr,\mathcal{N}}) \tag{3.4c}$$

$$= \begin{bmatrix} q_{goal} \\ \psi_{goal} \\ 0 \end{bmatrix}.$$

Also, we keep the time spent on each node by the nominal trajectories to use as a time varying constraint in the MPC in the form of a piecewise constant time function.

$$Node(t) = Node_i, \quad t = [t_{0,i},\ t_{0,i} + T_{pl,i}]. \tag{3.5}$$

After the optimization problem is solved for all the nodes along the shortest path, the nominal state and input trajectories are obtained as the union of the individual trajectories inside the nodes.

$$\boldsymbol{x}_{nom}(t) = \bigcup_{i=1,\dots,\mathcal{N}} \left(\begin{bmatrix} \boldsymbol{\eta}_{i,i+1}(t) \\ \boldsymbol{v}_{i,i+1}(t) \end{bmatrix} \right), \tag{3.6a}$$

$$\boldsymbol{u}_{nom}(t) = \bigcup_{i=1,\dots,\mathcal{N}} \boldsymbol{u}_{i,i+1}(t). \tag{3.6b}$$

The resulting nominal trajectories are smooth and continuous due to the nature of the proposed optimization problem. Note that, we didn't impose any initial or terminal

constraint to the input and we assumed that the input thrusts can be changed instantaneously. Only in high speed applications, we constrained the rate of change of the thrusts, to simulate the first order dynamics of the thrusters. Figure 3.2 illustrates the receding horizon trajectory optimization approach for an example map.

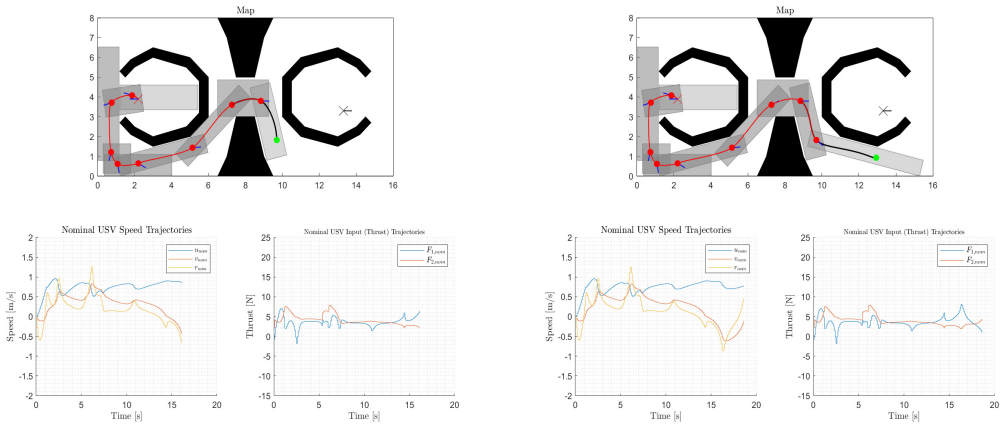


Figure 3.2: Illustration of the receding horizon trajectory optimization for 2 iterations. On the upper figures solid red lines are the planned position trajectories, red circles are the waypoints, blue arrows are the headings of the USV at the waypoints, black solid lines are the predicted position trajectories and green circles are the next waypoints. Red and black crosses are the starting and goal positions, respectively, and the black arrow is the goal heading of the USV. Plots below show the nominal velocity and input trajectories as the output of the optimization, corresponding to the position trajectories shown with solid red curves.

3.2 Motion Control

After we plan the nominal state and input trajectories, $\boldsymbol{x}_{nom}(t)$, $\boldsymbol{u}_{nom}(t)$, we linearize the nonlinear dynamics of the system around these nominal trajectories. Let,

$$\tilde{\boldsymbol{x}}(t) = \boldsymbol{x}(t) - \boldsymbol{x}_{nom}(t) \tag{3.7a}$$

$$\tilde{\boldsymbol{u}}(t) = \boldsymbol{u}(t) - \boldsymbol{u}_{nom}(t). \tag{3.7b}$$

In these expressions, $\tilde{\boldsymbol{x}}(t)$ and $\tilde{\boldsymbol{u}}(t)$ represents the deviation of the real state and input of the system from the nominal trajectories. After we linearize our system, we obtain,

$$\dot{\tilde{\boldsymbol{x}}}(t) = A(t)\tilde{\boldsymbol{x}}(t) + B(t)\tilde{\boldsymbol{u}}(t) \tag{3.8}$$

with

$$A(t) = \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right|_{\substack{\boldsymbol{x}(t)=\boldsymbol{x}_{nom}(t) \\ \boldsymbol{u}(t)=\boldsymbol{u}_{nom}(t)}} \tag{3.9a}$$

$$B(t) = \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\right|_{\substack{\boldsymbol{x}(t)=\boldsymbol{x}_{nom}(t) \\ \boldsymbol{u}(t)=\boldsymbol{u}_{nom}(t)}} \tag{3.9b}$$

$$\tag{3.9c}$$

Using such a formulation, if (3.8) is asymptotically stable, then the real states of the system converges to the nominal states and it is possible follow the generated nominal trajectories. In order to stabilize the linearized system, we use the MPC framework described in Section 2.5.

$$\begin{aligned}
\min_{\tilde{\boldsymbol{u}}()} \quad & \int_0^{T_P} [\tilde{\boldsymbol{x}}(t+\tau|t) Q \tilde{\boldsymbol{x}}(t+\tau|t) \\
& + \tilde{\boldsymbol{u}}^T(t+\tau|t) R \tilde{\boldsymbol{u}}(t+\tau|t)]\, d\tau && \text{(3.10a)} \\
\text{s.t.} \quad & \dot{\tilde{\boldsymbol{x}}}(k|t) = A(t)\tilde{\boldsymbol{x}}(k|t) + B(t)\tilde{\boldsymbol{u}}(k|t) && \text{(3.10b)} \\
& \tilde{\boldsymbol{\eta}}(k|t) + \boldsymbol{\eta}_{nom}(k|t) \in Node(k|t), \quad k \in [t,\ t+T_P] && \text{(3.10c)} \\
& \epsilon \boldsymbol{u}_{min} \leq \tilde{\boldsymbol{u}}(k|t) + \boldsymbol{u}_{nom}(k|t) \leq \epsilon \boldsymbol{u}_{max}, \quad k \in [t,\ t+T_P] && \text{(3.10d)} \\
& \tilde{\boldsymbol{x}}(t|t) = \tilde{\boldsymbol{x}}(t) && \text{(3.10e)}
\end{aligned}$$

In this problem, (3.10.c) and (3.10.d) are time varying constraints ensuring the closed loop system follows the nominal trajectories by avoiding any collision with the obstacles staying within the input limits. Note that, we do not constrain the velocities in the MPC problem since it is unlikely for the velocities to largely deviate from the nominal velocities. The ϵ factor multiplied with the input limits of the trajectory optimization is a relaxation factor for the constraints of the MPC to compensate the numerical errors and it is selected as $\epsilon = 1.05$ for the MATLAB implementation. For the Gazebo implementation, we simply used the Heron USV thrust limits for the MPC input constraints.

As mentioned earlier, the constraint (3.10.c) enforces the USV to be inside the correct node, and hence, inside the obstacle free region for the planned trajectories. Hence, as long as the constraint is satisfied, it is ensured that any collision with the obstacles is avoided which enhances the robustness of our algorithm. Note that, the MPC problem can be formulated and solved in discrete time with the convention used in Section 2.5. However, if the system dynamics are continuous, discretizing the time-varying dynamics using the Peano-Baker series [49][50] is not computationally feasible. In our implementation, the MATLAB model we used is continuous time model and we solved the MPC problem in continuous time. For the Gazebo implementation, we obtain the discrete time dynamics directly as the output of the system identification, so, we solved the discrete time counterpart of the MPC problem.

CHAPTER 4

IMPLEMENTATION AND RESULTS

Our algorithm starts with the generation of the Sparse Neighborhood Graph and finding the nodes and waypoints along the shortest path for a given map as explained in Section 2.2. Then, we solve the trajectory optimization problem given in Section 3.1 to generate nominal trajectories for the USV. Finally, we control the system using LTV MPC on the linearized dynamics over the generated trajectories as explained in Section 3.2. Figure 4.1 shows the overall structure of the algorithm.

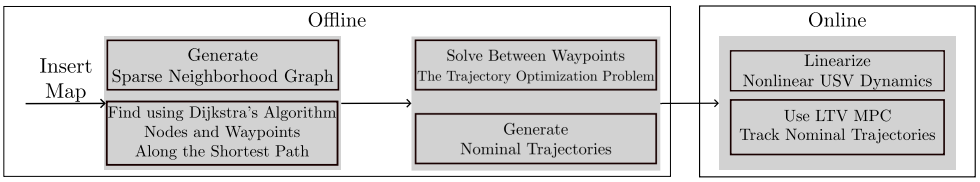


Figure 4.1: The Overall Algorithm

The generation of the nominal trajectories for the underactuated system during the planning phase removes a large computational cost for the real-time controller and results in the real-time control of the underactuated USV.

In this section, the implementation of the method is explained for, first, a predefined model implemented solely on MATLAB, second, the Clearpath Robotics Heron USV on the Gazebo simulator. Since the method depends heavily on the system model and the model of Heron USV does not exist in the literature, the system identification of the Heron USV is also conducted in this chapter. Finally, the results for both the MATLAB model and the Heron USV are shared and the results are compared with existing studies. All the experiments are conducted on a laptop with Intel i7 2.7 GHz processor running Ubuntu 20.04.

4.1 MATLAB Implementation

The proof of concept implementation of our method is conducted as a MATLAB simulation with a predefined USV model. For the nonlinear equations of motion of the simulated USV, Fossen's equations of motion are implemented with the parameters presented in Table 4.1.

Table 4.1: MATLAB Implementation USV Parameters

Parameter	m	I_z	X_u	Y_v	Y_r	N_v	N_r	$X_{\dot{u}}$	$Y_{\dot{v}}$	$Y_{\dot{r}}$	$N_{\dot{v}}$	$N_{\dot{r}}$	$X_{u\|u\|}$	$Y_{v\|v\|}$	$N_{r\|r\|}$	b
Value	10	0.65	-5.61	-3.51	0	0	0	-1.09	-1.05	-1.44	-1.44	-0.10	-2.31	-3.93	-0.26	0.26

To parameterize the trajectory optimization problem, first, we define the state and input vectors as (4.1).

$$\mathbf{x(t)} = \begin{bmatrix} x(t) \\ y(t) \\ \psi(t) \\ u(t) \\ v(t) \\ r(t) \end{bmatrix}, \quad \mathbf{u(t)} = \begin{bmatrix} F_1(t) \\ F_2(t) \end{bmatrix} \tag{4.1}$$

Using the hydrodynamic parameters and the underactuated Fossen's model, we obtain

the continuous time state space equations given in (4.2).

$$\dot{\mathbf{x}}(\mathbf{t}) = \mathbf{f}(\mathbf{x}(\mathbf{t}), \mathbf{u}(\mathbf{t})) \tag{4.2a}$$

$$= \begin{bmatrix} u(t)cos(\psi(t)) - v(t)sin(\psi(t)) \\ \\ u(t)sin(\psi(t) + v(t)cos(\psi(t)) \\ \\ r(t) \\ \\ -0.16r(t)^2 - 1.2v(t)r(t) - 0.63u(t) \\ -0.26u(t)|u(t)| + 0.11(F_1(t) + F_2(t)) \\ \\ -0.68v(t) + 2.88r(t)u(t) - 0.021u(t)v(t) \\ -0.13r(t)|r(t)| - 0.76v(t)|v(t)| + 0.51(F_1(t) - F_2(t)) \\ \\ -1.79v(t) + 10.22r(t)u(t) - 0.13u(t)v(t) \\ -0.82r(t)|r(t)| - 2v(t)|v(t)| + 3.2(F_1(t) - F_2(t)) \end{bmatrix} \tag{4.2b}$$

We solved the trajectory optimization problem in continuous time using the direct collocation method described in Chapter 2 with the parameterization explained in (2.37), (2.38), (2.39) and (2.40) with $T = 0.1sec$ and $J_c(u) = u^2$. After we obtain the nominal trajectories $\mathbf{x_{nom}}(\mathbf{t})$ and $\mathbf{u_{nom}}(\mathbf{t})$, we linearize the system to obtain $A(t)$ and $B(t)$ such that

$$\dot{\tilde{\mathbf{x}}}(\mathbf{t}) = A(t)\tilde{\mathbf{x}}(\mathbf{t}) + B(t)\tilde{\mathbf{u}}(\mathbf{t}) \tag{4.3}$$

with $\tilde{\mathbf{x}}(\mathbf{t}) = \mathbf{x}(\mathbf{t}) - \mathbf{x_{nom}}(\mathbf{t})$ and $\tilde{\mathbf{u}}(\mathbf{t}) = \mathbf{u}(\mathbf{t}) - \mathbf{u_{nom}}(\mathbf{t})$. For the MPC problem, we again used the direct collocation method on the linearized system with $T = 0.03sec$ to avoid discretizing the time-varying dynamics. For the MPC controller, we used the following cost matrices.

$$Q = \begin{bmatrix} 100 & 0 & 0 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 & 0 & 0 \\ 0 & 0 & 100 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.4}$$

$$R = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix} \tag{4.5}$$

We set the MPC prediction horizon to $T_{MPC} = 0.6sec$ and control horizon to $T_{control} = 0.03sec$ for all the experiments. For the MATLAB implementation, the trajectory optimization problem is solved using the `fmincon` solver of MATLAB, with the SQP algorithm described in Chapter 2 and the MPC problem is solved using the `quadprog` solver of MATLAB. After each MPC iteration, the system dynamics are integrated using the `ode45` solver of MATLAB using the thrust commands generated by MPC and the system dynamics given in (4.2).

4.2 Gazebo Implementation

After the proof of concept studies are conducted for the method on solely MATLAB environment, we examined the real time performance of our algorithm on a simulation environment with a physics engine. We conducted our experiments on the Gazebo simulation of the Clearpath Robotics Heron USV. Heron is an catamaran type underactuated USV, containing two thrusters on the differential drive configuration. Its important properties are presented on the Table 4.2. Heron USV is shown in Figure 4.2 and its gazebo environment is shown in Figure 4.3.

Table 4.2: Heron USV Properties

Dimensions [m]	1.35 x 0.98 x 0.32	Navigation	IMU, GPS
Weight [kg]	29	Communication	WiFi, USB, Ethernet, RS232
Rated Speed [m/s]	1.7	Rated Operation Time	2.5 Hours

Figure 4.2: Clearpath Robotics Heron USV [5].

4.2.1 Gazebo Simulator Setup

For the Gazebo simulation, we used the simulator provided by Clearpath Robotics [5]. This simulator is based on ROS Melodic running on a Ubuntu 18.04 machine.

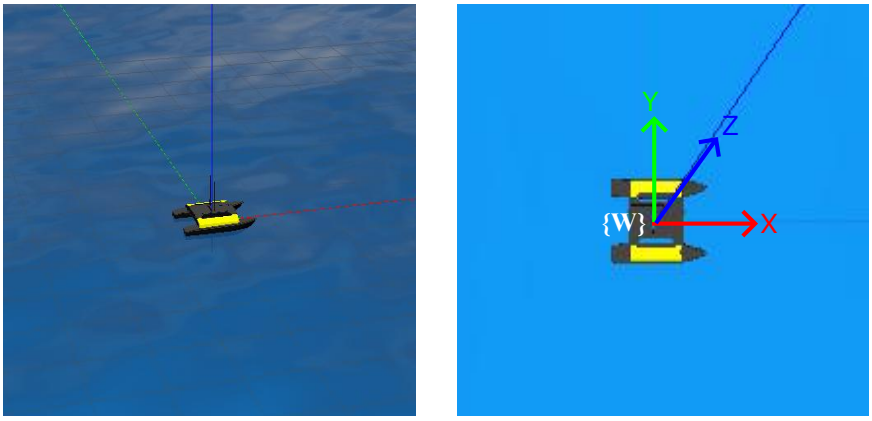


Figure 4.3: Heron USV Gazebo simulation environment. (a) shows the isometric view of the Heron USV in the simulation environment. (b) shows the world coordinate frame of the simulation environment. $\{W\}$ is the world coordinate system, with red, green and blue arrows representing the X, Y and Z axes, respectively.

So, we prepared a Docker image containing Ubuntu 18.04, the Heron Gazebo simulator, required ROS packages and a VNC service to conduct our simulations. We implemented our controllers and data acquisition setup on MATLAB running on the local machine. Figure 4.4 shows the overview of the simulation setup.

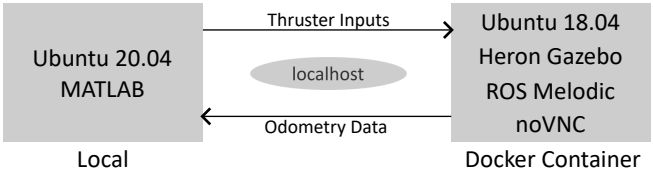


Figure 4.4: The overview of the simulation setup. The Gazebo simulation runs on a Docker container, containing the Heron simulator, required ROS packages and noVNC running on Ubuntu 18.04. The simulator publishes the odometry data to MATLAB environment running on the local machine. After the necessary computations, the MATLAB environment publishes the thrust commands to the simulator on the closed loop.

The original odometry data provided by the simulator is based on the world frame. However, we needed to obtain the position data on the world frame and the speed data on the base frame of the Heron to model our system with Fossen's equations of motion presented in Chapter 2. So we created the `/odometry_transformer` ROS

node for the coordinate frame transformation which outputs the transformed odometry data on the `/odometry/base` topic. The world and base frames are shown in Figure 4.3. The odometry topic of the Heron simulator `/odometry/filtered`

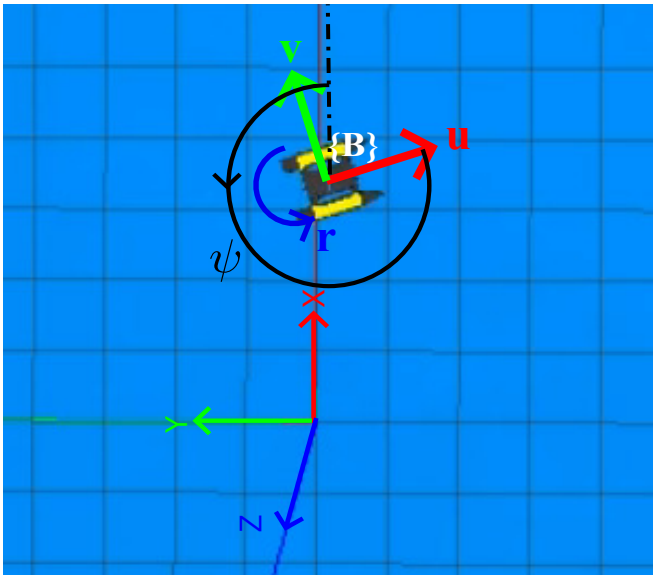


Figure 4.5: Overview of the world and base frames. $\{B\}$ represents the base frame of Heron. Vectors u and v represent the surge and sway axes of the USV respectively. ψ is the heading angle with respect to the world X axis and r represents the positive direction of the USV angular speed.

publishes the position and velocity of the USV with respect to the world frame presented in (4.6). In (4.6), $\mathbf{p^W}$ and $\mathbf{v^W}$ represents the position and velocity vector of the USV in the world frame, respectively.

$$\mathbf{x_{odom}} = \begin{bmatrix} \mathbf{p^W} \\ \mathbf{v^W} \end{bmatrix} = \begin{bmatrix} x \\ y \\ \psi \\ u^W \\ v^W \\ r^W \end{bmatrix} \tag{4.6}$$

In order to transform the published velocity vector to the base frame, at each instant that the odometry data is received, the rotation transformation presented in (4.7) is performed with the rotation matrix given in (4.8) that transforms the vector represented in the world frame to the base frame. Since the Heron USV is only allowed to

move in 2D, its z-axis is always aligned with the world frame z-axis. So, the rotation is purely with respect to the z-axis with an amount of the heading angle.

$$\mathbf{v^B} = R_W^B \mathbf{v^W} \tag{4.7}$$

$$R_W^B = \begin{bmatrix} cos\psi & -sin\psi & 0 \\ sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.8}$$

After defining the velocity transformations, the overall transformation matrix can be defined as (4.9). This transformation preserves the position vector in the world frame and outputs the velocity vector represented in the base frame.

$$\mathbf{J_{odom}} = \begin{bmatrix} I_{3x3} & 0 \\ 0 & R_W^B \end{bmatrix} \tag{4.9}$$

After defining the overall transformation matrix, the transformed odometry vector $\mathbf{x_{transformed}}$ is calculated as (4.10).

$$\mathbf{x_{transformed}} = \mathbf{J_{odom}} \mathbf{x_{odom}} \tag{4.10a}$$

$$= \begin{bmatrix} x \\ y \\ \psi \\ u^B \\ v^B \\ r^B \end{bmatrix} \tag{4.10b}$$

The odometry data is generated by fusing the IMU readings and the GPS data using Extended Kalman Filter on the node `ekf_localization_node`.

We acquire the transformed odometry data given in (4.10) on MATLAB through the node `matlab_global_node_67379`, created by the MATLAB-ROS bridge. To publish the thrust commands to the USV, MATLAB node publishes the topics `/heron/thrusters/0/input` and `/heron/thrusters/1/input` to the Gazebo simulator where the former one commands the left thruster and the latter one commands the right thruster. Each thruster of Heron has a thrust of 33.6 Newtons on the forward direction and -19.88 Newtons on the backwards direction [51]. The thrust commands are normalized between -1 to 1 using a spline interpolation before

they are send to the USV. The corresponding knot points for the spline are presented in Table 4.3.

Table 4.3: Heron USV Thrust to Input Mappings

Thrust [N]	Input
-19.88	-1.0
-16.52	-0.8
-12.6	-0.6
-5.6	-0.4
-1.4	-0.2
0.0	0.0
2.24	0.2
9.52	0.4
21.28	0.6
28.0	0.8
33.6	1.0

Finally, Docker container and the local machine communicates through the `localhost` of the local machine. Figure 4.6 shows the RQT graph of the Gazebo simulator of Heron USV.

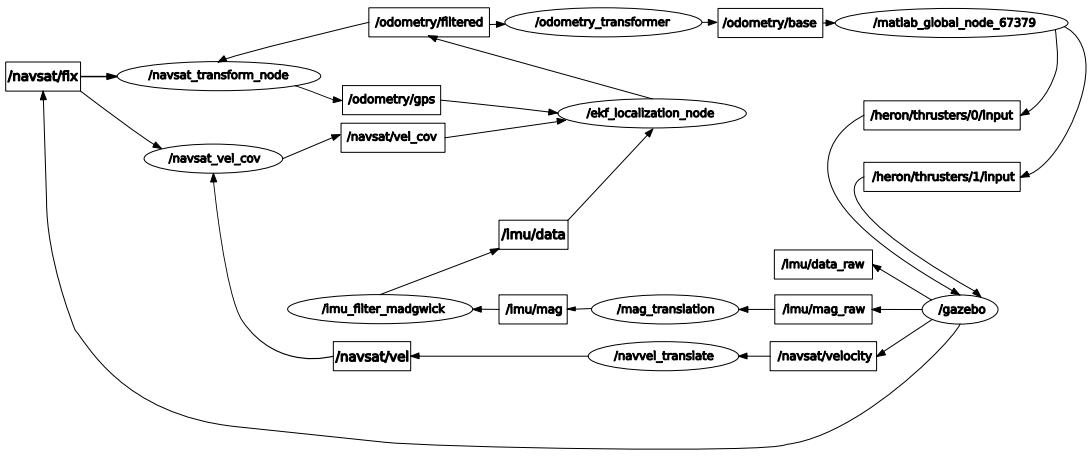


Figure 4.6: RQT graph of the Gazebo simulator of Heron USV.

4.2.2 System Identification on Heron USV

Our method developed in this study uses the system model heavily both in the motion planning and the control phases. However, there is no dynamic model for the Heron USV in the literature. So, we conducted system identification on Heron based on Fossen's equations of motion and NARMAX methods. We conduct the system identification in discrete time since we obtain the data in discrete steps through the Gazebo simulation. In this identification, we try to identify the overall equations of motion, not the real hydrodynamic parameters. We can write the speed dynamics of the USV as given in 4.11.

$$u[k+1] = \theta_1 u[k] + \theta_2 r[k]^2 + \theta_3 v[k] r[k] + \theta_4 u[k]|u[k]| + \theta_5 (F_1[k] + F_2[k]) \tag{4.11a}$$

$$\begin{aligned} v[k+1] = {} & \theta_6 v[k] + \theta_7 r[k] + \theta_8 u[k] v[k] + \theta_9 u[k] r[k] \\ & + \theta_{10} v[k]|v[k]| + \theta_{11} r[k]|r[k]| + \theta_{12}(F_1[k] - F_2[k]) \end{aligned} \tag{4.11b}$$

$$\begin{aligned} r[k+1] = {} & \theta_{13} v[k] + \theta_{14} r[k] + \theta_{15} u[k] v[k] + \theta_{16} u[k] r[k] \\ & + \theta_{17} v[k]|v[k]| + \theta_{18} r[k]|r[k]| + \theta_{19}(F_1[k] - F_2[k]) \end{aligned} \tag{4.11c}$$

After obtaining data at each sampling instant, we can compute the value of $u[k+1]$, $v[k+1]$ and $r[k+1]$ in terms of parameters $\theta_1,\ \theta_2, \ldots, \theta_1 9$, which makes the model linear in the parameters NARMAX model with n_x and n_u being equal to 1. Assuming we get N samples from the simulation, we can write the output vectors (4.12), (4.13) and (4.14) that consists of the values of $u[k+1]$, $v[k+1]$ and $r[k+1]$ for $k = 0, \ldots, N-1$.

$$\mathbf{y_1} = \begin{bmatrix} u[1] \\ u[2] \\ \vdots \\ u[N] \end{bmatrix} \tag{4.12}$$

$$\mathbf{y_2} = \begin{bmatrix} v[1] \\ v[2] \\ \vdots \\ v[N] \end{bmatrix} \tag{4.13}$$

$$\mathbf{y_3} = \begin{bmatrix} r[1] \\ r[2] \\ \vdots \\ r[N] \end{bmatrix} \tag{4.14}$$

Also, we can write the regressor matrices R_u, R_v and R_r that consists of the values of nonlinear terms at each sampling instant for the state equations of $u[k+1]$, $v[k+1]$ and $r[k+1]$ on their columns up to $N-1$ sampling instants. These matrices are presented in (4.15), (4.16) and (4.17).

$$R_u = \begin{bmatrix} u[0] & r[0]^2 & v[0]r[0] & u[0]|u[0]| & (F_1[0]+F_2[0]) \\ u[1] & r[1]^2 & v[1]r[1] & u[1]|u[1]| & (F_1[1]+F_2[1]) \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \tag{4.15}$$

$$R_v = \begin{bmatrix} v[0] & r[0] & u[0]v[0] & u[0]r[0] & v[0]|v[0]| & r[0]|r[0]| & (F_1[0]-F_2[0]) \\ v[1] & r[1] & u[1]v[1] & u[1]r[1] & v[1]|v[1]| & r[1]|r[1]| & (F_1[1]-F_2[1]) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \tag{4.16}$$

$$R_r = \begin{bmatrix} v[0] & r[0] & u[0]v[0] & u[0]r[0] & v[0]|v[0]| & r[0]|r[0]| & (F_1[0]-F_2[0]) \\ v[1] & r[1] & u[1]v[1] & u[1]r[1] & v[1]|v[1]| & r[1]|r[1]| & (F_1[1]-F_2[1]) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \tag{4.17}$$

We can also concatenate each output vector into vector $\mathbf{y}$, each parameter into vector $\boldsymbol{\theta}$ and define the overall regressor matrix $\mathbf{R}$ as shown in (4.18).

$$\mathbf{y} = \begin{bmatrix} \mathbf{y_1} \\ \mathbf{y_2} \\ \mathbf{y_3} \end{bmatrix} \quad \mathbf{R} = \begin{bmatrix} R_u & 0 & 0 \\ 0 & R_v & 0 \\ 0 & 0 & R_r \end{bmatrix} \quad \boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_{19} \end{bmatrix} \tag{4.18}$$

Finally, we can write the regression equation given in (4.19).

$$\mathbf{y} = \mathbf{R}\boldsymbol{\theta} \tag{4.19}$$

The only unknown for this equation is the parameter vector θ and we need to search for this vector. First, we need to define the regression error as (4.20).

$$\mathbf{e}(\theta) = \mathbf{y} - \mathbf{R}\boldsymbol{\theta} \tag{4.20}$$

Then, we can write the searching problem as to minimize the W norm of the regression error $\mathbf{e}$ as formulated in (4.21).

$$\min_{\boldsymbol{\theta}} \quad \boldsymbol{e}(\boldsymbol{\theta})^T W(\boldsymbol{\theta}) \boldsymbol{e}(\boldsymbol{\theta}) \tag{4.21}$$

In this formulation, W is the diagonal weighting matrix that weights each of the error terms during the search and it is selected as the inverse of the estimated error variances at each sampling instant. Using such a weighting gives the optimal weighting in terms of the maximum likelihood [52]. Since W also depends on the parameters, an iterative optimization algorithm should be used to find the value of W at each iteration. We used the SQP algorithm to solve this problem.

To conduct this system identification, we used the MATLAB System Identification Toolbox [53]. Using the toolbox, we first create a `idnlarx` object with the simulation input and output data. After that, we define the regressor terms configuring the `idnlarx.Regressors` property of the created object. We obtain the data at a sampling period of 0.1 seconds from the Gazebo simulation, so, we configure the `idnlarx.Ts` property to 0.1 seconds. Finally, we conduct the system identification with the `nlarx` function of the toolbox. We select the solver properties as given in Table 4.4 to conduct the system identification as explained before.

Table 4.4: nlarx Options

nlarxOptions.Focus	simulation
nlarxOptions.SearchMethod	fmincon
nlarxOptions.SearchOptions.Algorithm	sqp

We obtain the identified model through the properties of the output of the `nlarx` function. Assuming `model` is the output of the `nlarx` function, we get the model from the property `model.OutputFcn.LinearFcn`. To identify Heron USV model, we used the input and output sequences shown in Figure 4.7.

For the identification, we changed the input sequence every 60 seconds. First, we started with chirp inputs with an amplitude of 33.6 Newtons on the forward direction. At each sequence, we increased the phase difference on the right thruster with 30 degrees, up to 180 degrees. After that, we changed the chirp sequence to the backwards direction, with an amplitude of -19.88 Newtons. We made the same phase difference

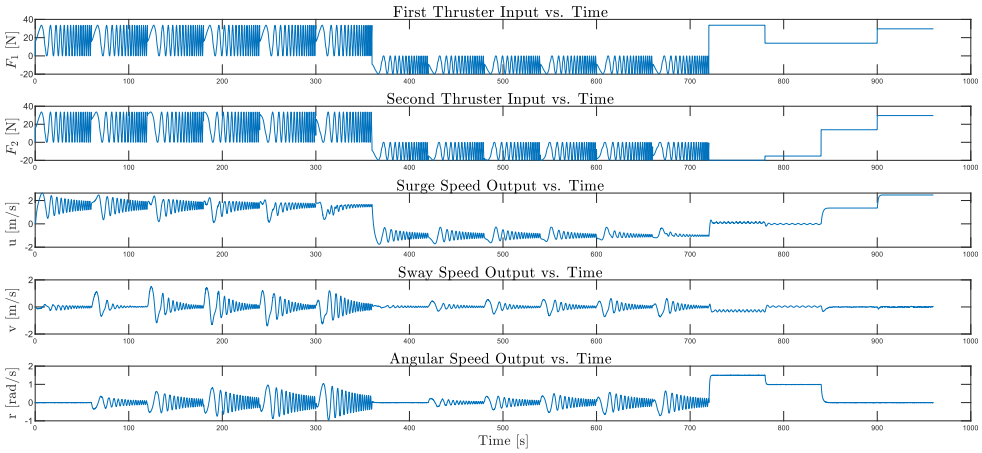


Figure 4.7: Input and output sequences used for the system identification.

change for every chirp sequence. After, we commanded 33.6 Newtons for the left thruster and -19.88 Newtons for the right thruster to further examine the sway dynamics of the USV. Then, we set 15 Newtons to left and -15 Newtons to right thruster to further examine the rotational velocity dynamics. After that, we set 15 Newtons to both thrusters to further examine the surge dynamics. Finally, we set 30 Newtons to both thrusters to further examine the dynamics near the forward saturation limit. After conducting the system identification, we obtained the fit to the simulation data given in Figure 4.8 and the percentage fit to the simulation data is given in Table 4.5.

Table 4.5: Percentage Fit of the Identified System

State	Fit
Surge u	96.53%
Sway v	82.13%
Rotational Velocity r	94.78%

Also, we obtained a Mean Square Error of $MSE = 0.005851$ for the overall simulation. After the system identification, we obtained the parameters for surge, sway and rotational velocity dynamics given in Tables 4.6, 4.7 and 4.8, respectively.

Table 4.6: Resulting Parameters for Surge Dynamics

Regressor	$u[k]$	$r[k]^2$	$v[k]r[k]$	$u[k]\|u[k]\|$	$F_1[k] + F_2[k]$
Parameter	0.9468	0.0020	0.906	-0.0069	0.0030

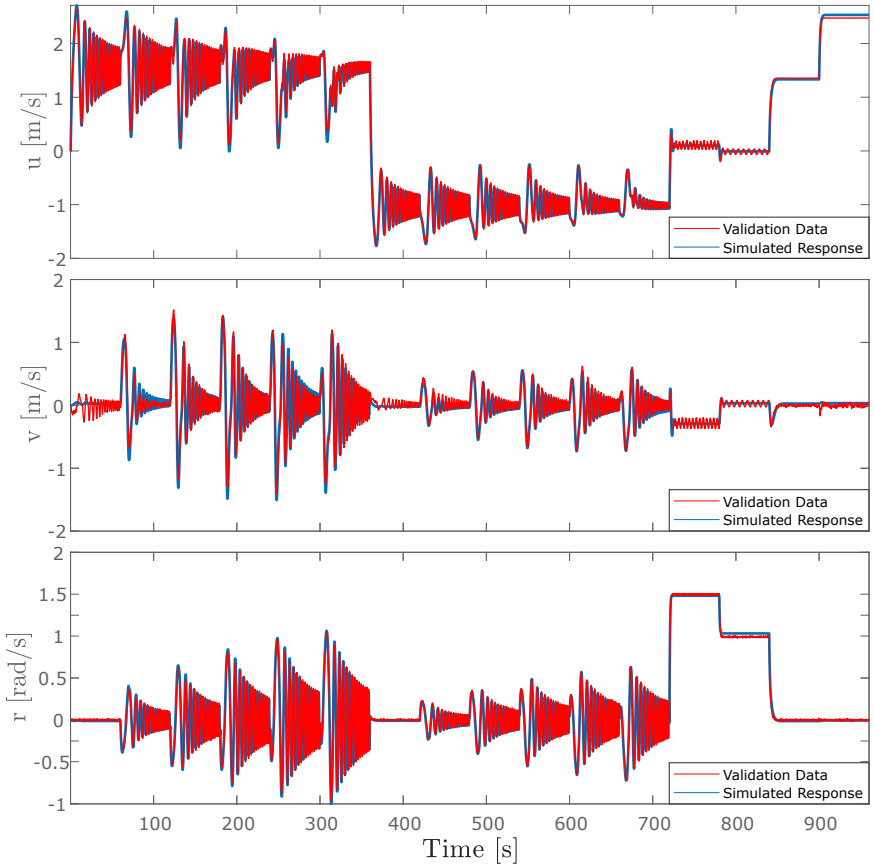


Figure 4.8: Fit to the simulation data.

Table 4.7: Resulting Parameters for Sway Dynamics

Regressor	$v[k]$	$r[k]$	$u[k]v[k]$	$u[k]r[k]$	$v[k]\|v[k]\|$	$r[k]\|r[k]\|$	$F_1[k] - F_2[k]$
Parameter	0.9327	0.0117	0.0060	-0.1016	0.0009	-0.0076	-0.0001

Table 4.8: Resulting Parameters for Rotational Velocity Dynamics

Regressor	$v[k]$	$r[k]$	$u[k]v[k]$	$u[k]r[k]$	$v[k]\|v[k]\|$	$r[k]\|r[k]\|$	$F_1[k] - F_2[k]$
Parameter	-0.0019	0.9692	0.0048	0.0046	0.0018	-0.0582	0.0032

After the identification, we validated our model in three different cases. First, we tried to give a step input at the forward saturation limit. The input sequence is shown in Figure 4.9. The validation data and the simulated response of the system are shown in Figure 4.10.

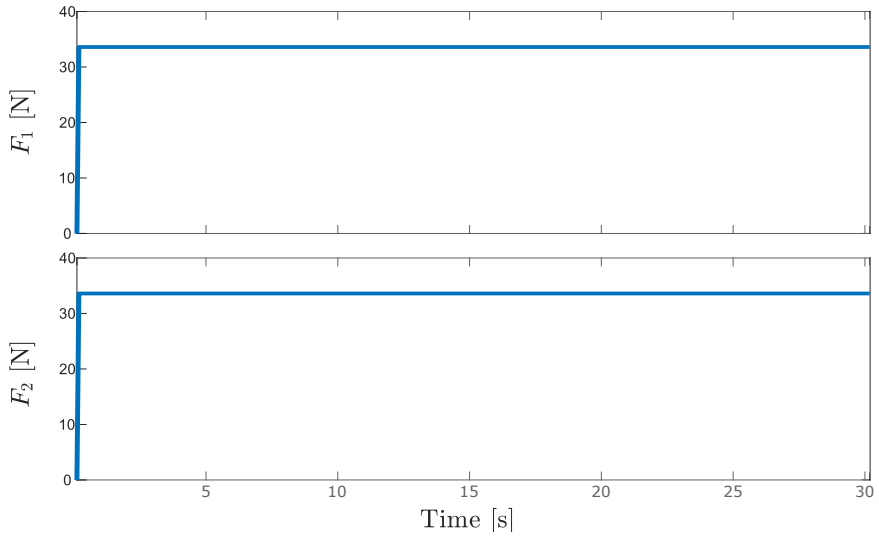


Figure 4.9: Input sequences for the first validation.

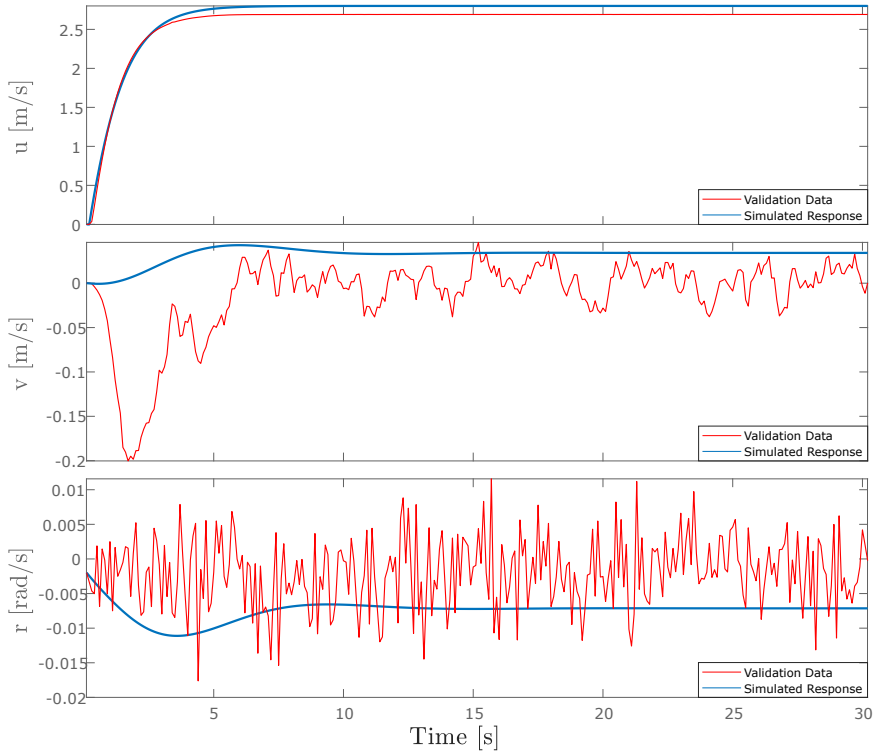


Figure 4.10: Output of the first validation.

Secondly, we tried to give a sinusoidal input to the thrusters with the amplitude of forward saturation limit. We set a phase difference between the thrusters. The input sequence is shown in Figure 4.11. The validation data and the simulated response of the identified system are shown in Figure 4.12.

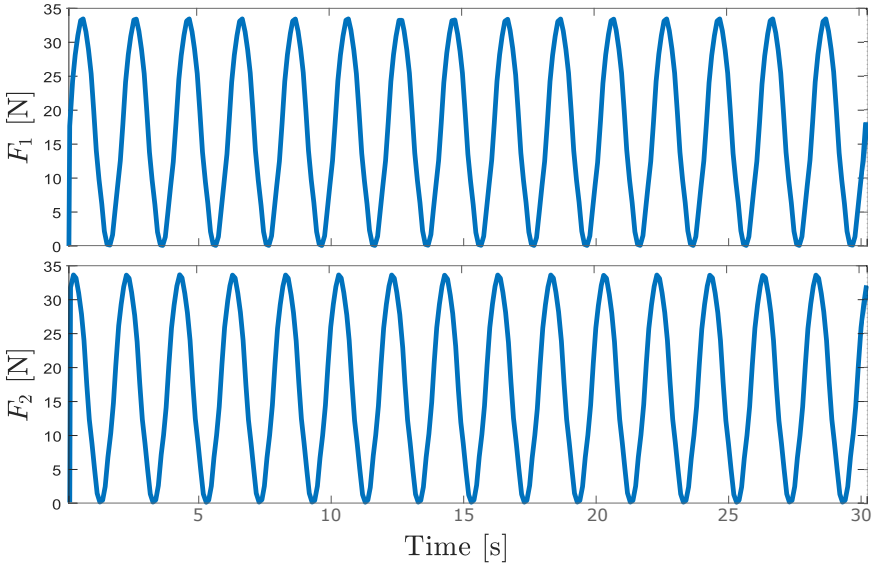


Figure 4.11: Input sequences for the second validation.

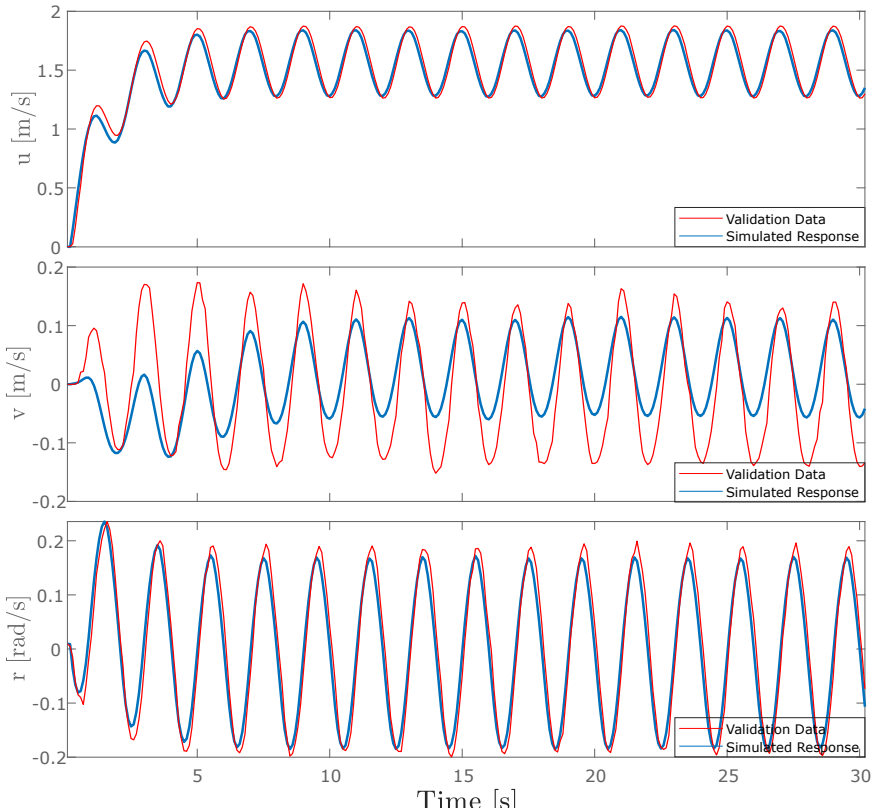


Figure 4.12: Output of the second validation.

Finally, we tried to give a step input to the thrusters at 17 Newtons to drive the USV at a medium speed. The input sequence is shown in Figure 4.13. The validation data

and the simulated response of the identified system are shown in Figure 4.14.

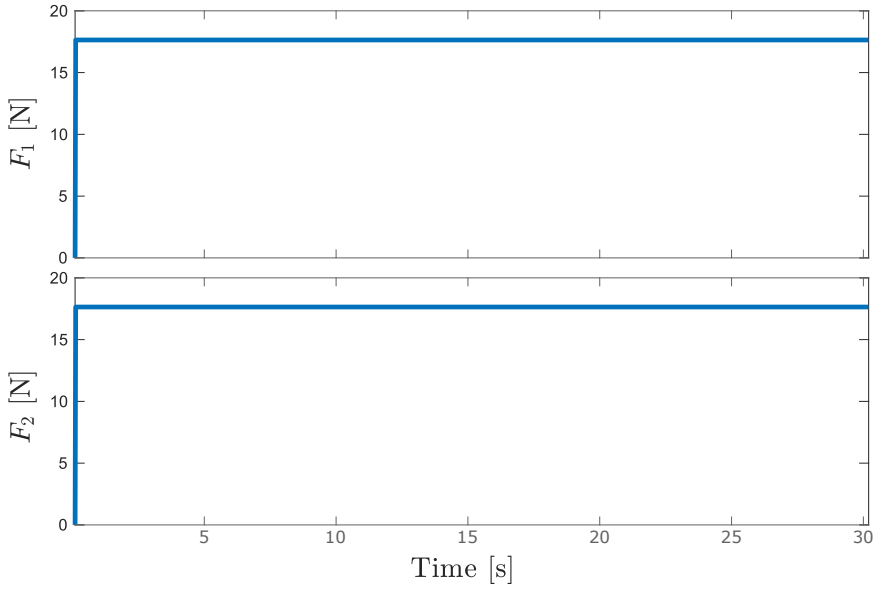


Figure 4.13: Input sequences for the third validation.

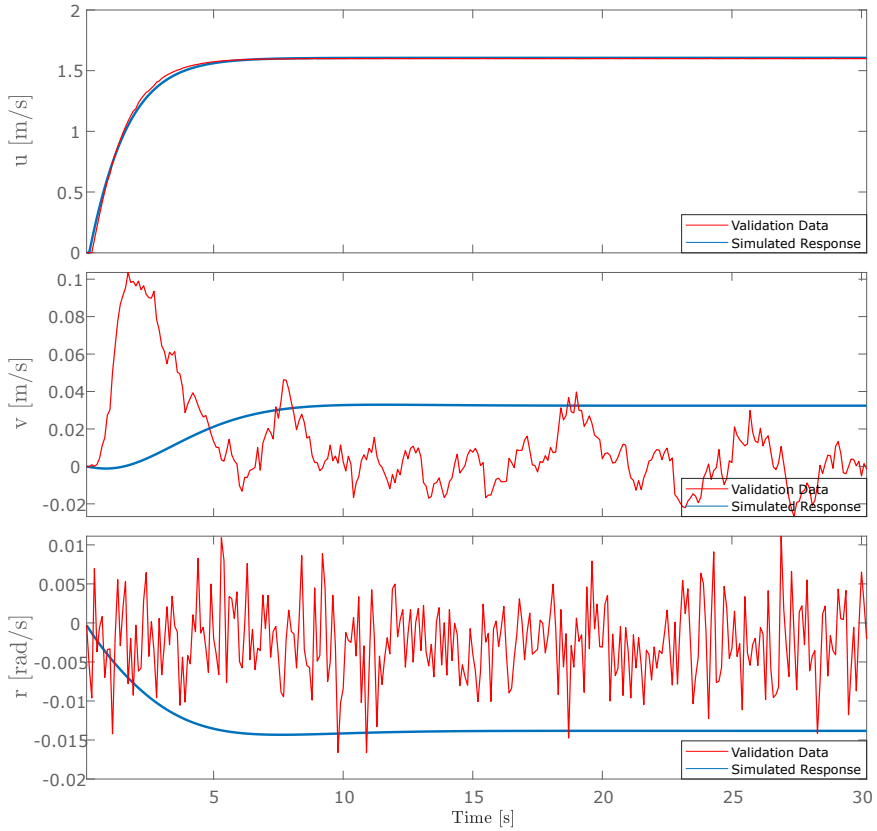


Figure 4.14: Output of the third validation.

Our validation shows that near the saturation limits and very low speeds, the identified system deviates the real system's response a small amount. Also, we can see that the noise on the sway and rotational velocity dynamics create a small bias for the identified system in these axes. Finally, we can see that the damping in the sway axis is overestimated as a result of this identification. We conduct the system identification based on the Fossen's equations of motion presented in Chapter 2, which is a simplified model. If we add more regressor terms in the identification process, we can get a more accurate result. However, we will use this model to test our algorithm and all these deviations will manifest themselves as a process noise during the operation. This way, we will test the robustness of our method against the process noise.

4.2.3 Implementation of the Method

After we identify our system, we obtain a discrete time nonlinear system representing the dynamics of Heron USV. Since the direct transcription method presented in Chapter 2 is an easier parameterization for discrete time systems, we will use this parameterization for the receding horizon trajectory optimization problem. Since we conducted our system identification with a sampling period of $T_s = 0.1sec$, we will use that period to solve the trajectory optimization and MPC problems. For the trajectory optimization problem, first, define the input vector.

$$\mathbf{u} = \begin{bmatrix} F_1[k_0] \\ F_1[k_0+1] \\ \vdots \\ F_1[k_0+N_{pr,i}-1] \\ F_2[k_0] \\ F_2[k_0+1] \\ \vdots \\ F_2[k_0+N_{pr,i}-1] \end{bmatrix} \tag{4.22}$$

Here, $N_{pr,i}$ is the prediction horizon of the planner and it is calculated as given in (4.23).

$$N_{pr,i} = \frac{T_{pr,i}}{T_s} \tag{4.23}$$

For the system dynamics, we use the identified model for the speed dynamics and the Euler discretization of the position dynamics as given in (4.24).

$$\mathbf{f}(\mathbf{x}[\mathbf{k}], \mathbf{u}[\mathbf{k}]) = \begin{bmatrix} x[k] + T_s(u[k]cos(\psi[k]) - v[k]sin(\psi[k])) \\ y[k] + T_s(u[k]sin(\psi[k]) + v[k]cos(\psi[k])) \\ \psi[k] + T_s r[k] \\ \theta_1 u[k] + \theta_2 r[k]^2 + \theta_3 v[k] r[k] \\ \quad + \theta_4 u[k]|u[k]| + \theta_5(F_1[k] + F_2[k]) \\ \theta_6 v[k] + \theta_7 r[k] + \theta_8 u[k] v[k] + \theta_9 u[k] r[k] \\ \quad + \theta_{10} v[k]|v[k]| + \theta_{11} r[k]|r[k]| + \theta_{12}(F_1[k] - F_2[k]) \\ \theta_{13} v[k] + \theta_{14} r[k] + \theta_{15} u[k] v[k] + \theta_{16} u[k] r[k] \\ \quad + \theta_{17} v[k]|v[k]| + \theta_{18} r[k]|r[k]| + \theta_{19}(F_1[k] - F_2[k]) \end{bmatrix} \tag{4.24}$$

with $\theta_1, \ldots, \theta_{19}$ being the identified parameters and the definitions given in (4.25).

$$\mathbf{x}[\mathbf{k}] = \begin{bmatrix} x[k] \\ y[k] \\ \psi[k] \\ u[k] \\ v[k] \\ r[k] \end{bmatrix}, \quad \mathbf{u}[\mathbf{k}] = \begin{bmatrix} F_1[k] \\ F_2[k] \end{bmatrix}, \quad \mathbf{x}[\mathbf{k}+\mathbf{1}] = \mathbf{f}(\mathbf{x}[\mathbf{k}], \mathbf{u}[\mathbf{k}]) \tag{4.25}$$

Using these definitions, we can use (2.26) to parameterize the state equations. The position constraints are the box constraints consisting of four line inequalities and together with the speed constraints, they can be generated using (2.29). After we obtain the nominal trajectories $\mathbf{x_{nom}}[\mathbf{k}]$ and $\mathbf{u_{nom}}[\mathbf{k}]$, we linearize the system to obtain $A[k]$ and $B[k]$ such that

$$\tilde{\mathbf{x}}[\mathbf{k}+\mathbf{1}] = A[k]\tilde{\mathbf{x}}[\mathbf{k}+\mathbf{1}] + B[k]\tilde{\mathbf{u}}[\mathbf{k}] \tag{4.26}$$

with $\tilde{\mathbf{x}}[\mathbf{k}] = \mathbf{x}[\mathbf{k}] - \mathbf{x_{nom}}[\mathbf{k}]$ and $\tilde{\mathbf{u}}[\mathbf{k}] = \mathbf{u}[\mathbf{k}] - \mathbf{u_{nom}}[\mathbf{k}]$. For the MPC problem, we use the direct shooting method on the linearized system since it is lightweight and

computationally efficient due to the reduced number of decision variables. For the MPC controller, we used the following cost matrices.

$$Q = \begin{bmatrix} 50 & 0 & 0 & 0 & 0 & 0 \\ 0 & 50 & 0 & 0 & 0 & 0 \\ 0 & 0 & 150 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.1 \end{bmatrix} \tag{4.27}$$

$$R = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix} \tag{4.28}$$

During the planning phase, we try to minimize the input effort to reach the goal. We set the MPC prediction horizon to $N_{MPC} = 40$ (4 seconds) and control horizon to 1 time-step for all the experiments. We solved the trajectory optimization problem using the interior-point algorithm of MATLAB `fmincon` solver and MPC problem using MATLAB `quadprog` solver. Finally, Before the start of each experiment, we align the Gazebo initial configuration to the MATLAB initial configuration. This way, we make the Gazebo model of the USV to run in a virtual environment, i.e., as if there are obstacles in the Gazebo environment even if there is not. Figure 4.15 shows the mentioned virtual environment.

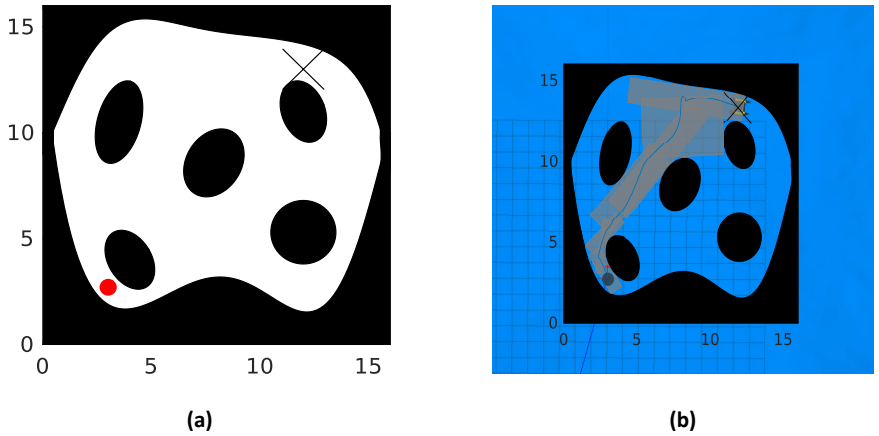
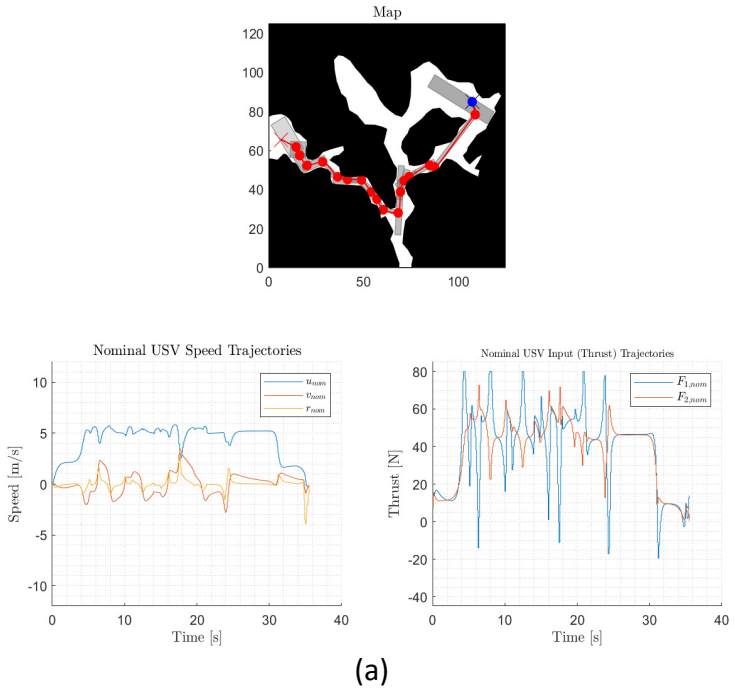


Figure 4.15: The virtual environment created for Gazebo simulation. (a) shows the map created in MATLAB. (b) shows the Gazebo simulation of the USV.

4.3 MATLAB Implementation Results

4.3.1 Results Without Process Noise

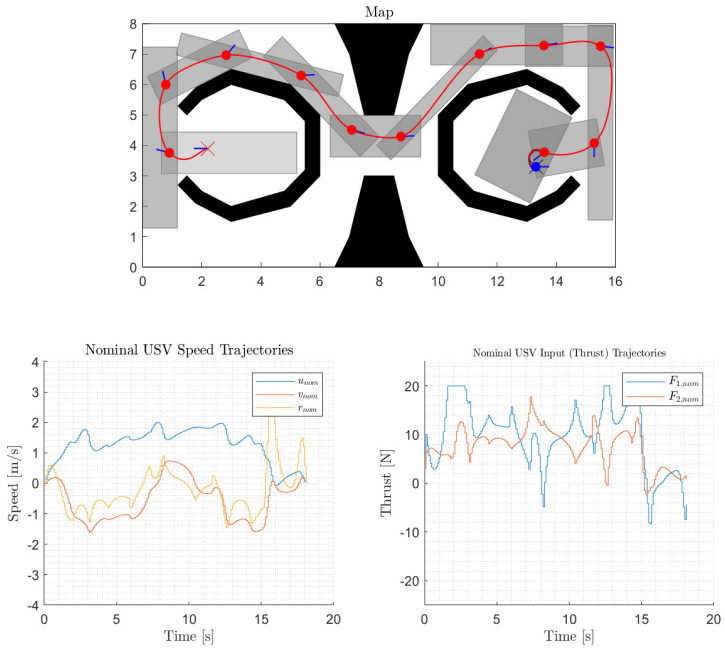
To test our algorithm, we first conducted simulations on two different maps without any process noise included. Note that these maps are the same maps used in [2] to better compare and contrast our algorithm. Figures 4.17 and 4.16 shows these maps and the planned trajectories connecting the starting point to the goal location. We conducted the simulation using the parameter sets given in Table 4.9.



(a)

Figure 4.16: Trajectory planning on Map (a). Upper plot shows the map and red solid curve is the planned trajectory. Red circles are the waypoints. Lower figures show the nominal speed and thrust trajectories for map (b).

In the planning phase of the MATLAB implementation, we did not constrained the sway speed v of the USV. The simulation results for closed loop control on the nominal trajectories are given in Figures 4.19 and 4.18. From the results, the real trajectories follow closely the nominal trajectories.



(b)

Figure 4.17: Trajectory planning on Map (b). Upper plot shows the map and red solid curve is the planned trajectory. Red circles are the waypoints. Lower figures show the nominal speed and thrust trajectories for Map (a).

Table 4.9: The Parameter Sets for the Test Simulations.

Map	a	b
v_{des}	6	2
$\begin{bmatrix} u \\ v \\ r \end{bmatrix}_{min}$	$\begin{bmatrix} -4 \\ -\infty \\ -10 \end{bmatrix}$	$\begin{bmatrix} -0.5 \\ -\infty \\ -3 \end{bmatrix}$
$\begin{bmatrix} u \\ v \\ r \end{bmatrix}_{max}$	$\begin{bmatrix} 12 \\ \infty \\ 10 \end{bmatrix}$	$\begin{bmatrix} 4 \\ \infty \\ 3 \end{bmatrix}$
$\begin{bmatrix} F_1 \\ F_2 \end{bmatrix}_{min}$	$\begin{bmatrix} -40 \\ -40 \end{bmatrix}$	$\begin{bmatrix} -10 \\ -10 \end{bmatrix}$
$\begin{bmatrix} F_1 \\ F_2 \end{bmatrix}_{max}$	$\begin{bmatrix} 80 \\ 80 \end{bmatrix}$	$\begin{bmatrix} 20 \\ 20 \end{bmatrix}$

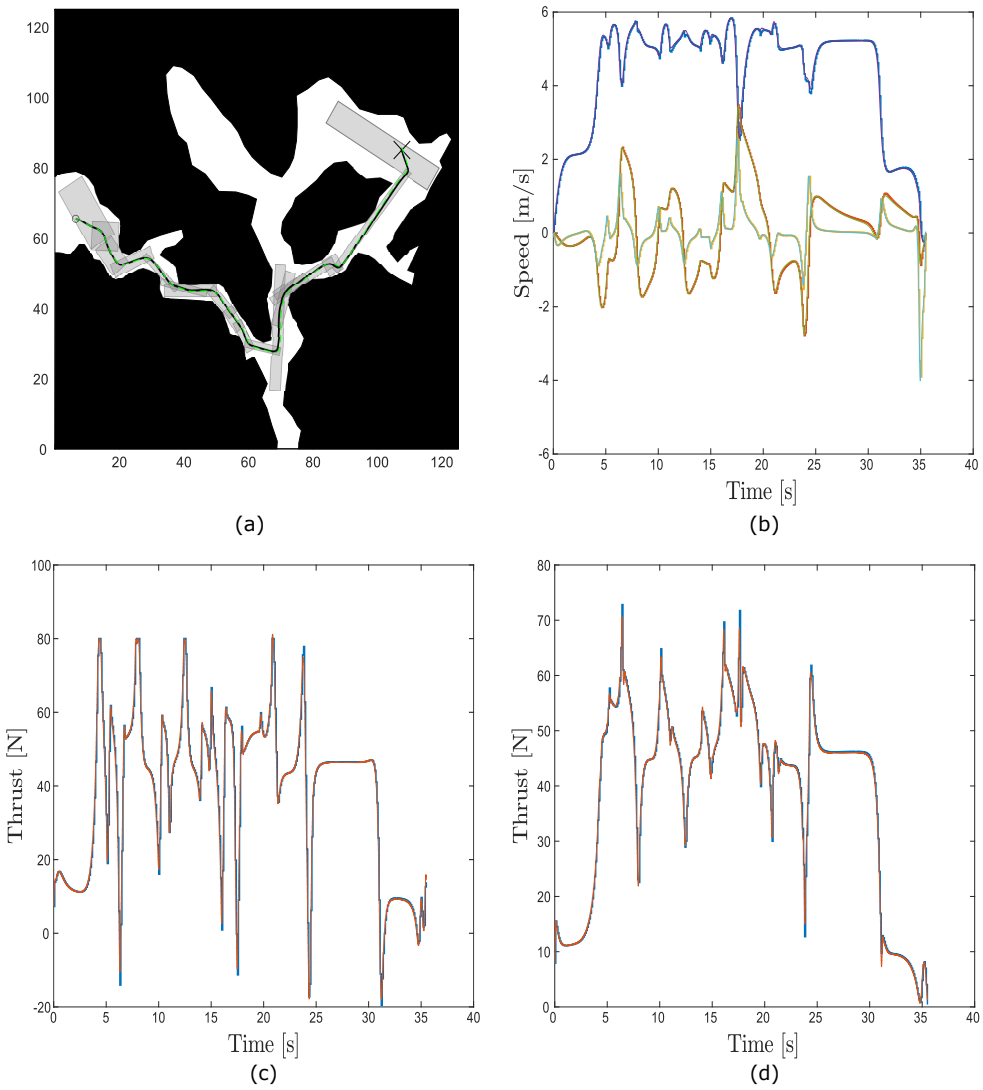


Figure 4.18: Simulation results for Map (a). Figure (a) shows the nominal position trajectory and the real closed loop trajectory. Solid black curve is the nominal trajectory and dashed green curve is the real trajectory. (b) (c) and (d) shows the nominal and real velocity and thrust trajectories as the output of the MPC. In (b), the nominal and real USV velocities are shown. Bold blue, red and yellow curves show the nominal USV velocities u, v and r, respectively. Thin purple, green and light blue curves show the real velocities on top of the nominal ones. In Figure (c) and (d), blue curves are the nominal thrust inputs F_1 and F_2, respectively. Red curves are the real thrust trajectories. Real trajectories closely follow the nominal trajectories within the constraints.

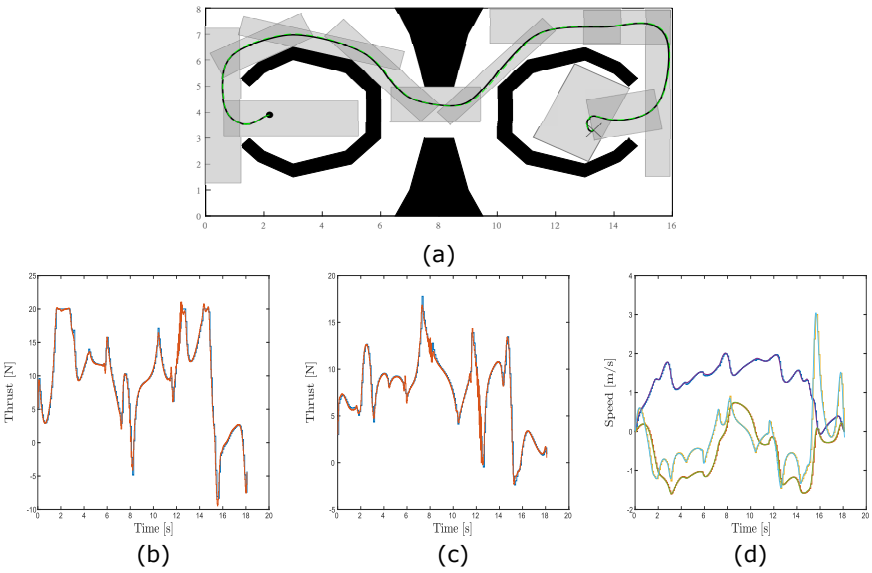


Figure 4.19: Simulation results for Map (b). Figure (a) shows the nominal position trajectory and the real closed loop trajectory. Solid black curve is the nominal trajectory and dashed green curve is the real trajectory. (b) (c) and (d) shows the nominal and real velocity and thrust trajectories as the output of the MPC. In figure (b) and (c), blue curves are the nominal thrust inputs F_1 and F_2, respectively. Red curves are the real thrust trajectories. In (d), the nominal and real USV velocities are shown. Bold blue, red and yellow curves show the nominal USV velocities u, v and r, respectively. Thin purple, green and light blue curves show the real velocities on top of the nominal ones. Real trajectories closely follow the nominal trajectories within the constraints.

4.3.2 Results in the Presence of Process Noise

After the tests without process noise, we tested the robustness of the method with Monte Carlo experiments in two different maps. The nominal trajectories for the first map is generated as Figure 4.20 using the same parameter set for map (b) of the Table 4.9, except, we changed the desired speed to the $v_{des} = 1$. We set the prediction horizon as 0.6 seconds and the control horizon as 0.03 seconds. We added the process noise to the input thrusts with $SNR = 1$ (signal-to-noise ratio). We conducted 500 experiments and only 1 of the Monte Carlo runs failed to reach the goal region with 99.8% success for the given set of parameters. The average CPU time of the MPC iterations is measured as $t_{CPU} = 0.0057$ seconds. Figure 4.21 shows the resulting position trajectories and an example MPC solution for the velocity and

thrust trajectories in the presence of the process noise.

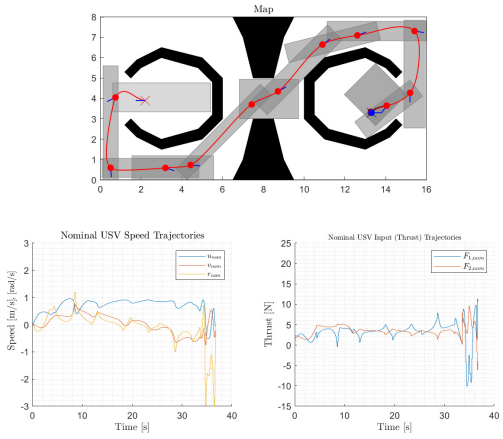


Figure 4.20: Nominal trajectories for the Monte Carlo experiments.

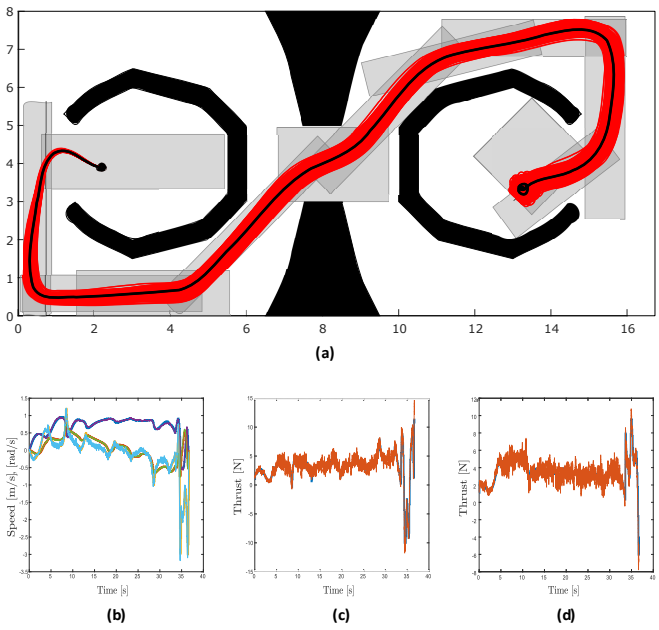


Figure 4.21: Results of the Monte Carlo experiments for the first map. (a) shows the closed loop position trajectories in the presence of process noise. (b), (c) and (d) shows the nominal and real velocity and thrust trajectories for an example run in the presence of the process noise.

In Figure 4.21(a), red traces are the USV trajectories during the tests. Solid black curve is the nominal position trajectory. In Figure 4.21(b), the nominal and real USV

velocities are shown. Bold blue, red and yellow curves show the nominal USV velocities u, v and r, respectively. Thin purple, green and light blue curves show the real velocities on top of the nominal ones. In Figures 4.21(c) and 4.21(d), blue curves are the nominal thrust inputs F_1 and F_2, respectively and red curves are the real thrust trajectories.

For the second map, the nominal trajectories are generated as Figure 4.22 using the planning parameters presented in 4.10.

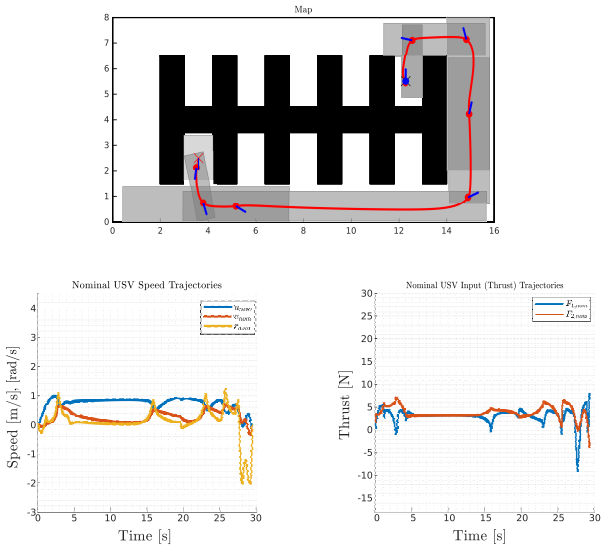


Figure 4.22: Nominal trajectories for the Monte Carlo experiments.

Table 4.10: The Parameter Set for the Monte Carlo Experiment in the Second Map.

v_{des}	$\begin{bmatrix} u & v & r \end{bmatrix}^T_{min}$	$\begin{bmatrix} u & v & r \end{bmatrix}^T_{max}$	$\begin{bmatrix} F_1 & F_2 \end{bmatrix}^T_{min}$	$\begin{bmatrix} F_1 & F_2 \end{bmatrix}^T_{max}$
1	$\begin{bmatrix} -0.6 \\ -\infty \\ -2 \end{bmatrix}$	$\begin{bmatrix} 3 \\ \infty \\ 2 \end{bmatrix}$	$\begin{bmatrix} -10 \\ -10 \end{bmatrix}$	$\begin{bmatrix} 20 \\ 20 \end{bmatrix}$

In this experiments, we set the prediction horizon as 0.6 seconds and the control horizon as 0.03 seconds. We added the process noise to the input thrusts with $SNR = 1$ (signal-to-noise ratio). We conducted 500 experiments and 6 of the Monte Carlo runs failed to reach the goal region with 98.8% success for the given set of parameters. The average CPU time of the MPC iterations is measured as $t_{CPU} = 0.0053$ seconds.

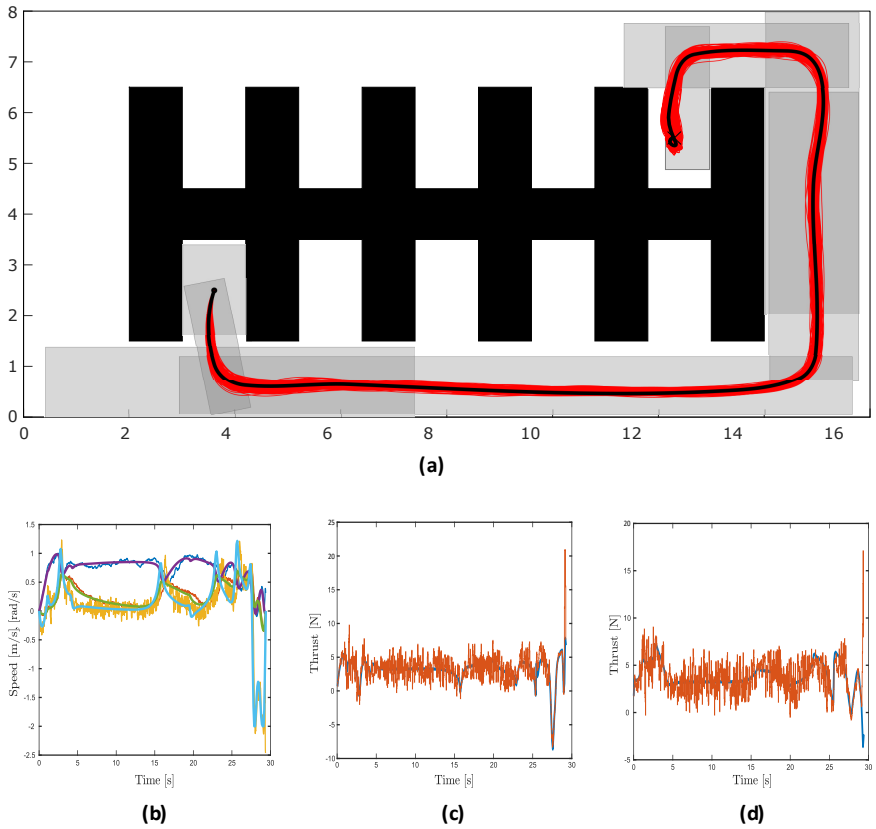


Figure 4.23: Results of the Monte Carlo experiments for the second map. (a) shows the closed loop position trajectories in the presence of process noise. (b), (c) and (d) shows the nominal and real velocity and thrust trajectories for an example run in the presence of the process noise.

In Figure 4.23(a), red traces are the USV trajectories during the tests. Solid black curve is the nominal position trajectory. In Figure 4.23(b), the nominal and real USV velocities are shown. Bold purple, cyan and yellow curves show the nominal USV velocities u, v and r, respectively. Thin blue, orange and red curves show the real velocities on top of the nominal ones. In Figures 4.23(c) and 4.23(d), blue curves are the nominal thrust inputs F_1 and F_2, respectively and red curves are the real thrust trajectories.

The results show that, our algorithm shows high robustness against external disturbances in the presence of perfect system model and the USV is able to follow the nominal trajectories even in the presence of high process noise. Also, the CPU time of the MPC iterations shows real time performance for the underactuated USV model.

4.4 Gazebo Implementation Results

For the Gazebo implementation, we tested our algorithm in four different maps. For these maps, during the planning phase, we used the following parameters.

Table 4.11: The Parameter Sets for the Gazebo Implementation Planning Phase.

	Map 1	Map 2	Map 3	Map 4
v_{des}	1.5	1	1	1
$\begin{bmatrix} u \\ v \\ r \end{bmatrix}_{min}$	$\begin{bmatrix} -0.9 \\ -0.5 \\ -1.5 \end{bmatrix}$	$\begin{bmatrix} -0.9 \\ -0.5 \\ -1.5 \end{bmatrix}$	$\begin{bmatrix} -0.8 \\ -\infty \\ -1.5 \end{bmatrix}$	$\begin{bmatrix} -0.8 \\ -\infty \\ -1.5 \end{bmatrix}$
$\begin{bmatrix} u \\ v \\ r \end{bmatrix}_{max}$	$\begin{bmatrix} 2 \\ 0.5 \\ 1.5 \end{bmatrix}$	$\begin{bmatrix} 2 \\ 0.5 \\ 1.5 \end{bmatrix}$	$\begin{bmatrix} 2 \\ \infty \\ 1.5 \end{bmatrix}$	$\begin{bmatrix} 2 \\ \infty \\ 1.5 \end{bmatrix}$
$\begin{bmatrix} F_1 \\ F_2 \end{bmatrix}_{min}$	$\begin{bmatrix} -15 \\ -15 \end{bmatrix}$	$\begin{bmatrix} -15 \\ -15 \end{bmatrix}$	$\begin{bmatrix} -15 \\ -15 \end{bmatrix}$	$\begin{bmatrix} -15 \\ -15 \end{bmatrix}$
$\begin{bmatrix} F_1 \\ F_2 \end{bmatrix}_{max}$	$\begin{bmatrix} 28 \\ 28 \end{bmatrix}$	$\begin{bmatrix} 28 \\ 28 \end{bmatrix}$	$\begin{bmatrix} 28 \\ 28 \end{bmatrix}$	$\begin{bmatrix} 28 \\ 28 \end{bmatrix}$

For the motion control, we do not impose any velocity constraint on the solver and set the input limits as the Heron USV thurster saturation limits, i.e., a maximum of 33.6 Newtons and a minimum of -19.88 Newtons.

4.4.1 Map 1

The first map, generated nodes and waypoints are shown in Figure 4.24.

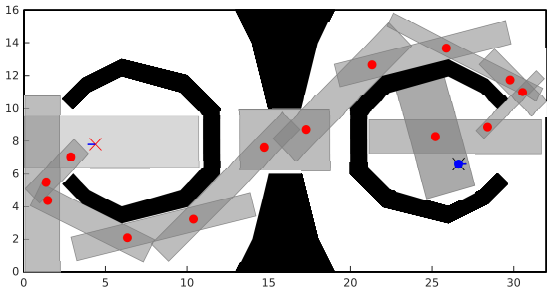


Figure 4.24: The first map for the Gazebo implementation. Grey rectangles are the generated nodes along the shortest path and the red dots are the waypoints. Red cross shows the initial position and blue arrow is the initial heading direction of the USV. Blue dot and arrow are the goal position and orientation, respectively.

After the trajectory optimization, we obtained the position, velocity and input nominal trajectories as given in Figure 4.25.

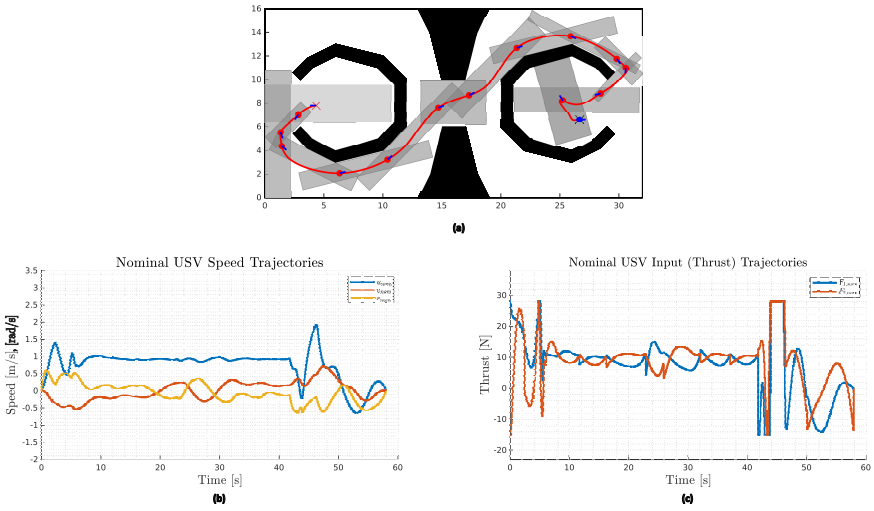


Figure 4.25: Trajectory optimization in map 1. In (a), the red curve is the nominal position trajectories. (b) shows the nominal velocity trajectories and (c) shows the nominal input trajectories.

After the nominal trajectories are generated, the Heron USV is controlled on these trajectories using MPC in the Gazebo environment. Figures 4.26 and 4.27 shows the resulting position and heading angle trajectories, Figures 4.28, 4.29 and 4.30 shows the resulting velocity trajectories, and 4.31 and 4.32 shows the resulting input trajectories, respectively.

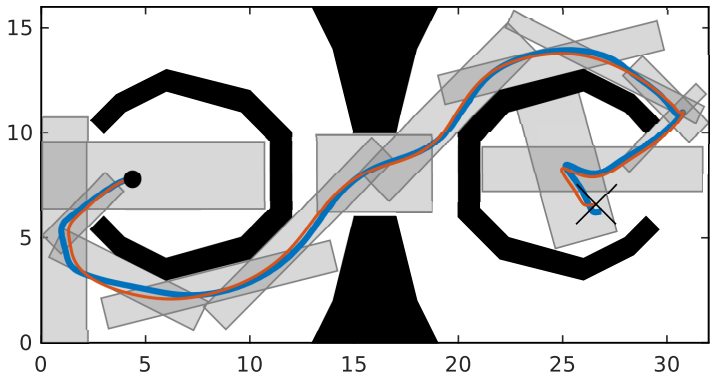


Figure 4.26: Heron USV position trajectory for map 1. Orange curve is the nominal position trajectory and blue curve is the real position trajectory of Heron.

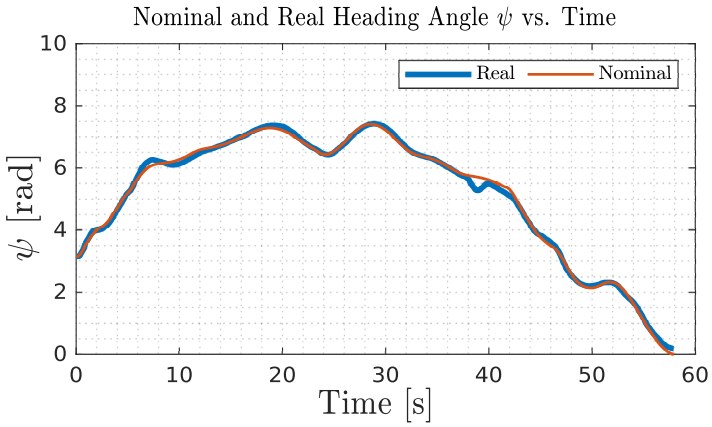


Figure 4.27: Heron USV heading angle trajectory for Map 1. Nominal trajectory is the planned trajectory and the real trajectory is Heron's trajectory during operation.

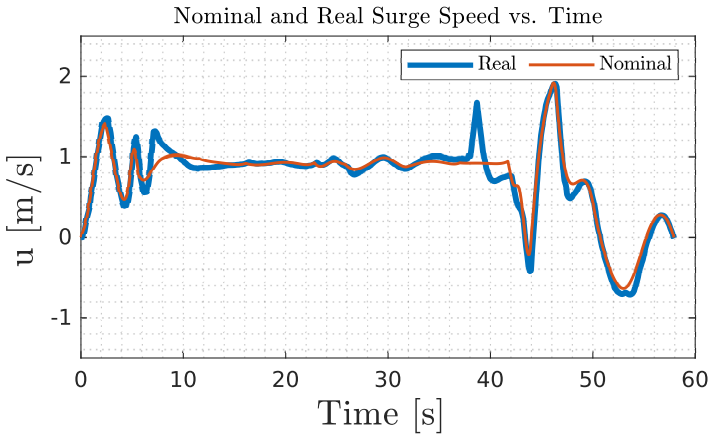


Figure 4.28: Heron USV surge velocity trajectories for Map 1.

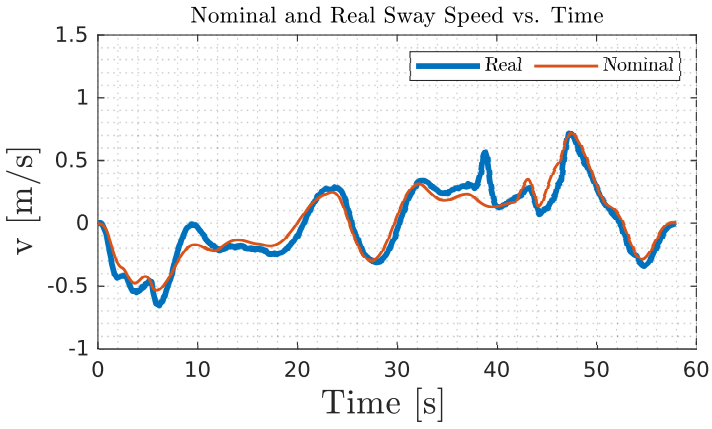


Figure 4.29: Heron USV sway velocity trajectories for Map 1.

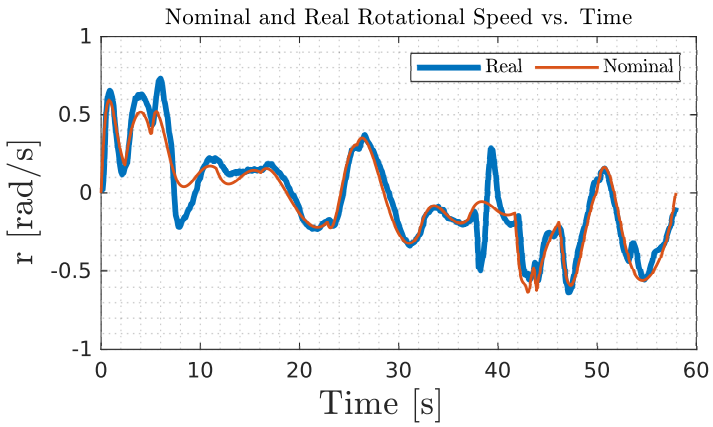


Figure 4.30: Heron USV rotational velocity trajectories for Map 1.

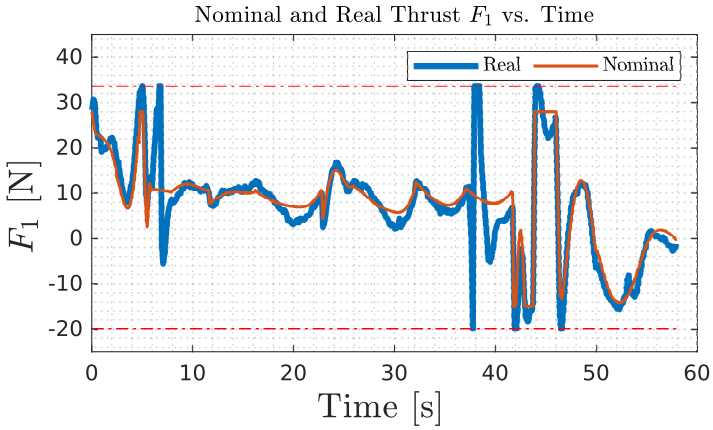


Figure 4.31: Heron USV left thruster input trajectories for Map 1.

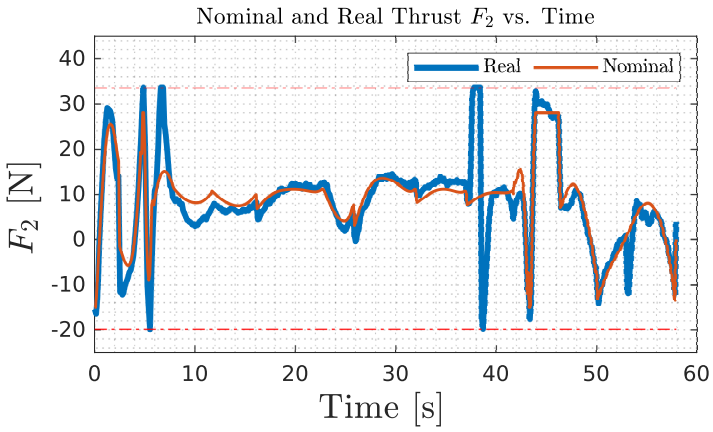


Figure 4.32: Heron USV right thruster input trajectories for Map 1.

In the trajectory optimization problem, we try to optimize the input effort of the system over the trajectories. To see the effect of the input effort optimization, we cal-

culated and presented the instantaneous mechanical power and cumulative energy consumption of the Heron USV. We calculated the instantaneous power and cumulative energy as given in (4.29). In this calculation, we included both the linear and rotational motion power consumption. For the rotational motion, we assumed the distance between Heron's thrusters is one meter. Also, the numerical results for map 1 are presented in Table 4.12.

$$P(t) = |u(t)\,(F_1(t) + F_2(t)))| + |r(t)\,(F_1(t) - F_2(t))| \tag{4.29a}$$

$$E(t) = \int_0^t P(\tau)\,d\tau \tag{4.29b}$$

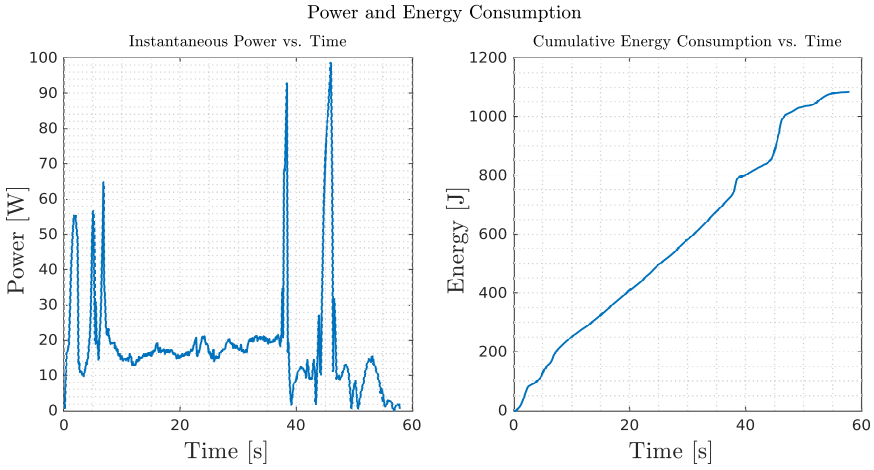


Figure 4.33: Heron USV power and energy consumption for map 1. The left figure shows Heron's instantaneous power consumption of and the right figure shows the cumulative energy consumption over time.

Table 4.12: Numerical Results for Map 1.

t_{nom}	57.9 seconds
t_{CPU}	0.0965 seconds
E_{total}	1084 Joules

In Table 4.12, the nominal time t_{nom}, average CPU time t_{CPU} for closing the loop in MATLAB and the total energy consumption E_{total} data are presented. The CPU time encompasses the time required to receive odometry data from Gazebo, solve the MPC problem and publish the thrust commands back to Gazebo.

4.4.2 Map 2

The second map, generated nodes and waypoints are shown in Figure 4.34.

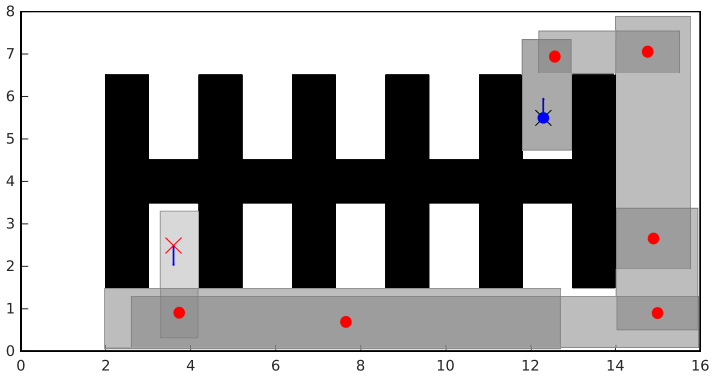


Figure 4.34: The second map for the Gazebo implementation.

After the trajectory optimization, we obtained the position, velocity and input nominal trajectories as given in Figure 4.35.

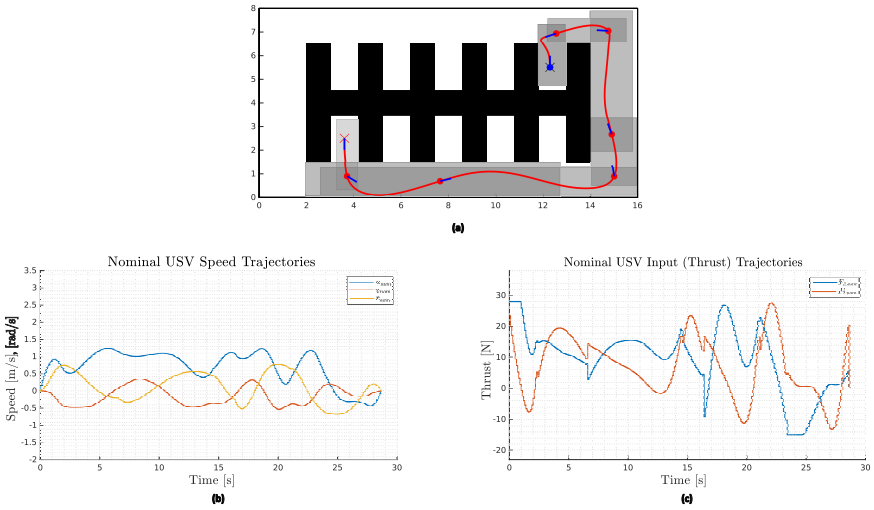


Figure 4.35: Trajectory optimization in map 2. In (a), the red curve is the nominal position trajectories. (b) shows the nominal velocity trajectories and (c) shows the nominal input trajectories.

After the motion control, Figures 4.36 and 4.37 shows the resulting position and heading angle trajectories, Figures 4.38, 4.39 and 4.40 shows the resulting velocity trajectories, and Figures 4.41 and 4.42 shows the resulting input trajectories, respectively.

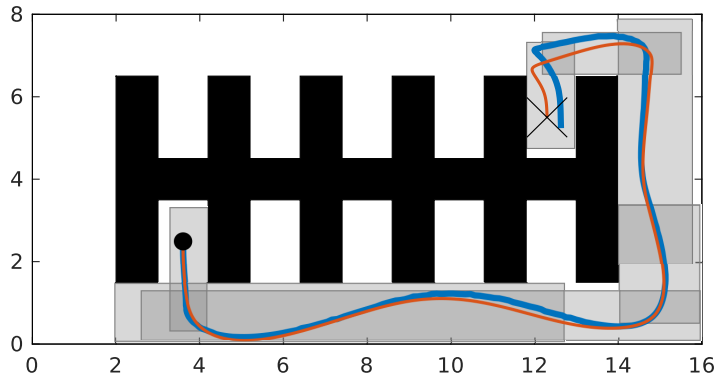


Figure 4.36: Heron USV position trajectory for Map 2. Orange curve is the nominal position trajectory and blue curve is the real position trajectory of Heron.

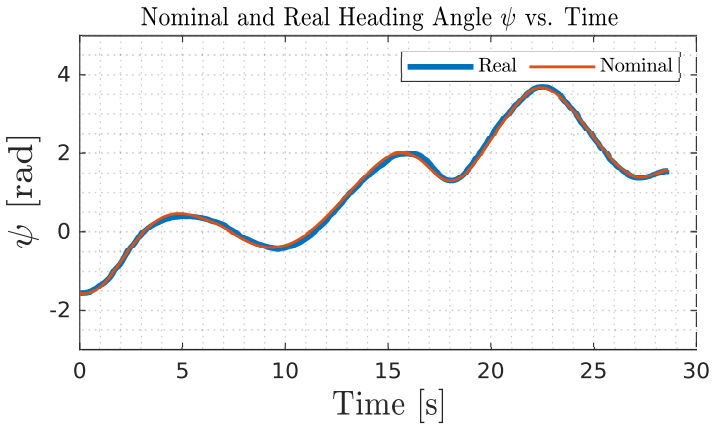


Figure 4.37: Heron USV heading angle trajectories for Map 2.

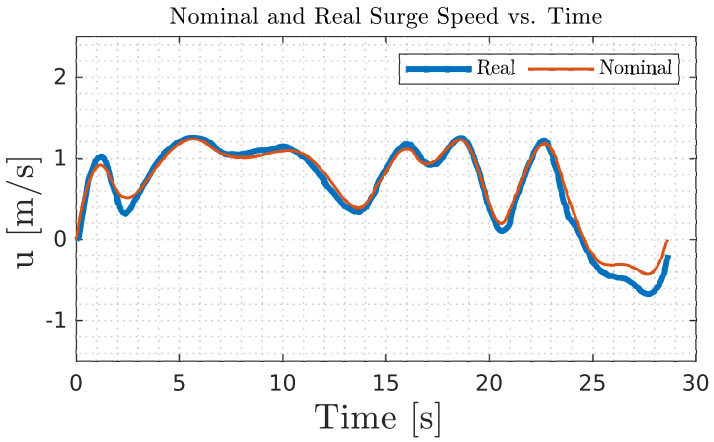


Figure 4.38: Heron USV surge velocity trajectories for Map 2.

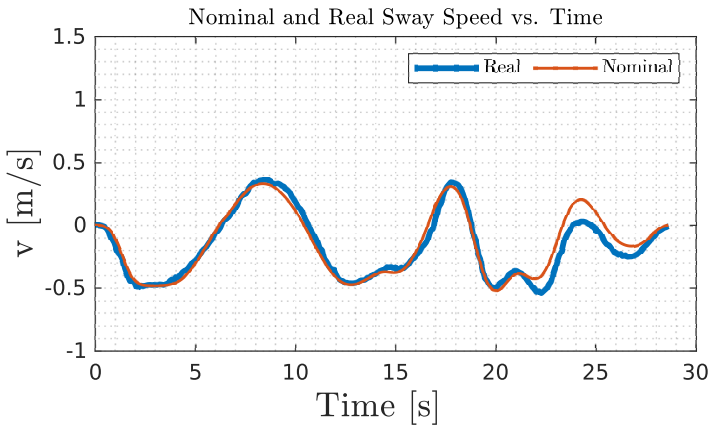


Figure 4.39: Heron USV sway velocity trajectories for Map 2.

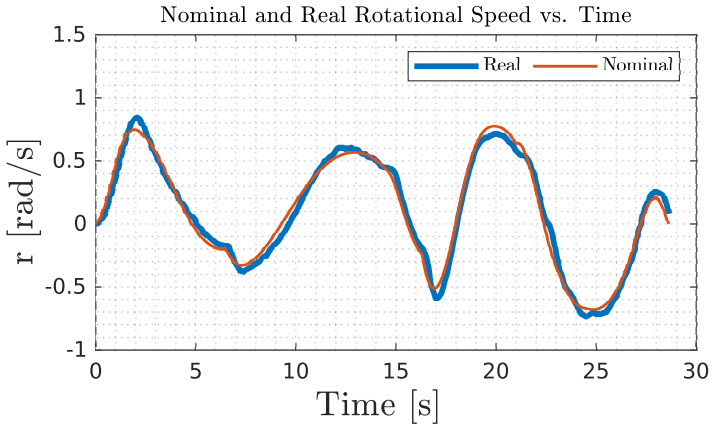


Figure 4.40: Heron USV rotational velocity trajectories for Map 2.

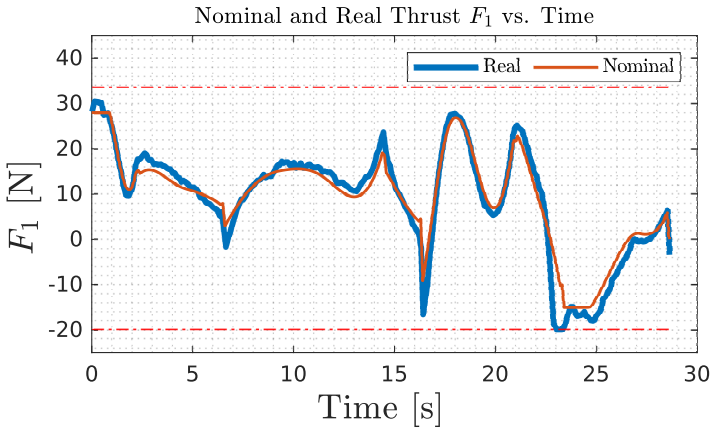


Figure 4.41: Heron USV left thruster input trajectories for Map 2.

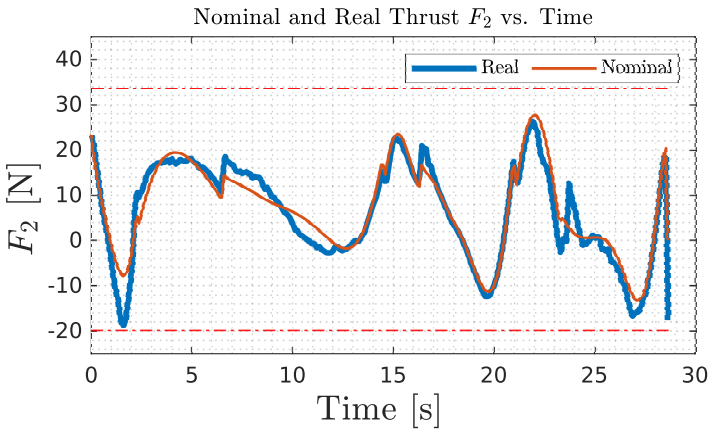


Figure 4.42: Heron USV right thruster input trajectories for Map 2.

Heron's instantaneous power and cumulative energy consumption over time is given in Figure 4.43. Also, the numerical results are given in Table 4.13.

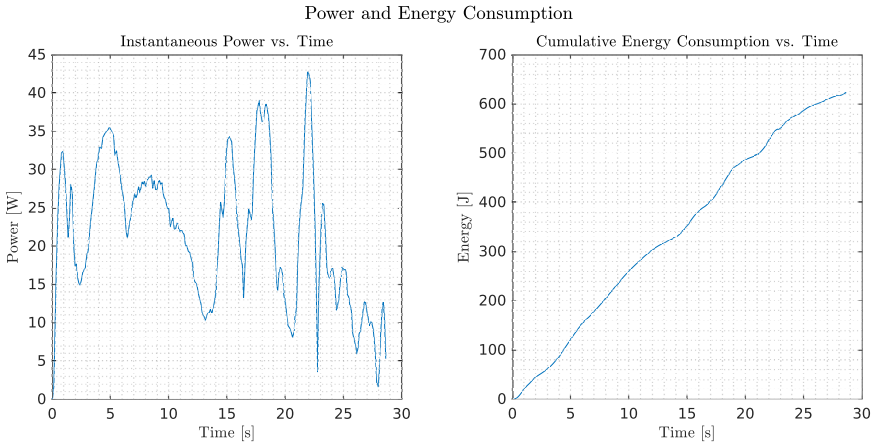


Figure 4.43: Heron USV power and energy consumption for Map 2.

Table 4.13: Numerical Results for Map 2.

t_{nom}	28.6 seconds
t_{CPU}	0.0989 seconds
E_{total}	622 Joules

4.4.3 Map 3

The third map, generated nodes and waypoints are shown in Figure 4.44.

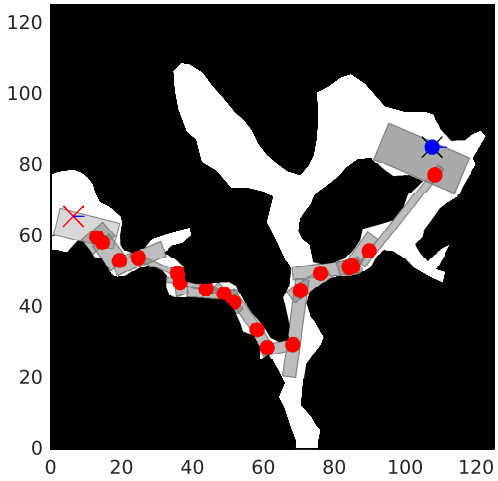


Figure 4.44: The third map for the Gazebo implementation.

After the trajectory optimization, we obtained the position, velocity and input nominal trajectories as given in Figure 4.45.

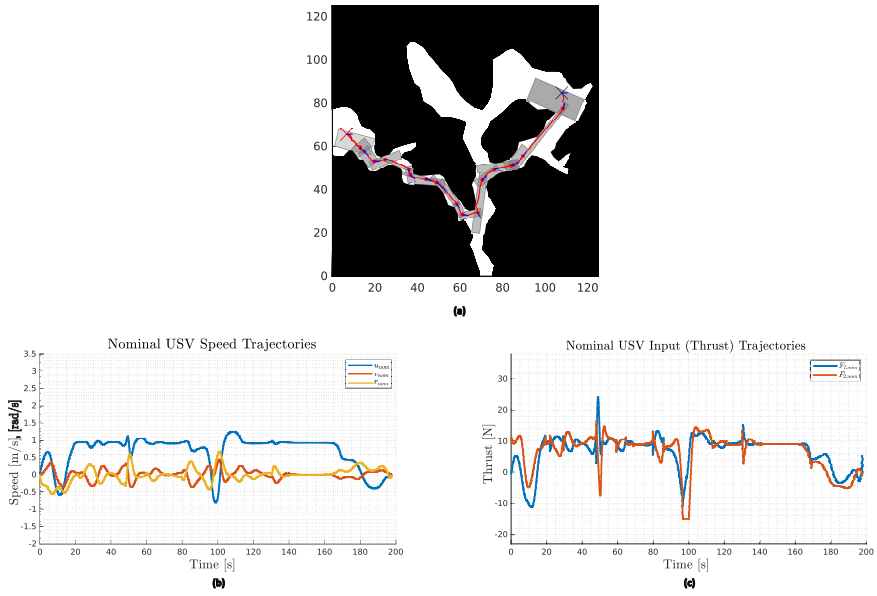


Figure 4.45: Trajectory optimization in map 3. In (a), the red curve is the nominal position trajectories. (b) shows the nominal velocity trajectories and (c) shows the nominal input trajectories.

After the motion control, Figures 4.46 and 4.47 shows the resulting position and heading angle trajectories, Figures 4.48, 4.49 and 4.50 shows the resulting velocity trajectories, and Figures 4.51 and 4.52 shows the resulting input trajectories, respectively.

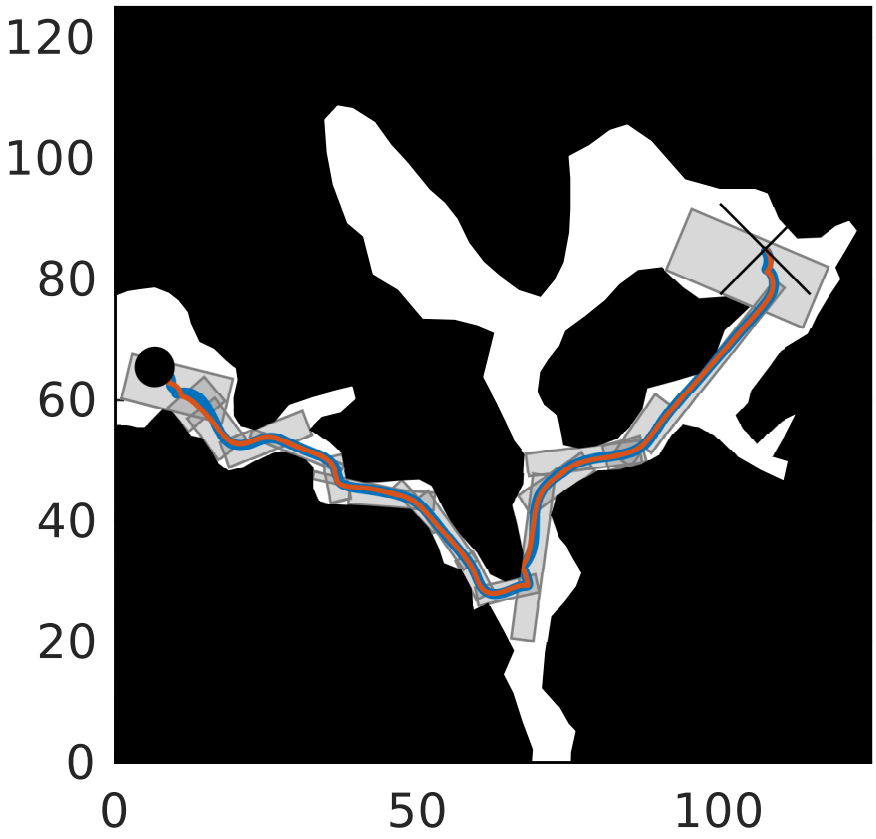


Figure 4.46: Heron USV position trajectory for Map 3. Orange curve is the nominal position trajectory and blue curve is the real position trajectory of Heron.

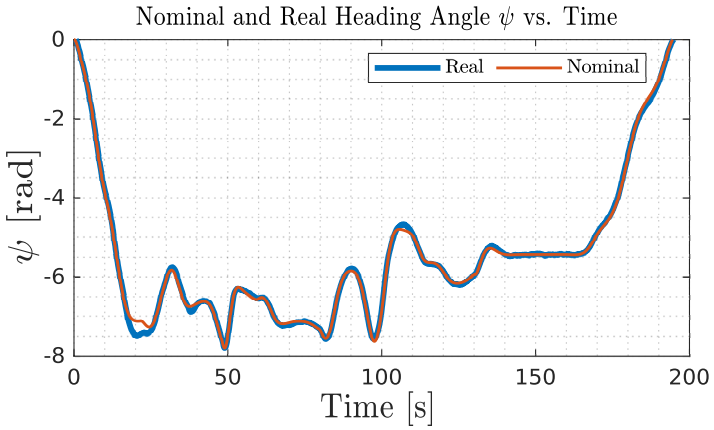


Figure 4.47: Heron USV heading angle trajectories for Map 3.

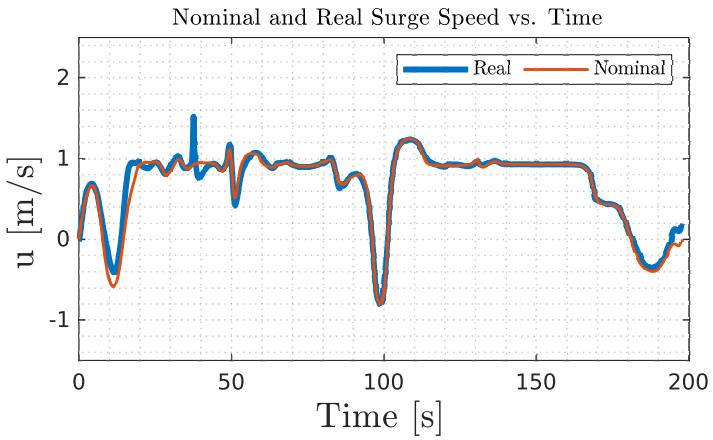


Figure 4.48: Heron USV surge velocity trajectories for Map 3.

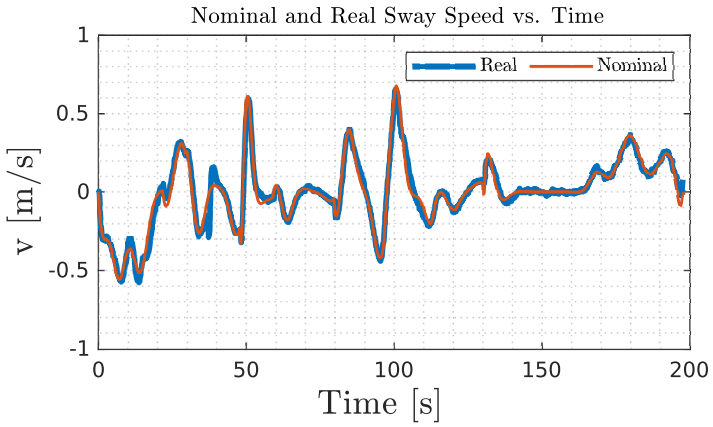


Figure 4.49: Heron USV sway velocity trajectories for Map 3.

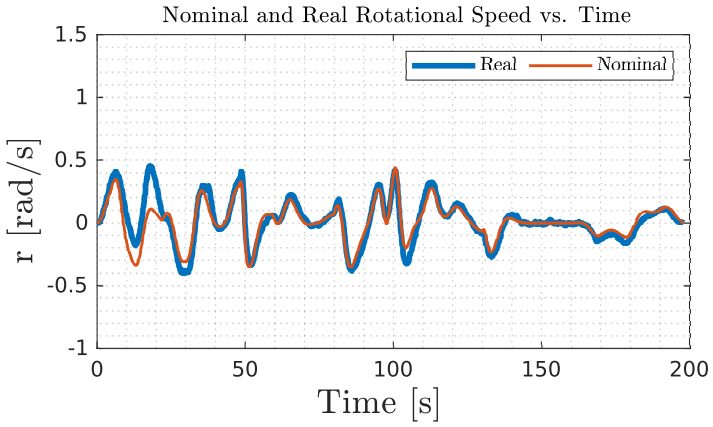


Figure 4.50: Heron USV rotational velocity trajectories for Map 3.

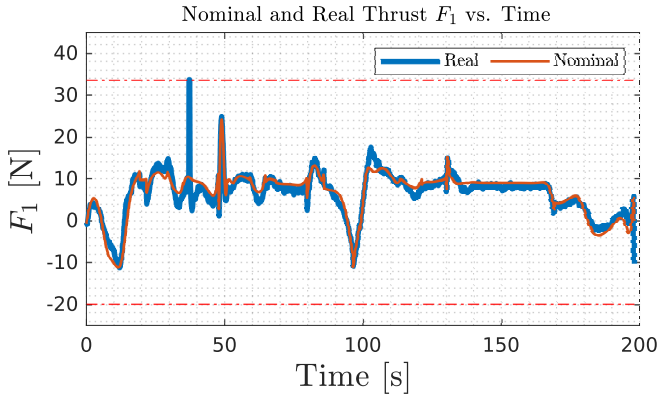


Figure 4.51: Heron USV left thruster input trajectories for Map 3.

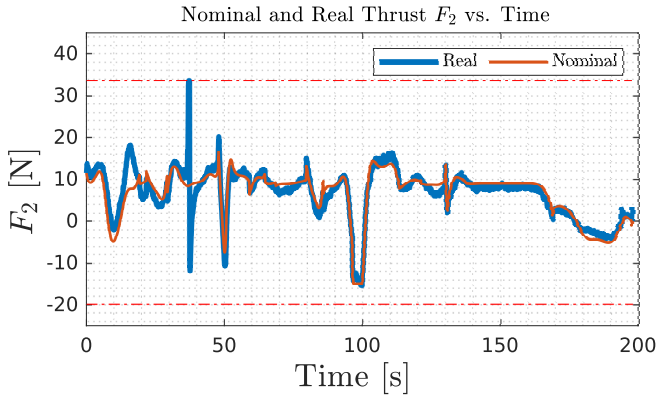


Figure 4.52: Heron USV right thruster input trajectories for Map 3.

Heron's instantaneous power and cumulative energy consumption over time is given in Figure 4.53. Also, the numerical results are given in Table 4.14

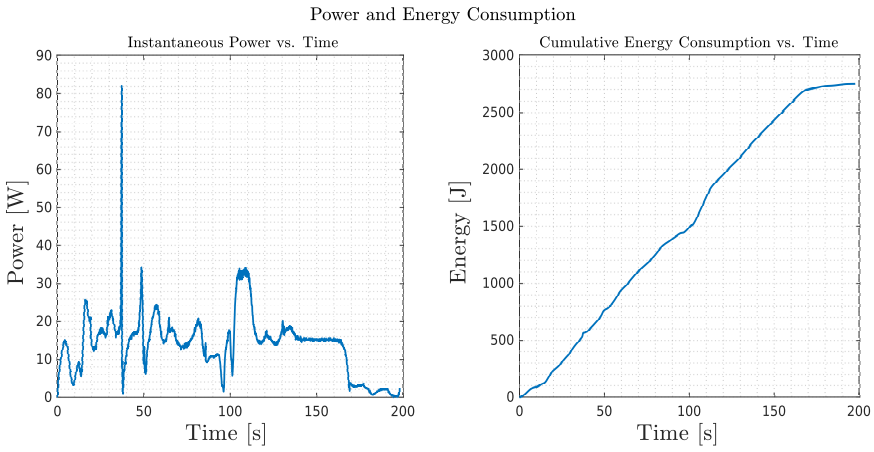


Figure 4.53: Heron USV power and energy consumption for Map 3.

Table 4.14: Numerical Results for Map 3.

t_{nom}	197.7 seconds
t_{CPU}	0.0748 seconds
E_{total}	2752 Joules

4.4.4 Map 4

The fourth map, generated nodes and waypoints are shown in Figure 4.54.

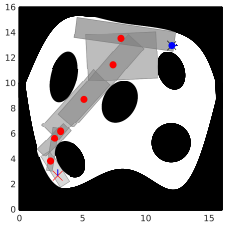


Figure 4.54: The fourth map for the Gazebo implementation.

After the trajectory optimization, we obtained the position, velocity and input nominal trajectories as given in Figure 4.55.

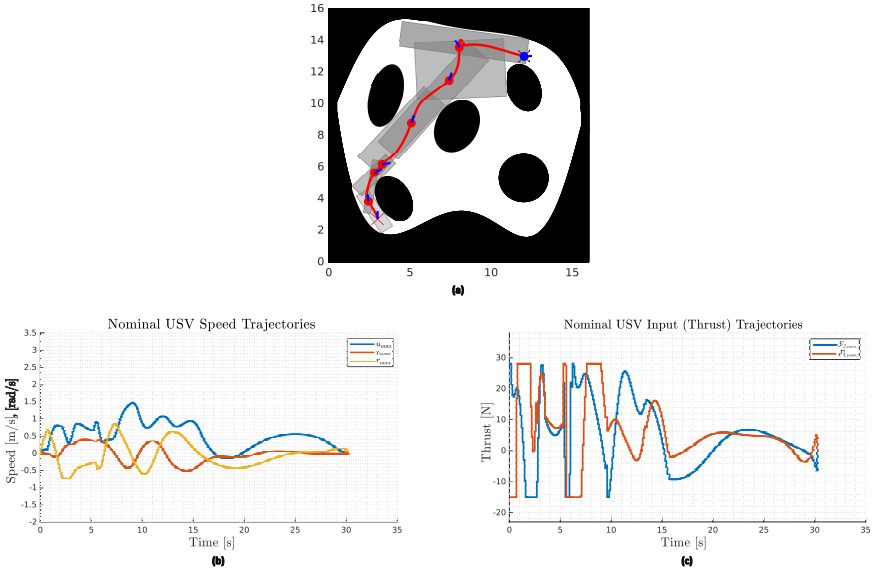


Figure 4.55: Trajectory optimization in Map 4. In (a), the red curve is the nominal position trajectories. (b) shows the nominal velocity trajectories and (c) shows the nominal input trajectories.

After the motion control, Figures 4.56 and 4.57 shows the resulting position and heading angle trajectories, Figures 4.58, 4.59 and 4.60 shows the resulting velocity trajectories, and Figures 4.61 and 4.62 shows the resulting input trajectories, respectively.

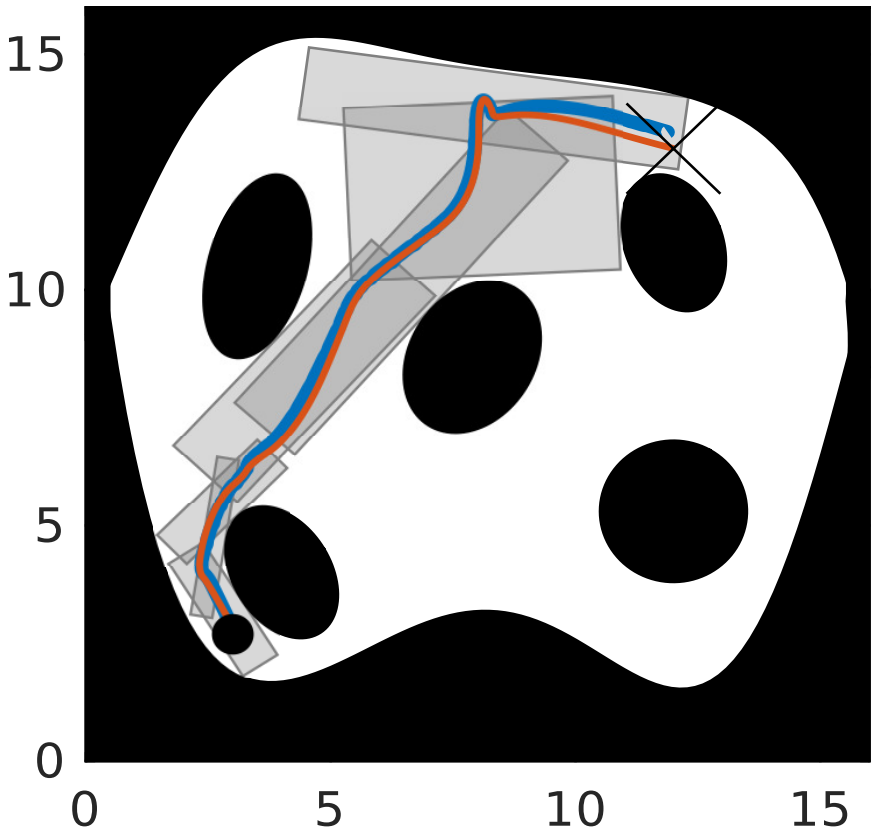


Figure 4.56: Heron USV position trajectory for Map 4. Orange curve is the nominal position trajectory and blue curve is the real position trajectory of Heron.

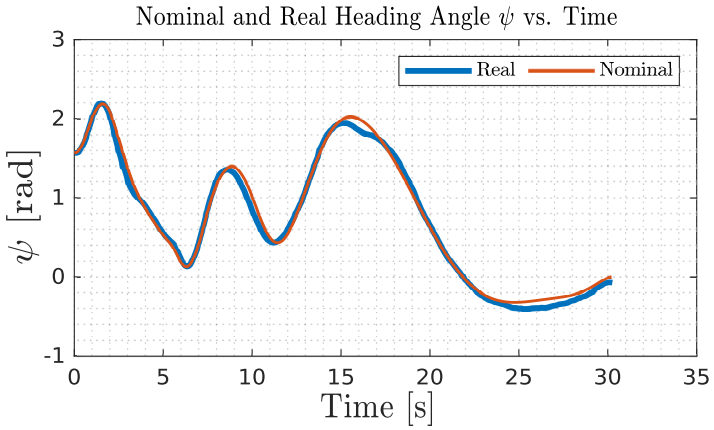


Figure 4.57: Heron USV heading angle trajectories for Map 4.

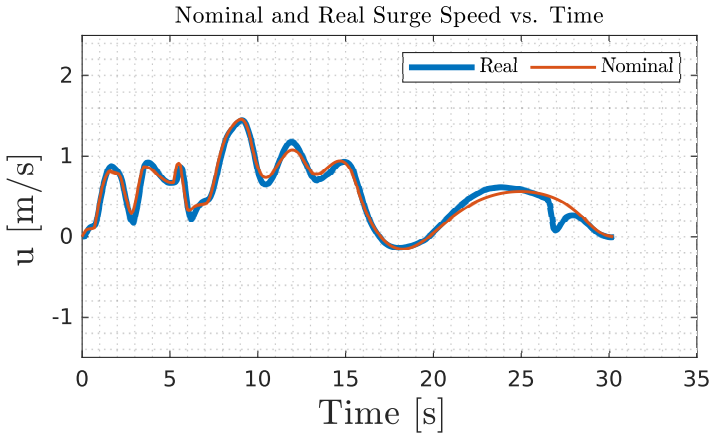


Figure 4.58: Heron USV surge velocity trajectories for Map 4.

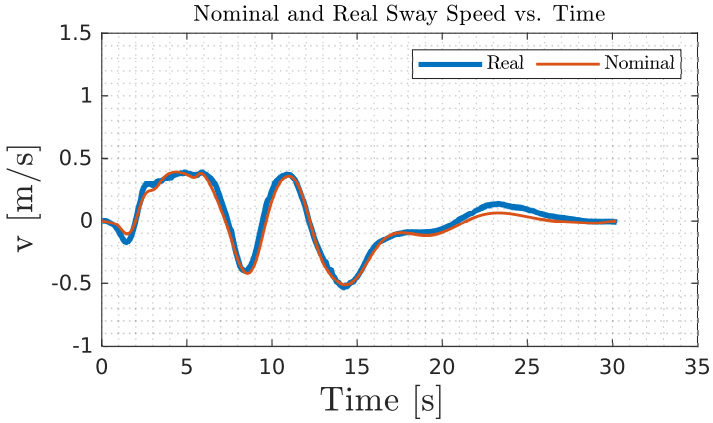


Figure 4.59: Heron USV sway velocity trajectories for Map 4.

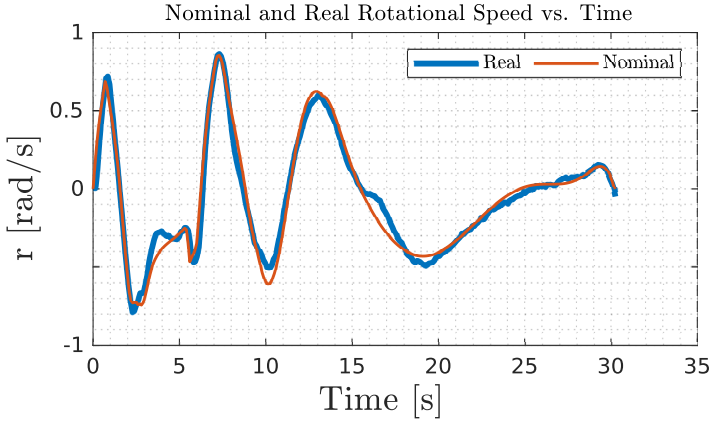


Figure 4.60: Heron USV rotational velocity trajectories for Map 4.

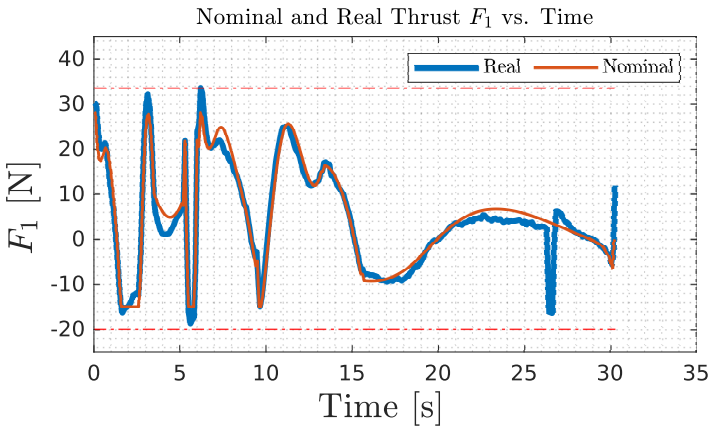


Figure 4.61: Heron USV left thruster input trajectories for Map 4.

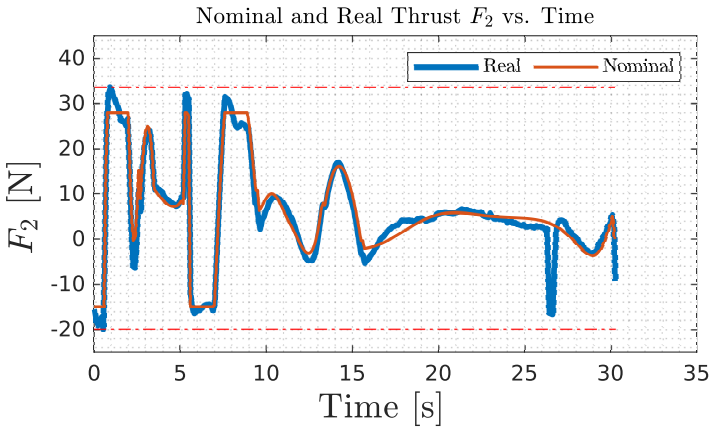


Figure 4.62: Heron USV right thruster input trajectories for Map 3.

Heron's instantaneous power and cumulative energy consumption over time is given in Figure 4.63. Also, the numerical results are given in Table 4.15

Table 4.15: Numerical Results for Map 4.

t_{nom}	30.2 seconds
t_{CPU}	0.1310 seconds
E_{total}	393 Joules

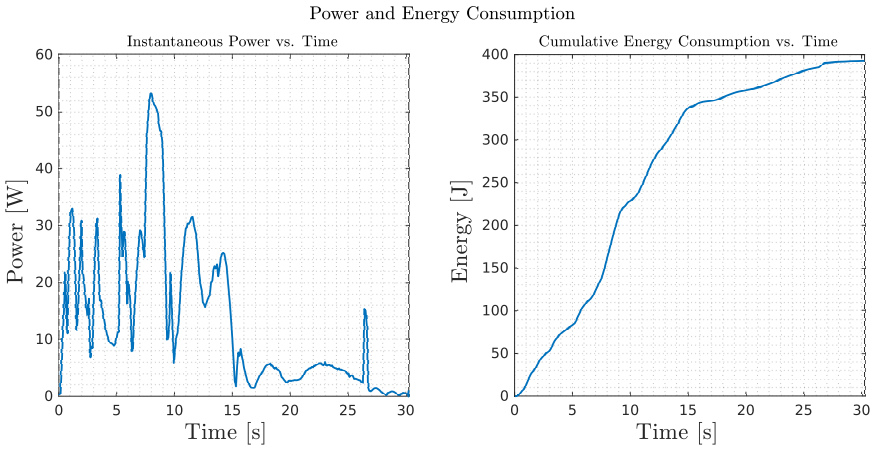


Figure 4.63: Heron USV power and energy consumption for Map 4. The left figure shows Heron's instantaneous power consumption of and the right figure shows the cumulative energy consumption over time.

4.5 Comparion With Previous Work

The method developed in this study mainly aims to increase the performance of the MPC-Graph method presented in [6] for the underactuated systems. Thus, we used the underactuated version of the fully actuated USV model used in [6] for the MATLAB simulations and we used the same maps presented in [6]. With MPC-Graph, it is reported that the Monte Carlo experiments conducted with the fully actuated USV in the map 4.20 shows 98% success. Our method shows a success rate of 99.8% even with the underactuated USV subject to more strict input limits. Figure 4.64 shows the result of their experiments on map 4.20 and an example solution for the thruster inputs.

In the MPC-Graph work, it is also reported that the underactuated USV model has a success rate of 71.2% in the map presented in Figure 4.22 for the Monte Carlo Experiments. However, our method shows a 98.8% success in the presence of process noise, which is a major improvement. Figure 4.65 shows the result of their experiments on map 4.22 and an example solution for the thruster inputs.

For the real-time performance, the MPC-Graph study reports that the CPU time required to solve one iteration of the MPC problem is $t_{CPU} = 2.04$ seconds due to the extensive prediction horizon required to stabilize the underactuated USV, and, it does

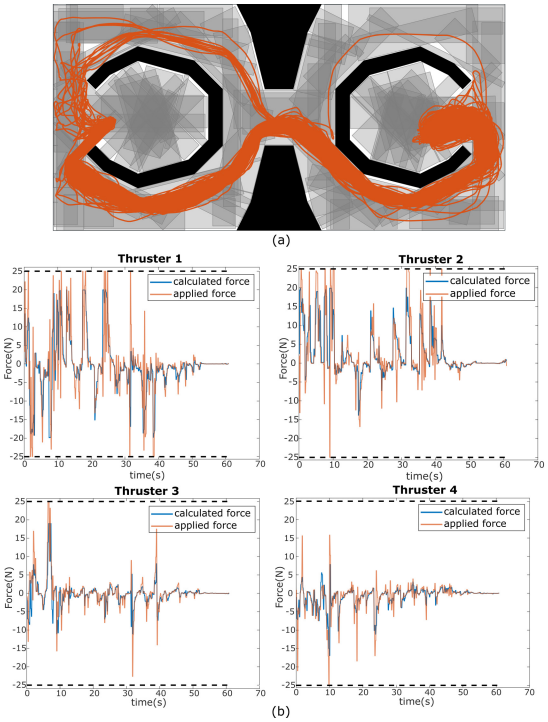


Figure 4.64: The Monte Carlo Experiment results presented in [6] for the fully actuated USV. (a) shows the successful trials that end up at the goal position. (b) shows the thruster commands for all four thrusters of the fully actuated USV. MPC-Graph shows 98% success during the trials with the fully actuated USV.

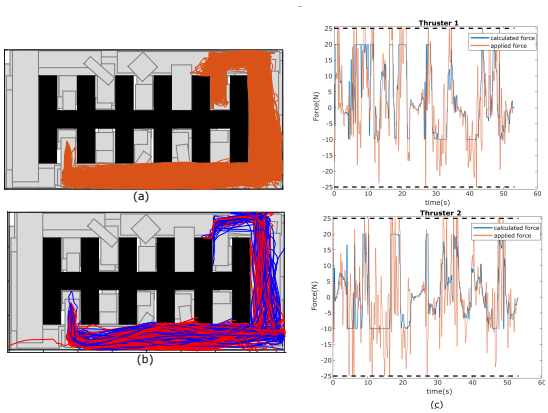


Figure 4.65: The Monte Carlo Experiment results presented in [6] for the underactuated USV. (a) shows the successful trials that end up at the goal position. (b) shows the failed trials. (c) shows the thruster commands of the underactuated USV. MPC-Graph shows 71.2% success during the trials with the underactuated USV.

not satisfy the real-time requirements for this reason. In our method, since we make the predictions of trajectories offline in the planning phase, we do not require such high prediction horizons for the MPC problem. Also, after we generate trajectories, we use a linear MPC on the linearized dynamics where MPC-Graph uses nonlinear MPC for the underactuated USV. For these reasons, we achieve a real-time performance, with CPU times near the sampling rate of 0.1 seconds to close the control loop during our experiments. Table 4.16 shows the comparison between the MPC-Graph and our method in terms of the time complexity for real-time operation.

Table 4.16: Time Complexity Comparison with MPC-Graph.

MPC-Graph Average	2.04 seconds
Map 1	0.0965 seconds
Map 2	0.0989 seconds
Map 3	0.0748 seconds
Map 4	0.1310 seconds

Our method also has the capability of defining a desired surge speed at the planning phase, which reduces the speed oscillations during the operation. One disadvantage of our method is the required computational power during the planning phase. We solve the optimization problem to generate trajectories for two full funnels, which results in very high time horizons for the optimizer. The CPU time required to solve the trajectory optimization problem increases exponentially with the time horizon in our observations. So, it might be very costly to solve the trajectory optimization problem for large funnels. On the other hand, MPC-Graph has a real-time performance in the planning phase.

For the power and energy comparison, MPC-Graph study does not include any power and energy analysis. However, Heron USV has a 14.4 Volts battery with a 29 Ampere-hour capacity, which is equal to 417.6 Watt-hours capacity. So, Heron USV battery is able to supply 417.6 Watts uninterrupted electrical power for one hour. It is also reported that the typical operation time for Heron USV at the rated conditions is 2.5 hours. So, we can assume that Heron draws 167 Watts for a typical operation. Thus, the maximum mechanical power dissipation of Heron USV for all four maps stays below the typical power consumption.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

In this work, we presented the method of receding horizon trajectory optimization over Sparse Neighborhood Graphs for the control of underactuated USVs. Our method conducts a trajectory optimization on the funnels generated by SNG algorithm to generate feasible nominal state and input trajectories to effectively control underactuated USVs. Our method starts with the generation of SNG on a given workspace for the USV. After we generate the graph, we conduct Dijkstra's Algorithm to find the node sequence along the shortest path that connects the starting configuration to the goal. After that, we generate waypoints for our USV on the intersection of these nodes. These waypoints serves as the local initial and final configurations for the trajectory optimization algorithm and the union of all these nodes serves as the operation region of the USV. After the waypoints are generated, we solve the receding horizon trajectory optimization problem at each node to generate nominal trajectories inside these nodes. The union of all these trajectories at each node along the shortest path is the nominal trajectories connecting the starting configuration the the goal. Then, we linearize the dynamics of the USV around these trajectories and drive the USV along the nominal trajectories inside the obstacle free region using a time-varying MPC policy. During the trajectory optimization, we enforce the generated trajectories to stay inside the each individual node, or funnel, and obey the speed and input limits. The use of MPC guarantees that the USV always stays inside the obstacle free region of the workspace of the USV since we constrain the system to stay inside the correct funnel at correct time using MPC. As long as the MPC problem is feasible, it is guaranteed that the USV reaches the goal configuration.

In our method, we employed a receding horizon strategy to obtain smooth trajectories for the USV. We observed that, if we don't take next funnels into account during the trajectory optimization, the configuration of the robot at the outlet of the individual funnel might be infeasible to reach the next waypoint. Also, we observed that the computational time required for the trajectory planning increases exponentially with the length of the funnel, so, we only take the next node into consideration during the trajectory optimization.

For the implementation, we used 2 different underactuated USV models, one simulated in MATLAB with a pre-defined model from a previous study and other is the Clearpath Robotics Heron USV. Since Heron has no dynamical model reported in the current literature, we conducted a NARMAX methods based nonlinear system identification. To capture its dynamics in every axis, we used an extensive training input. Our system identification showed nearly a perfect fit in the surge and rotational axes, however, it overestimated the sway axis damping since we use a simplified base model for the identification. We noticed that increasing the number of nonlinear terms in the sway axis might give a better fit on that axis. We utilised a Docker container to run the simulation of Heron USV and we closed the loop on MATLAB, communicating with the docker container. During these experiments, we noticed the communication delay between MATLAB and Gazebo environments.

We conducted two Monte Carlo experiments in the MATLAB simulations with a relatively high process noise to show the robustness of our method. We also conducted experiments on four different maps with Heron USV to show the real-time performance of our method. Finally, we reported the mechanical power demand of Heron during these experiments to show the power and energy consumption on optimized trajectories. Our method showed superior results on the real-time performance and robustness for the underactuated USV, and, the mechanical power consumption of Heron stayed below the reported nominal electrical power consumption on its user manual.

Even if our method shows superior results in the real-time control, the planning phase requires a very high computational power and far away from being used for real-time planning actions. We noticed that the reasons for the high computational time are the

length of the funnels and the software architecture. Also, our method currently can't generate an alternative trajectory towards the goal if it gets out of the funnel due to disturbances. However, these issues can be solved in the future, using a smart way to generate shorter funnels and migrating the software from MATLAB to C++ and ROS with a smarter software architecture, which can also decrease the communication delay mentioned earlier. As we reach the near real-time performance for the planning, we can use our algorithm to generate alternative trajectories for the USV on the run if it gets out of the funnel.

5.2 Future Work

Our main task will be to implement our method on physical Heron USV to examine its performance in real-world after this work.

Secondly, we will make the algorithmic and software architecture modifications explained in the previous section for a better planning performance.

We will also try to generalize the our receding horizon trajectory optimization method for other sequential composition methods, in both 2D and 3D applications for various underactuated systems in the future.

REFERENCES

[1] 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, 2020.

[2] S. Atasoy, O. K. Karagöz, and M. M. Ankarali, "Trajectory-free motion planning of an unmanned surface vehicle based on MPC and Sparse Neighborhood Graph," *IEEE Access*, vol. 11, pp. 47690–47700, 2023.

[3] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[4] R. Tedrake, *Underactuated Robotics*. MIT, 2023.

[5] Clearpath Robotics, "Heron Tutorials 0.1.1 Documentation." https://www.clearpathrobotics.com/assets/guides/kinetic/heron/index.html, 2020.

[6] S. Atasoy, "MPC-Graph: Nonlinear feedback motion planning using sparse sampling based neighborhood graph," January 2022.

[7] M. W. Spong, "Underactuated mechanical systems," in *Control Problems in Robotics and Automation* (B. Siciliano and K. P. Valavanis, eds.), (Berlin, Heidelberg), pp. 135–150, Springer Berlin Heidelberg, 1998.

[8] Y. Liu and H. Yu, "A survey of underactuated mechanical systems," *IET Control Theory & Applications*, vol. 7, no. 7, pp. 921–935, 2013.

[9] P. G. Morasso, "Motor control models: Learning and performance," in *International Encyclopedia of the Social & Behavioral Sciences (Second Edition)* (J. D. Wright, ed.), pp. 957–964, Oxford: Elsevier, second edition ed., 2015.

[10] C. Eilers, J. Eschmann, R. Menzenbach, B. Belousov, F. Muratore, and J. Peters, "Underactuated waypoint trajectory optimization for light painting photography," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1505–1510, 2020.

[11] R. YU and P. LEUNG, "Optimal partial harvesting schedule for aquaculture operations," *Marine Resource Economics*, vol. 21, no. 3, pp. 301–315, 2006.

[12] R. Martin, "Optimal control drug scheduling of cancer chemotherapy," *Automatica*, vol. 28, no. 6, pp. 1113–1123, 1992.

[13] R. Loxton, K. Teo, V. Rehbock, and W. Ling, "Optimal switching instants for a switched-capacitor DC/DC power converter," *Automatica*, vol. 45, no. 4, pp. 973–980, 2009.

[14] B. Açıkmeşe and L. Blackmore, "Lossless convexification of a class of optimal control problems with non-convex control constraints," *Automatica*, vol. 47, no. 2, pp. 341–347, 2011.

[15] M. Chyba, T. Haberkorn, R. Smith, and S. Choi, "Design and implementation of time efficient trajectories for autonomous underwater vehicles," *Ocean Engineering*, vol. 35, no. 1, pp. 63–76, 2008.

[16] K. Zheng, P. Huang, and G. P. Fettweis, "Optimal control of quadrotor attitude system using data-driven approximation of Koopman operator," *IFAC-PapersOnLine*, vol. 56, no. 2, pp. 834–840, 2023. 22nd IFAC World Congress.

[17] R. Bellman, "Dynamic programming," *Science*, vol. 153, no. 3731, pp. 34–37, 1966.

[18] W. B. Powell, "A unified framework for stochastic optimization," *European Journal of Operational Research*, vol. 275, no. 3, pp. 795–821, 2019.

[19] J. T. Betts, "Survey of numerical methods for trajectory optimization," *Journal of Guidance, Control, and Dynamics*, vol. 21, no. 2, pp. 193–207, 1998.

[20] J. T. Betts and I. Kolmanovski, *Optimal Control Preliminaries*, ch. Chapter 3, pp. 85–114. ASME, 1987.

[21] C. Hargraves and S. Paris, "Direct trajectory optimization using nonlinear programming and collocation," *AIAA J. Guidance*, vol. 10, pp. 338–342, 07 1987.

[22] X. Tang and J. Chen, "Direct trajectory optimization and costate estimation of infinite-horizon optimal control problems using collocation at the flipped legendre-gauss-radau points," *IEEE/CAA Journal of Automatica Sinica*, vol. 3, no. 2, pp. 174–183, 2016.

[23] I. M. Ross and M. Karpenko, "A review of pseudospectral optimal control: From theory to flight," *Annual Reviews in Control*, vol. 36, no. 2, pp. 182–197, 2012.

[24] P. T. Boggs and J. W. Tolle, "Sequential quadratic programming," *Acta numerica*, vol. 4, pp. 1–51, 1995.

[25] J. Nocedal and S. J. Wright, "Interior-point methods for nonlinear programming," *Numerical Optimization*, pp. 563–597, 2006.

[26] C. R. Cutler and B. L. Ramaker, "Dynamic matrix control, a computer control algorithm," *IEEE Transactions on Automatic Control*, vol. 17, p. 72, 1979.

[27] M. G. Forbes, R. S. Patwardhan, H. Hamadah, and R. B. Gopaluni, "Model predictive control in industry: Challenges and opportunities," *IFAC-PapersOnLine*, vol. 48, no. 8, pp. 531–538, 2015. 9th IFAC Symposium on Advanced Control of Chemical Processes ADCHEM 2015.

[28] Y. Ma, Z. Liu, T. Wang, S. Song, J. Xiang, and X. Zhang, "Multi-model predictive control strategy for path-following of unmanned surface vehicles in wide-range speed variations," *Ocean Engineering*, vol. 295, p. 116845, 2024.

[29] 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*, vol. PP, pp. 1–1, 05 2021.

[30] 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, p. 926, 06 2018.

[31] 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, 2017.

[32] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[33] 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.

[34] L. Kavraki, P. Svestka, J.-C. Latombe, and M. 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.

[35] S. M. LaValle, "Rapidly-exploring random trees : a new tool for path planning," *The annual research report*, 1998.

[36] R. R. Burridge, A. A. Rizzi, and D. E. Koditschek, "Sequential composition of dynamically dexterous robot behaviors," *The International Journal of Robotics Research*, vol. 18, no. 6, pp. 534–555, 1999.

[37] O. Arslan and U. Saranli, "Reactive planning and control of planar spring–mass running on rough terrain," *IEEE Transactions on Robotics*, vol. 28, pp. 567–579, 06 2012.

[38] D. Conner, H. Choset, and A. Rizzi, "Flow-through policies for hybrid controller synthesis applied to fully actuated systems," *Robotics, IEEE Transactions on*, vol. 25, pp. 136 – 146, 03 2009.

[39] R. Gregg, T. Bretl, and M. Spong, "Asymptotically stable gait primitives for planning dynamic bipedal locomotion in three dimensions," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 1695 – 1702, 06 2010.

[40] A. Rizzi, J. Gowdy, and R. Hollis, "Distributed coordination in modular precision assembly systems," *I. J. Robotic Res.*, vol. 20, pp. 819–838, 10 2001.

[41] 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.

[42] 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.

[43] O. Özdemir, "Feedback motion planning of a novel fully actuated unmanned surface vehicle via sequential composition of random elliptical funnels," December 2022.

[44] L. Yang and S. 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.

[45] 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.

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

[47] S. Billings and I. Leontaritis, "Parameter estimation techniques for nonlinear systems," *IFAC Proceedings Volumes*, vol. 15, no. 4, pp. 505–510, 1982. 6th IFAC Symposium on Identification and System Parameter Estimation, Washington USA, 7-11 June.

[48] S. Billings, *Models for Linear and Nonlinear Systems*, ch. 2, pp. 17–59. John Wiley & Sons, Ltd, 2013.

[49] H. Shiobara and N. Hori, "Numerical exact discrete-time-model of linear time-varying systems," in *2008 International Conference on Control, Automation and Systems*, pp. 2314–2318, 2008.

[50] G. D. Meena and S. Janardhanan, "Discretization of linear time-varying systems," in *2020 International Conference on Emerging Frontiers in Electrical and Electronic Technologies (ICEFEET)*, pp. 1–6, 2020.

[51] Clearpath Robotics, "Heron Simulator." https://github.com/heron/heron_simulator, 2022.

[52] MathWorks, "Loss Function and Model Quality Metrics." https://www.mathworks.com/help/ident/ug/model-quality-metrics.html, 2024. [Accessed 21-08-2024].

[53] MathWorks, "Nonlinear ARX Models." https://www.mathworks.com/help/ident/nonlinear-arx-models.html, 2024. [Accessed 24-08-2024].