EPFL

Thèse n° 10197

Implicit Distance Functions: Learning and Applications in Robotics

Présentée le 21 septembre 2023

Faculté des sciences et techniques de l'ingénieur
Laboratoire d'algorithmes et systèmes d'apprentissage
Programme doctoral en génie électrique

pour l'obtention du grade de Docteur ès Sciences

par

Mikhail KOPTEV

Acceptée sur proposition du jury

Prof. C. N. Jones, président du jury
Prof. A. Billard, directrice de thèse
Dr F. Pokorny, rapporteur
Prof. G. Chalvatzaki, rapporteuse
Prof. A. Alahi, rapporteur

École polytechnique fédérale de Lausanne

2023

The purpose of models is not to fit the data but to sharpen the question.
— Samuel Karlin

To whom it may concern...

Acknowledgements

I wish there was a way to know you're in the good old days
before you've actually left them.
— Andy Bernard

This thesis is a result of four years of my work, and it was made possible by many great people I encountered throughout my academic journey. The support, inspiration, and motivation I received from them were defining in many ways.

First, I sincerely thank my thesis director, Prof. Aude Billard. She provided me with the opportunity to pursue my Ph.D. at the Learning Algorithms and Systems Laboratory (LASA) and offered consistent support throughout the process. Her guidance, advice, and faith in my abilities have been invaluable. Prof. Billard allowed me the freedom to define my research direction while also offering the necessary mentorship. I am deeply grateful for the opportunity to work with her and for the support I received.

Next, I would like to thank the thesis jury — Prof. Colin Jones, Prof. Alexandre Alahi, Prof. Georgia Chalvatzaki, and Prof. Florian Pokorny — for their time and effort in reviewing this manuscript and assessing my work. The feedback I received from them has contributed to enhancing the quality of the thesis.

I want to thank Sergey Trofimov and Danil Ivanov for bootstrapping my academic career when I was just a Bachelor's student in Moscow. They taught me a lot about how to conduct research, how to present my work, and how to write papers. The semester projects I completed under their guidance formed the basis of my skills and led me to change universities to pursue my Master's degree. Without their influence, I may never have envisioned a future in academia, let alone obtaining a Ph.D.

I would also like to thank Muriel Richard for accepting me as an intern at the EPFL Space Center, a decision that enabled me to move to Switzerland and continue my journey at EPFL. Although brief, this internship was an enriching experience, during which I met many wonderful people and forged strong friendships with some. This opportunity significantly contributed to developing the skills required to begin my Ph.D. in LASA.

During my time at LASA, I had a chance to see multiple generations of researchers. When I

first arrived, I was fortunate to have senior colleagues such as Mahdi Khoramshahi, whose wisdom is remarkable, Salman Faraji, who was always very kind, thoughtful, and helpful; and Nadia Figueroa, who became a great source of inspiration for me. Nadia's help and support were invaluable; she became a co-supervisor and a friend. Nadia supported me during all four years of my Ph.D., assisted with structuring my research, and co-authored all my publications. She helped me to seamlessly navigate through all the challenges Ph.D. can offer, and I will always be grateful.

In LASA, I worked alongside many wonderful people: Lukas Huber, David Gonon, Alberic Lajarte, Yang Liu, Enrico Eberhard, Kunpeng Yao, Michael Bombile, Walid Amanhoud, Sthithpragya Gupta, among others. I want to thank them all for the time we spent together, for our countless work-related interactions and collaborations within the lab, as well as for the numerous activities and trips outside the lab. These talented and kind-hearted people made my time at LASA a truly memorable experience.

Throughout my time at LASA, I shared my office with three different individuals. Ilaria Lauzana warmly welcomed me, exhibiting exceptional helpfulness and friendliness. After Ila's departure, Carolina Correia became my office companion, making our time together nothing short of pleasant. We shared numerous common perspectives on life and work, and we always managed to find common ground. Finally, for the past year, Harshit Khurana has been my officemate, and his integrity, determination, and professionalism perfectly complement his warm and friendly nature. He always offered insightful and friendly advice when needed.

I separately thank two special people: Farshad Khadivar and Bernardo Fichera. Farshad is the kindest human being one can only imagine. He devotes himself fully to anything he is a part of, be it work, sports, or friendship. I will always strive to be as humble and kind as Farshad is. Bernardo has become a genuine friend with whom I share many interests. He inspires me to be more serious about sports, to think more about high philosophical matters, and to never stop learning. I am truly grateful for having the chance to meet both of them.

Finally, I must thank my dear Alina for her unconditional support starting from the day we met. You are always there for me, and you are the best person to share happiness with and navigate through hardships alongside. You bring meaning to my life, and nothing is more important to me than you.

Lausanne, June 26, 2023 M. K.

Abstract

In this thesis, we address the complex issue of collision avoidance in the joint space of robots. Avoiding collisions with both the robot's own body parts and obstacles in the environment is a critical constraint in motion planning and is crucial for ensuring the safety and stability of the robot. Unlike traditional operational-space control where tasks are projected to joint space through the Jacobian, we tackle collisions directly in joint space, where the robot can be treated as a material point. This brings new challenges, such as computational complexity of high-dimensional spaces, and the need for fast and accurate collision models. Nevertheless, this approach can lead to more stable and efficient solutions and reduce the potential for numerical instabilities and local minima that can arise in operational-space control.

In the first part of this thesis, we present a real-time self-collision avoidance method for controlling humanoid robots. This is achieved by learning the feasible regions of control in the robot's joint space and representing these regions as smooth self-collision boundary functions. The learned boundary functions are used as constraints in a real-time quadratic program-based inverse kinematic solver to generate collision-free motions. The humanoid robot's high-dimensional joint space poses challenges in learning efficient and accurate boundary functions. To resolve this, we partition the robot model into smaller, lower-dimensional sub-models. We also evaluate different state-of-the-art machine learning techniques for learning the boundary functions. Our approach has been validated on the 29 degree-of-freedom (DoF) iCub humanoid robot, showing highly accurate real-time self-collision avoidance.

In the second part of this thesis, we extend the method to efficiently compute the distance-to-collision between a robot in arbitrary configuration and a point in the robot's workspace. Our approach involves learning a neural implicit signed distance function expressed in joint space coordinates. The differentiable nature of the function enables us to efficiently compute gradients, generating a continuous repulsive vector field in joint space. With GPU parallelization, complex scenes with multiple obstacles can be processed in real-time. This high-resolution collision representation can be used to achieve real-time reactive collision-free control by integrating it as a constraint in an inverse kinematics solver or in a model predictive controller. We demonstrate the effectiveness of our approach through experiments with a 7 DoF robot in a reaching task with dynamic obstacles.

Lastly, we present a method for modulating dynamical systems using a sampling-based model

predictive control. The approach leverages on the properties of previously learned collision detection models, and involves locally perturbing the nominal DS to ensure a meaningful tangential component is always present near obstacles. The algorithm facilitates obstacle avoidance in the high-dimensional joint space of a robot and has been validated in simulations and real-world tests with a 7 DoF robot, successfully navigating a cluttered environment with moving concave obstacles while maintaining local stability of the nominal DS.

In conclusion, this thesis presents a novel approach to learn self-collisions and signed distances in the joint space of robots. The differentiable and parallelizable nature of the learned models allows for efficient integration into sampling-based methods, leading to a new technique for modulating dynamical systems for obstacle avoidance in high-dimensional joint space. Our methods have been validated on a 7 DoF manipulator and a 29 DoF humanoid robot.

Résumé

Dans cette thèse, nous abordons la question complexe de l'évitement de collisions dans l'espace conjoint des robots. Éviter les collisions avec les parties du corps du robot et les obstacles dans l'environnement est une contrainte critique dans la planification du mouvement et est essentiel pour assurer la sécurité et la stabilité du robot. Contrairement au contrôle de l'espace opérationnel traditionnel où les tâches sont projetées dans l'espace conjoint par la jacobienne, nous abordons directement les collisions dans l'espace conjoint, où le robot peut être traité comme un point matériel. Cela apporte de nouveaux défis, tels que la complexité de calcul des espaces de grande dimension et la nécessité de modèles de collision rapides et précis. Cependant, cette approche peut conduire à des solutions plus stables et efficaces et réduire le potentiel d'instabilités numériques et de minima locaux qui peuvent survenir dans le contrôle de l'espace opérationnel.

Dans la première partie de cette thèse, nous présentons une méthode d'évitement de collision en temps réel pour le contrôle de robots humanoïdes. Cela est réalisé en apprenant les régions réalisables de contrôle dans l'espace conjoint du robot et en représentant ces régions sous forme de fonctions de bordure de collision propre. Les fonctions de bordure apprises sont utilisées comme contraintes dans un solveur cinématique inverse à programme quadratique en temps réel pour générer des mouvements sans collision. La grande dimensionnalité de l'espace conjoint du robot humanoïde pose un défi pour l'apprentissage de fonctions de bordure efficaces et précises. Pour résoudre ce problème, nous partitionnons le modèle du robot en sous-modèles plus petits et de plus basse dimensionnalité. Nous évaluons également différentes techniques d'apprentissage automatique de pointe pour l'apprentissage des fonctions de bordure. Notre approche a été validée sur le robot humanoïde iCub à 29 degrés de liberté (DoF), montrant un évitement de collision en temps réel hautement précis.

Dans la seconde partie, nous étendons la méthode pour calculer efficacement la distance de collision entre un robot en configuration arbitraire et un point dans l'espace de travail du robot. Notre approche implique l'apprentissage d'une fonction de distance implicite neuronale exprimée dans les coordonnées de l'espace conjoint. La nature différentiable de la fonction nous permet de calculer efficacement les gradients, générant un champ de vecteurs répulsif continu dans l'espace conjoint. Avec la parallélisation GPU, des scènes complexes avec plusieurs obstacles peuvent être traitées en temps réel. Cette représentation de collision haute résolution peut être utilisée pour atteindre un contrôle sans collision réactif en temps

réel en l'intégrant en tant que contrainte dans un solveur de cinématique inverse ou dans un contrôleur prédictif de modèle. Nous démontrons l'efficacité de notre approche à travers des expériences avec un robot à 7 DoF dans une tâche de portée avec des obstacles dynamiques.

Enfin, nous présentons une méthode pour moduler les systèmes dynamiques en utilisant un contrôle prédictif de modèle basé sur des échantillons. L'approche s'appuie sur les propriétés des modèles de détection de collision précédemment appris et implique une perturbation locale du DS nominal pour garantir qu'une composante tangentielle significative est toujours présente près des obstacles. L'algorithme permet l'évitement d'obstacles dans l'espace conjoint de haute dimension d'un robot et a été validé dans des simulations et des tests en situation réelle avec un robot à 7 DoF, naviguant avec succès dans un environnement encombré avec des obstacles concaves en mouvement tout en maintenant la stabilité locale du DS nominal.

En conclusion, cette thèse présente une nouvelle approche pour apprendre les auto-collisions et les distances signées dans l'espace conjoint des robots. La nature différentiable et parallélisable des modèles appris permet une intégration efficace dans les méthodes basées sur des échantillons, conduisant à une nouvelle technique pour moduler les systèmes dynamiques pour l'évitement d'obstacles dans l'espace conjoint de haute dimension. Nos méthodes ont été validées sur un manipulateur à 7 DoF et un robot humanoïde à 29 DoF.

Contents

List of Figures

List of Tables

1 Introduction

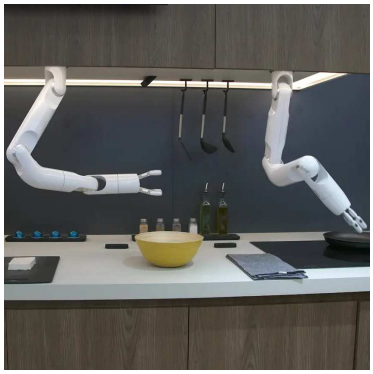
We are stepping through an explosion.
And we are living day to day and it doesn't look like it.
— Andrej Karpathy

1.1 Motivation

We live in a technological singularity. What is one hundred years? From a cosmic perspective, one hundred years is just a blink of an eye. Nevertheless, human civilization has transformed significantly in the past century, constantly increasing the pace of new discoveries and technological developments. As human physical capacities are quite limited, we develop new machinery to sustain the growth tempo: to mine resources at scale, to transport more goods, to build new structures, and to produce more things. Once machinery becomes more than just a masterfully engineered combination of physical processes (think of an internal combustion engine) and requires a computer system to control it (think of a modern car), it may be called a robotic system or, when localized into a single physical frame, a robot.

Robots and robotic systems have become indispensable part of human lives. The singularity-like development that we observe across every domain of human activity is directly connected to the use of robots. One particular example is a robotic arm. Such arms (refer to Figure 1.1) are utilized in countless factories and warehouses and are responsible for assembling, painting, welding, sorting, and packaging, among dozens of other applications. Robotic arms are capable of precise, fast, and consistent repetitions of almost any manufacturing task, driving production costs down and making cars, furniture, and consumer electronics widely accessible.

In industrial environments, robots operate behind safety barriers and are not expected to interact with humans. Precise position- or velocity-controlled robots may cause severe injuries if a human interferes with the robot's motion. However, robots will not always be confined to industrial applications. It is natural to envision a future in which robots constantly share


(a) Tesla Gigafactory Berlin.[1] (b) Samsung's Bot Chef.[2] (c) Franka Emika robot.

Figure 1.1: Robotic arms in (a) industrial, (b) domestic, and (c) research environments.

the environment with humans and interact with them. A robotic arm mounted on a mobile base may be used to assist a human worker in a warehouse or even be helpful in a domestic scenario, such as assisting with cooking and cleaning. In such situations, robot motion must be safe for humans, and, for motions that do not involve contact, **robots must be able to avoid collisions with humans and other objects**.

Environmental collisions are just one of many challenging constraints that should be considered when controlling a robot. In recent years, numerous companies have started developing humanoid robots. These robots are designed to interact with humans and perform tasks in a human-like manner. Control algorithms for such robots must consider balancing, locomotion, and whole-body control, in addition to collision avoidance. Moreover, the robot should avoid colliding with itself, which can occur due to redundancies in the robot's kinematics. Frequently, self-collisions could be harmful to the robot's hardware; therefore, while executing arbitrary tasks, **robots must be able to avoid self-collisions**.

In some cases, avoiding collisions and self-collisions can be seen as trivial task. As soon as a potential collision is detected, the robot can halt the motion and wait for the obstacle to move away. However, in some cases, it is not possible to just stop the motion. A humanoid robot may require first to stabilize itself to avoid losing balance, or the task may be constrained in time and can not be paused to avoid collisions. In other words, **robots must be able to reactively plan a collision-free motion**.

The primary goal of this thesis is to develop collision and self-collision avoidance methods for humanoid robots and robot manipulators. We approach the problem from two perspectives: (i) how to reliably and efficiently detect collisions and self-collisions, and (ii) how to reactively plan and execute collision-free motions.

[1] ©Tesla Inc. https://www.youtube.com/watch?v=7-4yOx1CnXE
[2] ©Samsung U.S. https://www.youtube.com/watch?v=OwA6-b1Z7aQ

1.2 Main concepts and approaches

We established that robots must be able to avoid collisions and self-collisions, as well as swiftly plan a collision-avoding motion in dynamic environments. But how can robot detect a collision? What does it mean to plan a collision-free motion? In this section, we provide an introductory overview of the main concepts and approaches that are used in the literature. For more details on the background and related works, please refer to Chapter 2.

1.2.1 Collision detection

Humans primarily perceive collisions through physical contact. When we touch an object, neural signals are transmitted to the brain, where the information is processed and enables us to detect the contact. We are also capable of anticipating collisions. For example, sitting in front of a monitor, we can almost sense a collision if we imagine extending an arm towards it. This perception is often referred to as *kinesthesia* (or *proprioception*) - the sense of self-movement, force, and body position. Proprioceptors, specialized neurons located in muscles, tendons, and joints, allow us to perceive our body's and limbs' positions. Combined with prior experience and our internal world model, kinesthesia not only allows us to detect collisions as they occur but also enables us to predict and anticipate them.

These two methods of collision detection - direct sensing and model-based anticipation - can be applied to robots as well. One approach involves equipping robots with artificial skin that detects contacts and conveys the relevant information to the controller. However, artificial skin modules can be costly and may add significant wiring to the robot, increasing system complexity. While promising, this approach is not yet mature nor widespread. An alternative method for detecting collisions involves using backdrivable joints in conjunction with force-torque sensors. This technique allows a robotic manipulator to detect a collision even with a lightweight object and promptly halt its motion.

However, proprioceptive sensing may not always be applicable. In addition to hardware limitations, such as the absence of artificial skin or internal sensors for collision detection, there may be tasks that require complete collision avoidance. For instance, a robot involved in painting must not touch the object being painted. In such cases, the robot must rely on sensing devices like cameras or lidars to plan collision-free motion. It may be equipped with these sensors or receive information from an external tracking system. To achieve real-time performance, sensor data must be rapidly processed, and the controller must quickly plan a collision-free trajectory. While this is not a major limitation in static environments, dynamic environments with moving obstacles require motion planning algorithms to be capable of swiftly replanning trajectories as obstacles are changing their positions.

Motion planning algorithms rely on robot models to predict the positions of the robot along the planned trajectory. Additionally, obstacle positions might be predicted if the motion law is known, or using various filtering techniques. Alternatively, obstacles can be considered

quasi-static, but only if the planner can generate collision-free motion at high frequencies. In that case the reactivity of the controller directly depends on it's operating frequency. For example, if the motion plan is generated once every second (or at 1 Hz), it will take one second for the robot to react to a collision. If the plan is generated at 100 Hz, the robot will react in 0.01 seconds. Such reduced reaction time enables collision avoidance for situations where obstacles are moving relatively fast.

To accommodate the requirement for rapid motion generation, numerous approaches exist to evaluate collisions between the robot's body and detected obstacles. The most straightworward method involves representing all collidable bodies as triangulated meshes, defined by a set of vertices and faces. However, this method is computationally demanding, particularly when the number of vertices and faces can surpass hundreds of thousands in relatively complex environments. Consequently, most motion planning algorithms employ simplified collision models.

One significant simplification entails using strictly convex shapes, which allows for the implementation of efficient collision-checking algorithms such as Gilbert–Johnson–Keerthi (GJK) (Gilbert et al., 1988). Additionally, collision bodies can be decomposed into various collision primitives, such as capsules, bounding boxes, or spheres. While this approach is computationally feasible and suitable for real-time use, mesh simplification may be unsuitable in certain situations. For example, a robot arm might possess a complex geometry that the approximate collision model fails to accurately capture. In such instances, the collision model may yield imprecise results, either excessively restricting the motion, or causing collisions that the planner did not detect.

Apart from the simple collision detection, optimization algorithms used in motion planning may rely on collision gradient, which points in the direction repelling the robot from the obstacle. However, this gradient is not generally available for collision detection algorithms, and researchers continuously strive to enhance existing algorithms to compute the collision gradient more effectively.

Recently, with the emergence of learning-based methods, researchers have begun exploring the potential of learning collision models from data. If the robot kinematics is assumed to be known, the dataset containing collided and free postured can be collected, and collision model can be learned. These learning-based approaches inherently provide collision gradients, as the underlying decision functions are smooth and differentiable. Notably, such methods often bypass the Cartesian representation and learn collisions as functions of robot joint configurations, which are more suitable for low-level robot controllers. It is still an open question whether such learned models can outperform traditional methods in terms of accuracy and efficiency, however, they are promising for real-time collision avoidance. *In this thesis, we explore the learning-based collision detection approach, and demonstrate its potential for real-time collision avoidance.*

1.2.2 Motion planning

Most robot motions are predominantly produced by electrical currents powering the motors connected to the robot's links. Although this may not necessarily apply to soft robots, hydraulic-powered joint systems, or other robot types, electric-motor based robots are quite widespread. Regardless of the power source, multiple layers of abstraction exist to enable robot motion control. The first layer involves low-level controllers that receive commands in the form of joint torques, velocities, or positions and convert them into the appropriate power output (e.g., electrical current or hydraulic pressure) for the actuators. The next layer features a controller that takes into account the robot's current state, detected contacts, environmental obstacles, and numerous other potential constraints, generating joint torques (or velocities or positions) in accordance with a high-level task. The final layer of abstraction encompasses the high-level tasks the robot needs to execute, such as moving between points A and B, lifting an object, or interacting with its environment.

Although developing low-level control demands substantial engineering effort, modern robots come equipped with control boxes that accept commands and account for basic constraints like gravity compensation. In certain limited cases, such as uncomplicated tasks in obstacle-free environments, a relatively simple controller may suffice to achieve the desired motion. However, in most situations, the controller must be capable of planning motion that fulfills the high-level task while adhering to all of the constraints mentioned above. This is the task of *motion planning*.

There are various approaches to motion planning problems. A fundamental challenge is that high-level tasks exist in the three-dimensional Euclidean space, while robot motion is executed and ultimately planned in the robot's joint space, with a dimensionality corresponding to the number of degrees of freedom (DoF) the robot possesses. For a robotic arm, there are typically six or seven DoF, while humanoid robots may have up to 30 DoFs.

Consider a 7-DoF robotic manipulator equipped with a gripper. The primary task of such a robot is to interact with objects using its end-effector. To pick up an object, the robot's end-effector must be positioned at the gripping location, often in a specified orientation. Consequently, the task is defined by a 6-DoF pose (three for position and three for orientation) of the end-effector. As the manipulator has 7-DoF, the robot is considered redundant, allowing it to achieve the desired task in various joint configurations. This redundancy is intentional so that the robot can utilize it to satisfy secondary constraints, such as joint limits and collision avoidance.

The prevailing approach to generate robotic motion involves planning the robot motion in the task space and then employing Inverse Kinematics (IK) methods to map the task-space commands into joint-space commands. For high-dimensional systems, analytic IK is often impractical, and IK must be solved numerically. When multiple constraints are considered, the IK problem is usually formulated as a constrained optimization problem and solved using various optimization techniques. Some constraints, like joint limits, are formulated directly in

joint space, while others, such as goal reaching and collision avoidance, are formulated in task space using the analytical forward kinematics (FK) expressions. A limitation of IK-based path-planning methods is the difficulty of considering whole-body collision constraints, as FK is a function of a robot joint configuration and a specific point on a robot's body. In other words, guaranteeing collision avoidance requires numerous FK-based minimal distance constraints, densely covering the entire robot body. *In this thesis, we demonstrate that collision avoidance can be expressed as a single constraint formulated as a function in the joint space.*

Another approach involves planning robot motion directly in the robot's joint space. For instance, one or multiple goal joint positions may be attained using an IK solver, and search algorithms are then used to find a path between the current joint configuration and the goal. However, such planning is computationally demanding, and, depending on sampling granularity, may take seconds or even minutes to plan motion in a high-dimensional joint space. Another group of algorithms for joint space motion planning are formulated as optimization problems, aiming to find an optimal path connecting the current joint position to the goal while satisfying various motion planning constraints. Generally, these algorithms strive to balance computational efficiency with trajectory quality, meaning that methods can be quick to compute but may produce suboptimal trajectories, or they can ensure desired goal-reaching but with slow planning.

If all obstacles are known to have a convex shape, it is possible to generate a motion law that guarantees collision-free goal-reaching. Obstacle convexity allows for simple strategies, such as boundary following, to ensure goal reaching consistently. Such a motion law can be computationally efficient and provide a high-frequency control loop. Although this may be applicable in task space when combined with IK solvers, in high-dimensional joint spaces, even a spherical obstacle near the robot has a non-trivial non-convex shape, rendering this approach infeasible in most cases. *In this thesis, we design a hybrid controller that combines reactive and planning-based approaches to achieve collision-free motion in high-dimensional joint spaces while leveraging the properties of learned collision models.*

1.3 Thesis Objectives and Structure

In the previous section, we introduced challenges related to collision detection and motion planning in robotics. This thesis aims to explore novel approaches for collision detection and reactive collision-free motion generation in the high-dimensional joint spaces of robots. We seek to utilize existing machine learning (ML) frameworks to develop efficient and robust methods for collision detection, and to effectively use these methods in various motion generation techniques, including the proposed novel hybrid controller. Hence, **the main objectives of this thesis are:**

(i) **employing a data-driven learning approach to obtain collision checking models defined in a robot's joint space, and**

(ii) **leveraging the properties of the learned collision models to achieve reactive collision-free motion generation.**

The manuscript consists of six chapters. **Chapter 1** introduces the main challenge of motion planning, while the remaining chapters are organized as follows:

Chapter 2

This chapter outlines the essential background information needed to understand the work presented in this thesis. We provide a detailed discussion on methods for detecting collisions and planning robotic motions. Additionally, we mention state-of-the-art approaches in related fields and corresponding publications.

Chapter 3

Further, we describe our proposed method for learning a self-collision detection model for a humanoid robot. We first formulate self-collision detection as a binary classification problem and then present an approach to learn a collision boundary in the robot's joint space. We examine various ML techniques, such as Support Vector Machines (SVM) and Neural Networks (NN), and compare their performances in the self-collision classification problem. We then utilize the proposed models to formulate a collision avoidance constraint in a Sequential Quadratic Programming (QP)-based IK solver and demonstrate the effectiveness of our approach in a picking scenario with the 32-DoF humanoid robot iCub. This chapter's material is adapted from Koptev et al. (2021).

Chapter 4

We reformulate the collision detection problem as a regression problem and propose learning minimal distances between a robot in a given configuration and points in its workspace. This enables generalizing the approach beyond self-collisions and enables collision detection between the robot and objects in its workspace. We investigate various Neural Network architectures to build the Neural Implicit Distance model and use its properties, such as batch processing, to enhance the performance of the state-of-the-art sampling-based Model Predictive Path Integral (MPPI) algorithm. Moreover, similarly to the previous chapter, we leverage the model gradients to enable the IK solver to reactively avoid collisions with moving objects in the robot's workspace. We verify our approach in experiments with the 7-DoF Franka Emika robotic arm. This chapter is summarized and published in Koptev et al. (2023).

Chapter 5

Building upon the proposed learning-based models, we aim to achieve reactive motion planning in the joint space of a 7-DoF manipulator. We employ a Dynamical System (DS) modulation approach that allows swift collision avoidance in the joint space, assuming a known obstacle-repulsion direction. This direction is provided by the previously introduced Neural

Implicit Distance model. To enhance the algorithm's navigation capabilities, we couple it with a policy generation controller that optimizes for perturbations near concave obstacles, enabling navigation in cluttered environments. This planner is based on the MPPI algorithm, and it efficiently utilizes the parallelization properties of the learned collision models. We evaluate the resulting hybrid controller and compare it to state-of-the-art approaches in a series of experiments with the 7-DoF Franka Emika robotic arm. The materials presented in this chapter are submitted to a robotics journal at the time of writing.

Chapter 6

In the final chapter, we summarize the main contributions of this thesis and discuss potential directions for future development of the proposed methods.

1.4 Thesis Assumptions

It is essential to understand the challenges this thesis aims to tackle and the challenges that are beyond the scope of this manuscript. Throughout this thesis, we consider several assumptions.

Assumption 1: Known Robot Shape

We always assume that we know the exact shape of the robot, including the precise kinematic model and collision meshes. We do not consider that the robot shape may change during the experiment or that the simulated robot differs from its hardware implementation.

Assumption 2: Simple Dynamics

Assumption 1 states that there is no sim-to-real gap for the robot model. While it can be true for kinematics, it rarely holds for models of robot's dynamics. Although various optimal control techniques may focus on maximizing robot performance, we assume that motion trajectories in this thesis are not challenging enough to require complex dynamics modeling. We assume that if a path exists in the robot's joint space, a feasible motion can be executed along this path. We do not investigate trajectories with very high velocity profiles that may be challenging to execute precisely in the real world.

Assumption 3: Known Obstacle Shape

We assume that we know the exact positions of the obstacles in the environment and their shapes. Throughout this thesis, we will approximate obstacles with a set of spheres, equivalent to the case of a point cloud where an obstacle is represented by many spheres of small radii. We do not consider the case of unknown obstacle shapes or unknown obstacle positions. Throughout the thesis, we use the OptiTrack system to track the position of obstacles near

the robot. A more applicable alternative method for real-world cases is to use camera-based algorithms to track obstacle positions. However, this is a challenging task and is beyond the scope of this thesis.

Assumption 4: Quasi-static Obstacles

We focus on situations where obstacles are considered quasi-static, meaning they can change their positions in the environment, but motion planning algorithms do not account for their dynamic properties, such as velocity or acceleration. Although solutions to estimate these dynamics exist, they typically require either detailed information on the object's characteristics for precise modeling or extensive data collection and subsequent dynamics model learning. Instead, we adopt a motion planning approach that rapidly adapts to new obstacle positions at each iteration (100+ Hz), enabling us to address sudden and unpredictable changes in the object's position without the need for a complex model of the object's dynamics or predicting its future position. With this approach, the object's motion speed is effectively limited by the algorithm's update rate, allowing for relatively fast obstacle motion when compared to human reaction speeds[1], while still remaining within the capabilities of the motion planning algorithm's update frequency.

[1] Typically, human reaction speed to a visual stimulus is 200-300 ms, which can be translated to an update rule of approximately 5 Hz.

2 Background

Everyone has a perspective. A *vision*.
— Jensen Huang

2.1 Foreword

In the previous chapter, we briefly discussed the main challenges and approaches in collision detection and motion planning.

This chapter presents the background for collision-free motion planning, along with state of the art works in the field. Section 2.2 covers the problem of collision detection, which is not strictly limited to robotics, but rather emerged from fields of geometry and computer graphics. In Section 2.3, we provide a comprehensive review of robotics-specific path- and motion-planning methods.

2.2 Collision detection

2.2.1 Sensing-based collision detection

As mentioned in the introduction, one way to address collisions in a robotic system is to detect them as they occur, relying on internal sensors. One approach involves covering the robot with tactile sensors imitating skin. This technique was explored by Schmitz et al. (2011), who developed a set of capacitive sensors to cover the body surface of the humanoid robot iCub. Experiments demonstrated that artificial skin enables high-precision contact detection and improves the performance of grasping tasks. An extensive review of tactile sensors for robotics can be found in Dahiya et al. (2013).

An alternative method for detecting collisions involves detecting external forces by means of sensors attached to the robot's joint motors. This technique was investigated by De Luca

et al. (2006), who demonstrated how collisions could be detected by measuring external forces using force/torque sensors within the kinematic chain. Haddadin et al. (2017) further explored this approach, detailing a collision event pipeline that leverages joint sensor information.

While these methods are undoubtedly useful when performing contact interactions between a robot and a human, our focus in this thesis is on collision-free motion planning, which requires predicting and preventing collisions before they occur.

2.2.2 Geometric collision detection

For complex motions, the controller must plan the motion in advance, using internal models of the robot's kinematics and dynamics to predict future positions based on control input. To account for collisions with the environment, obstacles must be modeled and checked for collisions with the robot's body.

The algorithm to detect collisions between multiple bodies primarily depends on the bodies' representations. The most straightforward way to represent a physical body in a computer simulation is a *polygonal mesh* represented by vertices and faces that define the 3D geometry of a body. A large number of different tests and queries, such as closest point computations and various intersection tests, can be performed on bodies represented with meshes. However, this approach is computationally expensive, especially for large meshes. Therefore, collision detection algorithms always consist of a broad phase, which uses overly simplified body representations, such as bounding boxes, to rule out improbable colliding body pairs; and a narrow phase, which uses more accurate representations, such as meshes, to detect actual collisions (Ericson, 2004).

Collision detection algorithms work almost exclusively with convex objects, as precise distance calculation for concave shapes can be obstructed by local minima. The most common way to represent a concave object is to use a convex hull, a convex shape that contains the original object. Collisions between strictly convex shapes can be calculated by efficient algorithms such as GJK (Gilbert et al., 1988) or Voronoi-clip (Mirtich, 1998). Modern collision detection libraries, such as Flexible Collision Library (FCL) (Pan and Manocha, 2016), combine multiple state-of-the-art methods along with various optimizations to enable quick and efficient collision checking in robotic simulations.

2.2.3 Collision gradient

To plan a collision-free motion, the controller must not only be able to detect collisions but also frequently require the collision gradient. The collision gradient is a vector that defines the direction of the fastest distance increment between two bodies. This vector is used in optimization-based motion planning methods to calculate the gradient of the cost function (Ratliff et al., 2009).

Authors of Escande et al. (2007) and Escande et al. (2014) address the problem of continuous gradients for distances between convex hulls. Their proposed collision representation can be used to formulate a minimal distance constraint in the optimization problem, enabling collision-free motion generation for humanoid robots. Zimmermann et al. (2022) propose a framework based on collision primitives and leverage the method's differentiability to formulate collision-avoidance constraints in various optimization-based motion planners.

2.2.4 Implicit collision representation

Another method to define a body geometry is *implicit* representation. In this approach, bodies are not described explicitly with faces and vertices but rather defined implicitly through a mathematical expression. Implicit objects are often described as a function mapping from 3D space to real numbers, $f : \mathbb{R}^3 \rightarrow \mathbb{R}$. An object boundary, frequently defined by the zero-set level $f = 0$, is called an implicit surface. Implicit geometry representation allows for fast collision detection and collision gradient computation (Ericson, 2004). While various primitives can be used for implicit geometry representation, it is not straightforward to contain a complex geometry within a single mathematical expression.

With the advent of GPU computing, universal approximators, such as Neural Networks (NNs), became increasingly accessible at scale. It became possible to encapsulate complex geometries using implicit signed distance representation. Park et al. (2019) present an approach to learn Signed Distance Fields (SDFs) representing arbitrary 3D meshes using Neural Networks. Mildenhall et al. (2020) extended this approach by learning not only the 3D geometry but also the object texture, allowing the generation of novel views of scenes with complicated geometry and appearance using Neural Radiance Fields (NeRFs).

The field of robotics also benefited from implicit geometry representation. Robotic controllers require checking for collisions with obstacles along the predicted robot trajectory and rely on collision gradients to plan collision-free motion. With such applications, implicit collision representation takes the form $f : \mathbb{R}^N \rightarrow \mathbb{R}$, where N is the dimensionality of configuration space (or C-space), that frequently (but not always) coincides with the robot's joint space.

Figueroa et al. (2018) leveraged SVM to model the self-collision boundary of a 14-DoF double manipulator setup and used the model's differentiability to formulate non-penetrability constraints in the IK solver (Salehian et al., 2018a). The sparse model representation allowed for precise control over the number of support vectors, enabling model size tuning to achieve the desired inference speed. Das and Yip (2020) and Zhi et al. (2021) utilized online sampling and active learning to adapt the learned SVM collision model from a static environment to a dynamic one. However, the update step requires adjusting the support vector set, which is a computationally expensive procedure, and real-time performance in dynamic environments could only be achieved for simpler robots with few DoFs.

Rakita et al. (2018) used NN to learn a self-collision cost for a humanoid robot and demon-

strated that such a constraint may be orders of magnitude faster than traditional collision detection methods. While they used a simplified robot geometry representation (line segments) and approximate a cost function instead of precise distances, this work was one of the first to use joint angles as input to a Neural Network to perform collision detection. Kew et al. (2020) learned collisions as a function of the combined robot state and parametrized environment state. They further showed its application in a parallelized rapidly-exploring random tree planner (RRT), proving that such a method can be more effective than traditional collision checking for batched queries. Additionally, the authors leveraged the model's differentiability to adjust path planning. While the authors did not apply the method to achieve real-time collision-free motion generation, they highlighted the inherent benefits of using NNs to check for collisions for batches of inputs.

Danielczuk et al. (2021) presented a trained model that can predict the occupancy grid using scene point-cloud and joint configuration of a robot as inputs. The learned model provides high collision detection rates and good performance in the case of batched queries. However, the model is reported to be unsuitable for real-time control due to slow point-cloud processing.

Chen et al. (2022) investigated the combined approach of learning the robot geometry and collision model using a camera imaging system to collect data for a Neural Network. They demonstrated how the model is capable of online morphology adjustment, by simulating a broken motor, or changing the shape of the end-effector. However, they only demonstrated this on a 4-DoF robot and did not study online motion generation, apart from simple gradient-following goal reaching.

Liu et al. (2022) suggested a neural implicit model to learn smooth distances at various scales. They demonstrated the framework's applicability to a dataset of 24-DoF human postures, learning the signed distance field as a function of human posture. This distance field was then used to model a collision avoidance constraint for reactive collision-avoiding robot motion generation.

In this thesis, we investigate the use of implicit collision representation for motion planning. Our contributions build upon the work by Fernandez (2018) and involve further development of the approach, including the introduction of a humanoid robot application and collision detection involving external obstacles. Although some recent works in the literature share similarities with our research, such as utilizing ML-based collision checking, it is essential to note that these publications emerged concurrently with the development of this thesis. Consequently, the overlaps can be attributed to the independent evolution of ideas within the research community.

Our primary contribution lies in developing a self-contained framework for both self-collision and external collision detection, and subsequently leveraging the strengths of learned distance functions to create a motion planning method. While acknowledging the overlaps with other works, this thesis provides its unique perspective on implicit collision representation and real-time motion planning, which have been validated through peer-reviewed publications.

2.3 Motion Planning

Motion planning has been a popular research area for nearly half a century, with a myriad of methods and algorithms proposed to address various aspects of the problem. Initially, research focused on path planning, which aims to find a trajectory connecting two points in the configuration space. Over time, the scope of planning has broadened to encompass more complex problems such as motion planning, which seeks a collision-free trajectory that satisfies constraints such as model kinematics and dynamics, model uncertainties, and multiple bodies (LaValle, 2006). This section offers an overview of prevalent motion planning methods used in robotics.

2.3.1 Offline Planning Methods

A widely adopted paradigm in robotic systems is the hierarchical approach, where a feasible path is first generated offline and then executed by a high-frequency controller. Consequently, a significant amount of academic effort has been devoted to developing various offline trajectory generation methods.

Resolution-complete methods, such as Dijkstra or A* algorithms, operate on a discretization of possible robot configurations. If the discretization is sufficiently fine-grained, a path between the start and goal configurations will be found within a finite time. These algorithms employ a greedy search assisted by various heuristic functions. However, resolution-complete methods are limited to low-dimensional C-spaces, as the number of points in the discretization grid increases exponentially with the dimensionality of the configuration space. Thus, discretizing high-dimensional C-spaces is computationally expensive (Hart et al., 1968). This phenomenon is known as the "curse of dimensionality."

Sampling-based path planners form a large family of algorithms that rely on sampling the robot's C-space and utilizing the explored states to find a path between the start and goal configurations. Early methods include the Probabilistic Roadmap Method (PRM) (Amato and Wu, 1996; Kavraki et al., 1996), which incrementally constructs a feasible region of C-space represented as a graph, where paths exist for neighboring nodes. Nevertheless, PRM can be inefficient in cases where it is infeasible to explore the entire state-space (e.g., high-dimensional joint space of a redundant robot). Rapidly-exploring Random Tree (RRT) (LaValle, 1998; Kuffner and LaValle, 2000) is a sampling-based method that constructs a tree of feasible states, with each node connected to its nearest neighbor. The tree is expanded by sampling a random state and connecting it to the nearest node. RRT can be considered as more efficient than PRM, because it does not require exploring the entire C-space. Both PRM and RRT are *probabilistically complete* methods, meaning that if a path connecting the initial and goal states exists, it will be found within a finite time. However, the number of nodes in the graph required to discover a path increases with the space dimensionality and additionaly grows larger when using smaller discretization steps. As a result, the computation time can vary from seconds to hours, depending on the complexity of the C-space.

Over the years, numerous modifications for PRM and RRT have been proposed, addressing various shortcomings, improving path quality (Karaman et al., 2011), and enhancing method performance. Some consider the robot's dynamics (Webb and van den Berg, 2013), while others aim to increase the method's performance to enable rapid plan generation (Kingston et al., 2019). For a comprehensive overview of sampling-based methods, we refer the reader to Kingston et al. (2018).

2.3.2 Optimization-based Motion Planning

The increasing computational power and development of new algorithms have led to the growing popularity of *optimization-based motion planning*. These methods minimize a state-dependent cost function, which encapsulates task-dependent constraints.

Ratliff et al. (2009) introduced Covariant Hamiltonian Optimization for Motion Planning (*CHOMP*), a trajectory optimization method that employs functional gradient techniques to iteratively refine an initial trajectory. This initial trajectory can be warm-started using another offline planner or chosen naively, possibly containing collisions. The optimized cost function takes into account trajectory smoothness and collision avoidance. Due to the non-convex constraints, gradient descent techniques may result in local minima. This issue is addressed by restarting the optimization with a perturbed initial trajectory. CHOMP has proven useful for generating smooth trajectories in robotic grasping and quadruped locomotion tasks (Zucker et al., 2013). However, the typical optimization time is on the order of seconds, making the method unsuitable for complex scenes with dynamic obstacles. Notably, the collision avoidance constraint relies on a precomputed distance field, which is not straightforward to adjust for new obstacle positions.

Kalakrishnan et al. (2011) proposed Stochastic Trajectory Optimization for Motion Planning (*STOMP*), which uses a derivative-free stochastic trajectory optimization method. The initial trajectory, which may be unfeasible, is perturbed stochastically, and the cost function for candidate trajectories is evaluated. This cost function considers obstacle avoidance, control input, and various task-related constraints. The planning time is comparable to that of CHOMP and is sub-second for 7-DoF robotic arm navigation in environments with multiple obstacles.

Schulman et al. (2014) presented *TrajOpt*, a trajectory optimization method that employs sequential convex optimization to find collision-free trajectories. Unlike STOMP and CHOMP, which treat the trajectory optimization problem as non-convex, TrajOpt locally approximates non-convex collision constraints using sequential convex optimization. Similar to the other methods, planning time heavily depends on the quality of the initial trajectory guess and may require several seconds for complex scenes.

Overall, trajectory optimization methods show promise, as computational advances will eventually enable faster trajectory optimization, facilitating reactive replanning in response to environmental changes.

2.3.3 Online Optimization Methods

The methods discussed above do not consider the possibility of state changes during motion execution. Although this assumption is reasonable for computer graphics and simulated environments, real-world applications may involve perturbations in the robot's state or changes in obstacle positions. Thus, a plan should depend on information gathered during execution.

In Section 2.3.2, we described state-of-the-art optimization techniques. Optimization-based motion planners can be classified as *online* or *offline*, depending on running frequency and considered constraints. This section focuses on methods that can be evaluated frequently enough (typically above 20 Hz) to incorporate changes in the environment and robot state, integrating these changes as feedback into the optimization process.

One of the earliest works approaching the robot navigation problem as an optimization problem is Quinlan and Khatib (1993). The authors proposed an algorithm to adjust previously generated trajectories (e.g., by offline planners) to account for new obstacle positions. Brock and Khatib (2002) further developed the method and demonstrated its applicability to complex experiments and high-dimensional robots. The algorithm is expressed in closed form, allowing evaluation at frequencies exceeding 1kHz, meaning that changes in obstacle configurations can be reactively reflected in the planned path. To this day, this approach is used in various applications, including collision avoidance in human-robot collaborative scenarios (Kot et al., 2022). Among the method's limitations is its locality, necessitating the invocation of a global planner to generate a new trajectory when the robot is far from the initially planned path.

One way to make optimization-based motion planning more applicable for real-time feedback is to generate plans not for the entire start-to-goal motion but for a finite (or receding) time horizon. System tasks and dynamics are posed as constraints in a nonlinear optimization problem, and feasible trajectories are generated dynamically. At each time step, the best motion over the limited horizon is determined; the first action is executed, the horizon is shifted one step forward, and the problem is solved again. Imperfect robot control resulting in imprecise trajectory following can be integrated as feedback into this optimization scheme, along with varying obstacle positions. This approach is known as *Model Predictive Control* (MPC).

The navigational capacities of MPC methods strongly depend on the horizon length, as algorithms with short look-ahead may struggle to escape local minima. However, such methods have a wide range of applications, including autonomous driving (Frasch et al., 2013) and humanoid robot control (Koenemann et al., 2015). Nevertheless, MPC applied to high-degree-of-freedom robots has shown that in real-time scenarios, the planning horizon cannot be made long enough to perform complex motions (Erez et al., 2013).

Recent advancements in computational hardware, such as GPU chips that enable massive parallelization, have given rise to a new generation of optimization methods that rely on sampling-based exploration to approximate the optimal input distribution (Williams et al.,

2017). These methods can handle complex dynamics and non-differentiable cost functions. For example, the authors of Bhardwaj et al. (2022) utilized the MPPI approach and demonstrated online trajectory optimization via sampling-based model predictive control operating at 125 Hz on a 7-DoF robotic manipulator. This method's optimization routine is similar to STOMP, but the receding horizon and GPU-parallelized treatment of samples enable high-frequency problem solving. This method can be considered *reactive* because it is capable of swiftly moving the robot away from collisions if an obstacle approaches the manipulator. Simultaneously, due to trajectory planning with a look-ahead horizon, this method can navigate relatively complex environments. However, the method may still require specific sampling heuristics, an increased number of samples, or an extended horizon to find paths in cluttered environments (Sacks and Boots, 2022; Yoon et al., 2022).

2.3.4 Planning with Feedback

Another approach to achieving a feedback plan is defining a state-dependent potential function (LaValle, 2006). An early work in this area is Khatib (1986), which introduced the *artificial potential method*. The goal state is modeled as an attractive field, while obstacles have a repulsive potential, acting similar to magnets. The motion plan involves following the global potential function's gradient, expressed as a weighted sum of local potentials. Despite its mathematical elegance and adaptability to changes in robot state and obstacle positions, this method has the significant drawback of conflicting objectives resulting in local minima, leading to stuck or oscillatory behaviors.

Rimon and Koditschek (1988) addressed the issue of local minima by defining a global *navigation function*, that, by definition, is free from local minima. To construct such a function, an exact implicit obstacles representation is required. Consequently, these functions are challenging to generalize for arbitrary obstacle shapes and multidimensional C-spaces. This concept further evolved into *harmonic potential functions*, which were inspired by physical processes such as heat transfer or fluid flow (Connolly and Grupen, 1993; Feder and Slotine, 1997). This method ensures global convergence to the attractor but is inherently two-dimensional and can be impractical for high-DoF robotic systems, as solving a partial differential equation is required. Additionally, such methods can only provide reactiveness with respect to robot state perturbation, and any changes in the environment will require recomputing the navigation functions, rendering these methods unapliccable in dynamic environments for high-DoF robots.

2.3.5 Dynamical Systems for Motion Generation

Methods described in Section 2.3.4 use navigation function to define a state-dependent motion law for the system. Such method of generating robotic motion is commonly reffered as Dynamical Systems (DS) approach. A Dynamical System is defined by a vector field that defines the system's state velocity (first-order DS) or acceleration (second-order DS). The DS

defines motion laws as closed-form functions, which typically depend on the state, inherently providing built-in feedback for situations when the robot deviates from the planned trajectory.

Generally, DSs are formulated as a set of autonomous (or non-autonomous) differential equations, enabling the approximation of a wide range of movement primitives for robotic systems. Khansari-Zadeh and Billard (2011) proposed a method to approximate recorded trajectories with a set of Gaussians, providing a globally stable autonomous DS that guarantees convergence to a fixed attractor. This approach allows for repeating reaching motions while providing stability guarantees and adapting motion plans for sudden changes in robot positions. Figueroa and Billard (2017) further utilized this Learning from Demonstrations (LfD) approach to encapsulate complex multistage tasks as a sequence of DSs. Shukla and Billard (2012) used Support Vector Regression (SVR) to formulate a multi-attractor dynamical system. Depending on the initial state of the system, the generated motion is locally asymptotically stable to the attractors within a finite region of attraction.

Dynamical Systems provide a motion law that can be computed at a very high rate (above 1 kHz) and adjust the command using the current robot state, thus enabling the accomplishment of highly dynamic tasks for robots. Khansari-Zadeh et al. (2012) used DS to estimate a hitting model in a minigolf task for a 7-DoF robotic arm, and Kim et al. (2014) demonstrated the use of DS to define object motion in the challenging problem of catching an object in-flight. Salehian et al. (2016) further extended this method to allow for soft catching while using DS control law to smoothly intercept an object in flight. Many other applications of DS for various robotic tasks are explored by Billard et al. (2022).

However, state-dependent DS formulations are not suitable for motion planning, as they do not take environmental constraints into account when computing system velocity. As previously mentioned, early approaches that rely on navigation functions to provide collision-free motion are limited to simpler configuration spaces with analytically defined obstacles or static scenarios. To address this issue, Khansari-Zadeh and Billard (2012a) proposed a method to *modulate* a DS to enable collision avoidance for generated motions. *Modulated dynamical systems* use a nominal DS and alter it with a modulation matrix that encapsulates collision-avoidance behavior and depends on the *implicit obstacle representation*.

Modulated DS can be used to provide a collision-free motion plan for a robotic system. This approach is similar in spirit to the navigation function approach, described previously, as it provides a globally stable motion plan. As the modulation is calculated in closed form[1], the plan can be evaluated at frequencies exceeding 1 kHz, readily incorporating quasi-static environments where obstacles can change positions during motion execution. The authors extended this work (Khansari-Zadeh and Billard, 2012b) for scenarios with fast-moving obstacles,

[1] In modulated dynamical systems, the primary computational complexity, aside from nominal dynamics, stems from calculating the modulation matrix, which is dependent on the distance function. When the distance function is represented by a complex model, such as a neural network, the computation remains closed form but may become non-negligible in terms of processing time and resources. This issue is further investigated in Chapter 5 of this thesis.

where the quasi-static assumption is no longer required. However, in both quasi-static and dynamic cases, the asymptotic stability and successful attractor reaching require obstacles to have strictly convex shapes, significantly limiting the method's applicability.

Huber et al. (2019) and Huber et al. (2022) further improved the Modulated DS approach by changing the structure of the modulation matrix. This allowed for overcoming strict convexity limitations and enabling avoidance of star-shaped obstacles. The method relies on selecting a reference point inside the obstacle, which could be problematic for obstacles formulated implicitly or in high-dimensional C-spaces. Additionally, this method only considers linear nominal DS, and requires local linearization to handle nonlinear dynemaics.

In this thesis, we utilize the modulation framework to ensure collision-free motion generation for high-dimensional robotic systems. We address the limitations of the DS modulation method, such as constraints on obstacle shape and the necessity to select a reference point inside the obstacle, by combining the low-level Modulated DS with a high-level Model-Predictive Planner. The latter is responsible for providing exit strategies from local minimas encountered by the modulated DS. Moreover, we leverage learned implicit distance representations to enable the application of the method in high-dimensional configuration spaces. We showcase the application of the method in various simulated and real-world scenarios, including a 7-DoF robotic arm operating in close proximity to a human who obstructs the robot's motion by creating concave arm traps.

3 Self-Collision Avoidance for Humanoid Robots

Knowing yourself is the beginning of all wisdom.
— Aristotle

3.1 Foreword

This thesis aims to address collision avoidance in various robotic systems and establish a reliable method for generating collision-free motion. In this chapter, we begin tackling this challenge by focusing on self-collision avoidance for redundant robotic systems. First, we present a framework that employs a data-driven approach to learn a self-collision boundary for a robot. Then, we demonstrate how to use the learned function for reactive collision-free motion generation.

The work presented in this chapter has been published in *M. Koptev, N. Figueroa and A. Billard, "Real-Time Self-Collision Avoidance in Joint Space for Humanoid Robots," in IEEE Robotics and Automation Letters, vol. 6, no. 2, pp. 1240-1247, April 2021, doi: 10.1109/LRA.2021.3057024.*

This work was also presented at the *IEEE International Conference on Robotics and Automation (ICRA) 2021*.

3.2 Introduction

A prime example of a highly redundant robotic system is a humanoid robot. Humanoid robots have a wide span of possible applications, potentially becoming universal assistants. In their morphology, such robots resemble humans, meaning that they also posess two arms, two legs, a torso and a head, and each limb can collide with other parts of the robot. Though they are difficult to control due to balancing, stepping, and whole-body control considerations (Faraji et al., 2014; Nava et al., 2017; Figueroa et al., 2020), they have many advantages. They do not require flat floors to operate, are capable of bi-manual dexterous manipulation, and

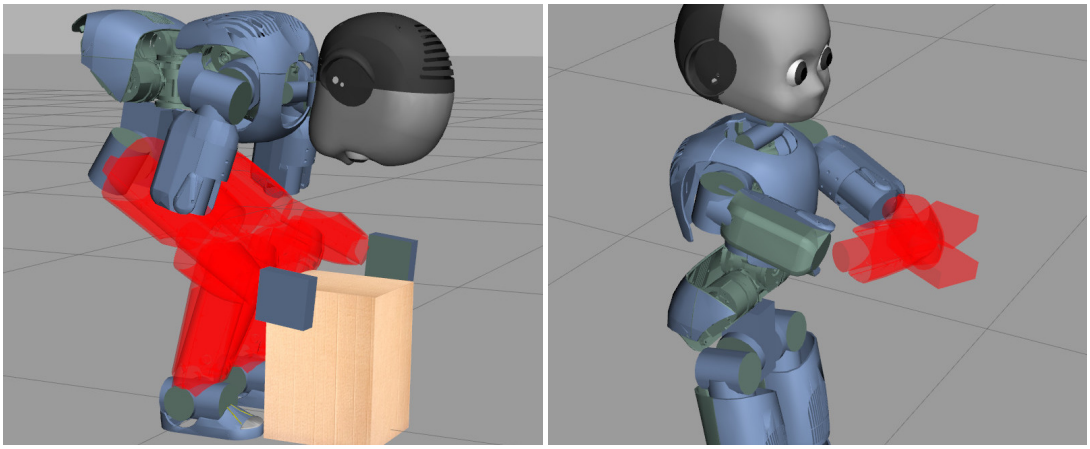


Figure 3.1: Self-collided postures example in simulation for humanoid robot iCub. Links in the collided state are highlighted red. On the left – arms collide with legs in the crouching scenario, on the right – self-collisions between the arms.

can extend their manipulation workspace by bending forward or crouching. Achieving such behaviors is challenging, as the control schemes must additionally prevent collisions with obstacles in the environment and between robot's own body parts. Figure 3.1 shows possible scenarios where self-collsions may occur. Robots with back-drivable joints (Ajoudani et al., 2014) or equipped with soft skin (Guadarrama-Olvera et al., 2019), may allow self-collisions or exploit self-contacts and compliance to achieve a goal. However, for robots not equipped with such capabilities, self-collision prevention is necessary to avoid hardware damage.

We focus on providing a real-time solution to the self-collision avoidance (SCA) problem for high-dimensional position-controlled humanoid robots. In the literature, the SCA problem is often tackled with *offline motion planning* algorithms that generate optimal collision-free paths in configuration space for robots in challenging and constrained environments (Kavraki et al., 1996; Kuffner and LaValle, 2000; Luna et al., 2020; Ratliff et al., 2009; Schulman et al., 2014). Yet, as the robot's joint space dimensionality increases, so does the computational and geometrical complexity of the problem. While these specific methods have been capable of generating collision-free trajectories for high-DoF humanoid robots that can be dynamically-stable, efficient and optimal, their main shortcoming is computation time. It may take seconds to minutes to generate a single feasible trajectory. Hence, their applicability to reactive and dynamic real-time control is limited.

To alleviate this, *online reactive approaches* for high-dimensional humanoid robots have been explored. These approaches rely on distance computations between simplified geometrical representations of segments of the robot's body; i.e. spheres, swept-sphere volumes, capsules, convex-hulls, patch-based bounding volumes, etc. Self-collisions are avoided by (a) using the distance functions as "repulsive potential" functions transformed to joint velocities for Jacobian-based inverse kinematics (IK) (Sugiura et al., 2006, 2007; Schwienbacher et al., 2011) and to joint torques for torque control schemes (De Santis et al., 2007; Dietrich et al., 2011); or

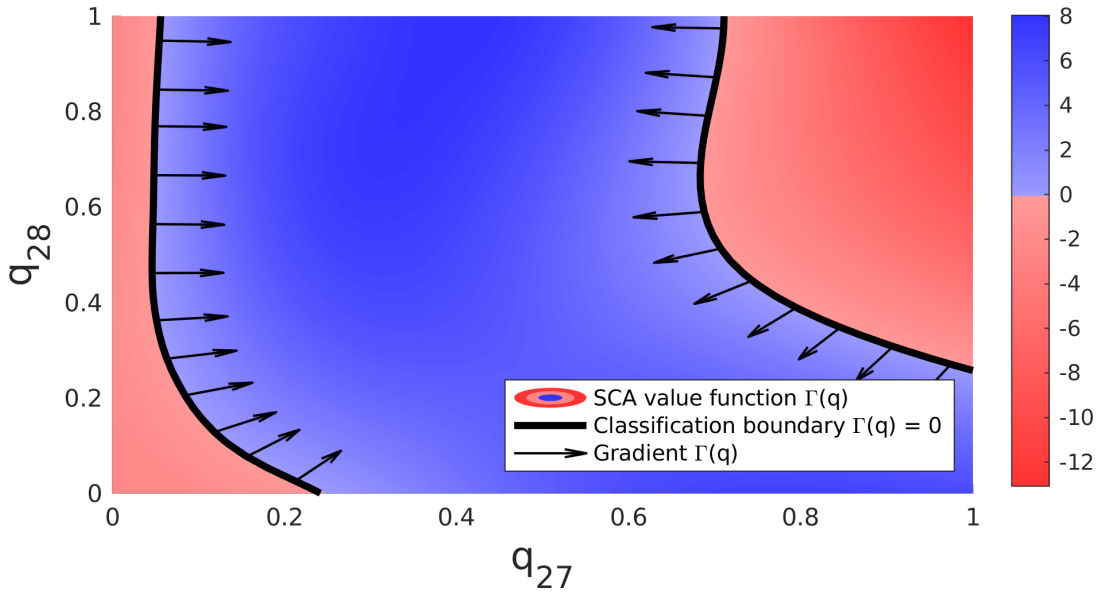


Figure 3.2: SCA value function $\Gamma(\boldsymbol{q})$ projected to two selected dimensions. Red areas denote negative class ($\Gamma(\boldsymbol{q}) < 0$) and collided configurations, whereas blue area represent feasible configurations ($\Gamma(\boldsymbol{q}) > 0$). Arrows inside collision-free region stand for gradient $\nabla\Gamma(\boldsymbol{q})$ that is pointing in the direction away from collisions.

(b) as constraints in optimization-based IK solvers (Stasse et al., 2008; Escande et al., 2014; Quiroz-Omaña and Adorno, 2019).

Although providing better computational efficiency than the *offline* methods, this family of approaches have several issues. First, the accuracy and computation time of collision avoidance relies heavily on the precision of the geometric representation. A highly accurate representation of the segments will yield highly accurate distance functions but are more computationally taxing; and vice-versa. Further, the former approaches are susceptible to numerical issues induced by Jacobian inversions and are not robust to local minima. This is handled by the latter approaches in which inequality constraints are defined as the distances between closest points on the robot body. Yet, in order for the optimizations to converge, the distance functions should be continuously differentiable (Stasse et al., 2008). This can be difficult to achieve. A distance function between two geometrical objects is continuously differentiable only if one object is convex and the other strictly convex (Escande et al., 2014). To guarantee convergence, Stasse et al. (2008); Escande et al. (2014) and Quiroz-Omaña and Adorno (2019) have proposed various novel geometrical representations and distance functions that ensure differentiability. However, they rely on convex geometrical representations and approximations which can over-constrain the problem; i.e. SCA might be too conservative.

Insights from previous work on SCA for multiple fixed based robot arms (Salehian et al., 2018b; Figueroa et al., 2018) show that self-collision regions of a multi-body robot are static and unique in joint space. Hence, a continuously differentiable function can be learned to model

the self-collision boundary between collided and free joint configurations (see Figure 3.2). Salehian et al. (2018b); Figueroa et al. (2018) use a sparse support vector machine (SVM) formulation for fast (~ 2ms) and accurate (~98%) collision detection and repulsive gradient computation. The SCA boundary functions and gradients were re-formulated as inequality constraints in a quadratic program (QP)-based optimization problem that is solved in real-time. This approach was validated on a 14-DoF dual-arm setup, however, it's scalability to higher-dimensional humanoid robots was not explored. Following this work, we propose to learn continuously differentiable boundary functions that can be approximated in real-time to prevent self-collisions while controlling the 29-DoF iCub humanoid. We show that the learned SCA models can be used to actively avoid self-collisions within a *real-time* IK solver (< 5ms computation time).

Self-collision boundary functions can be learned by sampling the robot's workspace for collided and non-collided joint space configurations. With a sufficiently representative dataset, several function approximation approaches could learn the boundary functions with high prediction accuracy. Since the dataset is collected offline, we can use precise triangle mesh representation of the iCub robot body to acquire a highly accurate sampling of the collision boundary. The challenge we face is that such datasets are in the order of millions of datapoints. Further, we seek to provide high SCA prediction accuracy and computationally efficiency while ensuring continuity requirements. We offer the former by decomposing the robot's 29-dimensional joint space into independent submodels of lower dimensionality, each covering a sub-combination of limbs and bodies. To learn the boundary functions, we explore and compare the sparse SVM used by Salehian et al. (2018b); Figueroa et al. (2018) known as Cutting-Plane Subspace Pursuit (CPSP) algorithm (Joachims and Yu, 2009b), GPU-accelerated feed-forward neural networks (NN) and GPU-accelerated SVM implementations.

Similar approach, presented by Fang et al. (2015), uses a SVM to detect self-collisions of the 31-DoF WALKMAN humanoid robot. There, the SVM is used as a prediction tool to accelerate the online computation of the pairs of closest points. Given these predictions, the actual Euclidean distances are used to formulate constraints fed to a task-prioritized optimization problem as in (Stasse et al., 2008). While this approach is computationally efficient ($10-20$ms), the reported misclassification rates of the learned SVMs are high and require an additional heuristic to improve prediction accuracy. Further, this approach may suffer from convergence issues due to the nondifferentiability of the constraints.

The idea of learning a model of a robot's workspace with SVM and using it to detect collision is also explored in (Das and Yip, 2020). An active learning scheme and kernel approximation are leveraged to efficiently detect collisions. They accelerate offline motion planning approaches such as rapidly exploring random trees (RRT) (LaValle, 2006) variants to orders of magnitude above the state-of-the-art. This work clearly demonstrates the computational advantage of using a learned collision function over classical pairwise distance searches. However, its scalability to high-dimensional humanoid robots and applicability to real-time IK solvers is yet to be demonstrated.

3.2.1 Chapter organization

Sections 3.3-3.4 describe the proposed dataset building technique and the analysis of boundary function approximation approaches. Section 3.5 describes the proposed IK optimization problem with the learned boundary functions as constraints. In Section 3.6,we show that our approach yields highly accurate real-time SCA for the 29-DoF iCub (Metta et al., 2010).

3.3 Problem Formulation

We seek to learn an SCA model for the iCub humanoid robot (Metta et al., 2010) in joint space to predict and prevent self-collisions. Our motivation is to avoid repetitive geometrical sampling and minimal distances computations by learning a continuous and continuously differentiable ($\mathcal{C}^1$) function $\Gamma(\boldsymbol{q}) : \mathbb{R}^n \rightarrow \mathbb{R}$ from a dataset of sampled postures $\boldsymbol{q} \in \mathbb{R}^n$. This function should represent the cost for a self-collision constraint, describing collided and free robot configurations. While $\Gamma(\boldsymbol{q}) \in \mathbb{R}$ can help determine how close a given posture is to a collided state, the gradient of this function, $\nabla\Gamma(\boldsymbol{q}) \in \mathbb{R}^n$, can be used to navigate existing path-planning methods away from the collision border, see Figure 3.2.

To learn $\Gamma(.)$, any machine learning function approximation technique could in principle be used. We set to compare three techniques: SVM, sparse SVM (CPSP), NN. In the case of SVM, the number of parameters (support vectors) grows linearly with the number of datapoints (Schölkopf and Smola, 2002). Hence, we additionally evaluate the sparse SVM version (CPSP) (Joachims and Yu, 2009a). We then contrast this to Neural Networks with a focus on finding the minimum number of neurons to achieve similar accuracy. We show that the resulting constraints from any of these techniques are continuously differentiable.

3.4 Self-Collision Boundary Learning

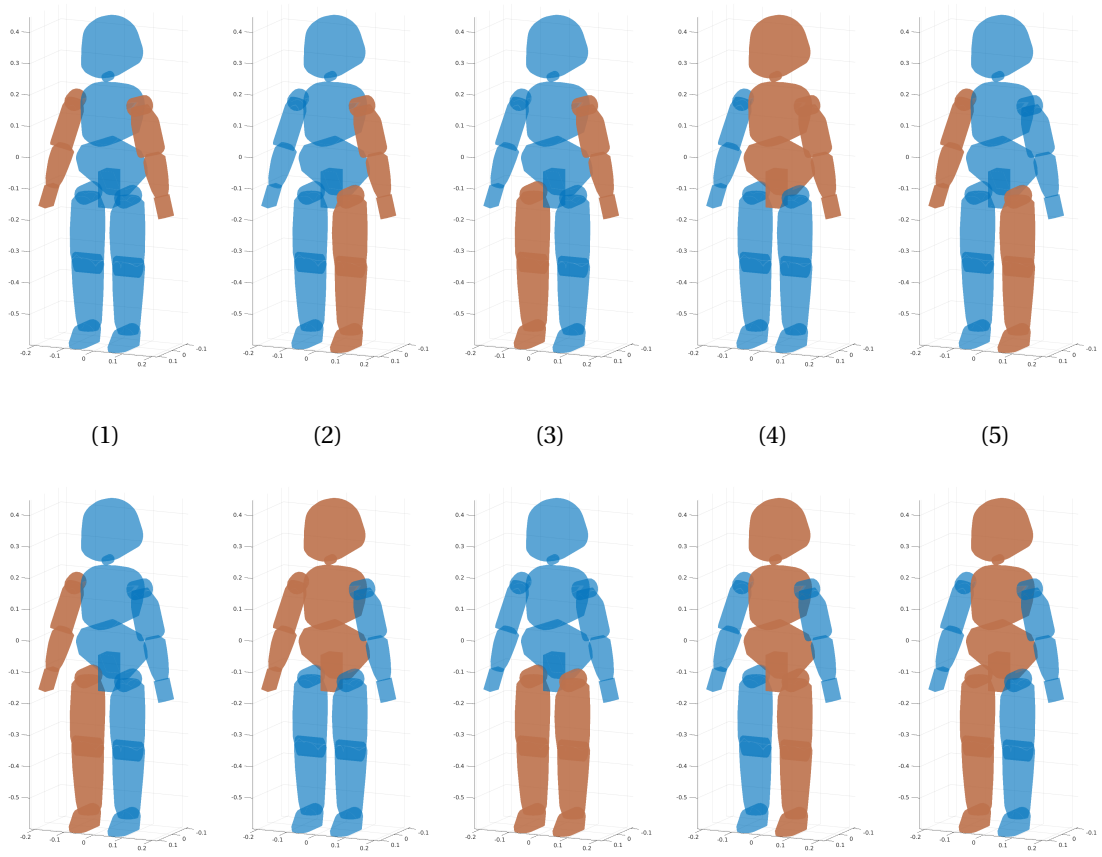
This section describes our approach applied to the iCub humanoid robot, yet it is applicable to any humanoid.

3.4.1 Self-Collision Dataset

To learn a self-collision detection function, we generate a dataset containing examples of both collided and collision-free classes. The iCub has 53 actuated joints (Metta et al., 2010). Excluding hands, eyes and the three joints associated with neck and head movements[1] reduces the number of actuated joints to 29. The resulting joint space of the iCub is equivalent to $\mathbb{R}^{29}$, as each joint q^i is revolute and has corresponding joint limits:

$$q_i^- < q_i < q_i^+, i = 1, ..., 29.$$

[1] The collision space with the head moving is quasi equivalent to the collision space with the head static.



(1) (2) (3) (4) (5)

(6) (7) (8) (9) (10)

Figure 3.3: Visualization of humanoid self-collision model separation into ten independent submodels. For each submodel 1-10 orange links are checked for collisions, while blue links' collisions are ignored. Neighbouring links collisions are also ignored. The features of each submodel correspond to the joints adjacent to highlighted links. For models 2, 3, 5 and 6 torso joints are also considered as features.

As noted by Fang et al. (2015) and evidenced in our experiments, due to the complex joint space geometry, learning a single SCA model for a high-dimensional humanoid robot that is *both highly accurate* and *computationally efficient* is challenging, because 29-dimensional joint space has a structure of collided and free regions too complex to be described with good accuracy. For that reason, we simplify the problem by decomposing the 29-DoF humanoid model into ten submodels of lower dimensionality, that independently describe possible collisions between robot links. The links, joints and feature dimensionality for every submodel are specified in Table 3.1 and visualized in Figure 3.3. Notably, certain pairs of submodels become symmetric if the torso joints are mirrored. By exploiting this symmetry, we reduce the number of models *to be learned* from ten to six. We indicate the symmetric submodel pairs in Table 3.1.

Table 3.1: Humanoid robot model separation (1-10). Entries marked with (^{*x}) are symmetrical to unmarked models x. Full-body is present as zero entry for comparison. Visualisation is presented in Figure 3.3.

N	Collided Links	Varied Joints	Features
0	Full body	Full body	29
1	Left arm, right arm	Left arm, right arm	14
2	Left arm, left leg	Left arm, left leg, torso	16
3	Left arm, right leg	Left arm, right leg, torso	16
4	Left arm, torso, head	Left arm, torso	10
5^{*3}	Right arm, left leg	Right arm, left leg, torso	16
6^{*2}	Right arm, right leg	Right arm, right leg, torso	16
7^{*4}	Right arm, torso, head	Right arm, torso	10
8	Left leg, right leg	Left leg, right leg	12
9	Left leg, torso, head	Left leg, torso	9
10^{*9}	Right leg, torso, head	Right leg, torso	9

While many works rely on geometrical simplifications of robot structure, we use precise triangle meshes. To detect collisions we use the Flexible Collision Library (FCL) (Pan and Manocha, 2016). The dataset is generated by a uniform random sampling with balancing heuristics. The dataset creation is structured so that 50% of samples are collided postures, 35% are feasible postures with a minimal distance across link pairs lower than a threshold (to have a better representation of collision boundary), and 15% of entries are uncollided postures without minimal distance requirements. For every submodel, we randomly sample robot configurations within the joint limits until the desired dataset size (for collided, within the threshold, and free postures) is reached.

The threshold value for 35% of close-to-collision configurations is set to 5 cm. Postures are considered collided if the minimal distance between any link pair is below 1 cm. Using joint limits q_i^-, q_i^+ all configurations are normalized, i.e. every feature belongs to $(0,1)$. As a result, for each submodel, we generate a *normalized* and *balanced* dataset, where 50% of samples are of collided configuration, and 50% are uncollided configurations including safe and close-to-collision postures. This procedure allows us to generate unlimited amounts of data, and datasets sizes are only constrained by the computation time.

3.4.2 Self-Collision Boundary Learning via SVM

To learn a self-collision boundary from the collected data, we first follow the classic SVM formulation, where the kernel trick is employed to lift the data to higher dimension space and find separation hyperplane in feature space. We use the radial basis function (RBF) kernel $K(\boldsymbol{q}_1, \boldsymbol{q}_2) = e^{-\gamma||\boldsymbol{q}_1 - \boldsymbol{q}_2||^2}$, where parameter γ defines kernel width. For a given robot

configuration $\boldsymbol{q}$, the SVM decision function $\Gamma(\boldsymbol{q})$ has the following form:

$$\begin{aligned}\Gamma(\boldsymbol{q}) &= \sum_{i=1}^{N_{sv}} \alpha_i y_i K(\boldsymbol{q}, \boldsymbol{q}_i) + b \\ &= \sum_{i=1}^{N_{sv}} \alpha_i y_i e^{-\gamma||\boldsymbol{q}-\boldsymbol{q}_i||^2} + b,\end{aligned} \tag{3.1}$$

and the equation for $\nabla\Gamma$ is naturally derived as follows:

$$\begin{aligned}\nabla\Gamma(\boldsymbol{q}) &= \sum_{i=1}^{N_{sv}} \alpha_i y_i \frac{\partial K(\boldsymbol{q}, \boldsymbol{q}_i)}{\partial \boldsymbol{q}} = \\ &= -2\gamma \sum_{i=1}^{N_{sv}} \alpha_i y_i e^{-\gamma||\boldsymbol{q}-\boldsymbol{q}_i||^2} (\boldsymbol{q} - \boldsymbol{q}_i).\end{aligned} \tag{3.2}$$

In (3.1), $\boldsymbol{q}_i$ ($i = 1, ..., N_{sv}$) are the support vectors from the training dataset, y_i are corresponding collision labels (-1 if posture is collided, $+1$ otherwise), $0 \leq \alpha_i \leq C$ are the weights for support vectors and $b \in \mathbb{R}$ is decision rule bias. Parameter $C \in \mathbb{R}$ is a penalty factor used to trade-off between errors minimization and margin maximization. Parameters α_i and b and the support vectors $\boldsymbol{q}_i$ are estimated by solving the optimization problem for the soft-margin kernel SVM (Schölkopf and Smola, 2002). With large datasets, the time requirements to minimize the corresponding problem grows, so it is important to utilize efficient computation methods. For efficient learning of the SVM model from the sampled data, we use ThunderSVM – an open-source library that utilizes GPUs to speed-up the computations (Wen et al., 2018).

For each of the submodels (Table 3.1) we must generate independent datasets and SVMs. Hence, the optimal hyperparameters C and γ are different for each submodel. To find C and γ we perform the procedure of grid-search in the following ranges: $C = [1, 5000]$ and $\gamma = [0.1, 10]$ for each model. For the grid-search, we use training and testing datasets with sizes of 100,000 and 200,000, respectively. This scheme offers faster parameter optimization with similar results to k-fold cross-validation. The grid-search results rely heavily on the sampling density and size of the cells in the grid. However, as we have multiple models to optimize, dense sampling on a large range would require excessive computations. Given that, for each model, we first localize optimal areas by performing two searches with the 5x5 grids and finally tune precisely with the 10x10 grid. Optimal hyperparameters for submodels are provided in Table 3.4.

After parameters C and γ are found for every submodel, we use 250,000 sampled postures to train each SVM. We then estimate classification accuracy and the confusion matrix with another 150,000 points previously unseen by the model. Performance validation is achieved by repeating the training five times, each time randomly selecting points for train and test datasets. To correctly classify collided and non-collided postures, we do not need to densely cover all state-space, but rather have enough points in the classification boundary's vicinity.

As our sampling procedure is built to include close-to-collision configurations, we find that 250,000 points are enough to learn a model with decent performance.

To evaluate the performance of the trained model, we use metrics such as classification *Accuracy* (A), *True Negative Rate* (TNR) and *True Positive Rate* (TPR). Those are computed using the elements of the confusion matrix:

$$\mathrm{TNR} = \frac{\mathrm{TN}}{(\mathrm{TN}+\mathrm{FP})}, \ \mathrm{TPR} = \frac{\mathrm{TP}}{(\mathrm{TP}+\mathrm{FN})}, \tag{3.3}$$

where TP stands for *True Positive*, TN – for *True Negative*, FP – for *False Positive* and FN – for *False Negative*. Negative class here is for the collided postures, and positive class means the robot configuration has no self-collisions. A high TPR (low FN) avoids classifying free configurations as collided and does not remove valid configurations from feasible space. On the other hand, a high TNR (low FP), will avoid labeling collided configurations as free. This is the most critical metric to be maximized, as we wish to avoid classifying self-collided configuration as feasible.

SVM Performance Evaluation

The trained models performance is provided in Table 3.4. Submodels distinguish collided and uncollided configurations with an average accuracy of 96.0%. TNR is above 94.5% for every submodel, while lowest value for TPR is 91%. It is worth noticing that 250,000 points randomly sampled for 16-DoF models (submodels 2, 3, 5, 6) correspond to the grid with a density of fewer than three points per axis. Such classification accuracy in the robot's joint space is not yet achieved by the traditional grid and voxel methods used in distance field computation. At the same time, the full-body model with 29 DoF has relatively low accuracy (~86%) even with a training dataset of one million entries.

Learned SVM models have total support vector count $N_{sv} = 236797$ (incl. symmetrical). To evaluate each model for a single query point, it is required to compute the squared euclidean distance from each support vector and calculate the sum of resulting RBF kernels (as in (3.1) and (3.2)). By means of optimized C and CUDA code, we can perform all computations efficiently and fast enough to be used online. The computations are easy to parallelize, so we compare the performance of single- and multithreaded CPU implementation, as well as GPU variant. The hardware used for computations is 4.2 GHz Intel i7-7700K and Nvidia GTX 1060. Parallelization for CPU is achieved by OpenMP, while GPU implementation uses a combination of custom CUDA kernels and CUBLAS routines with asynchronous memory copying and stream control to achieve maximum kernel concurrency. For accurate results, all computations are performed with double precision. The average time needed to evaluate one random query posture in three various ways (CPU sequential/parallel, GPU) is shown in Table 3.2. For better understanding of the iteration time, we include reference time for solving QP IK (more on that in Section 3.5).

	CPU (1 thread)	CPU (4 threads)	GPU
IK (10 iter.)	3.24 ms		
SCA (1 iter.)	7.74 ms	2.55 ms	**1.38** ms
IK+SCA (10 iter.)	116.81 ms	40.87 ms	**19.27** ms

Table 3.2: Computation time required to evaluate trained SVM classifier for one posture (averaged between 1000 trials).

From Table 3.2, it is seen that the use of GPU allows us to control the humanoid robot with self-collision constraints at an efficient rate of 50 Hz. However, the speed-up from multithreaded CPU implementation is only 2x. The model needs to have significantly more support vectors to achieve better GPU acceleration. With more CPU threads (latest processors having more than 16 cores), it should be possible to increase the CPU implementation speed. At the same time, GTX 1060 is a low-tier GPU released in 2016, and newer hardware with more CUDA cores can also significantly accelerate GPU computations.

Sparse SVM (CPSP) Performance Evaluation

The previous sub-section demonstrated that it is possible to learn the self-collision boundary in the robot's 29-dimensional joint space with kernel-based SVM. We also showed that, in order to achieve real-time performance, it is required to utilize high-performance CPU or even GPU. Often, the computational capabilities of robots' onboard computers are limited, and we cannot assume access to GPU. It is hence important to investigate other learning methods, which may result in self-collision models with fewer parameters. We follow the approach proposed by Figueroa et al. (2018) and apply the CPSP method (Joachims and Yu, 2009a) that reformulates the SVM optimization problem to put a strict upper bound on the number of support vectors. A key difference between CPSP and classical SVM is that support vectors $\boldsymbol{q}_i$ are not necessarily points that were part of the training dataset. However, the final expression for the functions $\Gamma(\boldsymbol{q}), \nabla\Gamma(\boldsymbol{q})$ ((3.1) and (3.2)), and the hyperparameters C and γ correspond to those of classical SVM. As in the SVM model learning, we repeat training-validation process five times, shuffling data between training and testing datasets.

To learn the sparse SVMs with CPSP optimization we use the SVM$^{\text{perf}}$ library (Joachims and Yu, 2009a). We set the support vector budget to $N_{max} = 1000$ for submodels, and $N_{max} = 10000$ for the single full-dimensional self-collision model (for evaluation purposes). The resulting accuracy for self-collision detection in submodels is 92.1% on average. FP do not surpass 4.6%, but FN reach 10% for some models. Full metrics for the performance of learned models are provided in Table 3.5. The sparse SVM method results in fast optimization and a reduced number of model parameters, yet with lower prediction accuracy compared to the classical SVM. This could be alleviated by setting a higher N_{max}, however, at the expense of increased model complexity and its computation time.

3.4.3 Self-Collision Boundary Learning via Neural Networks

Since we learn the SCA boundary functions as a classification problem, we also explore the use of Neural Networks (NN). By altering the depth (number of layers) and width (maximal number of nodes in a layer) of a NN, we can control the number of weight coefficients in the model, therefore controlling its size and evaluation time. The difficulty, however, lies in finding the optimal depth and width that will yield the best trade-off between *computational efficiency* and *prediction accuracy*. To ensure the necessary continuity properties for $\Gamma(.)$; i.e., that it should be $\mathcal{C}^1$, we can use continuous activation functions, such as sigmoid or hyperbolic tangent. Through empirical evaluation we found that a NN with $50-30-10$ hidden layer structure and tanh activation function (see Figure 3.4) is the optimal architecture to learn submodels. Several network architectures with various depths and widths were evaluated. The proposed architecture is the most computationally efficient that provides better prediction accuracy than the SVM counterparts. The NN parameters are weight matrices $\boldsymbol{W}_i$ and bias vectors $\boldsymbol{b}_i$, $i = 1,...,4$, their dimensions vary depending on the amount of neurons on i-th layer and dimensionality of state $\boldsymbol{q}$ across various submodels.

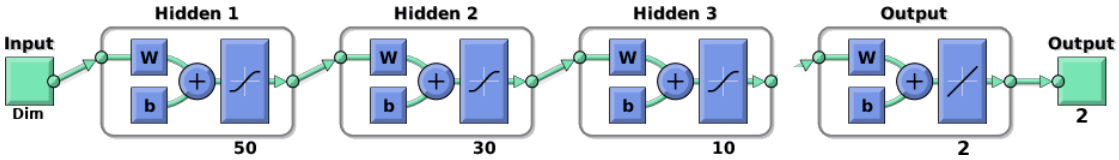


Figure 3.4: NN layers used to learn self-collision boundary function.

For a given posture $\boldsymbol{q}$, the NN outputs a pair of real numbers (p_1, p_2):

$$\begin{aligned}\left[p_1(\boldsymbol{q}), p_2(\boldsymbol{q})\right]^T = \boldsymbol{b}_4 + \boldsymbol{W}_4 \tanh\left(\boldsymbol{b}_3 + \boldsymbol{W}_3 \cdot \right.\\ \left. \tanh\left(\boldsymbol{b}_2 + \boldsymbol{W}_2 \tanh\left(\boldsymbol{b}_1 + \boldsymbol{W}_1 \boldsymbol{q}\right)\right)\right),\end{aligned} \tag{3.4}$$

that represent respective probabilities of belonging to two classes (free or collided) after normalization. As we are interested in a single-valued output for our self-collision boundary function, we set $\Gamma(\boldsymbol{q}) = p_2 - p_1$, so that $\Gamma(\boldsymbol{q}) = 0$ when $p_1 = p_2$, this i.e. posture belongs to the self-collision boundary. For collided postures $\Gamma(\boldsymbol{q}) < 0$, and for feasible postures $\Gamma(\boldsymbol{q}) > 0$, as in previously learned SVM models. Without affecting boundary detection, we may also skip the normalization step, so that $\Gamma(\boldsymbol{q})$ is not bounded. The gradient of SCA function is derived as $\nabla\Gamma(\boldsymbol{q}) = \dfrac{\partial p_2}{\partial \boldsymbol{q}} - \dfrac{\partial p_1}{\partial \boldsymbol{q}}$, where

$$\begin{aligned}\left[\frac{\partial p_1(\boldsymbol{q})}{\partial \boldsymbol{q}}, \frac{\partial p_2(\boldsymbol{q})}{\partial \boldsymbol{q}}\right]^T = -\boldsymbol{W}_1\boldsymbol{W}_2\boldsymbol{W}_3\boldsymbol{W}_4\left(\tanh\left(\boldsymbol{b}_1 + \boldsymbol{W}_1\boldsymbol{q}\right)^2 - 1\right)\\ \cdot\left(\tanh\left(\boldsymbol{b}_2 + \boldsymbol{W}_2\tanh\left(\boldsymbol{b}_1 + \boldsymbol{W}_1\boldsymbol{q}\right)\right)^2 - 1\right)\\ \cdot\left(\tanh\left(\boldsymbol{b}_3 + \boldsymbol{W}_3\tanh\left(\boldsymbol{b}_2 + \boldsymbol{W}_2\tanh\left(\boldsymbol{b}_1 + \boldsymbol{W}_1\boldsymbol{q}\right)\right)\right)^2 - 1\right).\end{aligned} \tag{3.5}$$

In (3.4) and (3.5) the hyperbolic tangent functions are applied element-wise for a vector input.

Table 3.3: Comparison for various learning methods used to learn self-collision detection function. All parameters represent sum or average value for such parameters across ten submodels (incl. symmetrical) from Table 3.1.

Learning Method	SVM	CPSP	NN
Model size, # of doubles	2099422	138000	**15322**
Total training time, s	11250	4762	**2106**
Avg. accuracy	0.960	0.920	**0.990**
Avg. TPR	0.949	0.878	**0.984**
Avg. TNR	0.973	0.964	**0.995**
Evaluation time, ms	1.38(GPU)	0.42(CPU)	**0.11**(CPU)

We use the PyTorch library to learn the weights of the NNs by optimizing the negative log likelihood loss with RMSProp running for 2000 epochs. We train six NNs to cover self-collisions for all submodels from Table 3.1. As training time is significantly lower than in SVM models, and the model size does not increase when a larger dataset of training points is used, we train each submodel with 900,000 postures to achieve better accuracy, and validate using 100,000 unseen points. We repeat train-test procedure five times, shuffling data between training and testing datasets. For the trained models, the mean accuracy is 99.0%, and the FNR is 0.08% on average. Full results for training and validation are provided in Table 3.6. NNs outperform SVMs in terms of accuracy, learning speed and evaluation time. Computation of values and gradients of function $\Gamma(.)$ learned via NNs requires $\sim$ 0.11ms on 4.2GHz Intel i7-7700K. This includes 15 consecutive evaluations of ten submodels from Table 3.1 (our IK algorithm requires at least 15 iterations to converge). We can compare our method with (Quiroz-Omaña and Adorno, 2019), where the Vector Field Inequalities approach is used to compute the distance function and its gradient to avoid collisions. In their approach, the single evaluation of Jacobian constraint takes $\sim$ 5.2ms with comparable CPU. The learned self-collision representation is visualized in Figure 3.5.

One might wonder if a single large NN is capable of modeling the SCA boundary as accurately as our decomposed SCA-NN approach. In the first row of Table 3.6 we include the evaluation metrics of a single 29-dimensional NN model without submodel decomposition. To find the optimal NN structure we performed grid-search on combinations of depth and width sizes, with ranges of 3-9 hidden layers and 50-250 neurons per layer. By training the networks on a larger dataset (2.5 million) and running it for 5000 epochs, we found a 4-layer (including output) NN with 250 neurons on each layer to be the optimal choice. However, the prediction accuracy of this model reaches at most 92-93%, which is not sufficient to justify the increase in model complexity ($\sim$ 130k weights) and computation time.

3.4.4 Comparison across methods

In Tables 3.4, 3.5 and 3.6 we provide all the performance metrics for the three approaches, respectively. Further quantitative comparison of performance across is shown in Table 3.3.

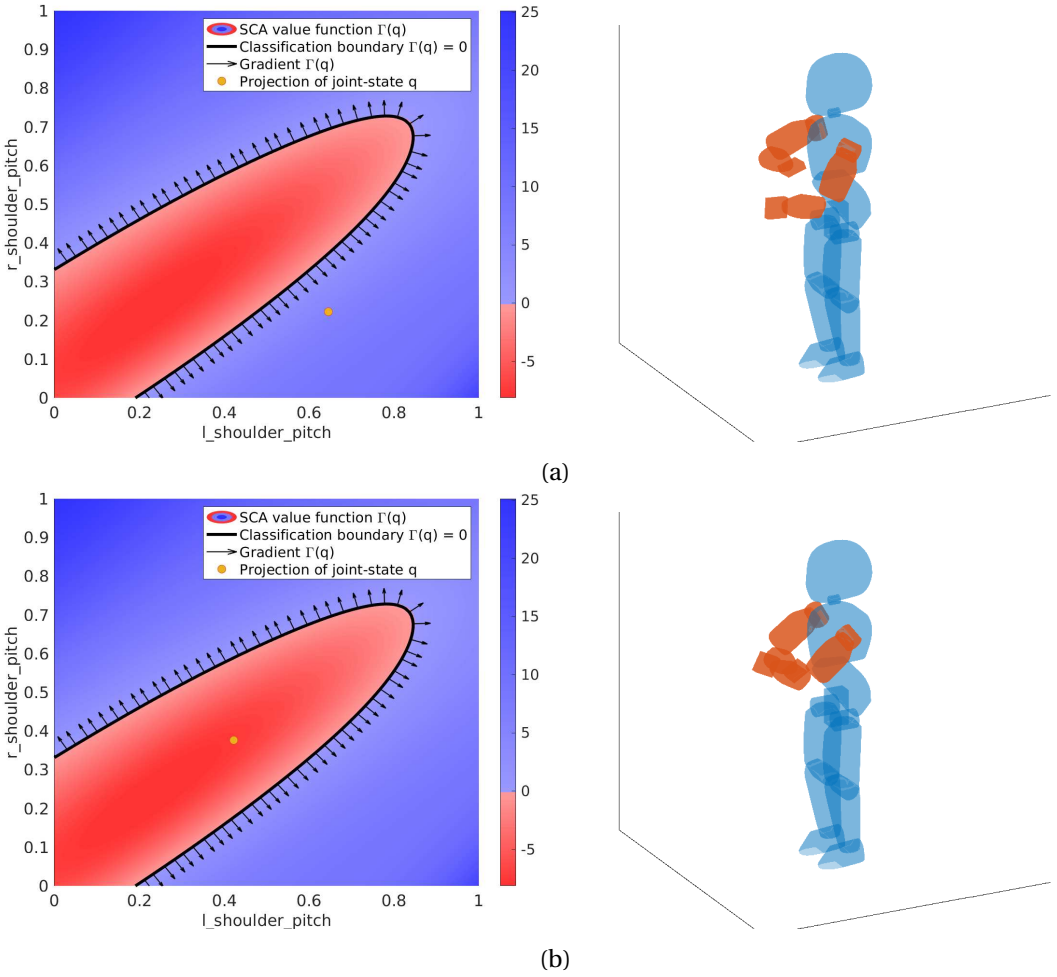


Figure 3.5: Visualization of the SCA boundary for the first submodel describing collisions between two arms (14-DoF). On the left is a two-dimensional projection of the joint space, and on the right is the robotic visualization. (a) depicts the uncollided state, while (b) illustrates a situation where both arms are colliding.

Such comparison depends heavily on the efforts spent on programming the techniques.[1] To provide a fair comparison, we used the most recent and efficient implementations offered for SVM (ThunderSVM), CPSP (SVM$^{\text{perf}}$), and NN (PyTorch). As expected, SVM provides better average accuracy as well as better TPR and TNR than CPSP, whereas CPSP model is significantly smaller and offers faster evaluation. NN gives the overall best performance with even fewer parameters and better accuracy, as well as better TPR and TNR and the best training and evaluation time. Notably, the NN's performance on the test data is very similar to its training performance. While this may be deemed as overfitting, this particular behavior can be justified in the self-collision detection scenario. In the simulations and experiments reported next, we used the function $\Gamma(.)$ learned via NNs, as it offers the best combination of classification accuracy and evaluation times for querying postures.

[1] The source code for this chapter is available at `https://github.com/epfl-lasa/Joint-Space-SCA`

Table 3.4: Performance and confusion matrix elements for self-collision models trained with classical kernel SVM. Each submodel is trained on 250,000 points dataset and tested on 150,000 unseen data points. Full model (0) is trained on 1,000,000 (one million) data samples. The elements of the confusion matrix are presented relatively to testing dataset size, and in the perfect scenario TP = TN = 0.5, while FP = FN = 0. All values (except for columns 1-4) are averaged between five training-validation runs. Standard deviations σ for each cell in columns 7-14 do not exceed 0.001.

N	Dim	C	γ	N_{sv}	Training time, s	Training accuracy	Test accuracy	TPR	TNR	TN	TP	FN	FP
0	29	100	0.1	182548	15632	0.891	0.864	0.855	0.875	0.438	0.426	0.074	0.062
1	14	2000	0.5	25767	2781	0.989	0.943	0.936	0.951	0.476	0.468	0.032	0.024
2 and 6	16	400	0.4	42923	1756	0.976	0.926	0.910	0.945	0.473	0.453	0.047	0.027
3 and 5	16	1000	0.4	32788	2962	0.977	0.942	0.925	0.962	0.481	0.461	0.039	0.019
4 and 7	10	100	1.5	15886	287	0.999	0.979	0.971	0.987	0.494	0.486	0.014	0.006
8	12	5000	0.5	13988	3418	0.991	0.976	0.971	0.982	0.491	0.485	0.015	0.009
9 and 10	9	500	0.5	6924	46	0.992	0.991	0.985	0.997	0.499	0.493	0.007	0.001

Table 3.5: Performance and confusion matrix elements for self-collision models trained with CPSP method. Each submodel is trained on 250,000 points dataset and tested on 150,000 unseen data points. Full model (0) is trained on 1,000,000 (one million) data samples. All values (except for columns 1-5) are averaged between five training-validation runs. Standard deviations σ for each cell in columns 7-14 do not exceed 0.002.

N	Dim	C	γ	N_{sv}	Training time, s	Training accuracy	Test accuracy	TPR	TNR	TN	TP	FN	FP
0	29	100	0.1	10000	117399	0.782	0.746	0.901	0.591	0.296	0.451	0.049	0.204
1	14	2000	0.5	1000	1052	0.916	0.906	0.869	0.943	0.476	0.434	0.066	0.028
2 and 6	16	400	0.4	1000	926	0.888	0.868	0.827	0.908	0.473	0.414	0.086	0.046
3 and 5	16	1000	0.4	1000	851	0.918	0.900	0.832	0.967	0.481	0.416	0.084	0.016
4 and 7	10	100	1.5	1000	676	0.938	0.936	0.891	0.981	0.494	0.446	0.054	0.009
8	12	5000	0.5	1000	826	0.965	0.964	0.939	0.989	0.491	0.470	0.030	0.006
9 and 10	9	500	0.5	1000	431	0.976	0.976	0.960	0.992	0.499	0.480	0.020	0.004

Table 3.6: Performance and confusion matrix elements for self-collision models trained with NNs. Submodels are trained on 900,000 points dataset and tested on 100,000 unseen data points. Full model (0) is trained on 2,500,000 points, and has the same feed-forward architecture with three hidden layers, but with 250 neurons on each layer. All values (except for columns 1-3) are averaged between five training-validation runs. Standard deviations σ for each cell in columns 5-12 do not exceed 0.001.

N	Dim	N_w	Training time, s	Training accuracy	Test accuracy	TPR	TNR	TN	TP	FN	FP
0	29	132750	2289	0.925	0.923	0.933	0.914	0.456	0.468	0.034	0.043
1	14	2612	348	0.999	0.998	0.997	1.000	0.500	0.499	0.002	0.000
2 and 6	16	2712	354	0.971	0.971	0.958	0.985	0.492	0.479	0.021	0.008
3 and 5	16	2712	350	0.989	0.988	0.980	0.996	0.498	0.490	0.010	0.002
4 and 7	10	2412	359	0.993	0.993	0.989	0.998	0.499	0.495	0.005	0.001
8	12	2512	343	0.994	0.994	0.989	0.998	0.499	0.495	0.006	0.001
9 and 10	9	2362	352	0.998	0.998	0.996	0.999	0.500	0.498	0.002	0.000

3.5 Application to online IK solver

We propose to use the learned self-collision models as constraints for an online Inverse Kinematics solver. Taking inspiration from the work by Salehian et al. (2018b), we model the self-collision boundary as a separation hyperplane and use it as a constraint for convex QP problem that can be solved in real-time. Compared to (Salehian et al., 2018b), our contribution is that we extend the method to the humanoid robot and demonstrate the use of multiple collision constraints. We propose the following quadratic problem to solve the IK (adapted from (Figueroa et al., 2020) and (Salehian et al., 2018b)):

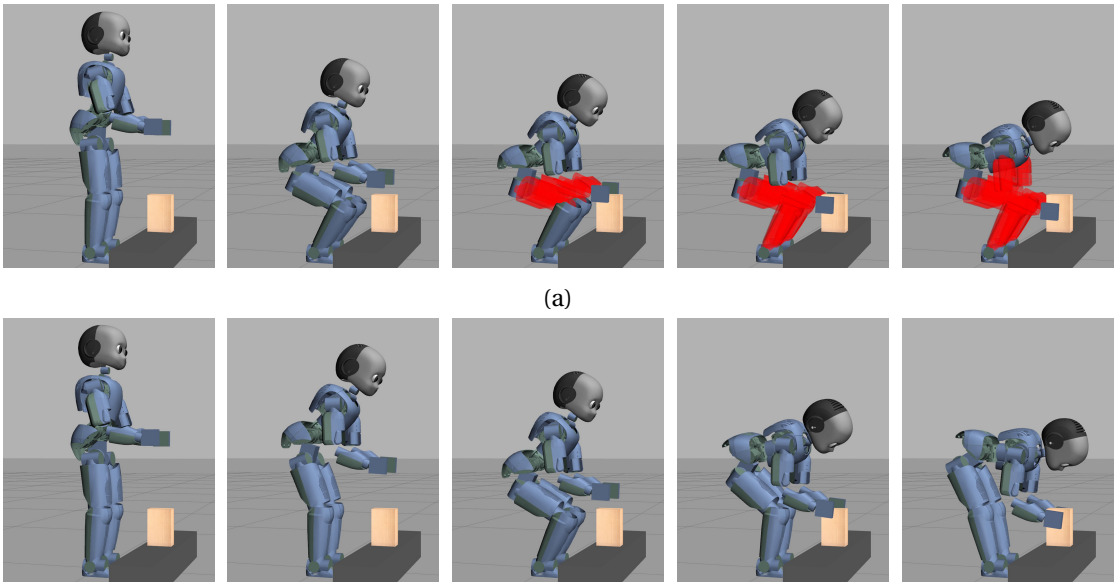
$$\begin{aligned} &\min_{\Delta \boldsymbol{q}, \boldsymbol{\delta}} \boldsymbol{\delta}^T \boldsymbol{Q} \boldsymbol{\delta} + \Delta \boldsymbol{q}^T \boldsymbol{R} \Delta \boldsymbol{q} \\ &\text{s.t.} \quad \begin{cases} f(\boldsymbol{q}) + \dfrac{\partial f(\boldsymbol{q})}{\partial \boldsymbol{q}} \Delta \boldsymbol{q} = \boldsymbol{x} + \boldsymbol{\delta} \\ q_i^- < q_i + \Delta q_i < q_i^+, \quad i = 1..29 \\ -\nabla \Gamma_i(\boldsymbol{q}) \Delta \boldsymbol{q} \leq \ln(\Gamma_i(\boldsymbol{q}) + 1), \quad i = 1..10. \end{cases} \end{aligned} \tag{3.6}$$

where $f(\boldsymbol{q})$ stands for the forward kinematics of the robot, $\boldsymbol{R}$ is equivalent to the well-known damping term in least-squares IK methods, and $\boldsymbol{Q}$ is a diagonal matrix that sets weights for the Cartesian tasks vector $\boldsymbol{x}$. Tasks include positions and orientations of the feet, hands, and the center of mass. Constraints in (3.6) represent a cartesian task $\boldsymbol{x}$ with slack $\boldsymbol{\delta}$, joint limits, and self-collision avoidance respectively. The last constraint forces the joint angles to move away from the self-collisions hyperplane as they approach it. When the robot is far from the boundary and $\Gamma_i(\boldsymbol{q}) > 0$, values $\ln(\Gamma_i(\boldsymbol{q}) + 1)$ are positive, which relaxes the set of self-collision inequality constraints, i.e., the robot accurately follows the desired end-effector trajectory. If i-th submodel is close to self-collision, then $\ln(\Gamma_i(\boldsymbol{q}) + 1)$ becomes negative, and the IK solver is forced to align joint motion $\Delta \boldsymbol{q}$ with corresponding gradient $\nabla \Gamma_i(\boldsymbol{q})$, thus moving the robot configuration away from the collision boundary and satisfying self-collision avoidance constraint. Recall that, due to symmetry, only six submodels (out of ten) are learned. To calculate Γ_i for symmetrical submodels ($i = 5, 6, 7, 10$), we mirror the torso pitch and yaw joints: $q_t^p \to 1 - q_t^p$ and $q_t^y \to 1 - q_t^y$, and invert the sign for the corresponding coordinates of $\nabla \Gamma_i$.

Equation (3.6) is a convex QP problem with equality and inequality constraints, hence, there is no closed-form solution for it. To solve it, we utilize CVXGEN solver, which generates C codes tailored for the specific formulation of the optimization problem (Mattingley and Boyd, 2012). The proposed control scheme is for the position-controlled robots, and after solving (3.6) the desired joint angles for the next control iteration are computed as $\boldsymbol{q}_{new} = \boldsymbol{q} + \Delta \boldsymbol{q}$.

3.6 Experimental Validation

In our experiment, we consider the picking-up scenario. The robot is supposed to bend from the standing posture and reach the object located on the ground. Without SCA constraints



(a)

(b)

Figure 3.6: The robot is trying the object close to the ground without (a) and with (b) proposed SCA constraints in the IK solver. Collided links are highlighted red. Refer to Video Link for the full experiment.

in Inverse Kinematics, the robot tends to have a self-collision between the elbow and the knee (Figure 3.6a). Not only this self-collision prevents the robot from reaching the target, but it is also harmful to the hardware, leading to torn cables and plates bending. However, with the proposed SCA constraints in the IK solver, the robot avoids that collision by shifting the hips and straightening the knees to maximize the distance between legs and arms links (Figure 3.6b).

Figs. 3.6a and 3.6b are from simulated environment, as it is easier to demonstrate self-collisions there. We also conducted the experiment with picking-up scenario on the real robot. As expected, iCub is able to pick up arbitrary positioned box avoiding self-collisions. Snapshots of one full picking-up motion are demonstrated in Figure 3.7, while grasping postures for three different box positions are shown in Figs. 3.8b - 3.8d.

3.6.1 Comparison with other methods

To evaluate the proposed method's performance in a general picking-up scenario, we perform the following benchmark. The box object is placed randomly in front of the robot (red area in Figure 3.8a), and three different IK algorithms are used to generate a grasping posture from a fixed initial state. IK methods include (i) the proposed method (as in (3.6)), (ii) QP solver without SCA constraint ((3.6) without last line), and (iii) the QP solver where collision-avoidance is a task based on repulsion in Euclidean space.

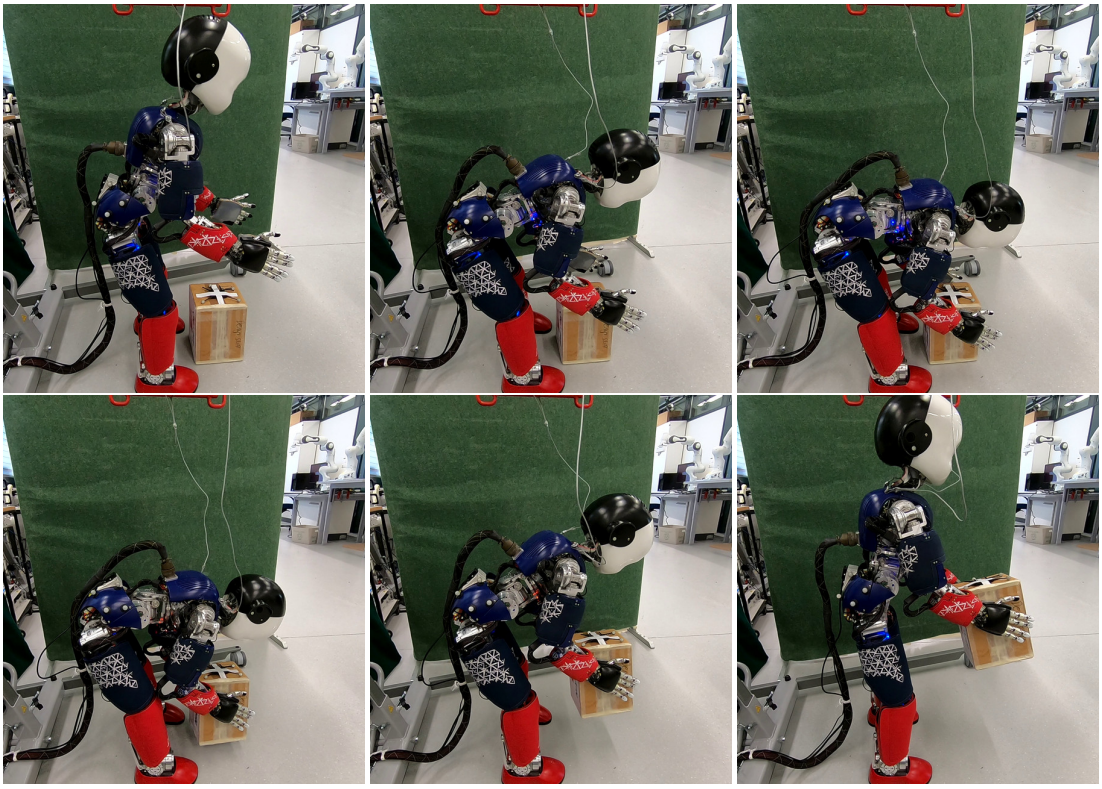


Figure 3.7: iCub is picking the box from the ground with the proposed SCA constraints. Refer to Video Link for the full experiment.

For the case of task-space collision avoidance constraint (iii), the QP problem (3.6) is modified to additionally minimize $\boldsymbol{\sigma}^T \boldsymbol{S}\boldsymbol{\sigma}$, where $\boldsymbol{S}$ is a diagonal weight matrix, and $\boldsymbol{\sigma}$ is a slack variable:

$$\boldsymbol{J}_i^{\text{SCA}}(\boldsymbol{q})\Delta\boldsymbol{q} = s\boldsymbol{v}_i^{\text{rep}} + \boldsymbol{\sigma}. \tag{3.7}$$

Here, $\boldsymbol{v}_i^{\text{rep}} \in \mathbb{R}^3$ is a vector connecting two closest points of the bodies in i-th submodel (i = 1...10) in task-space; i.e. $\boldsymbol{v}_i^{\text{rep}} = \frac{\boldsymbol{p}_1 - \boldsymbol{p}_2}{||\boldsymbol{p}_1 - \boldsymbol{p}_2||}$ where $\boldsymbol{p}_1, \boldsymbol{p}_2 \in \mathbb{R}^3$ are the closest points on the robot body. s is a scaling factor for $\boldsymbol{v}_i^{\text{rep}}$ to match the magnitude of $\Delta\boldsymbol{q}$. 3.7 replaces the proposed one (last line in (3.6)) with a traditional repulsive force and constraints approaches, similar to Sugiura et al. (2006, 2007); Schwienbacher et al. (2011) and (Stasse et al., 2008), respectively. To speed-up the distance computations, the geometry is approximated with convex hulls instead

Table 3.7: Self-collision avoidance constraints performance for 1000 random object positions.

IK method	r_c, %	d, cm	IK time, ms
No SCA (15 iter.)	63.26	6.56	3.41
Proposed method (15 iter.)	**4.86**	**5.78**	**3.61**
Jacobian-based SCA (15 iter.)	14.52	7.68	15.02
Jacobian-based SCA (30 iter.)	7.74	5.69	29.14

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

Figure 3.8: (a) - Area for random positions of the object in the benchmark. (b)-(d) - Grasping posture of the robot for various box positions.

of precise meshes, and FCL library is used to determine closest points on the bodies.

We check the final posture for the collision and calculate the error $d = \frac{1}{2}\left(||p_l^d - p_l^s|| + ||p_r^d - p_r^s||\right)$ between the desired positions for left and right hands $p_{\{l,r\}}^d$, and positions $p_{\{l,r\}}^s$ given by the IK solvers. We run the simulation 1000 times and compare the collision rate r_c, average hands position error d, and evaluation time for IK; the results are provided in Table 3.7. The collision rate r_c is lowest for the proposed method, while evaluation time is almost unaffected compared to basic IK without self-collision avoidance. At the same time, to achieve similar performance (in terms of mean task error d) with Jacobian-based SCA, it is necessary to perform more iterations of the Sequential QP IK; combined with repetitive distance calculations, that significantly slows down the computations.

3.7 Conclusion

In this chapter, we proposed a computationally efficient method to detect and avoid self-collisions for high-DoF robots, such as humanoids. We treat self-collisions as constraints in the robot's joint space, allowing for direct use in the control scheme. Apart from the demonstrated use in Sequential QP IK solver, the proposed self-collision avoidance constraint can be employed in other algorithms. For instance, it may serve as collision detection in RRT-navigation methods (Kuffner and LaValle, 2000), additionally providing a gradient to guide the sampling procedure. To some extent, it can replace distance field computations in (Ratliff et al., 2009). Although our method may require extensive computations to learn a self-collision boundary, such computations are performed only once. Furthermore, it provides a model for reliable and efficient self-collision detection, fitting the computation time within real-time limitations.

One drawback of our method is that it requires some heuristics, such as model separation, to reduce the overall dimensionality of the problem. Overall, we find that systems with dimensionality of up to 16 can be encapsulated into a single model, which would hold for almost any robotic arm. Moreover, the presented model separation strategy can be applied to any other humanoid robot. In one of the tangential studies (conducted as a semester project), we successfully applied a similar self-collision detection framework to a robotic hand, where each finger's collisions were learned separately. As a result, we are confident that the presented method can be applied to a wide range of robotic systems.

It would be interesting to investigate how sophisticated sampling procedures (for example, balancing the number of collisions between every link pair) may help learn a single 29-dimensional model of feasible postures. Additionally, it is possible to include other constraints, such as static balance, in the classification, thus expanding the unfeasible class.

4 Obstacle Avoidance for Robotic Manipulators

All models are wrong, but some are useful.
— George E. P. Box

4.1 Foreword

Previous chapter demonstrated that self-collisions of a redundant system could be represented as a static boundary in the high-dimensional joint space of a robot. When this boundary is approximated as a continuously differentiable function of class C^1, the gradients with respect to the input robot configuration essentially represent a repulsive vector field directly in the joint space of the robot; that repulsion can be used as a constraint in a control optimization routine.

In this chapter, this idea is extended to account for collisions with obstacles in the robot's workspace. We note that static collision boundaries in joint space can be interpreted as implicit signed distance-to-collision functions, if the robot state space is complemented with the Euclidean three-dimensional space. We present a novel approach for constructing an implicit distance function for distance evaluation between a robot in arbitrary configurations and any point in the three-dimensional workspace of the robot. The learned neural model allows for efficient and highly-parallelizable batched distance and gradient queries via GPU.

The work presented in this chapter has been published in *M. Koptev, N. Figueroa and A. Billard, "Neural Joint Space Implicit Signed Distance Functions for Reactive Robot Manipulator Control," in IEEE Robotics and Automation Letters, vol. 8, no. 2, pp. 480-487, Feb. 2023, doi: 10.1109/LRA.2022.3227860.*

Parts of this work were also presented at the *Motion Planning with Implicit Neural Representations of Geometry* workshop, held during the the *IEEE International Conference on Robotics and Automation (ICRA) 2022.*

4.2 Introduction

Motion planning is a crucial element of robotic control and has continuously attracted the interest of researchers throughout the years. Industrial, retail, and lately, domestic scenarios require robotic motion to satisfy certain constraints, such as energy minimization, smooth trajectories, goal-reaching, and orientation keeping. Yet, the most important constraint to satisfy in any motion planning task is to avoid collisions and self-collisions. Standard techniques avoid such collisions by computing distances between geometric approximations of the robot's body segments represented as spheres, ellipsoids, swept volumes and even meshes (Gilbert et al., 1988). One can then formulate constraints or costs for the motion planning algorithm of choice based on the computed minimum distances (Harrison et al., 2020).

While avoiding self-collisions is important, handling external collisions, i.e., collisions between the robot and its environment, is essential for a robot that must operate in constrained and dynamic environments. These collisions can be categorized into two types: collisions with static obstacles, such as furniture or walls, and collisions with moving obstacles, including other robots or humans. Avoiding moving obstacles is crucial for consumer technology, where robots should operate in an environment shared with humans, which assumes that the robot must always be prepared to avoid humans in its workspace to prevent possible injuries. With the advent of compliant collaborative robots equipped with force/torque sensors and robust vision systems, a robot can halt its motion when a collision is detected (Haddadin et al., 2017). Yet, advanced collaborative systems are expected to continue task execution while avoiding any type of obstacles in both a preemptive and reactive manner. Additionally, certain scenarios in human-robot interaction require maintaining direct contact between the robot and humans (Khoramshahi et al., 2020; Li et al., 2021). Thus, it is essential to have a notion of a repulsive vector field and corresponding tangential plane to handle the contact. Having those defined directly in joint space helps to plan more efficiently.

The problem of collision avoidance is generally treated as a constraint in a path-planning routine (Ratliff et al., 2009). Optimization and collision checking computations are unfeasible to perform in real-time, so optimal trajectories are computed offline before execution. However, if the obstacles near the robot are moving unpredictably, the control must be *reactive* and adjust the trajectory at execution time.

In this chapter, we assume the obstacles to be dynamic and, hence, investigate the applicability of the learned distance function to achieve real-time performance with two state-of-the-art *reactive* control methods: i) a collision-aware QP (quadratic program)-IK (inverse kinematics) solver that optimizes for the next immediate step and has shown real-time performance for high-dimensional humanoid robots (Faraji and Ijspeert, 2017), and ii) a sampling-based MPC that has a short look-ahead horizon (Williams et al., 2017; Bhardwaj et al., 2022). Both methods are reported to generate solutions with at least 100Hz frequency, which should allow real-time reactive obstacle avoidance.

4.2.1 Chapter organization

Section 4.3 presents the mathematical problem formulation, including assumptions, definitions and goals of this chapter. In Section 4.4 we present our proposed approach for learning a neural joint space implicit signed distance function (Neural-JSDF) for a given robot. Subsequently, Section 4.5 covers the application of the proposed Neural-JSDF to a QP-IK solver and sampling-based MPC approach for reactive control. Finally, we evaluate the performance of these two reactive control techniques in simulation and with real robotic experiments in Section 4.6.

4.3 Problem Formulation

We consider a robotic manipulator with m degrees of freedom and K links, whose state is described by joint angles $\boldsymbol{q} = [q^1, ..., q^m] \in \mathcal{C}$, where $\mathcal{C}$ stands for configuration space (C-space). All joints of the robot are revolute type and bounded by joint-limits $\boldsymbol{q} \in [q_{min}, q_{max}]$, thus $\boldsymbol{q} \in \mathcal{C} \subset \mathbb{R}^m$.

In this work, we seek to control a robotic manipulator in joint space to reach a task-space goal $\boldsymbol{x}^* \in \mathcal{SE}(3)$ for its end-effector that could be pre-defined or generated by a task-space control law, all the while handling collisions between the robot's body and static and dynamic objects in real-time; i.e., with a desired control loop frequency of ≥100Hz.

4.3.1 Assumptions & Definitions

Let us assume $\mathcal{B} \subset \mathbb{R}^3$ to be the set of points that geometrically describe the physical body of the robot. The points of the k-th link of the robot create subsets $\mathcal{B}_k \subset \mathcal{B}$, $k = 1, ..., K$. The robot's kinematics is known, with forward kinematics (FK) denoted by $f : \mathcal{C} \times \mathcal{B} \rightarrow \mathbb{R}^3$, mapping a robot configuration $\boldsymbol{q} \in \mathcal{C}$ and arbitrary robot body point $x \in \mathcal{B}$ to a workspace point $f(\boldsymbol{q}, \boldsymbol{x}) \in \mathbb{R}^3$. We can define minimal distances between the k-th link of the robot and arbitrary point $\boldsymbol{y} \in \mathbb{R}^3$ in the workspace:

$$d_k(\boldsymbol{q}, \boldsymbol{y}) = \min_{x \in \mathcal{B}_k} ||f(\boldsymbol{q}, \boldsymbol{x}) - \boldsymbol{y}||. \tag{4.1}$$

The workspace of the robot contains static and dynamic obstacles described by the sets of points $\mathcal{O}^s \subset \mathbb{R}^3$ and $\mathcal{O}^d \subset \mathbb{R}^3$, respectively. Dynamic obstacles can be generalized and include static, such that $\mathcal{O}^s \subset \mathcal{O}^d$, hence we will herein refer to the obstacle set as $\mathcal{O} \subset \mathbb{R}^3$ regardless of obstacles being static or dynamic. This definition allows us to write down minimal distances between the robot in configuration $\boldsymbol{q}$ and the environment as:

$$d_{min}(\boldsymbol{q}) = \min_{k, \boldsymbol{y}} d_k(\boldsymbol{q}, \boldsymbol{y}), \tag{4.2}$$

where $\boldsymbol{y} \in \mathcal{O}$ are points belonging to obstacles in the workspace. Naturally, if we seek to avoid collisions between the robot and the environment with a given margin l, any command position $\boldsymbol{q}$ must satisfy the condition $d_{min}(\boldsymbol{q}) > l$.

4.3.2 Goals

Two main goals are defined to achieve our objective:

- The *first goal* of this chapter is to learn a regression function $\boldsymbol{\Gamma}(\boldsymbol{q}, \boldsymbol{y})$ to approximate a K-dimensional minimal distance vector $\boldsymbol{d}(\boldsymbol{q}, \boldsymbol{y}) = [d_1(\boldsymbol{q}, \boldsymbol{y}), ..., d_K(\boldsymbol{q}, \boldsymbol{y})]$ with each entry representing the minimal distance between the k-th link and and arbitrary 3D point. Function $\boldsymbol{\Gamma}(\boldsymbol{q}, \boldsymbol{y})$ represents a distance field in the robot's workspace depending on the joint configuration of the robot. Additionally, $\frac{\partial \boldsymbol{\Gamma}(\boldsymbol{q}, \boldsymbol{y})}{\partial \boldsymbol{q}}$ is a vector field defined in the joint space of the robot, providing information on the direction to (and away from) potential collisions. We provide the theoretical derivation and properties of the proposed distance field $\boldsymbol{\Gamma}(\boldsymbol{q}, \boldsymbol{y})$ as well as the learning architecture and evaluation in Section 4.4.

- The *second goal* is to apply the learned function $\boldsymbol{\Gamma}(\boldsymbol{q}, \boldsymbol{y})$ and its gradient to enhance reactive control methods, by i) formulating a collision-avoidance constraint in a QP-based IK controller (as in previous chapter), and ii) formulating a collision-avoidance cost to leverage the parallelization properties in a sampling-based MPC approach (Bhardwaj et al., 2022). Corresponding method formulations are described in Section 4.5. Real-time experiments reported in Section 4.6.

4.4 Learning the Implicit Signed Distance Function

4.4.1 Implicit Signed Distance Field

Let us consider the expanded state-space $\mathbb{R}^m \times \mathbb{R}^3$, consisting of the robot state $\boldsymbol{q} \in \mathbb{R}^m$ and a Euclidean point $\boldsymbol{y} \in \mathbb{R}^3$ in the robot's workspace. For each expanded state, a unique minimal distance exists between the robot in configuration $\boldsymbol{q}$ and the 3D point at position $\boldsymbol{y}$. Hence, a static implicit distance field function exists in this space.

We propose to build a neural representation $\boldsymbol{\Gamma}(\boldsymbol{q}, \boldsymbol{y}) : \mathbb{R}^m \times \mathbb{R}^3 \rightarrow \mathbb{R}^K$, by learning the minimal distances between the robot's links and arbitrary points in the workspace. For typical redundant robotic manipulators $m < K$. For example, the 7-DoF Franka Emika Panda robotic manipulator can be defined with $m = 7$ and $K = 9$ (excluding the gripper fingers from the model), see Fig. 4.6. The resulting function $\boldsymbol{\Gamma}(\boldsymbol{q}, \boldsymbol{y}) = [\Gamma_1, ..., \Gamma_K]$ should represent minimal distances $\boldsymbol{d}(\boldsymbol{q}, \boldsymbol{y}) = [d_1(\boldsymbol{q}, \boldsymbol{y}), ..., d_K(\boldsymbol{q}, \boldsymbol{y})]$ between the point and each robot link. The partial derivative of the vector-valued distance field $\boldsymbol{\Gamma}(\boldsymbol{q}, \boldsymbol{y})$ with respect to $\boldsymbol{q}$ results in the following

Jacobian matrix, $\text{Jac}_{\boldsymbol{q}}\big(\boldsymbol{\Gamma}(\boldsymbol{q},\boldsymbol{y})\big) \in \mathbb{R}^{m\times K}$,

$$\text{Jac}_{\boldsymbol{q}}\big(\boldsymbol{\Gamma}(\boldsymbol{q},\boldsymbol{y})\big) = \frac{\partial \boldsymbol{\Gamma}(\boldsymbol{q},\boldsymbol{y})}{\partial \boldsymbol{q}} = \begin{bmatrix} \frac{\partial \Gamma_1(\boldsymbol{q},\boldsymbol{y})}{\partial \boldsymbol{q}_1} & \cdots & \frac{\partial \Gamma_K(\boldsymbol{q},\boldsymbol{y})}{\partial \boldsymbol{q}_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial \Gamma_1(\boldsymbol{q},\boldsymbol{y})}{\partial \boldsymbol{q}_m} & \cdots & \frac{\partial \Gamma_K(\boldsymbol{q},\boldsymbol{y})}{\partial \boldsymbol{q}_m} \end{bmatrix} \tag{4.3}$$

where each k-th column vector of $\text{Jac}_{\boldsymbol{q}}\big(\boldsymbol{\Gamma}(\boldsymbol{q},\boldsymbol{y})\big)$ corresponds to the gradient of each k-th scalar distance function $\Gamma_k(\boldsymbol{q},\boldsymbol{y})$ with respect to $\boldsymbol{q}$; i.e., $\text{Jac}_{\boldsymbol{q}}\big(\boldsymbol{\Gamma}(\boldsymbol{q},\boldsymbol{y})\big) = [\nabla_{\boldsymbol{q}}\Gamma_1(\boldsymbol{q},\boldsymbol{y}),\ldots,\nabla_{\boldsymbol{q}}\Gamma_K(\boldsymbol{q},\boldsymbol{y})]$ and $\nabla_{\boldsymbol{q}}\Gamma_k(\boldsymbol{q},\boldsymbol{y}) \in \mathbb{R}^m$ as,

$$\nabla_{\boldsymbol{q}}\Gamma_k(\boldsymbol{q},\boldsymbol{y}) = \frac{\partial \Gamma_k(\boldsymbol{q},\boldsymbol{y})}{\partial \boldsymbol{q}} = \sum_{i=1}^{m} \frac{\partial \Gamma_k(\boldsymbol{q},\boldsymbol{y})}{\partial \boldsymbol{q}_i}\hat{\boldsymbol{q}}_i, \tag{4.4}$$

with $\hat{\boldsymbol{q}}_i$ as the unit vector indicating the i-th dimension. Such gradient functions help explain how each joint influences the distance-to-collision of obstacle $\boldsymbol{y}$ with the k-th link. Each k-th distance function, $\Gamma_k(\boldsymbol{q},\boldsymbol{y})$, can be considered an admissible potential field for obstacle avoidance (Khatib, 1986; Rimon and Koditschek, 1992) in joint space. Consequently, $\nabla_{\boldsymbol{q}}\Gamma_k(\boldsymbol{q},\boldsymbol{y})$ becomes a repulsive joint space vector field, which is helpful for control (Vahrenkamp et al., 2012).

Therefore, learning $\boldsymbol{\Gamma}(\boldsymbol{q},\boldsymbol{y})$ will enable the evaluation of distances between the robot and points on the obstacles and using $\nabla_{\boldsymbol{q}}\boldsymbol{\Gamma}(\boldsymbol{q},\boldsymbol{y})$ to represent the repulsive vector field. Ideally, $\boldsymbol{\Gamma}(\boldsymbol{q},\boldsymbol{y}) = [d_1(\boldsymbol{q},\boldsymbol{y}),...,d_K(\boldsymbol{q},\boldsymbol{y})]$ for any combination of state $\boldsymbol{q}$ within joint-limits and point $\boldsymbol{y}$ in the robot's workspace. A Multi-Layer Perceptron (MLP) is used to learn $\boldsymbol{\Gamma}(\boldsymbol{q},\boldsymbol{y})$. The data collection procedure and network architecture are discussed in the following subsections.

4.4.2 Dataset Generation

Given the exact geometry of the robot's body (for example with 3D CAD models or URDFs) allows for the collection of a synthetic dataset of exact distance values $d_k(\boldsymbol{q},\boldsymbol{y})$, as in Eq. 4.1, for various $\boldsymbol{q}$ and $\boldsymbol{y}$ at training time. Each sample contains the robot state $\boldsymbol{q}$, workspace point $\boldsymbol{y}$, and target vector $\boldsymbol{d} = [d_1,...,d_K]$ consisting of minimal distances between links of the robot and point $\boldsymbol{y}$. Similarly to the previous chapter, we perform a uniform random sampling within the joint limits of the robot to generate the training dataset. The joint limits for training data are expanded by 5% in both directions to aid generalization for predictions close to joint limits. For each sampled robot configuration $\boldsymbol{q}$, 1000 various workspace points $\boldsymbol{y}$ are collected.

The final dataset contains five million entries, where 50% of configurations are collided or close to collision, meaning that $\exists k : d_k(\boldsymbol{q},\boldsymbol{y}) \leq 1\text{cm}$, and 50% are configurations with a minimal distance exceeding 1cm, such that $\forall k, d_k(\boldsymbol{q},\boldsymbol{y}) > 1\text{cm}$. Such balancing allows for a better approximation of the null iso-surface, which represents the exact geometry and surface of the robot's body. The collided half of the dataset is balanced to include collisions for links in equal proportions. Non-colliding data points are distributed uniformly between 1cm and

Table 4.1: A comparison of various layer sizes N and hidden layers amount D. Each architecture is trained on the same data for 10,000 epochs. The RMSE and standard deviation averaged between all links (and across five training instances) are provided for the 200k testing dataset.

N/D	**3**	**5**	**7**	**9**
64	4.88 (3.61)	3.63 (2.67)	3.50 (2.48)	**3.23 (2.22)**
128	3.43 (2.53)	2.38 (1.75)	**2.33 (1.62)**	2.47 (1.67)
256	2.74 (2.09)	**1.89 (1.38)**	2.07 (1.50)	2.07 (1.40)
512	2.71 (2.06)	**1.71 (1.23)**	1.77 (1.27)	1.96 (1.34)

100cm from collision. Another set of one million data points with non-modified joint limits is additionally collected for model testing and evaluation purposes.

Implementation: We compute distances between points and triangulated surfaces with code by Frisch (2022) in combination with the fast collision checking library (FCL) (Pan and Manocha, 2016) to calculate the signed distance between the mesh of the robot and the point in the workspace. The data collection procedure takes 90 minutes on a 12-core 3.7GHz CPU.

4.4.3 Network Architecture

Various network architecture choices were investigated to achieve the optimal performance. A simple fully-connected MLP was used as the baseline. The rectified linear activation function (ReLU) is used to provide faster forward and backward passes, and the root mean square error (RMSE) is used as the loss function.

The function $\mathbf{\Gamma}(\boldsymbol{q}, \boldsymbol{y})$ implicitly learns the robot's forward kinematics; thus, it might be helpful to build a feature vector as a concatenation of joint angles and their corresponding sine and cosine values: $\boldsymbol{q}_{in} = [q, cos(\boldsymbol{q}), sin(\boldsymbol{q})]$, where sin and cos are applied elementwise. Similar *positional encoding (p.e.)* is discussed in the original NeRF paper by Mildenhall et al. (2020), in which the authors conclude that it helps with learning of high-frequency features. Another perspective on the positional encoding is that it introduces nonlinear features frequently appearing in analytic FK equations, simplifying the regression task for NNs. To choose the optimal network parameters (number of layers and their size) for the MLP with positional encoding feature vector we performed a grid-search training with different depths, D, and layer sizes, N. For each pair of N and D, the network was trained on 2M training data points with different initial weights five times. The resulting average RMSE and its standard deviation are reported for 200k testing data points in Table 4.1. As can be seen, increasing the value of N improves the regression accuracy; however, as the network size is proportional to $\mathcal{O}(N^2 D)$ complexity, the smaller value $N = 256$ is chosen. For larger values of D, within the same amount of epochs, MLP fails to improve upon shallower networks (for $N > 128$). The chosen architecture used had $D = 5$ and $N = 256$.

The behavior of the RSME as we increase the depth of the networks in Table 4.1 suggests

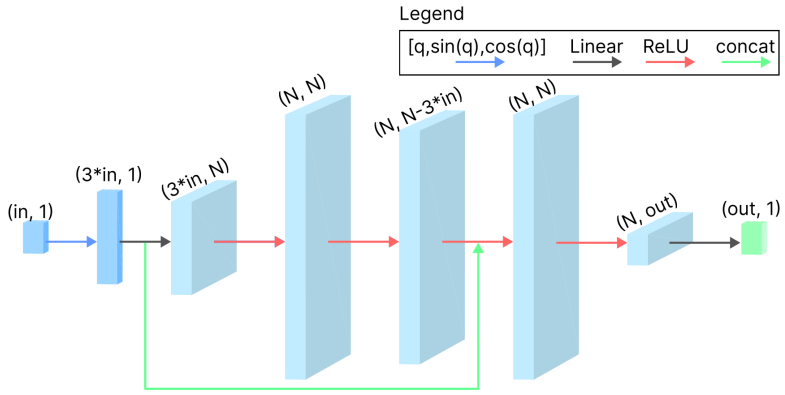


Figure 4.1: NN Architecture for the $\mathbf{\Gamma}(\boldsymbol{q}, \boldsymbol{y})$ function, featuring positional encoding and skip-connection option.

Table 4.2: A comparison of network architectures. Each trained on the same data for 10,000 epochs. The network parameters are $D = 5$ and $N = 256$. The results are averaged between ten trainings. p-value indicates the result of two-sample t-test run against first row of the table.

Architecture	Mean RMSE, cm	p-value
MLP	2.25 (1.66)	1.0000
MLP *(s.c.)*	2.28 (1.65)	0.9331
MLP *(p.e.)*	**1.89 (1.38)**	0.0002
MLP *(s.c + p.e.)*	1.93 (1.43)	0.0013

that we have a degradation problem; i.e., either we are over-fitting or we have vanishing and/or exploding gradients. One way to alleviate this is to introduce *skip-connections (s.c.)* between the input features and deeper layers of the network, essentially convexifying the training loss function. Apart from combating the vanishing/exploding gradients problem, we hypothesized that reintroducing the trigonometrical input features to deeper layers could yield better FK approximation. An example of the network with skip-connection between input and the fourth layer is provided in Fig. 4.1. Hence, we performed a further analysis of the MLP with and without p.e. and s.c. as shown in Table 4.2. Each architecture type is trained ten times, and the resulting RMSE distributions are studied by means of two-sample t-test. Each distribution is compared with the basic MLP (as in first row of Table 4.2), and resulting p-value is reported. The p-value indicates the likelihood of two distributions to be the same ($p = 1$) or to be different from each other ($p = 0$). For skip-connection $p = 0.93$, indicating that this feature is unlikely to improve the learning. The table shows that positional encoding leads to better distance approximation, while skip-connection does not bring significant improvements. Based on these findings, a simple MLP with $D = 5$ and $N = 256$ along with a positional encoding for the regression task is used. This network is similar to Fig. 4.1 but without the skip-connection. There, $in = 10$, with 7 for DoF and 3 for the workspace point position, and $out = 9$ representing d_k for $k = 1..9$ links.

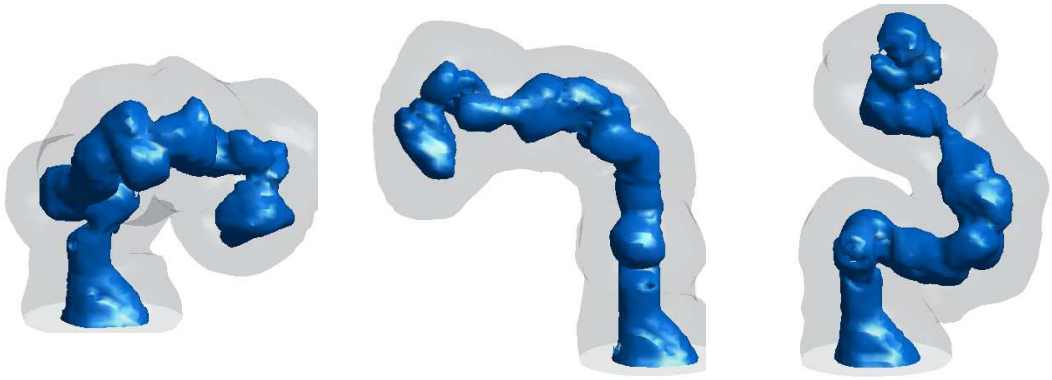


Figure 4.2: Illustration of the learned implicit distance isosurfaces $\mathbf{\Gamma}(\boldsymbol{q}, \boldsymbol{y}) = 0$ cm (solid) and $\mathbf{\Gamma}(\boldsymbol{q}, \boldsymbol{y}) = 10$ cm (transparent) for various configurations of the 7 DoF (Degrees-of-Freedom) Franka Emika Panda robot.

4.4.4 Learning Results

After selecting the hyperparameters, the NN was trained for 100,000 epochs, taking approximately 8h on RTX3090.[1]

Accuracy: Averaging between all links, $\mathbf{\Gamma}(\boldsymbol{q}, \boldsymbol{y})$ predicts minimum distances with an RMSE of 1.05 cm and a standard deviation of 0.81 cm. While the problem is posed as a regression, the ability of the learned model to distinguish between collided and free configurations, i.e., accuracy of predicting sign($\mathbf{\Gamma}(\boldsymbol{q}, \boldsymbol{y})$), was also investigated for configurations with $d_{min} < 3$cm. For such points, the classification accuracy of sign($\mathbf{\Gamma}(\boldsymbol{q}, \boldsymbol{y})$) is 0.90 when averaged between links. The performance of $\mathbf{\Gamma}(\boldsymbol{q}, \boldsymbol{y})$ in terms of RMSE and its standard deviation for each robot link is reported in Table 4.3. Fig. 4.2 shows iso-surfaces for different values of the learned function $\mathbf{\Gamma}(\boldsymbol{q}, \boldsymbol{y})$.

Computation Time: The computational performance of the learned function is investigated, and the results are presented in Table 4.4. It shows that batching improves the per-sample performance, which is expected with NN. Additionally, it shows that for a single sample, computations are more efficient on CPU. For reference, authors of (Montaut et al., 2022) report that one distance query between two convex hulls takes $0.6\mu s$ using standard Gilbert–Johnson–Keerthi (GJK) algorithm (Gilbert et al., 1988). The learned network calculates distances between $K = 9$ links and a single sphere, i.e., equivalent to performing nine such queries. Thus, if we assume that GJK is not parallelized, we can use $5.4\mu s$ as a baseline for a single robot-obstacle distance evaluation. If the control algorithm can benefit from batched distance queries (e.g. it is sampling based, or there are multiple obstacles), the NN can be up to 1000x faster than the standard GJK algorithm used to compute distances.

[1] The source code and supplementary video is available at `https://github.com/epfl-lasa/Neural-JSDF`

Table 4.3: Performance of learned function $\mathbf{\Gamma}(\boldsymbol{q}, \boldsymbol{y})$ in terms of RMSE and its standard deviation for close query points with $d_k \leq 10cm$, and far ones with $d_k > 10cm$, and in terms of the accuracy a_{coll} of predicting collisions, represented as sign($\mathbf{\Gamma}(\boldsymbol{q}, \boldsymbol{y})$) for configurations with $d_{min} < 3cm$.

Link, k	a_{coll}	**Close RMSE, cm**	**Far RMSE, cm**
1	0.98	0.37 (0.24)	0.40 (0.27)
2	0.99	0.32 (0.20)	0.43 (0.28)
3	0.98	0.38 (0.24)	0.46 (0.30)
4	0.97	0.56 (0.37)	0.57 (0.39)
5	0.93	0.73 (0.51)	0.68 (0.46)
6	0.88	1.02 (0.69)	1.06 (0.74)
7	0.87	1.29 (0.85)	1.25 (0.86)
8	0.83	1.40 (0.89)	1.46 (1.00)
9	0.76	1.83 (1.17)	2.06 (1.37)
Avg	0.90	1.04 (0.78)	1.06 (0.82)

Table 4.4: Computational performance of the learned function. All results are averaged on 10k runs with 12-core 3.7GHz CPU and RTX3090 GPU.

Batch size	**CPU**		**GPU**	
	Batch time, μs	**Sample time,** μs	**Batch time,** μs	**Sample time,** μs
1	32.141	32.141	74.340	74.340
10	115.998	11.600	79.148	7.915
100	426.743	4.267	98.689	0.987
1000	2341.552	2.342	103.147	0.103
10000	22734.501	2.273	281.336	0.028
100000	314058.185	3.141	2467.827	0.025

4.4.5 Prediction Errors Analysis for Learned Distance Function

In the previous subsection, we reported the RMSE for the learned distances. On average, the learned Neural Network predicts distances with an error of approximately 2 cm, with the RMSE increasing for robot links further down the kinematic chain. However, the averaged error does not capture all aspects of inaccuracies in the approximated distances. To provide a more comprehensive study of the learned function, we investigate the distribution of RMSE for points at different distances from the robot.

We uniformly and randomly sample 1 million model inputs in a 10-dimensional space (7 degrees of freedom for the robot and 3 degrees of freedom for the workspace points) and compare the Neural Network output with the ground truth. This additional model performance investigation uses a testing dataset that is different from the one used in the previous section, and it does not employ any balancing heuristics.

Figure 4.3 reports the distribution of the predicted distances by the Neural Network for various

distances between the robot and the query point. On average, the L1 error is always below 2 cm. However, it is important to note that at any scale, there exist adversarial query inputs that result in higher prediction errors. This effect becomes more pronounced for links further from the base. Specifically, for the first five links, even the outliers do not exceed an error of 5 cm. However, for the last link, there are inputs that lead to Neural Network prediction errors of up to 15 cm.

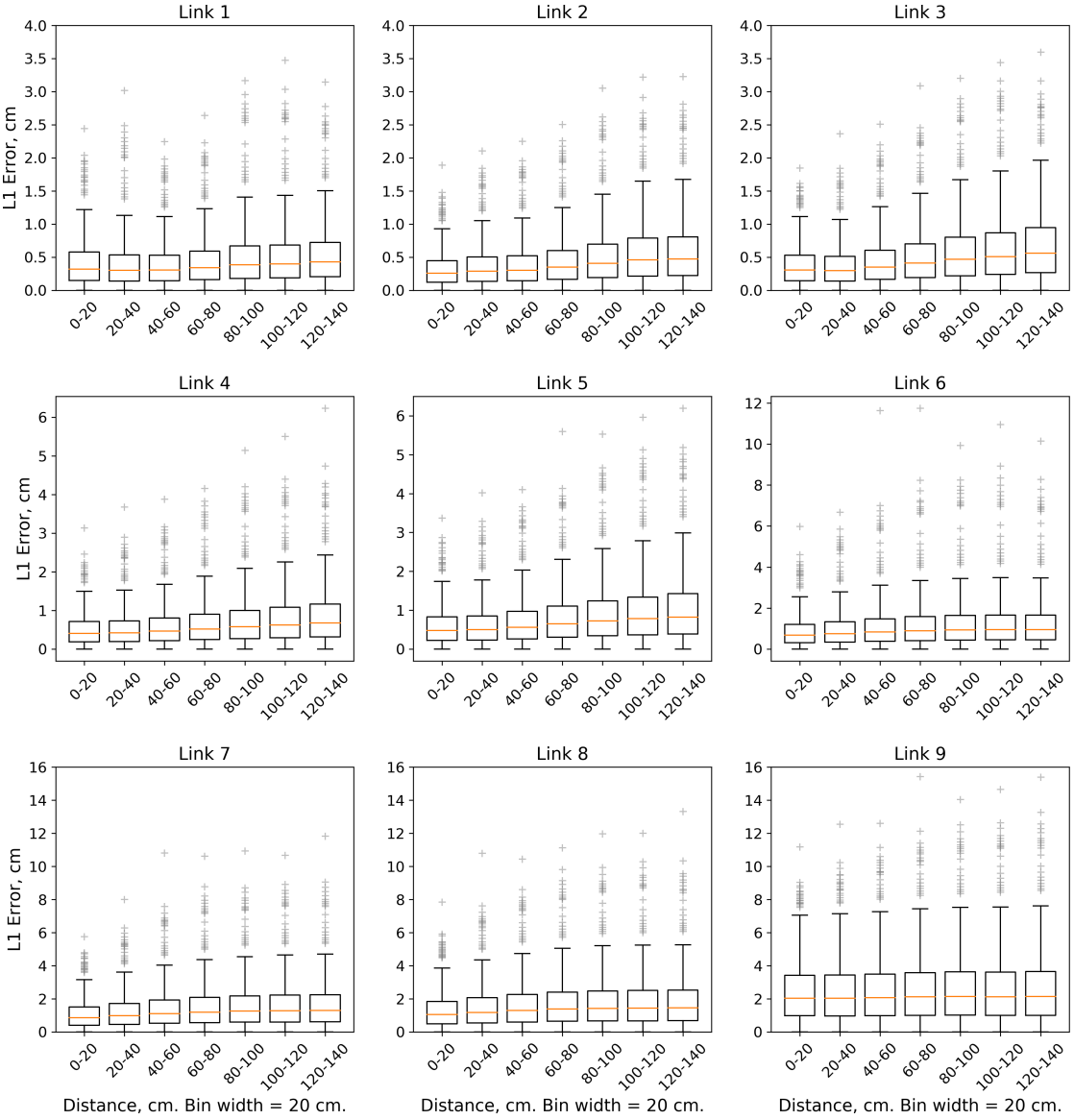


Figure 4.3: Error distribution for different distances between the robot and the workspace point. The nine subplots correspond to the nine links of the robot. Boxplots are presented for binned distances between the robot and the workspace point, with a bin width of 20 cm.

For a deeper exploration, we turn our attention to outliers, indicated by the gray crosses in Figure 4.3. On average, each link has approximately 20,000 outliers (out of 1M testing data).

Our investigation focuses on their distribution in relation to the boundaries of our sampled data. We propose a hypothesis: as a coordinate approaches the limits of our sampled data, the precision of the Neural Network prediction correspondingly diminishes.

To illustrate this, we plot a histogram showing the distribution of outliers based on their proximity to normalized sampling boundaries. For instance, if a robot joint state coordinate approaches q_{min}, the associated closeness value hovers near 0. Conversely, if the robot aligns with q_{max} on at least one joint angle, the corresponding joint limit closeness becomes 1. This concept also extends to workspace points, where values of 0 or 1 imply proximity to sampling boundaries (which are $\pm$ 1m from origin), while a value of 0.5 suggests the point is near the robot base.

We present these results in Figure 4.4. The data demonstrates a concentration of outliers close to the sampling boundary, and it's evident that our model offers significantly better predictions when inputs are beyond 20% from joint (or workspace) limits.

In light of this analysis, we suggest that future models should be trained on a broader data range, expanding the joint limits by up to 20%, compared to the 5% limit used in this thesis.

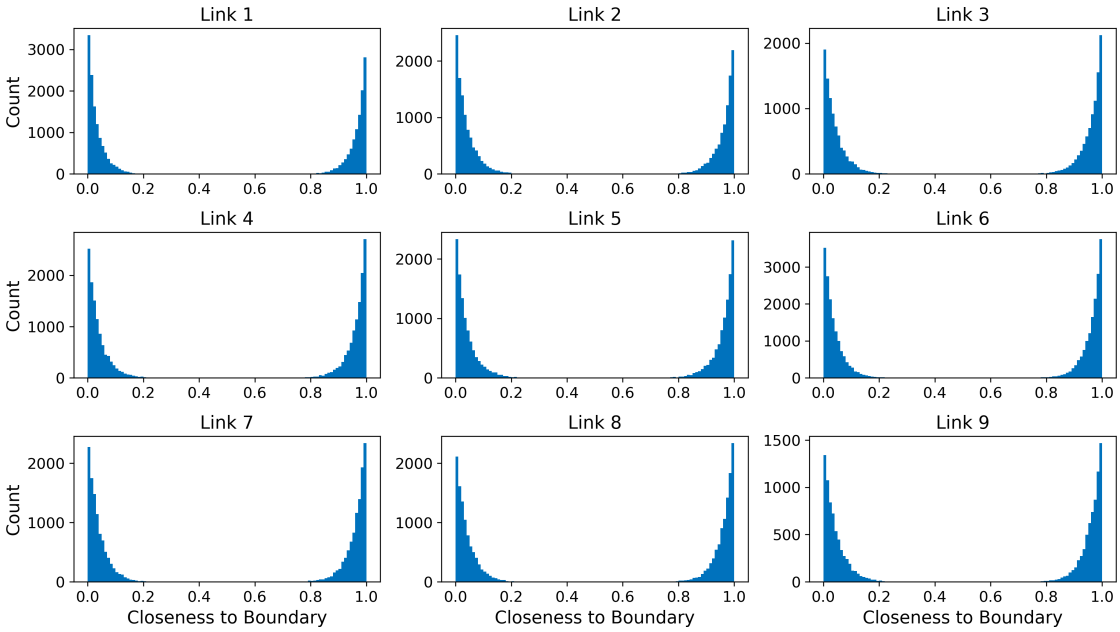


Figure 4.4: Histogram showing the distribution of outliers with respect to their proximity to the normalized sampling bounds. The x-axis denotes the closeness to the boundaries (0 signifies closeness to q_{min}, 1 to q_{max}, and 0.5 indicates neutral pose), while the y-axis represents the count of outliers.

4.5 Reactive Control with Neural-JSDF

As described previously, the learned function $\boldsymbol{\Gamma}(\boldsymbol{q}, \boldsymbol{y})$ approximates the distances between the robot's links in a given joint configuration, $\boldsymbol{q}$, and Euclidean point, $\boldsymbol{y}$, in its workspace. To treat collisions with moving, and possibly deforming obstacles of non-fixed shape and size we approximate the obstacle shape with $s = 1..S$ spheres with centers $\boldsymbol{y}_s$ and radii r_s. Note, point-cloud obstacle representations fit naturally in such framework, with $r_s = 0$, $\forall s$.

This section presents two approaches, i) QP-IK and ii) Sampling-based MPC, leveraging on various properties of the learned Neural-JSDF $\boldsymbol{\Gamma}(\boldsymbol{q}, \boldsymbol{y})$, such as repulsive gradient in joint space, and efficient batch input processing.

4.5.1 Reactive Collision-Avoidance IK

Similarly to the previous chapter, the learned function $\boldsymbol{\Gamma}(\boldsymbol{q}, \boldsymbol{y})$ can be used to formulate a constraint in a QP-IK solver:

$$\begin{aligned} &\min_{\Delta\boldsymbol{q}, \boldsymbol{\delta}} \boldsymbol{\delta}^T \boldsymbol{Q} \boldsymbol{\delta} + \Delta\boldsymbol{q}^T \boldsymbol{R} \Delta\boldsymbol{q} \\ &\text{s.t.} \quad \begin{cases} f(\boldsymbol{q}) + \dfrac{\partial f(\boldsymbol{q})}{\partial q} \Delta\boldsymbol{q} = \boldsymbol{x} + \boldsymbol{\delta} \\ q_i^- < q_i + \Delta q_i < q_i^+, \quad i = 1..m \\ -\nabla_{\boldsymbol{q}} \Gamma_k(\boldsymbol{q}, \boldsymbol{y}_s)^T \Delta\boldsymbol{q} \leq \ln(\Gamma_k(\boldsymbol{q}, \boldsymbol{y}_s) - r_s), \quad \forall k, s. \end{cases} \end{aligned} \tag{4.5}$$

In Equation 4.5, the goal is to minimize joint displacement $\Delta\boldsymbol{q}$ with damping term $\boldsymbol{R}$, to satisfy the kinematic constraints given by forward kinematics $f(\boldsymbol{q})$, Jacobian $\frac{\partial f(\boldsymbol{q})}{\partial \boldsymbol{q}}$ and cartesian task $\boldsymbol{x} \in \mathcal{SE}(3)$. These tasks may specify desired positions and orientations for chosen links, but further we assume they only contain tasks for the end-effector. Additionally, for situations where the solver fails to satisfy the reaching constraint, slacks $\boldsymbol{\delta}$ are introduced and minimized with corresponding weights $\boldsymbol{Q}$. The second constraint accounts for joint-limits, which are defined as q_i^- and q_i^+ for $i = 1..m$ joints.

Finally, the last line in (4.5) defines the proposed collision avoidance behaviour with the learned implicit signed distance function. It represents $K \times S$ constraints, one for each pair of $k = 1..K$ links and $s = 1..S$ spherical obstacles. $\Gamma_k(\boldsymbol{q}, \boldsymbol{y}_s)$ is the k-th component of the implicit distance function output vector, for $k = 1..K$ links. Finally, $\boldsymbol{y}_s$ are centers of spherical obstacles $s = 1..S$ with radii r_s. When the robot is far from collisions and $\Gamma_k(\boldsymbol{q}, \boldsymbol{y}_s) > 0, \forall k$, the set of constraints is relaxed. If the k-th link is close to the s-th obstacle, then $\ln(\Gamma_k(\boldsymbol{q}, \boldsymbol{y}_s) - r_s)$ becomes negative, and the IK solver is forced to align joint motion $\Delta\boldsymbol{q}$ with the corresponding gradient $\nabla_{\boldsymbol{q}} \Gamma_k(\boldsymbol{q}, \boldsymbol{y}_s)$, repelling the robot away from the collision boundary and satisfying collision avoidance constraint.

4.5.2 Sampling-Based Model Predictive Control

Additionally, we propose to use the learned Neural-JSDF in a more complex algorithm with a look-ahead horizon to address the local nature of the QP controller described in the previous section. Recent work (Bhardwaj et al., 2022) demonstrates sampling-based MPC for robotic manipulators leveraging on massive parallelization via a GPU to speed up FK and cost function computations. The learned function $\mathbf{\Gamma}(\boldsymbol{q}, \boldsymbol{y})$ fits naturally in such framework allowing for efficient batched collision checking. Below, the basics of Model Predictive Path Integral (MPPI) control are introduced.

For a discrete-time system at time t, the robot is controlled by a joint space acceleration command $\boldsymbol{u}_t$. This command is sampled from a policy $\pi_t = \Pi_{h=1}^{H} \pi_{t,h}$, where H is a look-ahead horizon, and policies $\pi_{t,h}$ are simple Gaussians defined by means $\boldsymbol{\mu}_{t,1}, ..., \boldsymbol{\mu}_{t,H}$ and covariances $\Sigma_{t,1}, ..., \Sigma_{t,H}$. At every iteration the sampling-based MPC algorithm, proposed in (Bhardwaj et al., 2022), samples a batch $\{\boldsymbol{u}_{i,h}\}_{i=1..N}^{h=1..H}$ of N control sequences of length H from current distribution π_t. After that, the roll-out states $\{\boldsymbol{x}_{i,h}\}_{i=1..N}^{h=1..H}$ are computed using an approximate dynamics function. The corresponding costs, $\{c_{i,h}\}_{i=1..N}^{h=1..H}$, are calculated as a weighted sum of goal-reaching, joint-limit avoidance, contingency stopping, and self- and environmental-collision avoidance costs.

The means and covariances of the Gaussian policies, $\pi_{t,h} = \mathcal{N}(\boldsymbol{u}_{t,h} | \boldsymbol{\mu}_{t,h}, \Sigma_{t,h})$, are then updated using a sample-based gradient of a risk-seeking objective function. The update rule for $\boldsymbol{\mu}_{t,h}$ is:

$$\boldsymbol{\mu}_{t,h} = (1 - \alpha_{\boldsymbol{\mu}}) \boldsymbol{\mu}_{t-1,h} + \alpha_{\boldsymbol{\mu}} \frac{\sum_{i=1}^{N} w_i \boldsymbol{u}_{i,h}}{\sum_{i=1}^{N} w_i} \tag{4.6}$$

where $\alpha_{\boldsymbol{\mu}}$ is a filtering coefficient, and

$$w_i = \exp \frac{-1}{\beta} \left(\sum_{h=1}^{H-1} \gamma^h c(\boldsymbol{x}_{i,h}, \boldsymbol{u}_{i,h}) + \gamma^H \hat{c}(\boldsymbol{x}_{i,H}, \boldsymbol{u}_{i,H}) \right) \tag{4.7}$$

is a cost-based weighting coefficient for sampled trajectories. In Equation 4.7, $c(x, u)$ is a task-specific cost, $\hat{c}(x, u)$ is the terminal cost, and $\gamma \in [0, 1]$ is a discount factor to balance between immediate rewards and final goal. We refer the reader to (Williams et al., 2017; Bhardwaj et al., 2022) for a detailed derivation of the MPPI algorithm. There, authors prove the connection between such update rule and optimal solution of Hamilton-Jacobi Bellman equation. To some extent, this algorithm is again presented in Chapter 5, but with different optimization objectives and a different cost function.

The distinctive property of MPPI is that it can handle a wide range of cost formulations, including the following formulation (Bhardwaj et al., 2022):

$$\begin{aligned} c(\boldsymbol{x}, \boldsymbol{u}) = & \alpha_p c_{\text{pose}} + \alpha_s c_{\text{stop}} + \alpha_j c_{\text{joint}} + \\ & \alpha_m c_{\text{manip}} + \alpha_c (c_{\text{coll}} + c_{\text{self-coll}}), \end{aligned} \tag{4.8}$$

where goal-reaching, contingency stopping, joint limit avoidance, manipulability and collision costs are combined with respective weights. In Equation 4.8, the collision cost, c_{coll}, is discrete, yielding $c_{\text{coll}} = 1$ if the robot is in collision with the environment, and $c_{\text{coll}} = 0$ otherwise. We use $\text{sign}(\boldsymbol{\Gamma}(\boldsymbol{q}, \boldsymbol{y}))$ to efficiently perform batched collision queries for computing the cost function.

4.6 Evaluation in Simulation and on Real Robot

4.6.1 Evaluation scenarios and metrics

We evaluate the performance of the reactive control methods introduced in the previous section (QP-IK and MPPI) using the learned Neural-JSDF for collision avoidance in two scenarios of different complexity. In both cases, the robot's goal is to perform a reaching motion from initial configuration, to the goal defined in the task space.

Scenarios: Scenario (A) includes two disjoint spherical obstacles in front of the robot. Scenario (B) imitates human presence in the robot's workspace (refer to Fig. 4.5). Human body is approximated with 30 spheres of varying radii, fitting the torso, head, and right arm. Both control methods are expected to work with at least 50Hz frequency, providing reactive behavior for avoiding moving obstacles. However, as dynamic information (such as obstacles velocity) is not directly incorporated into avoidance constraints, these methods are tested in a quasi-static environment. Additionally, that helps to achieve reproducible results. The benchmark consists of 100 experiments for randomized positions of collision obstacles. For scenario A the vertical placement of the obstacles differs between the experiments, while scenario B simulates the human with different arm positions, obstructing the robot's workspace. The robot's initial configuration and goal position are constant across the experiments.

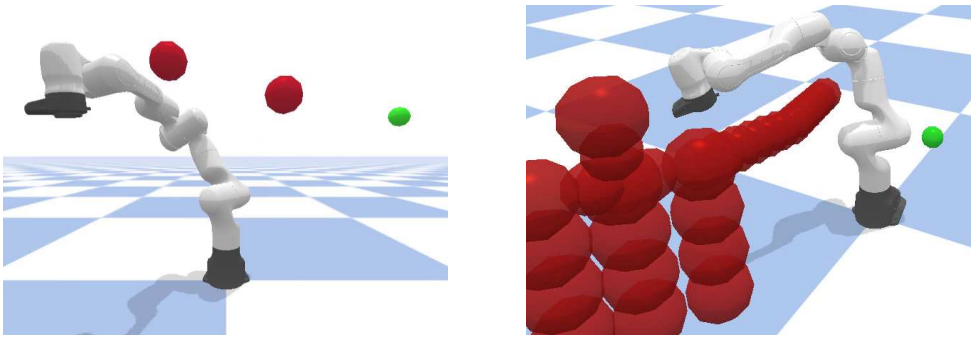


Figure 4.5: Benchmark scenarios for the reaching task. Goal is depicted as a green sphere, obstacles are represented with red spheres. (left) Scenario A, where spheres placement is different across the experiments, (right) Scenario B, human shape approximated with 30 spheres, with variable arm placement.

	QP(Sph)	QP(NN)	MPC(Sph)	MPC(NN)
Success rate, %	1.00	1.00	1.00	1.00
Reaching time, s	4.53	3.24	4.17	2.53
Clearance, cm	1.32	1.14	0.97	0.84
Frequency, Hz	217	249	66	92

Table 4.5: Performance comparison for Scenario A (two disjoint spheres).

	QP(Sph)	QP(NN)	MPC(Sph)	MPC(NN)
Success rate, %	0.91	0.92	1.00	1.00
Reaching time, s	5.76	4.09	4.30	3.17
Clearance, cm	1.24	1.21	0.86	0.82
Frequency, Hz	204	220	49	68

Table 4.6: Performance comparison for Scenario B (human-shaped obstacle approximated with 30 spheres).

Metrics: The results are provided in Tables 4.5 and 4.6. There, success rate indicates a number of experiments with successful goal reaching (i.e. no collisions and free from local minima), and clearance stands for average minimal distance-to-collision across successful trajectories. Both methods were tuned to behave conservatively and not allow for any collisions. Overall, QP is more reactive than MPC, but due to QP's local nature and lack of optimization horizon, it has a lower success rate in a more complex scenario.

4.6.2 Implementation details and discussion

Baseline Approach (Spheres Approximation)

Links of the robot are approximated with 55 spheres of various radii as shown in Fig. 4.6. This number of spheres and their radii were chosen to best represent the actual geometry of the robot without being overly conservative. For simplicity, p_k and r_k are used to denote a sphere belonging to the $k-$th link of the robot. For each obstacle sphere with a center $\boldsymbol{y}_s$, and a radius r_s, the closest sphere, (p_k, r_k), belonging to the robot can be found. The distance between the robot and the obstacle is then $d_{k,s} = ||\boldsymbol{y}_s - p_k|| - r_s - r_k$, and $\boldsymbol{v}_{k,s}^{rep}$ is a vector connecting two spheres, that acts as a repulsion in task space. For the QP-IK, collision avoidance constraints in Equation 4.5 are then replaced with the following:

$$(\boldsymbol{J}_k^{rep}(\boldsymbol{q})\Delta\boldsymbol{q})^T \boldsymbol{v}_{k,s}^{rep} \leq \ln(d_{k,s}), \tag{4.9}$$

where $\boldsymbol{J}_{k,s}^{rep}(\boldsymbol{q})$ is a Jacobian for the corresponding point on the robot body, and $\boldsymbol{v}_{k,s}^{rep}$ is scaled to match the magnitude of $\Delta\boldsymbol{q}$. These constraints cover all pairs of links and obstacles, forcing the solver to generate motion in a tangential plane to the obstacles, when the distance to collision $d_{k,s}$ is small.

Essentially, our method, as described in Equation 4.5, formulates collision avoidance constraints directly in high-dimensional joint space, with the corresponding tangent plane belonging to that space. In contrast, the above-formulated baseline in Equation 4.9 uses the Jacobian to approximate forward kinematics and avoid collisions using the tangent plane but in the task space $\mathbb{R}^3$.

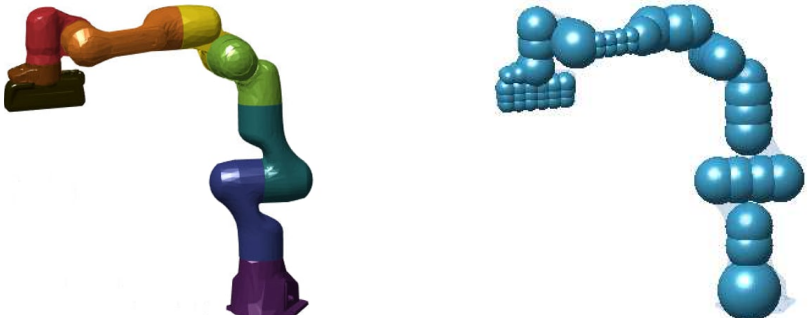


Figure 4.6: (left) Mesh representation of the Franka Emika, coloured segments represent the $K = 9$ links, and (right) sphere approximation (with $S = 55$) of the robot geometry.

QP-IK

CVXGEN (Mattingley and Boyd, 2012) is used to solve the described QP problem. The values of $\boldsymbol{\Gamma}$ and $\nabla_{\boldsymbol{q}}\boldsymbol{\Gamma}$ are evaluated on the 12-core 3.7Ghz CPU and passed to the solver. We evaluate the performance of the proposed collision avoidance IK solver using the learned Neural-JSDF $\boldsymbol{\Gamma}(\boldsymbol{q}, \boldsymbol{y})$ compared to formulating the collision avoidance constraint with the traditional method to detect collisions in task-space. Our method efficiently combines the learned function $\boldsymbol{\Gamma}(\boldsymbol{q}, \boldsymbol{y})$ and its gradient to create the repulsive force around obstacles, showing a 15% increase of computation speed for the IK solver over using the baseline technique for collision detection. However, in general, QP approach is prone to local minima, especially in case of non-convex obstacles, demonstrating worse success rate in Scenario B. It could be useful as a collision-aware IK solver but it requires additional high-level planning for more complex scenarios.

MPPI

While the authors of (Bhardwaj et al., 2022) claim that the operating frequency of their MPPI implementation exceeds 100 Hz, they clarify that this is the case only for a static environment with precomputed distance map. Assuming that obstacles are not fixed, and since it is impossible to pre-compute and store the scene distances, then the distance evaluations must be repeated at each iteration. Since the sampling and rollouts are performed in joint space, the use of our learned function is justified. At each iteration, it is required to calculate distances between the environment and $N \cdot H$ robot configurations. The input for the

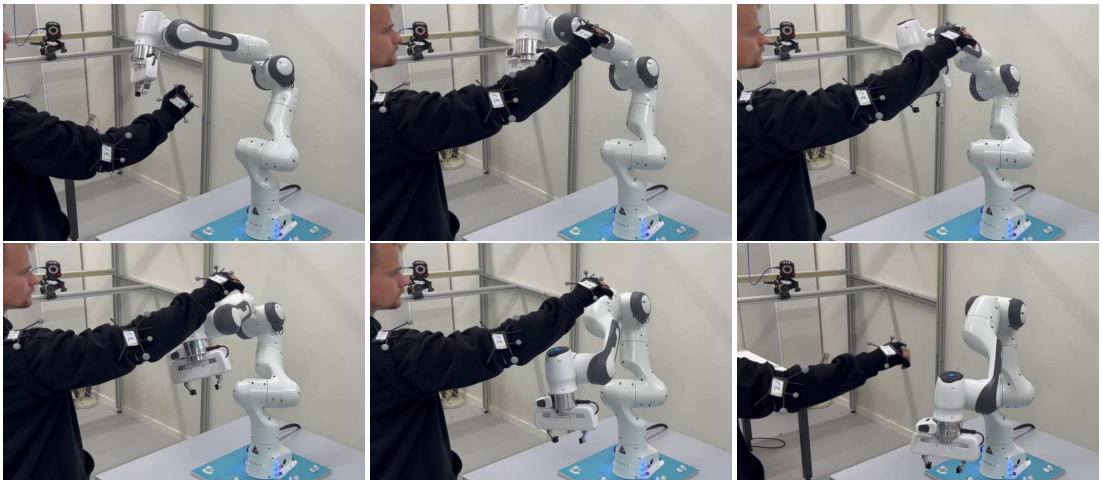


Figure 4.7: Snapshots of a reaching task with the human upper body as an obstacle approximated with a collection of 30 spheres.

learned function $\mathbf{\Gamma}(\boldsymbol{q}, \boldsymbol{y})$ is repeated S times, once for each spherical obstacle position; thus, a fall in performance is expected with the increase in the number of spherical obstacles. As a baseline to compare with, we again choose to approximate the robot body with spheres and evaluate collisions by calculating task-space distances between these spheres and spherical obstacles in the robot's workspace. Such baseline would seem computationally cheaper than NN evaluation; however, a highly parallelized forward pass of the neural network turns out to be faster. Tables 4.5 and 4.6 show the comparison between the naive distance computation between obstacle spheres and robot geometry approximated with spheres. All experiments were conducted on a 12-core 3.7 GHz CPU and RTX3090 GPU. For the case when there are $S = 2$ obstacles in the workspace, our approach shows significantly faster iterations at 92 Hz versus 66 Hz. On average, distance computation using the learned function $\mathbf{\Gamma}(\boldsymbol{q}, \boldsymbol{y})$ is 40% faster than the baseline. Additionally, Fig. 4.8 demonstrates how this controller frequency scales with the value S. This is showcased in the *multimedia attachment* where we present multiple simulations of both scenarios with increasing obstacle speeds and adversarial behaviors.

Comparison

As shown in the evaluations reported in Tables 4.5 and 4.6 and Figure 4.8, both reactive control approaches perform well in terms of avoiding collisions using the learned Neural- JSDF. While QP-IK is faster and exhibits more reactive behavior, it may get stuck in local minima in the optimization, or fail to recover from an odd joint configuration caused by the instantaneous reactivity as shown for both scenarios in the *multimedia attachment.* On the other hand, while the MPC is slower than QP-IK, it manages to avoid the problematic regions where obstacles are moving very fast, escaping oscillatory reactive behaviors exhibited by the QP-IK and ultimately reaching the target.

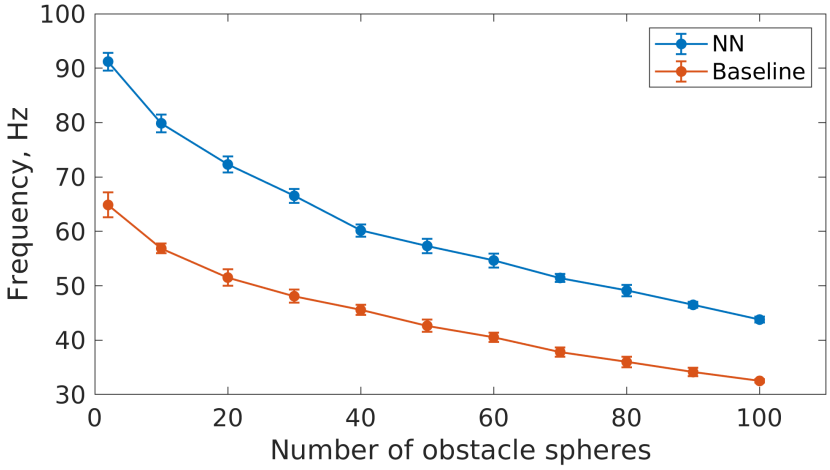


Figure 4.8: The MPPI frequency (Hz) and variance as a function of number of spherical obstacles, S, in the workspace of the robot. The blue line is for collision checking via learned function and the red is for the baseline with spherical approximation of the robot. Values averaged on 10 runs.

Real Robot Experiment

We further validated the approach on a real Franka Emika Panda being controlled at 1kHz by a custom low-level torque controller, similar to (Bhardwaj et al., 2022). Obstacles and the goal are tracked with OptiTrack at a 120Hz rate. We replicate scenarios A and B and stress test the system with adversarial obstacle motions as shown in the *multimedia attachment*. Fig. 4.7 demonstrates a successful reaching task while avoiding colliding with the human.

4.7 Conclusion

This chapter presented our method for learning the minimal distances between the robot and its environment as a neural implicit distance function. This distance is a function of the robot joint state and the coordinates of a point in its workspace. We can efficiently compute distances between the robot in arbitrary configuration and obstacles represented as a set of spheres of various radii. The gradient of the learned function with respect to inputs can be treated as the repulsive vector in the joint space of the robot, allowing for collision avoidance constraint formulation in the QP Inverse Kinematics solver. While the learned network contains many parameters, due to high parallelization, it still outperforms the naive baseline in terms of controller frequency. This property allows efficient use of the learned function as a collision-checker in sampling-based MPC control.

While we have investigated scenarios where obstacles in the robot's workspace are approximated with spheres, this method can also be used with obstacles represented as point clouds. We leave corresponding implementations for future work. Another interesting direction would

be to expand the use of repulsion gradients and apply them as a heuristic to guide the sampling to further improve the performance of the MPPI approach, similar to (Kew et al., 2020). The distance-to-collision and cosine similarity between the sampled acceleration and repulsion direction could be used to introduce re-projected samples that navigate the robot around the obstacle.

While this thesis focuses on quasi-static scenarios, it is worth noting that the derivative of a learned implicit distance function with respect to the Euclidean point position yields a repulsive vector in the task space. This vector can be used in motion planning that considers obstacle dynamics and, when compared with the obstacle velocity vector, may determine whether an obstacle is approaching or moving away from the robot. This information can be utilized to further improve the performance of the motion planner.

5 Collision-Free Motion Generation

The whole is greater than the sum of its parts.
— Aristotle

5.1 Foreword

In previous chapters, we introduced a framework for learning self-collisions in redundant robots and implicit distance functions, which provide signed distance field approximations and enable distance calculations between the robot and points within its workspace. Our approach takes robot joint positions as input to the learned model, and the output consists of the distance-to-collision. Since these models are differentiable, either analytically or through autograd, we can obtain a joint-space vector representing the repulsion direction. This vector is valuable as it can be employed to formulate collision avoidance constraints in various motion generation algorithms. To evaluate the performance of the learned models, we applied them to different control schemes, such as QP-IK for generating collision-free motions and sampling-based MPC.

We demonstrated that the learned models can efficiently detect collisions and perform comparably or better than traditional collision-checking methods. However, we also identified the limitations of the motion generation algorithms employed. Sequential QP-IK is susceptible to local minima since it generates motion incrementally for one step at a time. Consequently, the robot may become stuck in specific configurations, e.g., when trapped near a highly concave obstacle. Sampling-based MPC (STORM, MPPI) is more robust due to its look-ahead horizon, enabling it to plan exit strategies in cluttered environments. However, the method's sampling procedure does not utilize potential heuristics provided by the learned models, which could improve its convergence properties. Although the MPC method's frequency is close to 100 Hz, we observed in numerous experiments that it is not highly reactive and cannot guarantee rapid collision avoidance. Therefore, our next contribution in this thesis involves developing a motion generation algorithm capable of generating reactive collision-free motions while

leveraging the learned models to enhance the method's convergence properties.

In this chapter, we introduce a hybrid controller that combines Modulated DS and sampling-based MPC approaches. DS offers highly reactive lower-level control, generating collision-free motions at frequencies close to 1 kHz. Since this method cannot guarantee goal-reaching in complex environments, we combine it with sampling-based exploration, which generates escape strategies for situations where the robot is stuck in local minima. Both control levels (DS and MPC) rely on learned collision-checking models to estimate distances and apply repulsive gradient heuristics.

5.2 Introduction

Modern robotic systems are expected to operate in a variety of environments and interact with static and dynamic objects, as well as other agents, such as humans, animals, and other robots. While direct contact may be necessary to accomplish certain tasks, almost any motion requires a reaching phase where the robot must move its end-effector to a desired location while avoiding collisions between the robot body and obstacles or and other agents. For future reference, we assume that the obstacles in the environment are moving unpredictably, and reactive motion assumes feasible plan generation on frequencies above 100 Hz.

Techniques to generate collision-free motion can be categorized into two paradigms: offline and feedback/reactive planning (LaValle, 2006). In the former, the problem is divided into two phases - offline planning of collision-free paths (typically sampling-based algorithms or trajectory optimization) followed by trajectory execution (Spong et al., 2020). However, depending on the complexity of the environment and the dimensionality of the robot configuration space, offline planning may take a significant amount of time, which hinders operation in dynamic environments with moving obstacles and active agents.

Early works in reactive collision-free motion generation rely on local alternation of the system's dynamics in the vicinity of obstacles via Artificial Potential Fields (Khatib, 1986) or navigation functions (Rimon and Koditschek, 1992). Such classical feedback motion planning techniques, although reactive, are vulnerable to local minima and limited to parametric obstacle representations such as convex (spheres, ellipsoids, etc) or star-shaped obstacles. To ensure reactive collision-free motions with convergence guarantees the modulated dynamical system (DS) based approach was introduced in (Khansari-Zadeh and Billard, 2012a). In this approach, the nominal robot motion is generated by a DS that defines a vector field in the task space for end-effector motion (in this case, an additional Inverse Kinematics controller is required) or in joint space to describe the whole-body state-dependent motion law. The proposed modulation locally changes this law in order to navigate around obstacles.

Local modulation of DS for obstacle avoidance is performed with relatively low computation cost, however, convergence guarantees can only be ensured for conservative obstacle types such as convex (Khansari-Zadeh and Billard, 2012a) or star-shaped (Huber et al., 2019; Billard

et al., 2022). Further, a saddle-point local minimum trajectory still remains for such parametric shapes. These shortcomings limit the applicability of reactive DS modulation approaches to complex spaces with arbitrarily shaped obstacles that are hard to convexify such as joint space of redundant manipulators.

Receding horizon path planning approaches such as Model Predictive Control (MPC), can be used to generate collision-free trajectories in real-time (Erez et al., 2013; White et al., 2022). However, these methods are often computationally expensive, and finding a balance between computation speed and trajectory quality can be challenging (Bhardwaj et al., 2022). The computational complexity of MPC methods grows cubically with state dimensionality and time horizon (Richter et al., 2012), which can be prohibitive for cases when the robot with a high number of degrees of freedom (DoF) is required to reactively navigate in cluttered environments while avoiding obstacles.

5.2.1 Contributions

In this chapter, we aim to combine the advantages of DS motion planning and sampling-based MPC to allow for instantaneous obstacle avoidance and swift feasible trajectory generation for any type of environment. We adopt a modulated DS approach (Khansari-Zadeh and Billard, 2012a; Billard et al., 2022) as a primary motion generator and empower it with a popular sampling-based MPC algorithm known as Model Predictive Path Integral (MPPI) control (Williams et al., 2017), to navigate the robot away from the local minima caused by concave obstacles. We prove theoretically that the impenetrability and local convergence properties of the modulated DS are preserved with the MPC-based additive terms. Additionally, we focus on the challenging problem of joint space control and validate the proposed method using a 7-DoF robot. Example of a reaching task executed by the robot is shown in Figure 5.2. Apart from presenting a simulated benchmark, we demonstrate the collision avoidance controller running at 500 Hz (on a modern laptop CPU) with a real robot, generating reaching motions while avoiding the human that tries to obstruct the robot reaching motion with a concave arm trap.

5.2.2 Chapter organization

Section 5.3 presents the mathematical problem formulation, including assumptions, definitions, and goals of this chapter. In Section 5.4, we begin by presenting the background of the DS modulation method, as it is the basis of our approach. Furthermore, Sections 5.4 and 5.6 cover the proposed approach for modulating DS with MPC. Finally, Section 5.7 presents the validation of the method in simulation and on a real robot.

5.3 Problem definition

5.3.1 Assumptions & Definitions

We consider a robot with K physical links and d degrees of freedom (DoF). The joint state of the robot is defined as a vector of joint angles $\boldsymbol{q} = [q^1, ..., q^d] \in \mathbb{R}^d$. Additionally, it is assumed that all the joints of the robot are revolute and are subject to joint limits, $\boldsymbol{q} \in [\boldsymbol{q}_{min}, \boldsymbol{q}_{max}]$. The desired robot motion is defined by a continuous time-invariant nominal DS of the following form:

$$\dot{\boldsymbol{q}} = \boldsymbol{f}(\boldsymbol{q}), \tag{5.1}$$

which defines a state-dependent vector field guiding the robot state towards a goal (attractor $\boldsymbol{q}^* \in \mathbb{R}^d$). The function $\boldsymbol{f}(\boldsymbol{q})$ is autonomous, i.e., the law governing the evolution of the system depends solely on the system's current state $\boldsymbol{q}$ and not on time. While the system defined in (5.1) is time-invariant, the system state $\boldsymbol{q} = \boldsymbol{q}(t)$ changes over time. For better readability, time-dependency of the state variable is omitted in the remainder of the text.

We assume that the nominal DS $\boldsymbol{f}(\boldsymbol{q})$ is known and globally asymptotically stable (G.A.S.) to the attractor $\boldsymbol{q}^*$ wrt. to a Lyapunov function $V(\boldsymbol{q}, \boldsymbol{q}^*) : \mathbb{R}^d \rightarrow \mathbb{R}$. Such G.A.S. DS can be designed by user (Salehian et al., 2018a), or learned from a demonstration (LfD) (Khansari-Zadeh and Billard, 2011; Kronander et al., 2015; Figueroa and Billard, 2018; Shavit et al., 2018), and can be of arbitrary form (e.g. linear, nonlinear). While the nominal DS may implicitly avoid collisions in some scenarios, such as when an LfD-generated trajectory avoids static obstacles, it is not always guaranteed that it will do so in all scenarios. As shown in (Wang et al., 2022), DS-based LfD can ensure reachability to a target $\boldsymbol{q}^*$, but cannot ensure invariance to unsafe boundaries, unless a boundary function is explicitly defined and enforced through modulation or control-barrier functions (Taylor et al., 2020).

Obstacles (static or dynamic) can be defined as a set of points $\mathcal{O} \subset \mathbb{R}^3$ in the workspace of the robot. We assume that obstacles are known and are represented either by a point cloud or by a set of spheres. Moreover, we consider obstacles of arbitrary shape, including concave ones. Based on previous chapter, we define function $\boldsymbol{\Gamma}(\boldsymbol{q}, \boldsymbol{y}) = [\Gamma_1, ..., \Gamma_K]$, which represents minimal distances between point $\boldsymbol{y} \in \mathcal{O}$ and each link ($k = 1, ..., K$) of the robot in configuration $\boldsymbol{q}$. Function $\boldsymbol{\Gamma}(\boldsymbol{q}, \boldsymbol{y})$ is differentiable ($\mathcal{C}^2$), and its partial derivative with respect to $\boldsymbol{q}$ results in the following Jacobian matrix:

$$\mathrm{Jac}_q\big(\boldsymbol{\Gamma}(\boldsymbol{q}, \boldsymbol{y})\big) = \begin{bmatrix} \frac{\partial \Gamma_1(\boldsymbol{q}, \boldsymbol{y})}{\partial q_1} & \cdots & \frac{\partial \Gamma_K(\boldsymbol{q}, \boldsymbol{y})}{\partial q_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial \Gamma_1(\boldsymbol{q}, \boldsymbol{y})}{\partial q_d} & \cdots & \frac{\partial \Gamma_K(\boldsymbol{q}, \boldsymbol{y})}{\partial q_d} \end{bmatrix} \tag{5.2}$$

where each k-th column vector of $\mathrm{Jac}_q\big(\boldsymbol{\Gamma}(\boldsymbol{q}, \boldsymbol{y})\big)$ corresponds to the gradient of each k-th scalar

distance function $\Gamma_k(\boldsymbol{q}, \boldsymbol{y})$ with respect to $\boldsymbol{q}$. For simplified notation, we use

$$\Gamma(\boldsymbol{q}) = \min_{\boldsymbol{y} \in \mathcal{O}} \min_{k=1,\dots,K} \Gamma_k(\boldsymbol{q}, \boldsymbol{y}), \tag{5.3}$$

and denote the corresponding column of $\text{Jac}_q\big(\Gamma(\boldsymbol{q}, \boldsymbol{y})\big)$ as $\nabla\Gamma(\boldsymbol{q})$. In this way, $\Gamma(\boldsymbol{q})$ represents the minimal distance between the robot in configuration $\boldsymbol{q}$ and the nearest obstacle, and $\nabla\Gamma(\boldsymbol{q})$ represents the corresponding repulsive gradient in the joint space of the robot. Similarly, we assume the existence of function $\Gamma_{self}(\boldsymbol{q})$, which represents the minimal distance between the robot in configuration $\boldsymbol{q}$ and its own links. The gradient of $\Gamma_{self}(\boldsymbol{q})$ with respect to $\boldsymbol{q}$ is denoted as $\nabla\Gamma_{self}(\boldsymbol{q})$. Please refer to previous chapters for more details on learning and applications of such distance functions.

5.3.2 Goals

The main goal of this chapter is to design a modulation algorithm that reshapes $\boldsymbol{f}(\boldsymbol{q})$ from (5.1) with respect to the task-space obstacles $\mathcal{O}$ defined by $\Gamma(\boldsymbol{q})$ and self-collision boundary defined by $\Gamma_{self}(\boldsymbol{q})$, enabling the robot to (a) *avoid concave obstacles in joint space* while (b) *maintaining local stability of the nominal DS* (Equation (5.1)), and a controller capable of (c) *computing the required modulation with high frequency* to enable avoidance of moving obstacles.

To this end, we aim to leverage the parallelization and differentiability properties of the learned function $\Gamma(\boldsymbol{q})$, by combining the modulation framework (Khansari-Zadeh and Billard, 2012a; Huber et al., 2019) with sampling-based model predictive control (Bhardwaj et al., 2022; Williams et al., 2017). The vision is to achieve the desired properties such as local stability of the modulated DS and the ability to reactively avoid moving obstacles of arbitrary shape, within a single unified framework.

5.4 DS Modulation with Deflected Obstacle-Tangent Dynamics

In this section, we provide the necessary background on the DS modulation method, as it is essential for understanding our approach. For more detailed preliminaries on dynamical systems, please refer to Appendix A.

5.4.1 Dynamical System Modulation

A nominal Dynamical System (Equation (5.1)) defines a state-dependent vector field that can be altered, or modulated, by rotating the field with a modulation matrix $\boldsymbol{M}(.)$, where $\boldsymbol{M}$ can depend on different variables. The *modulated Dynamical System* is then defined as:

$$\dot{\boldsymbol{q}} = \boldsymbol{M}(.)\boldsymbol{f}(\boldsymbol{q}), \tag{5.4}$$

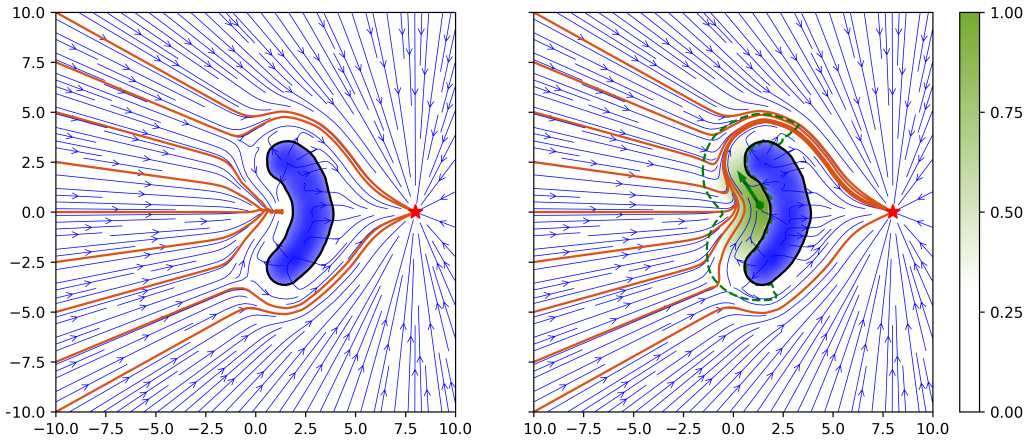


Figure 5.1: Two-dimensional toy example demonstrating the behavior of the modulated DS in presence of obstacle (blue). The system is linear with stable attractor (red) at (8,0). Orange lines indicate trajectories integrated forward in time. **(left)** Modulated DS (Khansari-Zadeh and Billard, 2012a) has local minima in the concave region of the obstacle. **(right)** Our method avoids the local minima by adding one navigation kernel with parameters estimated via sampling-based MPC. Green arrow indicates the center $\hat{q}_k$ and the direction $\hat{g}_k$ of the added deflection. Green shaded area shows the kernel activation region.

Depending on the choice of $\boldsymbol{M}(.)$, this DS can exhibit various behaviors. For example, $\boldsymbol{M}(\boldsymbol{q})$ is a matrix-valued function of the system state $\boldsymbol{q}$, and it can be activated differently in different regions of the state space, reshaping the nominal DS $\boldsymbol{f}(\boldsymbol{q})$. Importantly, if we aim to preserve some stability properties of nominal DS, certain limitations must be applied to the modulation matrix $\boldsymbol{M}(.)$.

Intuitively, when starting from a globally asymptotically stable (G.A.S) DS $\boldsymbol{f}(\boldsymbol{q})$, reshaping it with a full rank and locally active matrix $\boldsymbol{M}(\boldsymbol{q})$ only guarantees that the system remains stable (no spurious attractors are introduced but limit cycles can arise) and that trajectories can be kept arbitrarily close to the attractor $\boldsymbol{q}^*$, if they start close enough (which is equal to local asymptotic stability of the attractor) (Kronander et al., 2015).

DS modulation using matrix $\boldsymbol{M}(\boldsymbol{q}, \mathcal{O}) \in \mathbb{R}^{d\times d}$ that depends on the obstacles configuration $\mathcal{O}$, can be used to achieve obstacle avoidance. Further, we omit the matrix $\boldsymbol{M}(\boldsymbol{q})$ dependency on obstacles state $\mathcal{O}$ for a simpler notation, and always assume that $\boldsymbol{M}$ is a function of both state $\boldsymbol{q}$ and obstacle configuration $\mathcal{O}$. To allow for obstacle avoidance, the following modulation matrix is proposed in (Khansari-Zadeh and Billard, 2012a):

$$\boldsymbol{M}(\boldsymbol{q}) = \boldsymbol{E}(\boldsymbol{q})\boldsymbol{D}(\boldsymbol{q})\boldsymbol{E}(\boldsymbol{q})^{-1}, \tag{5.5}$$

where $\boldsymbol{M}(\boldsymbol{q})$ is composed through eigenvalue decomposition. In Equation (5.5), $\boldsymbol{D}(\boldsymbol{q})$ is a diagonal scaling matrix and $\boldsymbol{E}(\boldsymbol{q})$ is an orthogonal matrix defined as:

$$\boldsymbol{E}(\boldsymbol{q}) = \begin{bmatrix} \boldsymbol{n}(\boldsymbol{q}) & \boldsymbol{e}_1(\boldsymbol{q}) & \dots & \boldsymbol{e}_{d-1}(\boldsymbol{q}) \end{bmatrix}. \tag{5.6}$$

The first column of the matrix $\boldsymbol{E}(\boldsymbol{q})$ consists of the obstacle normal $\boldsymbol{n}(\boldsymbol{q})$[1]. The remaining columns, $\boldsymbol{e}_i(\boldsymbol{q})$, form a $(d-1)$-dimensional orthonormal basis for the tangent space at the state $\boldsymbol{q}$.. The tangential hyperplane formed by this basis is referred to as the *deflection* hyperplane in (Khansari-Zadeh and Billard, 2012a). Note that it is not a necessary condition for this basis to be orthonormal. Any basis that forms a tangential hyperplane to the normal $\boldsymbol{n}(\boldsymbol{q})$ and has linearly independent columns is valid. However, by enforcing orthonormality on the basis, we can exploit the fact that $\boldsymbol{E}(\boldsymbol{q})$ becomes orthogonal and $\boldsymbol{E}(\boldsymbol{q})^{-1} = \boldsymbol{E}(\boldsymbol{q})^T$, thereby improving numerical stability.

The diagonal matrix $\boldsymbol{D}(\boldsymbol{q})$ is defined as:

$$\boldsymbol{D}(\boldsymbol{q}) = \begin{bmatrix} \lambda_n(\boldsymbol{q}) & 0 & \dots & 0 \\ 0 & \lambda_\tau(\boldsymbol{q}) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_\tau(\boldsymbol{q}) \end{bmatrix}, \tag{5.7}$$

and is composed of two eigenvalues $\lambda_n(\boldsymbol{q})$ and $\lambda_\tau(\boldsymbol{q})$ defined as a function of the distance of the system from the obstacle $\Gamma(\boldsymbol{q})$.

While the definition of $\Gamma(\boldsymbol{q})$ may vary, the general idea is that $\boldsymbol{D}(\boldsymbol{q})$ should become identity matrix when the system is far from collisions, $\lambda_n(\boldsymbol{q})$ should approach 0 as the system is getting closer to collision, and $\lambda_n(\boldsymbol{q}) = 0$ at the obstacle boundary $\boldsymbol{Q}_\mathcal{O}$. In essence, the modulation (5.5) redistributes the flow of the nominal DS (5.1) based on proximity to the obstacle, thus allowing obstacle avoidance.

However, the collision avoidance properties of such modulation are restricted to a subset of convex obstacles. There is also an edge case within the subset of states located on the obstacle boundary $\boldsymbol{Q}_\mathcal{O}$. To ensure *obstacle impenetrability* by design, $\lambda_n(\boldsymbol{q}) = 0$ for all $\boldsymbol{q} \in \boldsymbol{Q}_\mathcal{O}$, causing the matrix $\boldsymbol{M}(\boldsymbol{q})$ to lose rank and create a nullspace. Consequently, velocities may vanish, leading to the emergence of spurious attractors and destabilization of the system's dynamics.

[1] $\boldsymbol{n}(\boldsymbol{q})$ is a unit vector, orthogonal to the obstacle boundary, which can be obtained, for example, by normalizing the gradient of $\Gamma(\boldsymbol{q})$.

Definition 1 (Impenetrability 1 (Neumann boundary condition)) *Consider an obstacle whose boundary is described by a smooth function* $\Gamma(\boldsymbol{q}) = 0$*, where* $\Gamma : \mathbb{R}^d \rightarrow \mathbb{R}$ *is a continuously differentiable function* ($\mathcal{C}^1$)*. Obstacle boundary normal is represented as* $\boldsymbol{n}(\boldsymbol{q}) = \frac{\nabla\Gamma(\boldsymbol{q})}{||\nabla\Gamma(\boldsymbol{q})||_2}$*. A vector field* $\boldsymbol{f}(\boldsymbol{q})$ *is not penetrating the obstacle boundary if its projection onto the obstacle normal* $\boldsymbol{n}(\boldsymbol{q})$ *vanishes on the obstacle's surface* $\Gamma(\boldsymbol{q}) = 0$*:*

$$\boldsymbol{f}(\boldsymbol{q})^T \boldsymbol{n}(\boldsymbol{q}) = 0 \qquad \forall \boldsymbol{q} : \Gamma(\boldsymbol{q}) = 0. \tag{5.8}$$

Definition 1 is derived from (Khansari-Zadeh and Billard, 2012a). Nonetheless, this definition is only valid for the continuous case, whereas any robotics application necessitates discretizing the dynamics. Specifically, for the case convex of concave obstacle, if the integration timestep δt is sufficiently large, a single iteration could bring the system state inside the obstacle. Conversely, for concave obstacles, an additional mode of failure arises, in which any motion in the tangential plane will violate the obstacle boundary if the system starts on the obstacle surface. Therefore, we propose a more general definition of impenetrability for the discretized case. Notably, it is still applicable for continuous dynamics.

Definition 2 (Impenetrability 2) *Consider an obstacle whose boundary is described by isosurface* $\Gamma(\boldsymbol{q}) = 0$*, where* $\Gamma : \mathbb{R}^d \rightarrow \mathbb{R}$*. A vector field* $\boldsymbol{f}(\boldsymbol{q})$ *is not penetrating the obstacle boundary, if for any trajectory* $\{\boldsymbol{q}\}_t$ *that starts outside the obstacle boundary, i.e.,* $\Gamma(\boldsymbol{q}_0) \geq 0$*, and evolving according to* $\dot{\boldsymbol{q}} = \boldsymbol{f}(\boldsymbol{q})$*, the following holds:* $\Gamma(\{\boldsymbol{q}\}_t) \geq 0, \forall t$*.*

However, Definition 1 is still useful for analyzing the stability properties of modulation. Spurious attractors, which can be can be local minima or saddle points, are induced when the nominal DS and obstacle normal are collinear at the obstacle boundary, i.e. $|\langle \boldsymbol{f}(\boldsymbol{q}), \boldsymbol{n}(\boldsymbol{q})\rangle| = ||\boldsymbol{f}(\boldsymbol{q})||$ and $\langle \boldsymbol{f}(\boldsymbol{q}), \boldsymbol{e}_i(\boldsymbol{q})\rangle = 0$ holds for all i. Refer to Figure 5.1 for demonstration of the local minima created with a simple concave obstacle.

An improvement upon the modulation framework for obstacle avoidance to avoid spurious attractors and preserve stability is proposed in (Huber et al., 2019, 2022). Instead of using the obstacle normal $\boldsymbol{n}(\boldsymbol{q})$ as the first column of $\boldsymbol{E}(\boldsymbol{q})$, the authors propose to instead use the reference direction $\hat{\boldsymbol{r}}(\boldsymbol{q}, \boldsymbol{q}^r)$ defined as:

$$\hat{\boldsymbol{r}}(\boldsymbol{q}, \boldsymbol{q}^r) = \frac{\boldsymbol{q}^r - \boldsymbol{q}}{||\boldsymbol{q}^r - \boldsymbol{q}||}, \tag{5.9}$$

where $\boldsymbol{q}^r$ is a point belonging to the obstacle.

While discarding the orthonormality of the basis $\boldsymbol{E}(\boldsymbol{q})$, this modification allows for the system to avoid obstacles with non-convex boundaries that can be represented as star-shaped obstacles. The reference direction $\hat{\boldsymbol{r}}(\boldsymbol{q}, \boldsymbol{q}^r)$ is not orthogonal to any of the basis vectors $\boldsymbol{e}_i$, thus

$\langle \hat{\boldsymbol{r}}(\boldsymbol{q}, \boldsymbol{q}^r), \boldsymbol{f}(\boldsymbol{q}) \rangle$ is projected onto tangential space, and there exists a tangential vector flow on the obstacle boundary, alleviating the local minima issue.

Interestingly, the improved modulated DS approach using reference direction (5.9) to construct $\boldsymbol{E}(\boldsymbol{q})$ can be interpreted as a nonlinear control system with the following form (See Appendix B.1):

$$\dot{\boldsymbol{q}} = \underbrace{\boldsymbol{M}(\boldsymbol{q}) \cdot \boldsymbol{f}(\boldsymbol{q})}_{\text{Modulated Dynamics}} + \underbrace{\boldsymbol{g}(\boldsymbol{q}, \boldsymbol{q}^r)}_{\text{Input}}, \tag{5.10}$$

where $\boldsymbol{g}(\boldsymbol{q}, \boldsymbol{q}^r)$ is a nonlinear function that depends on the current state $\boldsymbol{q}$ and the obstacle reference point $\boldsymbol{q}^r$ and pushes the robot state away from local minima. This interpretation connects the DS modulation approach to optimal control domain, as the tangent space vector flow can be thought of as a virtual control input applied to the system. Thus, $\boldsymbol{q}^r$ can be considered a control variable that is forcing the modulation to avoid $\boldsymbol{M}(\boldsymbol{q}) \cdot \boldsymbol{f}(\boldsymbol{q}) \to 0$ at the obstacle boundary.

While Equation (5.10) alleviates the creation of spurious attractors by inducing additional tangent space velocities through the additive nonlinear function $\boldsymbol{g}(\boldsymbol{q}, \boldsymbol{q}^r)$, such term is difficult to control and scale to high-dimensional problems. Specifically, $\boldsymbol{g}(\boldsymbol{q}, \boldsymbol{q}^r)$ is dependent on the projection of the nominal DS $\boldsymbol{f}(\boldsymbol{q})$ and the placement of reference point $\boldsymbol{q}^r$. Finding a suitable $\boldsymbol{q}^r$ is not trivial for high-dimensional concave obstacles, thus it may be challenging to build an algorithm that guarantees the collision avoidance and convergence in case of multiple complex obstacles. Additionally, this method introduces the need to invert the matrix $\boldsymbol{E}(\boldsymbol{q})$ at every iteration, which can be computationally expensive for high-frequency applications in high-dimensional systems.

5.4.2 Locally Deflected Obstacle-Tangent Space Dynamics

Inspired by the state-input system interpretation introduced in Equation (5.10), we present our approach for creating meaningful obstacle-tangent deflections that are locally active in the regions of potential local minima of the modulated nominal DS. To alleviate the shortcomings of (Khansari-Zadeh and Billard, 2012a) and (Huber et al., 2019), we propose the following formulation for modulating the DS:

$$\begin{aligned} \dot{\boldsymbol{q}} &= \boldsymbol{M}(\boldsymbol{q}) \left(\boldsymbol{f}(\boldsymbol{q}) + \alpha(\boldsymbol{q}) \boldsymbol{g}(\boldsymbol{q}) \right) \\ &= \underbrace{\boldsymbol{M}(\boldsymbol{q}) \cdot \boldsymbol{f}(\boldsymbol{q})}_{\text{Standard Modulation}} + \underbrace{\alpha(\boldsymbol{q})}_{\in [0,1]} \underbrace{\boldsymbol{M}(\boldsymbol{q}) \cdot \boldsymbol{g}(\boldsymbol{q})}_{\substack{\text{Explicit Tangent} \\ \text{Space Dynamics}}} \end{aligned} \tag{5.11}$$

where $\boldsymbol{M}(\boldsymbol{q}) \in \mathbb{R}^{d \times d}$ is a modulation matrix as defined in Equations (5.5)-(5.7) and (5.13), $\boldsymbol{f}(\boldsymbol{q}) : \mathbb{R}^d \to \mathbb{R}^d$ is the nominal DS, and $\boldsymbol{g}(\boldsymbol{q}) : \mathbb{R}^d \to \mathbb{R}^d$ is a vector field that is explicitly defined to be locally tangential to the obstacle boundary, i.e. $\boldsymbol{g}(\boldsymbol{q}) \perp \boldsymbol{n}(\boldsymbol{q})$, with $\boldsymbol{n}(\boldsymbol{q})$ being the obstacle normal.

Continuous scalar-valued activation function $\alpha(\boldsymbol{q}) : \mathbb{R}^d \rightarrow \mathbb{R}$ indicates the regions in state-space where the obstacle-tangential vector field $\boldsymbol{g}(\boldsymbol{q})$ is active; with $\alpha(\boldsymbol{q}) \in [0,1]$. When the nominal modulated DS term is close to local minima (i.e. $\boldsymbol{M}(\boldsymbol{q}) \cdot \boldsymbol{f}(\boldsymbol{q}) \rightarrow 0$), then $0 < \alpha(\boldsymbol{q}) \leq 1$, otherwise, $\alpha(\boldsymbol{q}) = 0$. Obstacle-tangential velocities $\boldsymbol{g}(\boldsymbol{q})$ are thus introduced only to avoid generation of spurious attractors. Figure 5.1 demonstrates the possible deflection added in the vicinity of local minima.

The advantage of Equation (5.11) over (Khansari-Zadeh and Billard, 2012a) and (Huber et al., 2019) is that $\boldsymbol{g}(\boldsymbol{q})$ is independent of $\boldsymbol{f}(\boldsymbol{q})$ or any reference point $\boldsymbol{q}^r$, opening the possibility of defining such tangential vector field with external optimization techniques analogous to a stabilizing optimal control input. This allows an intuitive and explicit control over the tangential component of the vector field yielding an inherent boundary impenetrable modulated DS.

5.4.3 Distance Function Adaptation

Our goal is to use a learned model of the true distance to collision, as defined in Equation (5.3), to define the collision boundary. To preserve the collision avoidance properties of the modulation $\boldsymbol{M}(\boldsymbol{q})$ defined in Equation (5.5), we need to properly define the functions $\lambda_n(\boldsymbol{q})$ and $\lambda_\tau(\boldsymbol{q})$, which are used to construct the diagonal gains matrix $\boldsymbol{D}(\boldsymbol{q})$.

We employ a parametrized sigmoid function:

$$\begin{aligned} \sigma(d_1, d_2, \lambda_1, \lambda_2, k) = \\ = \lambda_1 + \frac{\lambda_2 - \lambda_1}{1 + \exp\left(-k\left(\Gamma(\boldsymbol{q}) - \frac{(d_1+d_2)}{2}\right)\right)}, \end{aligned} \tag{5.12}$$

which smoothly connects the constant values λ_1 and λ_2 as a function of distance $\Gamma(\boldsymbol{q})$. The parameter k controls the width of the transition, and the offsets d_1 and d_2 define the transition mid-point.

To define $\lambda_n(\boldsymbol{q})$, we use $\lambda_1 = 0$ and $\lambda_2 = 1$, while for $\lambda_\tau(\boldsymbol{q})$, $\lambda_1 = 2$ and $\lambda_2 = 1$. Distance parameters d_1 and d_2 are set to be $1cm$ and $10cm$ respectively, and parameter k is set to $k = 2$. Thus, diagonal elements of matrix $\boldsymbol{D}(\boldsymbol{q})$ are defined as:

$$\begin{aligned} \lambda_n(\boldsymbol{q}) &= \sigma(1cm, 10cm, 0, 1, 2) \\ \lambda_\tau(\boldsymbol{q}) &= \sigma(1cm, 10cm, 2, 1, 2) \end{aligned} \tag{5.13}$$

With such construction, $\lambda_n(\boldsymbol{q}) = \lambda_\tau(\boldsymbol{q}) = 1$ (making matrix $\boldsymbol{D}(\boldsymbol{q})$ to be identity) for $\Gamma(\boldsymbol{q}) >$ 10 cm. For situations when distance-to-collision is lower than threshold $\Gamma(\boldsymbol{q}) = 1$ cm, $\lambda_n(\boldsymbol{q}) = 0$ and $\lambda_\tau(\boldsymbol{q}) = 2$, guaranteeing non-penetrability.

Theorem 1 (Impenetrability) *Consider an obstacle in the task space of the robot, represented by a set of points $\mathcal{O} \subset \mathbb{R}^3$. Let $\Gamma(\boldsymbol{q}) : \mathbb{R}^d \rightarrow \mathbb{R}$ be a continuous and continuously differentiable ($\mathcal{C}^1$)*

function that expresses the minimal distance between the robot in state $\boldsymbol{q}$ and set $\mathcal{O}$. System (5.11) *preserves the impenetrability of the obstacle.*
Proof: *See Appendix B.2.* ■

5.4.4 Explicit Tangent Space Dynamics as Navigation Kernels

We propose to construct the vector field $\boldsymbol{g}(\boldsymbol{q})$ that will be projected onto the tangent space of the obstacle boundary as a sum of K local dynamics $\hat{\boldsymbol{g}}_k$, which we refer to as *navigation kernels*, activated by radial-basis functions as follows:

$$\boldsymbol{g}(\boldsymbol{q}) = \sum_{k=1}^{K} \hat{\boldsymbol{g}}_k \exp\left(-\gamma_k ||\boldsymbol{q} - \hat{\boldsymbol{q}}_k||\right), \tag{5.14}$$

where $\hat{\boldsymbol{q}}_k$ is the centroid of k-th navigation kernel, and γ_k regulates kernel width. Vector $\hat{\boldsymbol{g}}_k$ is a linear local dynamics activated in the vicinity of the kernel center $\hat{\boldsymbol{q}}_k$.

Essentially, formulation (5.14) allows $\boldsymbol{g}(\boldsymbol{q})$ to be defined by a set of K navigation kernels, so that $\boldsymbol{g}(\boldsymbol{q})$ is present when the state $\boldsymbol{q}$ is close to the k-th kernel center $\hat{\boldsymbol{q}}_k$, and exponentially decays as the distance to the kernel center increases. If kernels are placed close to the obstacles (so that $\Gamma(\hat{\boldsymbol{q}}_k) < \varepsilon$), then the added tangential modulation will be stronger in the vicinity of the obstacle, and will decay as the distance to the obstacle increases.

5.4.5 Navigation Kernel Activation

The navigation kernels defined in Equation (5.14) are activated in the deflected dynamics (Equation (5.11)) by a state-dependent scalar-valued function $\alpha(\boldsymbol{q}) \in [0,1]$, that is used to reduce the effect of the vector field $\boldsymbol{g}(\boldsymbol{q})$ when the system is not in the local minima induced by the obstacle. This function consists of three components:

$$\alpha(\boldsymbol{q}) = \begin{cases} \alpha_\Gamma(\boldsymbol{q}) \cdot \alpha_n(\boldsymbol{q}) \cdot \alpha_f(\boldsymbol{q}), & ||\boldsymbol{q} - \boldsymbol{q}^*|| > r \\ 0, & ||\boldsymbol{q} - \boldsymbol{q}^*|| \leq r \end{cases} \tag{5.15}$$

where $\alpha_\Gamma(\boldsymbol{q})$ is equal to one when the system is close to the obstacle, and decays to zero with the increase of distance to collision. For example, we can define $\alpha_\Gamma(\boldsymbol{q}) = 1 - \lambda_n(\boldsymbol{q})$, rendering navigation kernels inactive for any robot state that is further than 10cm from collision. Values of $\alpha_n(\boldsymbol{q})$ change between 0 and 1 depending on the angle between the obstacle normal and the nominal DS direction. Finally, $\alpha_f(\boldsymbol{q})$ continuously changes from 1 to 0 as the system state $\boldsymbol{q}$ approaches the nominal DS attractor $\boldsymbol{q}^*$.

Additionally, we enforce $\alpha(\boldsymbol{q}) = 0$ within a ball B_r with radius r enclosing the attractor $\boldsymbol{q}^*$, to be able to preserve local stability of the attractor. Refer to Figure 5.1 for visualization of active area for a kernel placed close to the concave obstacle. Exact mathematical definitions for $\alpha_{\{\Gamma,n,f\}}$ are provided further in section 5.6.2.

Proposition 1 (L.A.S) *Consider a modulated DS in the form* $\dot{\boldsymbol{q}} = \boldsymbol{M}(\boldsymbol{q})\boldsymbol{f}(\boldsymbol{q})$. *Assume that* $\boldsymbol{f}(\boldsymbol{q})$ *is G.A.S. at the attractor* $\boldsymbol{q}^*$, *modulation matrix* $\boldsymbol{M}(\boldsymbol{q})$ *has full rank* $\forall \boldsymbol{q} \in \mathbb{R}^d$ *and is equal to an identity matrix in a ball* B_r *with radius* r *centered at* $\boldsymbol{q}^*$, *i.e.,* $\boldsymbol{M}(\boldsymbol{q}) = \mathbb{I}_{d\times d}$, $\forall \boldsymbol{q} : \boldsymbol{q} \in B_r$. *The modulated DS with the tangential deflection component (as in Equations 5.11, 5.14, and 5.15) in a form* $\dot{\boldsymbol{q}} = \boldsymbol{M}(\boldsymbol{q})\big(\boldsymbol{f}(\boldsymbol{q}) + \alpha(\boldsymbol{q})\boldsymbol{g}(\boldsymbol{q})\big)$ *is locally asymptotically stable (L.A.S.) at the attractor* $\boldsymbol{q}^*$ *if the activation parameter* $\alpha(\boldsymbol{q}) = 0, \forall \boldsymbol{q} : \boldsymbol{q} \in B_r$.

Proof: *See Appendix B.3.* ■

5.4.6 Navigation Kernel Parameter Optimization

The combination of K kernels allows to define a policy that is locally active, and is able to provide meaningful tangential components to the nominal DS enabling navigation around obstacles. The described policy is defined by direction $\hat{\boldsymbol{g}}_k$, kernel center $\hat{\boldsymbol{q}}_k$, and kernel width γ_k for each k-th kernel respectively. To determine these parameters, we propose to use MPPI approach similar to (Bhardwaj et al., 2022).

The MPPI algorithm provides a framework for finding the optimal parameters of a navigation policy by sampling a distribution of system parameters and then evaluating the corresponding cost of each trajectory generated by these samples. In our proposed formulation, we apply MPPI to learn the parameters of the navigation kernels. Specifically, the MPPI algorithm will be used to search over the space of $\hat{\boldsymbol{g}}_k$, which represents the local dynamics associated with each kernel.

5.5 Navigation Kernel Parameters Optimization via Sampling-based MPC (MPPI)

5.5.1 MPPI Algorithm Description

The idea behind the MPPI method is to sample the system parameters from a distribution, and then to use the sampled parameters to generate a set of trajectories. The cost of each trajectory is computed, and used to change the distribution and bias the sampling procedure towards the parameters that lead to the lower cost. Eventually, the procedure converges to parameters that minimize the defined cost function (Yoon et al., 2022).

Let's consider an autonomous discrete-time Dynamical System of the form

$$\boldsymbol{q}_h = \tilde{\boldsymbol{f}}(\boldsymbol{q}_{h-1}, \boldsymbol{g}_1, \ldots, \boldsymbol{g}_K), \tag{5.16}$$

where $\boldsymbol{q}_h$ is the state vector at time-index h and parameters $\boldsymbol{g}_{k=1..K}$ alter the system's behavior. If $\boldsymbol{g}_k = 0, \forall k$, then the system behaves as some nominal DS.

At each iteration, N candidate parameter sets $\{\boldsymbol{g}_{i,k}\}_{i=1..N}^{k=1..K}$ are sampled from K independent

multivariate Gaussian distributions such that $\boldsymbol{g}_{i,k} \sim \mathcal{N}_k(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$, and then used to generate N trajectories by propagating system (5.16) forward in time for H timesteps. We denote resulting trajectories (or *roll-outs*) as $\{\boldsymbol{q}_{i,h}, \dot{\boldsymbol{q}}_{i,h}\}_{i=1..N}^{h=1..H}$. After that, the cost for each trajectory is defined as $c_i = c(\boldsymbol{q}_{i,1}, \ldots, \boldsymbol{q}_{i,H})$. The cost is designed according to the desired bahvior, i.e. to prefer trajectories that are close to the nominal DS, or to enable obstacle avoidance, or to be compliant with secondary tasks. Corresponding weights are then caclulated as

$$w_i = \exp\left(-\frac{c_i}{\beta}\right), \tag{5.17}$$

and the means of the parameter distributions are updated as follows:

$$\boldsymbol{\mu}_k^{new} = (1-\alpha_\mu)\boldsymbol{\mu}_k + \alpha_\mu \frac{\sum_{i=1}^N w_i \boldsymbol{g}_{i,k}}{\sum_{i=1}^N w_i}, \tag{5.18}$$

essentially shifting the distribution center towards the parameters that lead to the lowest cost. The covariance matrix may also be updated:

$$\begin{aligned} \boldsymbol{\Sigma}_k^{new} =& (1-\alpha_\Sigma)\boldsymbol{\Sigma}_k + \\ &+ \alpha_\Sigma \frac{\sum_{i=1}^N w_i (\boldsymbol{g}_{i,k} - \boldsymbol{\mu})(\boldsymbol{g}_{i,k} - \boldsymbol{\mu}_k)^T}{\sum_{i=1}^N w_i}. \end{aligned} \tag{5.19}$$

After updating distributions parameters with $\boldsymbol{\mu}_k^{new}$ and $\boldsymbol{\Sigma}_k^{new}$, each parameter $\boldsymbol{g}_k$ is chosen either as the one with lowest corresponding cost c_i, or as a weighted sum (last term in (5.18)), or as $\boldsymbol{\mu}_k^{new}$, and the system is propagated forward in time for one timestep, and the process is repeated.

In (5.18)-(5.19), α_μ and α_Σ are the learning rates for the mean and covariance matrix, respectively. The parameter $\beta > 0$ is a temperature parameter that controls the amount of exploration. The higher the temperature, the more the distribution is spread out, and the more the exploration is performed. Authors of Williams et al. (2017) prove that such process iteratively converges to the optimal parameters that minimize the cost function.

5.5.2 MPPI Application

Navigation vector field $\boldsymbol{g}(\boldsymbol{q})$ is defined as sum of locally active navigation kernels (5.14), and each kernel is defined by three parameters: local deflection direction $\hat{\boldsymbol{g}}_k$, kernel placement $\hat{\boldsymbol{q}}_k$ and kernel width γ_k.

We propose to utilize the MPPI framework to determine the local navigation direction $\hat{\boldsymbol{g}}_k$ of each kernel. Additionally, numerous exploration trajectories produced by the MPPI algorithm at each iteration may be used to determine the placement of the navigation kernels, i.e. values of $\hat{\boldsymbol{q}}_k$.

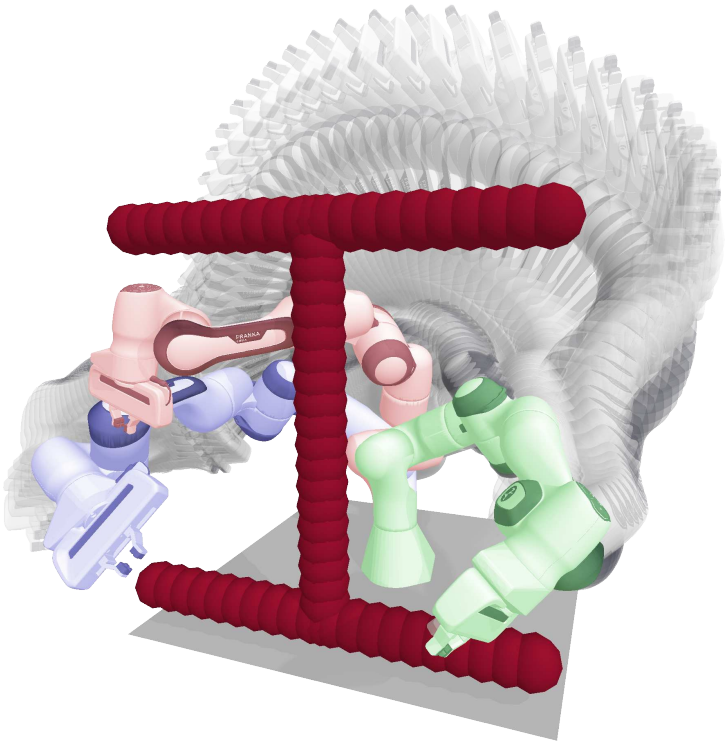


Figure 5.2: The robot begins in the blue state (start joint configuration) and must reach the attractor configuration (green state) while avoiding a concave task-space obstacle represented by a set of red spheres. The nominal robot dynamics is a linear motion in joint space towards the attractor. The standard modulated DS approach (Khansari-Zadeh and Billard, 2012a) exhibits a local minimum (represented by the red robot state) in this scenario. Our proposed method is able to reactively navigate around the obstacle and reach the attractor successfully.

While kernel widths γ_k can also be optimized via MPPI, for simplicity we fix $\gamma_k = \hat{\gamma}, \forall k$ as a hyperparameter. Kernel placement $\hat{\boldsymbol{q}}_k$ is determined during exploration using the roll-out states. The only parameter set to find via MPPI is then $\{\hat{\boldsymbol{g}}_k\}_{k=1..K}$. Such approach enables exploration of a large range of potential navigation strategies in the vicinity of the obstacle.

At each iteration of the MPPI algorithm, parameter set $\{\hat{\boldsymbol{g}}_{i,k}\}_{i=1..N}^{k=1..K}$ is sampled from K multivariate Gaussian distributions that are parameterized by $\{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1..K}$. We then use the sampled parameters to generate N roll-outs $\{\boldsymbol{q}_{i,h}, \dot{\boldsymbol{q}}_{i,h}\}_{i=1..N}^{h=1..H}$ by propagating the system dynamics H timesteps forward in time.

Next, the cost of each trajectory is computed, which is then used to bias the sampling procedure to achieve a lower cost. The weights for each sample are calculated using Equation (5.17), and the mean of the parameter distributions is updated using Equation (5.18). Once the new sampling distribution has been determined, we update the system state, sample new sets of parameters, and the process repeats.

An example of a placed navigation kernel with parameters optimized using MPPI is demon-

strated in Figure 5.1. Notably, in this two-dimensional toy example, the navigation strategy is relatively straightforward - track the obstacle boundary in either the left or right direction. However, in three dimensions, the tangential plane consists of an infinite number of directions to track the obstacle boundary. As dimensionality increases, it becomes exponentially more challenging to find valid navigation strategies. We apply the MPPI algorithm to a 7-DoF robot arm, where an infinite number of navigation strategies can be explored. Examples of successful navigation strategies for such cases are illustrated in Figures 5.2 and 4.7.

5.5.3 Cost Function

At each iteration of the MPPI algorithm, a cost for each roll-out $\{\boldsymbol{q}_{i,h}, \dot{\boldsymbol{q}}_{i,h}\}_{i=1..N}^{h=1..H}$ is computed. The cost function encodes desired robot behavior, and is used to bias the sampling procedure towards the parameters that minimize the cost. Cost function can be defined as a weighted sum of several cost terms. In this work, we consider the following cost terms:

Goal Reaching

The first cost term is a goal reaching cost, which is defined as follows:

$$c_i^{goal} = ||\boldsymbol{q}^* - \boldsymbol{q}_{i,H}||_2, \tag{5.20}$$

where $\boldsymbol{q}^*$ is the attractor of the nominal DS (5.1), and $\boldsymbol{q}_{i,H}$ is the position of the robot at the end of the i-th roll-out.

We only consider the final position of the roll-out, as sampled trajectory may temporarily move away from the goal while navigating around the obstacle. This cost penalizes trajectories that do not approach to the goal in the integration horizon H.

(Self-)Collision Avoidance

Distance estimator $\Gamma(\boldsymbol{q})$ provides true distance between the robot and the closest obstacle. As the intended behavior of the motion planner is to navigate around the obstacles, a continuous function penalizing close proximity to the obstacles may restrict the exploration of the space of possible navigation strategies. Therefore, we use a binary collision detection function that penalizes only trajectories that collide with the obstacle. The collision avoidance cost is defined as follows:

$$c_i^{coll} = \sum_{h=1}^{H} c_{i,h}^{coll}, \tag{5.21}$$

where

$$c_{i,h}^{coll} = \begin{cases} 0, & \text{if } \Gamma(\boldsymbol{q}_{i,h}) > 0 \\ 1, & \text{otherwise.} \end{cases} \tag{5.22}$$

For self-collision avoidance cost $c_i^{self\text{-}coll}$, we use the same cost function, but with the distance estimator $\Gamma(\boldsymbol{q})$ replaced by $\Gamma_{self}(\boldsymbol{q})$.

Joint Limits Avoidance

We treat joint limits violation in the same way as collisions. Each state is penalized for violation of the joint limits, and the cost is defined as follows:

$$c_i^{lim} = \sum_{h=1}^{H} c_{i,h}^{lim}, \tag{5.23}$$

where

$$c_{i,h}^{lim} = \begin{cases} 0, & \text{if } \boldsymbol{q}_{i,h} \in [\boldsymbol{q}_{min}, \boldsymbol{q}_{max}] \\ 1, & \text{otherwise.} \end{cases} \tag{5.24}$$

Since we assume the DS control in the joint space, we are not concerned by low-manipulability issues, thus do not dampen the joints close to joint-limits.

Stagnation Avoidance

The cost function also includes a term that penalizes trajectories that do not move with time. This cost is defined as follows:

$$c_i^{stay} = \frac{c_i^{goal}}{\max(\varepsilon, ||\boldsymbol{q}_{i,1} - \boldsymbol{q}_{i,H}||_2)}. \tag{5.25}$$

This cost encourages the exploration, while penalizing the stagnant trajectories away from the attractor, thus helping to avoid getting stuck in local minima. To some extent, this cost also acts against appearance of limit cycles, that are theoretically possible for the DS modulation approach. However, the length of a cycle should correspond to the horizon of the MPC path prediction. As the trajectories converge to $\boldsymbol{q}^*$, this cost becomes zero.

Nominal DS Similarity

Another cost component we propose to use measures the difference between the nominal DS and the actual trajectory. This cost is defined as follows:

$$c_i^{DS} = \sum_{h=1}^{H} \Gamma(\boldsymbol{q}_{i,h}) ||\dot{\boldsymbol{q}}_{i,h} - f(\boldsymbol{q}_{i,h})||_2, \tag{5.26}$$

and allows the robot to deviate from the nominal vector field when close to the obstacle.

Total Cost

The total cost for each trajectory is defined as a weighted sum of the cost terms defined above:

$$c_i = \sum_{t \in T} w_t c_i^t, \tag{5.27}$$

where w_t is the weight of the cost term t, and T is the set of cost terms $T = \{goal, coll, self\text{-}coll, lim, stay, DS\}$.

5.5.4 New kernel placement

MPPI algorithm is essentially a sampling-based motion planner generating new explorative trajectories at each iteration. For each point $\boldsymbol{q}_{i,h}$ at runtime the distance-to-collision $\Gamma(\boldsymbol{q}_{i,h})$, local obstacle repulsion $\boldsymbol{n}$ and the distance-to-goal $||\boldsymbol{q}^* - \boldsymbol{q}_{i,h}||_2$ are evaluated. We may use this exploration to better place the navigation kernels, and add them dynamically as new obstacles are detected along the sampled trajectories.

If stable region of the nominal DS attractor is not reached yet, the distance-to-collision is less than a threshold parameter, and system is within the region of possible local minima a new navigation kernel can be placed. Local minima detection can be characterized as scalar product value $\langle \boldsymbol{n}(\boldsymbol{q}), \hat{f}(\boldsymbol{q}) \rangle$ being close to -1, where $\hat{f}(\boldsymbol{q}) = \frac{\boldsymbol{f}(\boldsymbol{q})}{||\boldsymbol{f}(\boldsymbol{q})||}$ is a normalized direction of nominal DS, and $\boldsymbol{n}(\boldsymbol{q})$ is obstacle normal. Additionally we consider the proximity to the existing set of K navigation kernels. Overall, the criteria for placing a new navigation kernel are:

$$\{\hat{\boldsymbol{q}}_k\}_{k=1}^{K+1} = \boldsymbol{q}_{i,h} \cup \{\hat{\boldsymbol{q}}_k\}_{k=1}^{K} \tag{5.28a}$$

$$\Updownarrow$$

$$\begin{cases} ||\boldsymbol{q}^* - \boldsymbol{q}_{i,h}||_2 > \delta_*, & \Leftrightarrow \text{ Far from the nominal attractor.} \\ \Gamma(\boldsymbol{q}_{i,h}) < \delta_\Gamma, & \Leftrightarrow \text{ Close to collision with obstacle.} \\ \langle \boldsymbol{n}(\boldsymbol{q}), \hat{f}(\boldsymbol{q}) \rangle < -1 + \delta_n, & \Leftrightarrow \text{ Possible local minima for standard modulation.} \\ \min\limits_{k=1..K} ||\boldsymbol{q}_{i,h} - \hat{\boldsymbol{q}}_k||_2 > \delta_k. & \Leftrightarrow \text{ Far from existing navigation kernels.} \end{cases} \tag{5.28b}$$

Parameters $\delta_*, \delta_\Gamma, \delta_n, \delta_k$ are strictly positive and their values determine the density of the navigation kernels placed during exploration. Notably, some conditions of kernel placing duplicate the mechanisms of kernel activations α_Γ, α_n and α_f. In other words, even if a kernel is placed close to attractor $\boldsymbol{q}^*$, it would not get activated, because $\alpha_f = 0$. However, such redundancy provides more tunability for the system, and reduces the overall amount of navigation kernels placed.

After the new candidate for a navigation kernel is found, it is added to the existing set of kernels, centered at $\hat{\boldsymbol{q}}_{k+1} = \boldsymbol{q}_{i,h}$, has kernel width $\gamma_k = \hat{\gamma}$, and tangential direction $\boldsymbol{g}_{k+1} = \boldsymbol{\mu}_{k^*}$,

where k^* stands for the index of closest existing kernel: $k^* = \min_k ||\boldsymbol{q}_{i,h} - \hat{\boldsymbol{q}}_k||_2$. This efficiently warmstarts the optimization of kernel parameters with the parameters of the closest existing kernel.

The sampling policy is also changed to include the new mean $\boldsymbol{\mu}_{k+1} = \boldsymbol{\mu}_{k^*}$ and covariance $\boldsymbol{\Sigma}_{k+1} = \boldsymbol{\Sigma}_{k^*}$. We preserve the similar parameters between closest navigation kernels to avoid undesired oscillatory trajectories. The described MPPI method is detailed in Algorithm 1 in Appendix B.4.

5.6 Implementation Details

5.6.1 Tail Effect Compensation

Modulation $\boldsymbol{M}(\boldsymbol{q})$ defined in Equation (5.5) reshapes the vector field according to defined obstacle basis $\boldsymbol{E}(\boldsymbol{q})$ and weights matrix $\boldsymbol{D}(\boldsymbol{q})$. This modulation only depends on the distance to collision $\Gamma(\boldsymbol{q})$ and does not take into account the direction of the vector field. This can lead to a situation where the robot is slowed down by the modulation even if it is moving away from the obstacle. This effect is called the tail effect (Khansari-Zadeh and Billard, 2012a). In order to compensate for this effect, we redefine normal direction gain $\lambda_n(\boldsymbol{q})$ as follows:

$$\lambda'_n(\boldsymbol{q}) = \lambda_v(\boldsymbol{q}) + (1 - \lambda_v(\boldsymbol{q}))\lambda_n(\boldsymbol{q}), \tag{5.29}$$

where $\lambda_v(\boldsymbol{q})$ is a sigmoid function that is defined as:

$$\lambda_v(\boldsymbol{q}) = \frac{1}{1 + \exp\left(-100\langle \boldsymbol{n}(\boldsymbol{q}), \frac{\boldsymbol{f}(\boldsymbol{q})}{||\boldsymbol{f}(\boldsymbol{q})||}\rangle\right)}, \tag{5.30}$$

such that $\lambda_v(\boldsymbol{q})$ continuously indicates whether the nominal vector field is pointing towards the obstacle ($\lambda_v(\boldsymbol{q}) = 0$) or away from it ($\lambda_v(\boldsymbol{q}) = 1$). Thus, $\lambda'_n(\boldsymbol{q})$ is equal to $\lambda_n(\boldsymbol{q})$ if the nominal vector field is pointing towards the obstacle, and is equal to 1 if the nominal vector field is pointing away from the obstacle, even if the distance to collision is small.

5.6.2 Navigation Kernel Activation Implementation

For the local activation of navigation kernels, we define parameters $\alpha_\Gamma(\boldsymbol{q}), \alpha_n(\boldsymbol{q})$ and $\alpha_f(\boldsymbol{q})$ from Equation (5.15) using the λ_n and λ_v.

$$\begin{aligned} \alpha_\Gamma(\boldsymbol{q}) &= 1 - \lambda_n(\boldsymbol{q}), \\ \alpha_n(\boldsymbol{q}) &= 1 - \lambda_v(\boldsymbol{q}), \\ \alpha_f(\boldsymbol{q}) &= max(1, ||\boldsymbol{q} - \boldsymbol{q}^*||). \end{aligned} \tag{5.31}$$

With such definition, even though navigation kernels are primarily activated by RBFs, they are disabled for situations where distance to collision is large, essentially compressing RBFs closer to the obstacle. Additionally, the tail effect is compensated, as $\alpha_n(\boldsymbol{q})$ is equal to 0 if the nominal vector field drives the robot away from the obstacle. Note, that $\alpha_\Gamma(\boldsymbol{q})$ and $\alpha_n(\boldsymbol{q})$ could be merged into a single function using previously defined $\lambda'_n(\boldsymbol{q})$.

5.6.3 Navigation Kernel Tangentiality

As stated in Section 5.4, we define navigation kernel dynamics to be locally tangential to the obstacle boundary. While parameters $\boldsymbol{g}_k$ are sampled from a full d-dimensional Gaussian distribution, we may impose the following transformation on the final policy:

$$\boldsymbol{g}(\boldsymbol{q}) := \boldsymbol{M}^{\perp}(\boldsymbol{q})\boldsymbol{g}(\boldsymbol{q}), \tag{5.32}$$

where $\boldsymbol{M}^{\perp}(\boldsymbol{q})$ is constructed as $\boldsymbol{M}^{\perp} = \boldsymbol{E}\boldsymbol{D}^{\perp}\boldsymbol{E}^{T}$, and $\boldsymbol{D}^{\perp}$ is an identity matrix with zero as the first element of the first column. With such transformation it is guaranteed that $\boldsymbol{g}(\boldsymbol{q}) \perp \boldsymbol{n}(\boldsymbol{q})$. While this orthogonality is not crucial for obstacle avoidance, as by design any vector field $\boldsymbol{g}(\boldsymbol{q})$ is modulated by $\boldsymbol{M}(\boldsymbol{q})$, it is still a nice property that restricts navigation kernels to only provide meaningful tangential deflection, and not modulate the DS motion in undesired directions.

5.6.4 Discrete System Obstacle Impenetrability

Theorem 1 ensures that continuous dynamics defined by Equation (5.11) do not penetrate the obstacle boundary $\Gamma(\boldsymbol{q}) = 0$. However, the Neumann condition is insufficient to guarantee impenetrability for discrete-time dynamics, especially when the integration step is large in comparison to the speed of displacement of the robot, as this can often be the case in real-world robotics applications. Specifically, for convex obstacles, $\boldsymbol{n}(\boldsymbol{q})^T\dot{\boldsymbol{q}}$ on the obstacle boundary does not guarantee impenetrability, as the system can be brought through the obstacle boundary if the timestep δt is large and the previous state is not on the boundary (thus possibly having a nonzero velocity component along the obstacle normal). For concave obstacles, any non-infinitesimal displacement in a tangential plane while on the obstacle boundary within the concavity will also violate the obstacle boundary.

In practical applications, ensuring the non-penetrability of obstacles requires additional considerations. Apart from issues arising from numerical integration, the obstacle's normal is provided by a learned network, which only approximates the normal, thus not guaranteeing impenetrability. To mitigate that, a combination of safety threshold ε, local obstacle repulsion field $\boldsymbol{f}_{\text{rep}}$, and a sufficiently small numerical integration step δt is necessary.

By design, for system (5.11), $\dot{\boldsymbol{q}} \perp \boldsymbol{n}(\boldsymbol{q})$ for all $\boldsymbol{q} : \{\Gamma(\boldsymbol{q}) \leq 1cm\}$. In applications with discretized system integration, the robot state may still penetrate the safety threshold $\Gamma(\boldsymbol{q}) = \varepsilon = 1cm$. When this occurs, a small repulsive force, $\boldsymbol{f}_{\text{rep}}(\boldsymbol{q}) = k_{\text{rep}}\boldsymbol{n}(\boldsymbol{q})$, is added to the right-hand side of Equation (5.11) to move the state away from the obstacle boundary and maintain the safety

threshold. Note that $\boldsymbol{f}_{\text{rep}}(\boldsymbol{q}) = 0$ for all $\boldsymbol{q} : \{\Gamma(\boldsymbol{q}) \geq 1cm\}$.

Taking these factors into account, impenetrability for obstacles in discrete system integration can only be violated if a single integration step results in $\Gamma(\boldsymbol{q}_t) \geq \varepsilon$ transitioning to $\Gamma(\boldsymbol{q}_{t+1}) < 0$, or if the repulsion magnitude k_{rep} is insufficient. The first case can be addressed by reducing the numerical integration step δt to an adequately small value. As our algorithm operates at approximately 500Hz, the value of δt is set to 0.002, ensuring that the integration steps are small enough to prevent overcoming the safety threshold in a single step.

Assuming that the system integration is performed using the Euler method, the magnitude k_{rep} can be estimated by calculating the current undesired penetration through the threshold: $\varepsilon - \Gamma(\boldsymbol{q})$ and introducing the offset with a timestep correction: $k_{\text{rep}} = \dfrac{\varepsilon - \Gamma(\boldsymbol{q})}{\delta t}$. As a result, the repulsion force added to offset the threshold penetration is defined as

$$\boldsymbol{f}_{\text{rep}}(\boldsymbol{q}) = \begin{cases} \dfrac{\varepsilon - \Gamma(\boldsymbol{q})}{\delta t}\boldsymbol{n}(\boldsymbol{q}), & \forall \boldsymbol{q} : \Gamma(\boldsymbol{q}) < \varepsilon \\ 0, & \text{otherwise.} \end{cases} \tag{5.33}$$

The threshold $\varepsilon = 1cm$ (defined in Equation (5.13)) in combination with a small timestep $\delta t = 0.002$ ensures impenetrability of the obstacle boundary $\Gamma(\boldsymbol{q}) = 0$ in our practical application.

Overall, the continuous dynamics (5.11) take the following discrete form:

$$\dot{\boldsymbol{q}}_t = \boldsymbol{M}(\boldsymbol{q}_t)\left(\boldsymbol{f}(\boldsymbol{q}_t) + \alpha(\boldsymbol{q}_t)\boldsymbol{M}^{\perp}(\boldsymbol{q}_t)\boldsymbol{g}(\boldsymbol{q}_t)\right) + \boldsymbol{f}_{\text{rep}}(\boldsymbol{q}_t) \tag{5.34}$$

$$\boldsymbol{q}_{t+1} = \boldsymbol{q}_t + \dot{\boldsymbol{q}}_t \delta t. \tag{5.35}$$

5.6.5 Algorithm Implementation

Authors of Bhardwaj et al. (2022) report frequencies of up to 125 Hz for sampling-based MPC scheme similar to ours, however, it is only possible due to simple integration of the roll-outs, where the state is linear function of the control input. That enables the evaluation of the state sequence by means of lower-triangular integration tensor, achieving parallelization both across multiple timesteps and multiple samples. However, we rely on $\Gamma(\boldsymbol{q})$ evaluation for each timestep for modulation matrix calculation, making the system nonlinear, and restricting parallel computation in time domain. Mainly because of that our MPC is limited to 10-30 Hz depending on number of samples, obstacles and horizon length. The main computation bottleneck of our algorithm is the evaluation of $\Gamma(\boldsymbol{q})$ and $\Delta\Gamma(\boldsymbol{q})$, followed by QR decomposition of obstacle normal to construct the tangential space basis. The latter is not comparable in amount of operations with MLP evaluation, however, it restricts the use of algorithm to CPU.

While the method leverages the parallel execution for multiple exploration samples, and is fully suitable for GPU implementation, we found that QR decomposition of obstacle bases can only

be efficient on CPU, thus all setup runs exclusively on CPU. Newer generations of processors with relatively large cache allow quick computation of the MLP, and overall performance is satisfactory. Notably, more traditional methods to evaluate distance to collision and repulsive gradient would lead to even more significant slowdowns.

As such frequencies may still not be sufficient for environments where obstacles are not static, we leverage the modular structure and concurrent execution to improve the performance. A simple one-sample one-timestep integration of modulated DS is not as computationally expensive and can be performed with frequencies up to 700 Hz. We asynchronously compute the modulated DS with latest known Modulation Policy, and stream the data to the low-level impedance controller that is executed at 1 kHz. This allows us to achieve real-time performance in dynamic environments.

Importantly, the MPC does not need to operate continuously in the background, using up computational resources. The MPC sampling is activated only if the predicted robot state comes close to navigation kernels, and it is not engaged in situations where there is no risk of local minima induced by obstacles. In practice, the MPC can run asynchronously at all times, but the number of samples can be increased dynamically. Generally, a single sample is required to predict the future trajectory and identify local minima. Once the minima is detected and a navigation kernel is placed, the number of samples increases to determine the optimal deflection direction. When the nominal propagated motion encounters no local minima (or navigation kernels), the sample complexity can be reduced back to a single sample for trajectory prediction. This pattern is further explored in Section 5.7.

Nominal DS is a parameter to both Modulated DS module and MPC module, and obstacles postions are streamed using Optitrack at 120 Hz. Notably, we do not use ROS for streaming, instead we opt for a lightweight setup based on ZeroMQ publisher-subscriber sockets (Hintjens, 2013).

Overall flowchart of the implemented algorithm is shown in Figure 5.3. All frequencies are reported for Apple Silicon M2 3.7 GHz CPU.[1]

5.6.6 Computational Complexity

The proposed hybrid controller combines two underlying algorithms - DS and MPC. For both algorithms, the major part of computations is related to state propagation, particularly the computation of the Modulation matrix, which depends on the implicit distance function. This function is represented by a Multi-Layer Perceptron (MLP) with analytical computational complexity $O(s \cdot d \cdot n + m \cdot n^2)$, where s is the number of input queries; d is the input dimension (robot DoFs); m is the network depth; and n is the number of neurons per layer. Additionally, the number of inputs $s = N \cdot p$ can be expressed as the product of the number of robot states N and the number p of collision spheres (or points) in the robot's environment. The MPC

[1] The source code for this chapter is available at `https://github.com/epfl-lasa/OptimalModulationDS`

algorithm performs these computations sequentially for H steps along the horizon, resulting in a total computational complexity for the sampling-based MPC of $O(H(s \cdot d \cdot n + m \cdot n^2))$. Although the integration steps in MPC can be parallelized for multiple states and obstacles, the nonlinear integration procedure requires sequential evaluation of the time dimension. Consequently, the MPC evaluation frequency does not exceed 30 Hz in our experiments.

The DS can be considered as a single-step, single-sample MPC, and its computational complexity can then be expressed as $O(d \cdot n + m \cdot n^2)$. Given the relatively small size of the MLP used ($n = 256, m = 4$), the DS can be evaluated at frequencies up to 500-1000Hz.

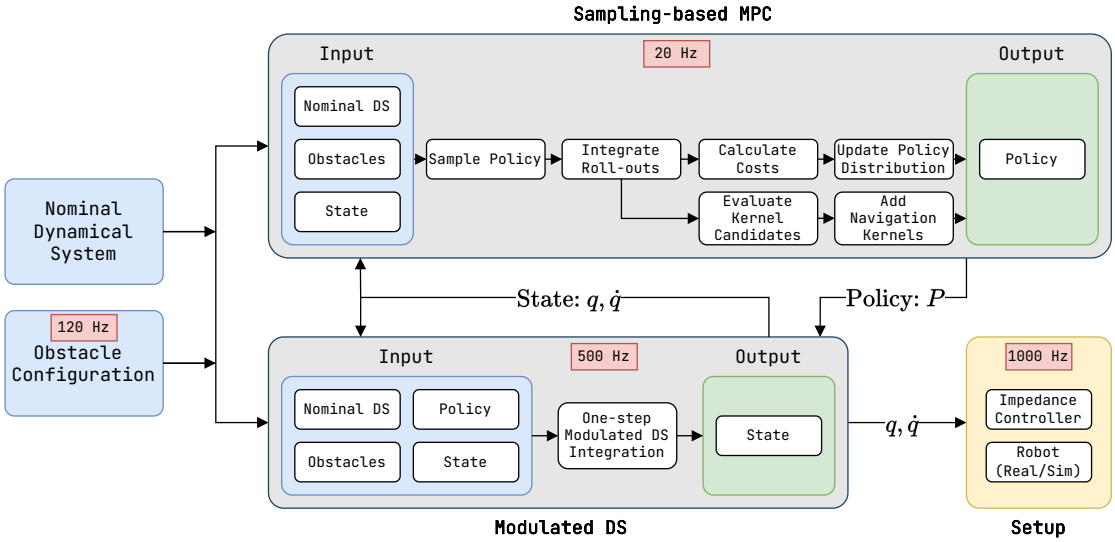


Figure 5.3: Flowchart of the implemented algorithm. Message passing between modules is performed asynchronously. Approximate operating frequencies for each module (shown in red boxes) are estimated for execution on Apple Silicon M2 3.7 GHz CPU.

5.7 Evaluation

We systematically evaluate the proposed method in a simulated environment on a 7-DoF Franka Panda robot. We aim to compare the performance of our approach with two state-of-the-art methods that are capable of reactive robot motion towards the given goal configuration while avoiding collisions with obstacles in the robot's workspace. Specifically, we compare the methods based on their ability to avoid collisions, achieve the goal and run at a high frequency. Additionally, we demonstrate the application of our method for a nonlinear joint space DS both in simulation and on a real robot.

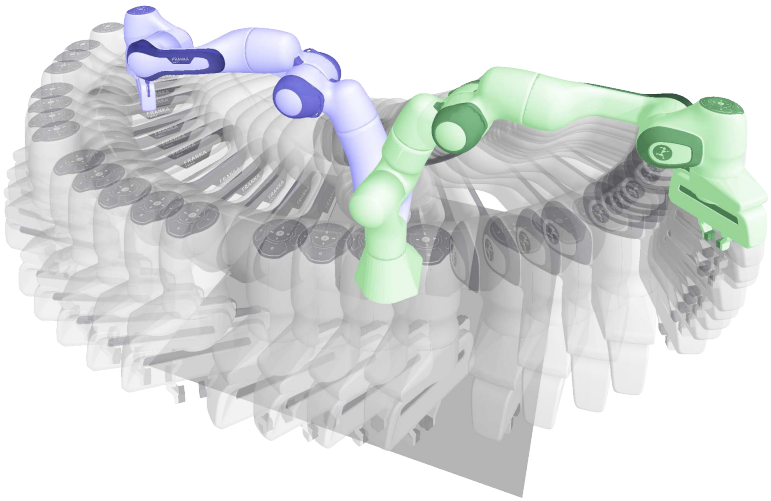


Figure 5.4: Reaching trajectory used in the benchmark. Robot is driven from the initial position (in blue) to the goal position (green) by linear DS defined in joint space. Concave obstacles are placed in the swept area in front of the robot to perform collision-avoidance benchmark.

5.7.1 Experimental Setup

We consider a reaching task, in which the robot has to perform a reaching motion towards a goal configuration while avoiding obstacles in the workspace. We compare our method with a standard DS modulation framework introduced in (Khansari-Zadeh and Billard, 2012a), and with a sampling-based MPC method (STORM) introduced in (Bhardwaj et al., 2022). We define a single desired joint state $\boldsymbol{q}^*$ and use a simple linear dynamical system as a nominal DS:

$$\dot{\boldsymbol{q}} = -\frac{\boldsymbol{q} - \boldsymbol{q}^*}{||\boldsymbol{q} - \boldsymbol{q}^*||}, \tag{5.36}$$

that is normalized to achieve uniform velocity profile during the task execution. The normalization is not performed when the system is close to the attractor, to avoid numerical instabilities.

Standard modulation and our method are designed to act in the joint space given some nominal DS, however the STORM implementation considers costs defined in task space (i.e., end-effector position and orientation). To enable a fair comparison, we redefine the goal reaching and orientation costs as a quadratic cost similar to Equation (5.20). This emulates the linear DS motion in the joint space, and allows us to compare the performance of methods on the same task. We use the same cost weights and hyperparameters as provided in the publicly available implementation[1] of STORM. Additionally, we leverage our previous work (Koptev et al., 2023) that introduced a Neural Network based Γ-function for collision checking, which improves the performance (in terms of frequency) of the method.

[1] https://github.com/NVlabs/storm

To make meaningful comparison in the reaching scenario, we design the experiment such that final goal state q^* describes position in the left part of robot's workspace, and initial position q_{init} is located in the right. To move between these two positions robot has to traverse through the workspace in front of it, as shown in Figure 5.4.

Table 5.1: Success rate comparison for three methods involving robot motion in obstructed environment with various obstacle sizes. Values are averaged between 100 runs, and means (and standard deviations) are reported.

Method/Obstacle Size	**2**	**3**	**4**	**5**	**6**	**7**
Ours	**1.00**	**1.00**	**1.00**	**1.00**	**0.99**	**0.98**
ModDS	1.00	1.00	0.86	0.72	0.65	0.62
STORM	1.00	0.98	0.82	0.81	0.80	0.79

Table 5.2: Number of iterations comparison for three methods involving robot motion in obstructed environment with various obstacle sizes. Values are averaged between 100 runs, and means (and standard deviations) are reported.

Method/Obstacle Size	**2**	**3**	**4**	**5**	**6**	**7**
Ours	446 (23)	476 (47)	524 (83)	563 (144)	571 (154)	626 (391)
ModDS	451 (39)	481 (51)	534 (113)	535 (79)	554 (102)	556 (104)
STORM	**416 (73)**	**452 (89)**	**477 (75)**	**467 (86)**	**463 (119)**	**454 (95)**

Table 5.3: Trajectory time (in seconds) comparison for three methods involving robot motion in obstructed environment with various obstacle sizes. Values are averaged between 100 runs, and means (and standard deviations) are reported.

Method/Obstacle Size	**2**	**3**	**4**	**5**	**6**	**7**
Ours	0.82 (0.06)	0.96 (0.12)	0.97 (0.19)	1.18 (0.37)	1.21 (0.37)	1.36 (0.94)
ModDS	**0.74 (0.06)**	**0.86 (0.09)**	**0.86 (0.19)**	**0.88 (0.12)**	**0.98 (0.18)**	**0.93 (0.18)**
STORM	4.85 (0.98)	5.34 (1.08)	6.46 (1.04)	6.05 (1.22)	5.54 (1.48)	6.26 (1.27)

For the benchmark, we consider a static cross-shaped sphere structure placed in front of the robot. We investigate methods' performance for various sizes of the obstacle. Some of the sizes are shown in Figure 5.5. In general, as the size of the obstacle increases, the robot's workspace becomes more constrained. For each obstacle size we vary the height of the cross center and its distance from the robot across 100 reaching motions. Benchmark results are presented in Tables 5.1, 5.2, 5.3 and 5.4. We consider reaching successful if it is performed within 10 seconds. Otherwise, we do not consider it in calculations of average number of iterations and trajectory time.

Table 5.4: Frequency (in Hz) comparison for three methods involving robot motion in obstructed environment with various obstacle sizes. Values are averaged between 100 runs, and means (and standard deviations) are reported.

Method/Obstacle Size	**2**	**3**	**4**	**5**	**6**	**7**
Ours (DS)	22 (1)	22 (3)	21 (2)	22 (2)	20 (1)	19 (2)
Ours (MPC)	548 (16)	498 (14)	546 (47)	485 (26)	473 (19)	468 (20)
ModDS	**613 (5)**	**562 (5)**	**622 (17)**	**607 (9)**	**567 (7)**	**598 (3)**
STORM	86 (7)	84 (8)	74 (3)	76 (4)	75 (3)	74 (7)

Table 5.5: MPC activation metrics for the benchmark experiments. The columns indicate obstacle size, the number of trajectories where more than one navigation kernel was placed (i.e., MPC was required), and the corresponding proportion of iterations where the path passed by the navigation kernels along with number of kernels placed. Values are averaged over 100 runs, with means (and standard deviations) reported. MPC activation rate and number of kernels is averaged only for trajectories where MPC was active.

Obstacle Size	**2**	**3**	**4**	**5**	**6**	**7**
Trajectories requiring MPC, rate	0.11	0.11	0.28	0.41	0.51	0.56
MPC Activation, rate	0.42 (0.16)	0.53 (0.18)	0.64 (0.19)	0.72 (0.16)	0.72 (0.15)	0.73 (0.15)
Number of kernels	1.8 (0.8)	2.7 (0.8)	2.8 (1.5)	2.6 (1.9)	2.7 (1.9)	2.7 (2.5)

5.7.2 Discussion

The effectiveness of our proposed method in navigating in the presence of obstacles is demonstrated in Tables 5.1-5.4, where it outperforms two other methods with consistently higher success rates. While the planning part of our algorithm is slower than STORM, the running frequency is comparable to that of a standard DS approach. It is worth noting that our algorithm runs on a CPU, while STORM efficiently leverages GPUs. However, the slow planning loop is not crucial for our method, as the primary modulated DS achieves motion reactivity at a high frequency. We should also note that this benchmark only evaluates the algorithms in quasi-static environments and does not consider their performance in dynamic scenarios. Future work could address this limitation by incorporating obstacle velocities into collision-avoidance algorithms. Nonetheless, our method is capable of reacting to dynamic changes in the environment due to its high frequency of operation, as demonstrated in experiments with a real robot.

The performance parameters of the MPC component are displayed in Table 5.5. As mentioned earlier, the MPPI-based sampling policy optimization is only necessary when the robot trajec-

Figure 5.5: Two-, three-, and four- spheres long obstacle cross configurations used in the benchmark.

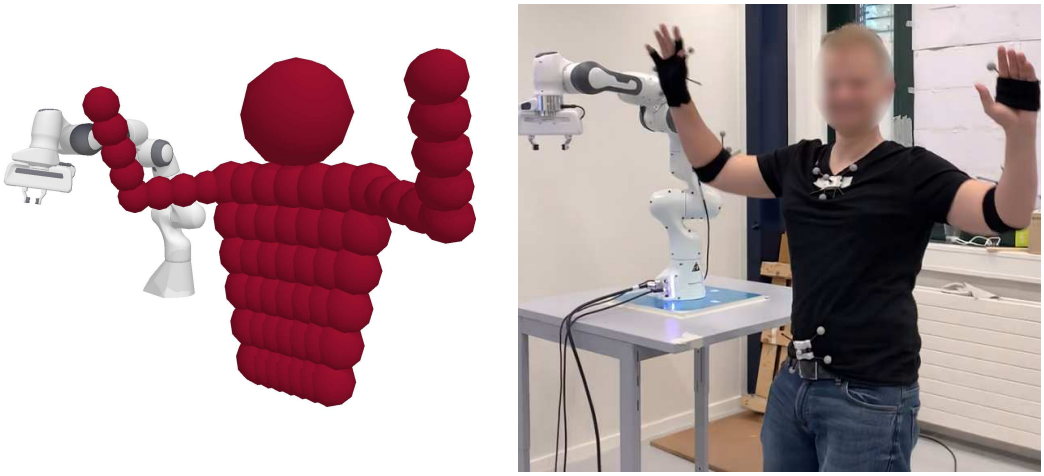


Figure 5.6: Spherical approximation of human upper body. 70 spheres of variable radii represent head, torso and arms. Optitrack markers are used to determine six keypoints located on the human body.

tory comes close to possible local minima induced by obstacles. The number of trajectories where the MPC is invoked to explore exit policies increases with environmental complexity. However, in certain situations (e.g., when the obstacle is not very close to the robot), the MPC might be unnecessary as the nominal DS delivers required collision-free reaching motion. Furthermore, for trajectories utilizing the MPC, it does not have to be employed throughout the entire trajectory but can be applied only when the robot passes by the obstacle. The corresponding MPC activation rate parameter grows with the amount of obstacles in the robot's workspace. Lastly, Table 5.5 reveals that an average of 2-5 navigation kernels is required to complete the reaching task in the proposed benchmark scenario. This value can be adjusted by configuring the hyperparameters such as kernel width $\hat{\gamma}$ and kernel placement thresholds $\delta_{\{\Gamma,n,k\}}$.

In the case of a dynamic unstructured environment, it is possible that previously placed navigation kernels may become irrelevant after obstacles have moved. This issue is addressed

in our method through the coefficient α_Γ (as defined in Equation (5.31)) that deactivates the deflection as obstacle moves away. If an obstacle once again comes close to such a kernel, MPPI will automatically begin adjusting the kernel's parameters.

5.7.3 Real Robot Experiments

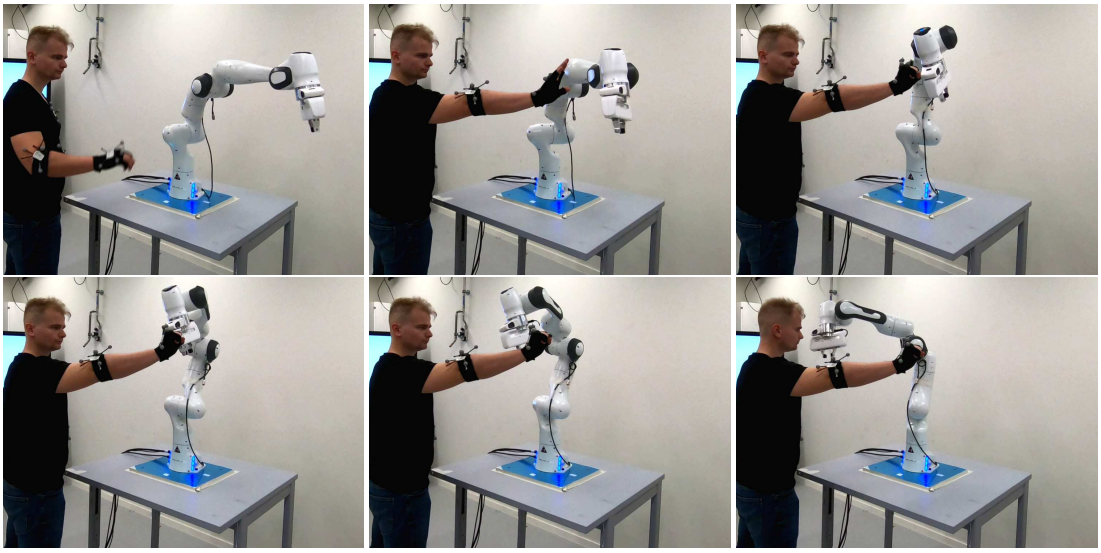
We demonstrate the reactive collision-avoidance properties of our method on a setup involving Franka Emika. We use a simple reaching trajectory similar to the one used in simulated benchmark. We then command the robot to alternate between these nominal dynamics to perform cyclic motion between two attractors following the reaching pattern. We fit a human shape with 70 spheres (see Figure 5.6) with variable radii, and randomly obstruct the robot workspace during the task execution. The keypoints on the human are tracked with OptiTrack at a 120Hz rate. Modulated DS with latest available policy is evaluated at frequency of at least 500Hz, and MPPI asynchronously updates the navigation policy at approximately 20Hz. The robot is then controlled at 1 kHz by a custom low-level torque controller. Figure 5.7 demonstrates the snapshots of the experiment in which human dynamically appears in the robot workspace and obstructs the motion.

5.7.4 Moving Obstacles

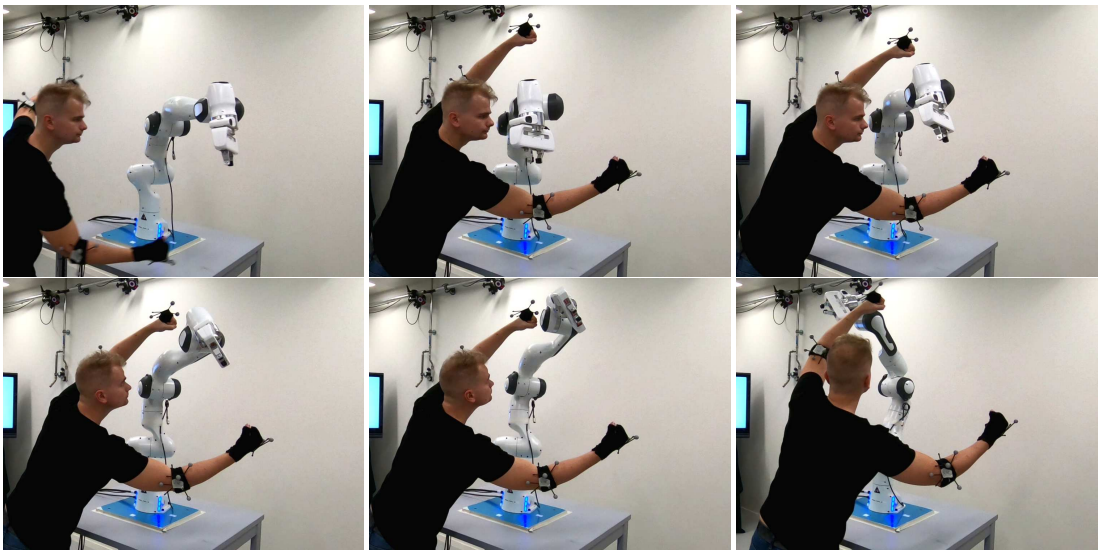
The designed hybrid controller can provide collision-free motion, even in the presence of moving obstacles. One of the key assumptions in this thesis is that the environment is quasi-static, where the controller does not consider the motion law of the obstacles. Given that the update rate of the Modulated DS exceeds 500 Hz, the controller's reaction time is approximately 2ms. Assuming that obstacles are located within the sensing range of the OptiTrack system and that the robot's reaction time is negligible, we can estimate the maximum permissible obstacle speed. The controller initiates collision avoidance repulsion when an obstacle is closer than 1 cm (see Sec. 5.6.4). With this in mind, our algorithm is suitable for scenarios where obstacles move at speeds slower than 1 cm per 2 ms, which translates to 5 m/s. While this is a significantly high speed, the critical assumption of negligible robot reaction time typically does not hold in practical scenarios. Specifically, the Franka robot's end-effector is limited to a velocity of 2 m/s, implying that the robot cannot avoid a collision if the obstacle is moving at speeds greater than 2 m/s.

5.8 Conclusion and Future Work

We have presented a method to introduce deflections into a Modulated Dynamical System to enhance the navigation capabilities of the original approach proposed by Khansari-Zadeh and Billard (2012a). Our method retains the properties of the Modulated DS, such as local stability and non-penetrability of obstacle boundaries.



(a)



(b)

Figure 5.7: Two experiments in which human swiftly obstructs the robot motion with (a) raised arm, and (b) concave arm trap. Robot reactively avoids the collision and navigates the concavity, continuing the reaching task execution. Time difference between subsequent images is approximately 700 milliseconds.

We utilize Sampling-based Model Predictive Control to optimize the deflection parameters and leverage the Neural Signed Distance Function for real-time performance in the 7-dimensional joint space of a robotic manipulator.

Our proposed hybrid controller enables real-time, collision-free motion generation for any nominal DS motion. The underlying modulated DS can effectively control collision-free robot movement at frequencies up to 600 Hz, while the navigation policy updates at 20 Hz, allowing

for pathplanning in non-static and rapidly changing environments.

Future work could address some limitations of our approach, such as the generation of motion exclusively in joint-space. While this is natural for robots, it may not be ideal for real-world tasks. The hybrid controller could be enhanced by adding an Inverse Kinematics layer that calculates the desired final joint positions for each robotic task. Additionally, considering multiple DS attractors could fully exploit the redundancy of robot kinematics.

Another potential improvement involves developing a fully dynamic controller that considers not only obstacle positions but also their velocities, adding a new level of reactivity to the generated motions. This approach could be further refined by incorporating filtering and prediction of obstacle movements, enhancing the MPC policy generation.

6 Conclusions

In this chapter, we summarize and review the main contributions of this thesis. We then discuss the limitations of the proposed methods and outline the potential future work.

6.1 Contributions

This thesis draws inspiration from kinesthesia, the inherent sense of self-movement, bodily force, and positional awareness. Proprioceptors, that are specialized neurons located within muscles, tendons, and joints, enable humans and animals to perceive their body and limb positions. This internal perception operates independently from external senses like vision and touch. The proprioceptive system is fundamental for our internal body schema, that serves as the basis for controlling human body movements (Head and Holmes, 1912). Humans develop a model of their body immediately after birth, and continuously refine it throughout their lifespan (Hoffmann et al., 2010). With this concept in mind, we focused on developing a collision avoidance system that is based on learned collision models.

The primary contribution of this dissertation is the provision of a generic framework for robot collision avoidance, which is based on implicit distance functions. We outlined a data-driven approach for learning collision detection functions and demonstrate their application in various algorithms that generate collision-free motion. First, we concentrated on self-collision detection in redundant humanoid robots and subsequently extend this approach to general collision avoidance. Finally, we developed a hybrid controller that leverages the properties of the learned collision representations to achieve reactive, collision-free motion generation for redundant robots.

In Chapters 1 and 2, we offered an introductory overview of the robot collision avoidance problem, addressing the state-of-the-art in collision detection and motion planning. While collision detection has its origins in computer graphics and geometry, motion planning primarily belongs to the field of robotics. We discussed the limitations of existing methods and outline potential directions for ML-based techniques.

In Chapter 3, we proposed a method that utilizes data-driven machine learning techniques to construct a self-collision classifier. We compared the performance of SVM and NN methods applied to this classification task. The constructed classifier takes joint positions of a redundant robot as input and provides a binary output indicating the robot's self-collision state. Since the underlying decision function is continuously differentiable, we can acquire gradients with respect to the input (joint space angles). These gradients can be efficiently used to formulate a collision avoidance constraint for a motion generation algorithm. We used the QP IK method to compare our learned function with a more traditional approach relying on Jacobians to translate repulsion constraints into the robot's joint space. We validated the method on a 32-DoF humanoid robot, iCub, in both simulation and real-world experiments, demonstrating that the learned collision model can generate collision-free motions for the robot. To our knowledge, our work represents one of the first instances of learning a self-collision function in high-dimensions (>30 degrees of freedom) and its application to a whole-body control of a humanoid robot.

In Chapter 4, we built upon the proposed self-collision avoidance framework and extend it to account for collisions with objects in the environment. We employed a data-driven approach to construct a collision detection function, posing the problem as a regression task. Our goal is to learn minimal distances between the robot and its environment as a function of robot state and environment point coordinates. Consequently, we obtained an implicit signed distance approximator that representats the robot's signed distance field in continiously differentiable way. We used this function to formulate the collision avoidance constraint in QP IK, demonstrating real-time collision-free motion generation. To overcome the local nature of this technique, we leveraged the parallelization properties of NNs, using our implicit distance function to check for collisions in Sampling-Based MPC controllers. We validated the method on a 7-DoF Franka Emika Panda robot, showcasing collision-free motion in reaching tasks. This chapter further advances the learning-based approach to collision detection and establishes the foundation for implicit distance representation in robotics, along with examples of its application. Our method is among the first to utilize learned distances to represent robot collision geometry, and we anticipate that this approach will gain increased adoption in the future.

In Chapter 5, we addressed the limitations of the motion generation methods employed in previous chapters by introducing a hybrid controller. This controller combines the reactivity of the Modulated DS approach, generating collision-free motions at high frequencies (up to 1 kHz), with the exit strategies provided by the Sampling-Based MPC method. The learned collision and self-collision models serve as integral components of the proposed method. We utilized parallel distance computations to increase the sampling frequency of the MPC and rely on collision gradients to modulate the DS for collision avoidance. We validated the method on a 7-DoF robotic arm, demonstrating swift and reactive motion generation, as well as the ability to seamlessly recover from concave traps surrounding the robot. Although the concept of a hybrid approach controller, which combines fast and slow algorithms, is not entirely new, we believe that our work is the first to integrate a DS-based obstacle avoidance

method with Model Predictive Control. We demonstrate the efficiency and potential of such an approach in robotics.

6.2 Limitations and future work

The work presented in this thesis provides a comprehensive framework for collision-free motion generation, encompassing both collision model learning and motion generation algorithms. Nonetheless, there remain some limitations that future research could address.

The proposed method for learning self-collisions can be applied to any robot with a known kinematic model but works best for robots with fewer DoFs, such as robotic arms. With an increasing number of DoFs, the collision boundary learning precision degrades, necessitating the division of a 32-DoF humanoid robot into ten submodels. Due to the symmetry properties of the robot's kinematics, only six models need to be learned; however, this still increases the method's complexity. Various techniques, such as data augmentation or model architecture optimization, could improve the method's performance for higher DoF robots.

Similar considerations apply to the collision avoidance method presented in Chapter 4, where the implicit distance function is learned. It is important to consider the trade-off between model complexity and inference speed. Increasing the number of parameters (i.e., layers and neurons) or incorporating architectural features like batch normalization or various regularization techniques could enhance the method's performance. However, this would also increase computation time, which is crucial for real-time motion generation.

The extended method for computing distances between the robot and obstacles in the environment requires a point coordinate to estimate the distance. In this thesis, we assumed that each obstacle could be approximated with a set of spheres, calculated distances to the sphere centers, and subtracted the radii afterward. We investigated situations involving 10-100 spheres in the robot workspace and demonstrated real-time performance. For the limit case of null radii, the obstacle representation can be considered as a point cloud, suitable for the majority of robotic tasks. However, the performance of distance computation will suffer when facing a large number of points common for point-cloud representations, hindering real-time performance. Additional techniques, such as introducing a broad phase with crude spherical approximation and a narrow phase with more precise point clouds, could be employed to enhance the method's performance.

The learned implicit distance function assumes that precise obstacle positions are known. While this thesis demonstrates experiments using an OptiTrack motion tracking system in a laboratory environment, such considerations may be unrealistic in real-world scenarios. Environment point clouds may be obtained using LiDARs or RGB-D cameras, or, alternatively, modern computer vision techniques enhanced with deep learning methods can estimate obstacle positions using standard cameras. Integrating such components into the developed collision avoidance framework would enable its application in real-world situations.

The hybrid controller presented in Chapter 5 considers a quasi-static scenario, relying on the controller to update the motion plan at high frequency (above 500 Hz) and adapt to swift changes in obstacle positions. However, for truly dynamic obstacle avoidance, obstacle velocities or even accelerations should be considered when generating a robot motion plan. For example, MPC may consider predicted obstacle positions along the integrated trajectory of the robot state in the rollouts. Additionally, obstacle velocity can be compared with the learned model gradients to determine whether the obstacle is approaching or receding from the robot, and thus inform the robot's avoidance strategy. Developing such a method, along with filtering and obstacle position prediction, would enable the controller's use in highly dynamic scenarios.

A Technical Preliminaries on Dynamical Systems

A.1 Dynamical Systems

In this thesis, we utilize dynamical systems (DS) to define the desired motion of the robot. Moreover, we adopt the modulation approach, introduced in Khansari-Zadeh and Billard (2012a), to reshape the DS in the vicinity of obstacles, enabling collision-free navigation. While these concepts are briefly discussed in Chapter 5, this appendix offers a more in-depth examination of the DS stability and modulation[2]

A continuous DS is typically represented by a set of differential equations that depend on time $t \in \mathbb{R}^+$, a vector of state variables $\boldsymbol{q} \in \mathbb{R}^d$, and a vector of control variables $\boldsymbol{u} \in \mathbb{R}^m$:

$$\dot{\boldsymbol{q}} = \boldsymbol{f}(t, \boldsymbol{q}, \boldsymbol{u}), \qquad \boldsymbol{f} : \mathbb{R}^+ \times \mathbb{R}^d \times \mathbb{R}^m \to \mathbb{R}^d. \tag{A.1}$$

State variables $\boldsymbol{q}$ completely define the state of a system. Specifically, this thesis assumes that $\boldsymbol{q}$ represents the joint angles of the fully actuated robotic system. Equation (A.1) is also referred to as the state equation. Particular case, when the DS does not explicitly considers control $\boldsymbol{u}$, is called *unforced* DS:

$$\dot{\boldsymbol{q}} = \boldsymbol{f}(t, \boldsymbol{q}), \qquad \boldsymbol{f} : \mathbb{R}^+ \times \mathbb{R}^d \to \mathbb{R}^d. \tag{A.2}$$

In cases where unforced DS does not depend on time and is strictly state-dependent, the DS is considered autonomous:

$$\dot{\boldsymbol{q}} = \boldsymbol{f}(\boldsymbol{q}), \qquad \boldsymbol{f} : \mathbb{R}^d \to \mathbb{R}^d. \tag{A.3}$$

When employing a DS to control robot motion, it is vital to analyze the system's behavior, whether it converges to a desired state, exhibits a limit cycle, or diverges to infinity. These behaviors are all determined by the stability of the DS.

[2] The material of this appendix is adapted from (Khalil, 2002)

A.2 Stability Analysis of Dynamical Systems

The notion of DS stability requires defining the *equilibrium points*.

Definition 3 (Equilibrium point) *A point $\boldsymbol{q}^* \in \mathbb{R}^d$ is an equilibrium point of the DS* (A.2) *if it has the property that whenever the state of the system starts at $\boldsymbol{q}^*$, it remains there for all time.*

For autonomous systems (A.3), the equilibrium points $\boldsymbol{q}^*$ can be found by finding the real roots of the equation:

$$\boldsymbol{f}(\boldsymbol{q}) = 0.$$

In Chapter 5, we consider that nominal motion is defined by an autonomous DS (A.3). Therefore, we focus on the stability analysis of autonomous DS. The stability of an equilibrium point $\boldsymbol{q}^*$ is determined by the behavior of the system in the vicinity of $\boldsymbol{q}^*$.

Definition 4 (Equilibrium point stability) *The equilibrium point $\boldsymbol{q} = \boldsymbol{q}^*$ of autonomous DS* (A.3) *is*

- *locally stable if for any $\varepsilon > 0$, there exists $\delta = \delta(\varepsilon) > 0$ such that*

$$||\boldsymbol{q}(0) - \boldsymbol{q}^*|| < \delta \Rightarrow ||\boldsymbol{q}(t) - \boldsymbol{q}^*|| < \varepsilon, \qquad \forall\, t \geq 0$$

- *unstable if it is not stable*
- *locally asymptotically stable (L.A.S) if it is stable and δ exists such that*

$$||\boldsymbol{q}(0) - \boldsymbol{q}^*|| < \delta \Rightarrow \lim_{t\to\infty} ||\boldsymbol{q}(t) - \boldsymbol{q}^*|| = 0$$

This definition implies that an equilibrium point is stable if and only if it is possible to remain arbitrarily close to it, provided that the trajectory starts close enough. Moreover, if trajectories approach the equilibrium point over time, the equilibrium point is considered asymptotically stable. When the point $\boldsymbol{q}^*$ is asymptotically stable, its *basin of attraction* is defined as all points from which the system converges to $\boldsymbol{q}^*$.

Definition 5 (Global asymptotic stability) *The equilibrium point $\boldsymbol{q} = \boldsymbol{q}^*$ of autonomous DS* (A.3) *is globally asyptotically stable (G.A.S.), if the asymptotic stability holds for any intitial point:*

$$\lim_{t\to\infty} ||\boldsymbol{q}(t) - \boldsymbol{q}^*|| = 0, \qquad \forall\, \boldsymbol{q}(0) \in \mathbb{R}^d$$

An alternative approach to define and analyze stability of a nonlinear dynamical system involves usage of Lyapunov's method. In particular, the following theorem is foundational to this method:

Theorem 2 (Stability of equilibrium point in autonomous nonlinear DS) *Let $\boldsymbol{q}^*$ be an equilibrium point for* (A.3)*, $D \in \mathbb{R}^d$ be a domain containing $\boldsymbol{q}^*$. Let $V : D \to \mathbb{R}$ be a continuously differentiable function such that*

$$V(\boldsymbol{q}^*) = 0 \qquad and \qquad V(\boldsymbol{q}) > 0 \ in \ D - \{\boldsymbol{q}^*\} \tag{A.4}$$

$$\dot{V}(\boldsymbol{q}) \leq 0 \ in \ D. \tag{A.5}$$

Then, $\boldsymbol{q} = \boldsymbol{q}^*$ is stable. Moreover, if

$$\dot{V}(\boldsymbol{q}) < 0 \ in \ D, \tag{A.6}$$

then $\boldsymbol{q} = \boldsymbol{q}^*$ is locally asymptotically stable.

Proof: *Please refer to (Khalil, 2002).* ■

Such function $V(\boldsymbol{q})$ is called a *Lyapunov function*. To establish a global asymptotic stability, a following theorem is used:

Theorem 3 (Barbashin-Krasovskii) *Let $\boldsymbol{q}^*$ be an equilibrium point for* (A.3)*. Let $V : \mathbb{R}^d \to \mathbb{R}$ be a continuously differentiable function such that*

$$V(\boldsymbol{q}^*) = 0 \qquad and \qquad V(\boldsymbol{q}) > 0, \quad \forall \boldsymbol{q} \neq \boldsymbol{q}^* \tag{A.7}$$

$$||\boldsymbol{q}|| \to \infty \Rightarrow V(\boldsymbol{q}) \to \infty \tag{A.8}$$

$$\dot{V}(\boldsymbol{q}) < 0 \quad , \forall \boldsymbol{q} \neq \boldsymbol{q}^* \tag{A.9}$$

then $\boldsymbol{q} = \boldsymbol{q}^*$ is globally asymptotically stable.

Proof: *Please refer to (Khalil, 2002).* ■

A.3 DS Modulation

Dynamical system (Equation (A.3)) defines a state-dependent vector field that can be altered, or *modulated*, by rotating the field with a modulation matrix $\boldsymbol{M}(.)$, which can depend on different variables[1]. The *modulated DS* is then defined as:

$$\dot{\boldsymbol{q}} = \boldsymbol{M}(.)\boldsymbol{f}(\boldsymbol{q}), \tag{A.10}$$

Depending on the choice of $\boldsymbol{M}(.)$, this DS can exhibit various behaviors. For example, $\boldsymbol{M}(\boldsymbol{q})$ is a matrix-valued function of the system state $\boldsymbol{q}$, and it can be activated differently in different regions of the state space, locally reshaping the nominal DS $\boldsymbol{f}(\boldsymbol{q})$. The simplest local state activation is the rotation:

$$\boldsymbol{M}(\boldsymbol{q}) = \mathbb{I} + (\boldsymbol{R} - \mathbb{I})e^{-||\boldsymbol{q}-\boldsymbol{q}_0||}, \tag{A.11}$$

with $\boldsymbol{R}$ being a rotation matrix, and $\boldsymbol{q}_0$ being the center of rotation. The effect of modulation vanishes exponentially away from this point.

A.4 DS Modulation for Obstacle Avoidance

A.4.1 Obstacle Representation

The key element of any obstacle avoidance strategy, including DS modulation and other path-planning methods, is the representation of the obstacle. Works Khansari-Zadeh and Billard (2012a); Huber et al. (2019); Salehian et al. (2018a) consider a definition in which obstacle detection function $\Gamma(\boldsymbol{q})$ is designed to be a scalar valued function of the system state $\Gamma(\boldsymbol{q}) : \mathbb{R}^d \rightarrow \mathbb{R}$, such that the isosurface $\Gamma(\boldsymbol{q}) = 1$ defines the collision boundary:

$$\begin{aligned} \Gamma(\boldsymbol{q}) < 1 \quad &\Leftrightarrow \quad \text{Inside obstacle (collision)} \\ \Gamma(\boldsymbol{q}) = 1 \quad &\Leftrightarrow \quad \text{Obstacle surface (contact)} \\ \Gamma(\boldsymbol{q}) > 1 \quad &\Leftrightarrow \quad \text{Outside obstacle (collision-free)} \end{aligned} \tag{A.12}$$

Here, it is important to note that $\Gamma(\boldsymbol{q})$ depends on both the system state $\boldsymbol{q}$ and the obstacles configuration $\mathcal{O}$; however, the latter dependence is frequently omitted for simpler notation. Additionally, $\Gamma(\boldsymbol{q})$ is assumed to be differentiable, i.e. $\Gamma \in \mathcal{C}^1$ for all $q \in \mathbb{R}^d$. Furthermore, the obstacle normal is defined as

$$\boldsymbol{n}(\boldsymbol{q}) = \frac{\nabla\Gamma(\boldsymbol{q})}{||\nabla\Gamma(\boldsymbol{q})||_2}. \tag{A.13}$$

This normal vector $\boldsymbol{n}(\boldsymbol{q})$ indicates the joint space direction from the obstacle and helps define the repulsive gradient, which pushes away from the obstacle at $\boldsymbol{q}$. The use of such a vector by the controller is essential for maintaining the desired collision-free behavior.

[1] We adapt the description of DS modulation from Billard et al. (2022)

A.4.2 Modulation Matrix for Obstacle Avoidance

DS modulation using matrix $\boldsymbol{M}(\boldsymbol{q}, \mathcal{O}) \in \mathbb{R}^{d\times d}$ that depends on the obstacles configuration $\mathcal{O}$, can be used to achieve obstacle avoidance. To allow for obstacle avoidance, the following modulation matrix can be used:

$$\boldsymbol{M}(\boldsymbol{q}) = \boldsymbol{E}(\boldsymbol{q})\boldsymbol{D}(\boldsymbol{q})\boldsymbol{E}(\boldsymbol{q})^{-1}, \tag{A.14}$$

where $\boldsymbol{M}(\boldsymbol{q})$ is composed through eigenvalue decomposition. Here, $\boldsymbol{D}(\boldsymbol{q})$ is a diagonal scaling matrix and $\boldsymbol{E}(\boldsymbol{q})$ is an orthogonal matrix defined as:

$$\boldsymbol{E}(\boldsymbol{q}) = \begin{bmatrix} \boldsymbol{n}(\boldsymbol{q}) & \boldsymbol{e}_1(\boldsymbol{q}) & \dots & \boldsymbol{e}_{d-1}(\boldsymbol{q}) \end{bmatrix}. \tag{A.15}$$

The first column of $\boldsymbol{E}(\boldsymbol{q})$ consists of the obstacle normal $\boldsymbol{n}(\boldsymbol{q})$ (as given in Equation (A.13)), while the remaining columns $\boldsymbol{e}_i(\boldsymbol{q})$ form a $(d-1)$-dimensional basis of the tangent space at the state $\boldsymbol{q}$.

The diagonal matrix $\boldsymbol{D}(\boldsymbol{q})$ is defined as:

$$\boldsymbol{D}(\boldsymbol{q}) = \begin{bmatrix} \lambda_n(\boldsymbol{q}) & 0 & \dots & 0 \\ 0 & \lambda_\tau(\boldsymbol{q}) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_\tau(\boldsymbol{q}) \end{bmatrix}, \tag{A.16}$$

and is composed of two eigenvalues $\lambda_n(\boldsymbol{q})$ and $\lambda_\tau(\boldsymbol{q})$ defined as:

$$\lambda_n(\boldsymbol{q}) = 1 - \frac{1}{\Gamma(\boldsymbol{q})}, \quad \lambda_\tau(\boldsymbol{q}) = 1 + \frac{1}{\Gamma(\boldsymbol{q})}. \tag{A.17}$$

Here, $\Gamma(\boldsymbol{q})$ determines the distance of the system in state $\boldsymbol{q}$ from the obstacle (as defined in Equation (A.12)). As the system moves away from the obstacle (i.e. $\Gamma(\boldsymbol{q})$ increases), $\boldsymbol{D}(\boldsymbol{q})$ becomes more similar to the identity matrix. On the other hand, when close-to-collision (i.e. $\Gamma(\boldsymbol{q})$ approaches 1), $\lambda_n(\boldsymbol{q})$ tends toward 0, cancelling out the normal component of the vector field and preventing the system from violating collision boundary; while $\lambda_\tau(\boldsymbol{q})$ grows closer to 2, increasing the component of the vector field tangential to the obstacle boundary and thus increasing the system velocity in the direction tracking the collision boundary.

In essence, the modulation (A.14) redistributes the flow of the nominal DS (A.3) based on proximity to the obstacle, thus allowing obstacle avoidance.

B Appendix of Chapter 5

B.1 Reference Direction Modulation Matrix Decomposition

The standard modulation approach presented in Khansari-Zadeh and Billard (2012a) defines the modulation matrix $\boldsymbol{M}(\boldsymbol{q})$ as:

$$\boldsymbol{M}(\boldsymbol{q}) = \boldsymbol{E}(\boldsymbol{q})\boldsymbol{D}(\boldsymbol{q})\boldsymbol{E}(\boldsymbol{q})^{-1}, \tag{B.1}$$

where $\boldsymbol{D}(\boldsymbol{q})$ is a diagonal matrix, and $\boldsymbol{E}(\boldsymbol{q})$ is an orthogonal matrix defined as:

$$\boldsymbol{E}(\boldsymbol{q}) = \begin{bmatrix} \boldsymbol{n}(\boldsymbol{q}) & \boldsymbol{e}_1(\boldsymbol{q}) & \dots & \boldsymbol{e}_{d-1}(\boldsymbol{q}) \end{bmatrix}. \tag{B.2}$$

The first column of $\boldsymbol{E}(\boldsymbol{q})$ consists of the obstacle normal $\boldsymbol{n}(\boldsymbol{q})$, while the remaining columns $\boldsymbol{e}_i(\boldsymbol{q})$ form a $d-1$-dimensional orthonormal basis of the tangent space at the state $\boldsymbol{q}$. For simpler notation, we drop the state dependency in matrix expressions further.

The improved approach introduced in Huber et al. (2019) proposes to use the alternative matrix $\boldsymbol{E}'$ by replacing vector $\boldsymbol{n}(\boldsymbol{q})$ in Equation (B.2) with a reference direction vector $\boldsymbol{r}(\boldsymbol{q},\boldsymbol{q}^r)$. We may express the reference vector $\boldsymbol{r}(\boldsymbol{q},\boldsymbol{q}^r)$ as:

$$\boldsymbol{r}(\boldsymbol{q},\boldsymbol{q}^r) = \boldsymbol{n}(\boldsymbol{q}) + \boldsymbol{a}(\boldsymbol{q},\boldsymbol{q}^r), \tag{B.3}$$

rendering modulation matrix in Huber et al. (2019) to be:

$$\begin{aligned} \boldsymbol{M}' =& \boldsymbol{E}'\boldsymbol{D}\boldsymbol{E}'^{-1} = \\ =& (\boldsymbol{E}+\boldsymbol{A})\,\boldsymbol{D}\,(\boldsymbol{E}+\boldsymbol{A})^{-1}, \end{aligned} \tag{B.4}$$

where $\boldsymbol{A}$ is a residual matrix defined as:

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{a}(\boldsymbol{q},\boldsymbol{q}^r) & 0 & \dots & 0 \end{bmatrix}. \tag{B.5}$$

If $\boldsymbol{a}(\boldsymbol{q}) \neq 0$, then rank of matrix A is 1, and, following the lemma from Miller (1981),

$$(\boldsymbol{E}+\boldsymbol{A})^{-1} = \boldsymbol{E}^{-1} - \frac{1}{1+b}\boldsymbol{E}^{-1}\boldsymbol{A}\boldsymbol{E}^{-1}, \tag{B.6}$$

where $b = trace(\boldsymbol{A}\boldsymbol{E}^{-1})$. We further denote

$$\boldsymbol{B} = -\frac{1}{1+b}\boldsymbol{E}^{-1}\boldsymbol{A}\boldsymbol{E}^{-1}.$$

Equation (B.4) can be rewritten as:

$$\begin{aligned}
\boldsymbol{M}' &= (\boldsymbol{E}+\boldsymbol{A})\,\boldsymbol{D}\,(\boldsymbol{E}+\boldsymbol{A})^{-1} = \\
&= \boldsymbol{E}\boldsymbol{D}\,(\boldsymbol{E}+\boldsymbol{A})^{-1} + \boldsymbol{A}\boldsymbol{D}\,(\boldsymbol{E}+\boldsymbol{A})^{-1} = \\
&= \boldsymbol{E}\boldsymbol{D}\left(\boldsymbol{E}^{-1}+\boldsymbol{B}\right) + \boldsymbol{A}\boldsymbol{D}\left(\boldsymbol{E}^{-1}+\boldsymbol{B}\right) = \\
&= \boldsymbol{E}\boldsymbol{D}\boldsymbol{E}^{-1} + \boldsymbol{E}\boldsymbol{D}\boldsymbol{B} + \boldsymbol{A}\boldsymbol{D}\boldsymbol{E}^{-1} + \boldsymbol{A}\boldsymbol{D}\boldsymbol{B} = \\
&= \boldsymbol{M} + \boldsymbol{E}\boldsymbol{D}\boldsymbol{B} + \boldsymbol{A}\boldsymbol{D}\boldsymbol{E}^{-1} + \boldsymbol{A}\boldsymbol{D}\boldsymbol{B}
\end{aligned} \tag{B.7}$$

We may denote the last three terms as $\boldsymbol{g}(\boldsymbol{q}, \boldsymbol{q}^r)$, then the final expression for modulation matrix $\boldsymbol{M}'$ is:

$$\boldsymbol{M}' = \boldsymbol{M} + \boldsymbol{g}(\boldsymbol{q}, \boldsymbol{q}^r), \tag{B.8}$$

essentially rendering the modulated system from Huber et al. (2019) to be:

$$\begin{aligned}
\dot{\boldsymbol{q}} &= \left(\boldsymbol{M} + \boldsymbol{g}(\boldsymbol{q}, \boldsymbol{q}^r)\right)\boldsymbol{f}(\boldsymbol{q}) = \\
&= \boldsymbol{M}\boldsymbol{f}(\boldsymbol{q}) + \boldsymbol{g}(\boldsymbol{q}, \boldsymbol{q}^r)\boldsymbol{f}(\boldsymbol{q}),
\end{aligned} \tag{B.9}$$

that can in turn be rewritten as:

$$\dot{\boldsymbol{q}} = \boldsymbol{M}\boldsymbol{f}(\boldsymbol{q}) + \boldsymbol{g}(\boldsymbol{q}, \boldsymbol{q}^r). \tag{B.10}$$

The latter formulation is equivalent to a control law, where modulated DS is deflected (or stabilized, depends on the choice of interpretation) by a nonlinear term $\boldsymbol{g}(\boldsymbol{q}, \boldsymbol{q}^r)$, which can be considered as a deflection or stabilizing control input through the lense of a control-input nonlinear systems from optimal control theory. The external control variable here is then the reference state $\boldsymbol{q}^r$, that defines the reference direction $\boldsymbol{r}(\boldsymbol{q}, \boldsymbol{q}^r)$.

B.2 Impenetrability of Obstacle Boundary for Explicit Tangent Space Dynamics Deflection

By Definition 1, impenetrability requires that normal velocity at boundary points $\boldsymbol{Q}_{\mathcal{O}} = \{\boldsymbol{q} \in \mathbb{R}^d : \Gamma(\boldsymbol{q}) = 0\}$ vanishes, i.e., $\boldsymbol{n}(\boldsymbol{q})^T\dot{\boldsymbol{q}} = \nabla\Gamma(\boldsymbol{q})^T\dot{\boldsymbol{q}} = 0,\ \forall \boldsymbol{q} \in \boldsymbol{Q}_{\mathcal{O}}$. The vector field modulated by the tangent space deflection is given by $\dot{\boldsymbol{q}} = \boldsymbol{M}(\boldsymbol{q}) \cdot (\boldsymbol{f}(\boldsymbol{q}) + \alpha(\boldsymbol{q})\boldsymbol{g}(\boldsymbol{q}))$ (Equation (5.11)).

If $\boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{0}$, then $\boldsymbol{n}(\boldsymbol{q})^T(\boldsymbol{M}(\boldsymbol{q})\boldsymbol{f}(\boldsymbol{q})) = 0$ on the boundary, and impenetrability for obstacles is

guaranteed following the proof of Theorem 2 in Khansari-Zadeh and Billard (2012a). Since $\boldsymbol{g}(\boldsymbol{q}) \perp \boldsymbol{n}(\boldsymbol{q})$ (see Equation 5.32 in Section 5.6.3), then $\boldsymbol{g}(\boldsymbol{q})$ induces no velocities along the normal direction, meaning impenetrability is preserved.

For completeness we restate the proof for the standard modulation including the explicit tangent space dynamics. Formally, impenetrability for continuous state dynamics can be proven if $\boldsymbol{n}(\boldsymbol{q})^T \dot{\boldsymbol{q}} = \boldsymbol{n}(\boldsymbol{q})^T \boldsymbol{M}(\boldsymbol{q}) \cdot (\boldsymbol{f}(\boldsymbol{q}) + \alpha(\boldsymbol{q})\boldsymbol{g}(\boldsymbol{q})) = 0$ as shown below:

$$\begin{aligned} \boldsymbol{n}(\boldsymbol{q})^T \dot{\boldsymbol{q}} &= \boldsymbol{n}(\boldsymbol{q})^T \boldsymbol{E}(\boldsymbol{q}) \boldsymbol{D}(\boldsymbol{q}) \boldsymbol{E}(\boldsymbol{q})^{-1} \big(\boldsymbol{f}(\boldsymbol{q}) + \alpha(\boldsymbol{q})\boldsymbol{g}(\boldsymbol{q})\big) \\ &= \begin{bmatrix} 1 \\ \boldsymbol{0}_{d-1} \end{bmatrix}^T \begin{bmatrix} 0 & 0 \\ 0 & \mathrm{diag}(\boldsymbol{2})_{d-1} \end{bmatrix} \boldsymbol{E}(\boldsymbol{q})^{-1} \big(\boldsymbol{f}(\boldsymbol{q}) + \alpha(\boldsymbol{q})\boldsymbol{g}(\boldsymbol{q})\big) \\ &= \boldsymbol{0}_d^T \boldsymbol{E}(\boldsymbol{q})^{-1} \big(\boldsymbol{f}(\boldsymbol{q}) + \alpha(\boldsymbol{q})\boldsymbol{g}(\boldsymbol{q})\big) = 0 \end{aligned} \tag{B.11}$$

The second line in Equation (B.11) is obtained by plugging in the following definitions: Let $D = \begin{bmatrix} \lambda_n & 0 \\ 0 & \Lambda_\tau \end{bmatrix} \in \mathbb{R}^{d \times d}$ with $\lambda_n \in \mathbb{R}$ being the eigenvalue corresponding to the direction along the normal $\boldsymbol{n}(\boldsymbol{q})$ and $\Lambda_\tau = \mathrm{diag}(\lambda_\tau) \in \mathbb{R}^{d-1}$ being the eigenvalues corresponding to the orthonormal eigenbasis tangential to the normal $E_\tau = [\boldsymbol{e}_1, \ldots, \boldsymbol{e}_{d-1}] \in \mathbb{R}^{d \times d-1}$ used to construct $E = [\boldsymbol{n}(\boldsymbol{q}) \;\; E_\tau]$. Further, recall that $\forall \boldsymbol{q} \in \boldsymbol{Q}_{\mathcal{O}} \implies \lambda_n = 0, \lambda_\tau = 2$, (as defined in Equations (5.12) and (5.13)). Since the first eigenvector of E is $\boldsymbol{n}(\boldsymbol{q})$ then $\boldsymbol{n}(\boldsymbol{q})^T \boldsymbol{n}(\boldsymbol{q}) = 1$ and $\boldsymbol{n}(\boldsymbol{q})^T E_\tau(\boldsymbol{q}) = \boldsymbol{0}_{d-1}$ rendering the term $\boldsymbol{n}(\boldsymbol{q})^T \boldsymbol{E}(\boldsymbol{q}) \boldsymbol{D}(\boldsymbol{q}) = \boldsymbol{0}_d^T$, thus proving impenetrability at the boundary for Equation (5.11). ■

B.3 Local Asymptotic Stability of the Modulated DS with the Tangential Deflection Component

If the activation parameter $\alpha(\boldsymbol{q})$ is equal to zero in a ball B_r enclosing the attractor $\boldsymbol{q}^*$, then the deflection component $\boldsymbol{g}(\boldsymbol{q})$ vanishes in B_r and the DS is identical to the standard modulated DS:

$$\boldsymbol{M}(\boldsymbol{q})(\boldsymbol{f}(\boldsymbol{q}) + \alpha(\boldsymbol{q})\boldsymbol{g}(\boldsymbol{q})) = \boldsymbol{M}(\boldsymbol{q})(\boldsymbol{f}(\boldsymbol{q}) + 0\boldsymbol{g}(\boldsymbol{q})) = \boldsymbol{M}(\boldsymbol{q})\boldsymbol{f}(\boldsymbol{q}), \quad \forall \boldsymbol{q} \in B_r. \tag{B.12}$$

Therefore, we need to prove that standard modulated DS $\boldsymbol{M}(\boldsymbol{q})\boldsymbol{f}(\boldsymbol{q})$ is locally asymptotically stable in B_r.

The original dynamics $\boldsymbol{f}(\boldsymbol{q})$ are globally asymptotically stable, which implies (following Theorem 3 in Appendix A) the existence of a Lyapunov function $V : \mathbb{R}^d \to \mathbb{R}$ such that:

$$V(\boldsymbol{q}^*) = 0 \qquad \text{and} \qquad V(\boldsymbol{q}) > 0, \quad \forall \boldsymbol{q} \neq \boldsymbol{q}^* \tag{B.13}$$

$$||\boldsymbol{q}|| \to \infty \Rightarrow V(\boldsymbol{q}) \to \infty \tag{B.14}$$

$$\dot{V}(\boldsymbol{q}) < 0 \quad , \forall \boldsymbol{q} \neq \boldsymbol{q}^* \tag{B.15}$$

Let B_r be be a ball centered at the $\boldsymbol{q}^*$ with a radius r small enough s.t. $\boldsymbol{M}(\boldsymbol{q}) = \mathbb{I}_{d\times d}$. In other words, modulation matrix $\boldsymbol{M}$ is equal to identity in close vicinity of the attractor. Thus, inside B_r we have

$$\dot{\boldsymbol{q}} = \boldsymbol{M}(\boldsymbol{q})\boldsymbol{f}(\boldsymbol{q}) = \mathbb{I}_{d\times d}\boldsymbol{f}(\boldsymbol{q}) = \boldsymbol{f}(\boldsymbol{q}), \quad \forall \boldsymbol{q} \in B_r. \tag{B.16}$$

Let $D \subset B_r$ be the largest level set of V that lies entirely inside B_r. For any $\boldsymbol{q}_0 \in D$, the modulated dynamics is exactly equal to the original dynamics $\dot{\boldsymbol{q}} = \boldsymbol{f}(\boldsymbol{q})$. Hence, $V(\boldsymbol{q}) > 0$ and $\dot{V}(\boldsymbol{q}) < 0$ holds for all $\boldsymbol{q} \in D$, which proves (see Theorem 2 in Appendix A) that the system is L.A.S. at $\boldsymbol{q}^*$ with region of attraction given by D. ■

B.4 MPPI deflection optimization algorithm

Algorithm 1: MPPI Modulated DS motion

Input : Nominal DS $f(\boldsymbol{q})$, attractor $\boldsymbol{q}^*$,
Cost function $c = c(\{\boldsymbol{q}, \dot{\boldsymbol{q}}\})$,
Obstacle points $\mathcal{O}$,
Initial robot state $\boldsymbol{q}_{init}$,

Output : Optimal trajectory $\{\boldsymbol{q}_s, \dot{\boldsymbol{q}}_s\}_{s=1..S}$
Modulation parameters $\{\hat{\boldsymbol{g}}_k, \hat{\boldsymbol{q}}_k\}_{k=1..K}$

Parameters : $N, H, \Delta t, \Delta\tau$

/* Initialize the iteration counter, kernel counter, and current state */
$(s, K, \boldsymbol{q}_s) \leftarrow (0, 0, \boldsymbol{q}_{init})$
/* Main loop */
while $||\boldsymbol{q}_s - \boldsymbol{q}^*|| > 0$ **do**
  /* Explore N trajectories */
  for $i \in [1, N]$ **do**
    /* Sample navigation parameters */
    for $k \in [1, K]$ **do**
      $\hat{\boldsymbol{g}}_{i,k} \leftarrow \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$
    /* Integrate for H timesteps */
    $\boldsymbol{q}_0 \leftarrow \boldsymbol{q}_s$
    for $h \in [1, H]$ **do**
      /* Evaluate nominal DS */
      $\dot{\boldsymbol{q}}_{nom} = f(\boldsymbol{q}_{h-1})$
      /* Evaluate Modulation */
      $M \leftarrow M(\boldsymbol{q}_{h-1}, \mathcal{O})$
      /* Evaluate Navigation Kernels */
      $\dot{\boldsymbol{q}}_{nav} = g(\boldsymbol{q}_{h-1}, \{\hat{\boldsymbol{g}}_{i,k}, \hat{\boldsymbol{q}}_k\}_{k=1}^{K}, \mathcal{O})$
      /* Get total velocity */
      $\dot{\boldsymbol{q}}_{i,h-1} = M(\dot{\boldsymbol{q}}_{nom} + \dot{\boldsymbol{q}}_{nav})$
      /* Integrate */
      $\boldsymbol{q}_{i,h} = \boldsymbol{q}_{i,h-1} + \dot{\boldsymbol{q}}_{i,h-1}\Delta t$
    /* Evaluate cost of trajectory */
    $c_i = c(\{\boldsymbol{q}_{i,h}, \dot{\boldsymbol{q}}_{i,h}\}_{t=1..H})$
  /* Update sampling policy */
  for $k \in [1, K]$ **do**
    $(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \leftarrow$ *Update using* c_i (Eqns. 5.17-5.19)
  /* Add new navigation kernels */
  for $i \in [1, N]$ **do**
    for $h \in [1, H]$ **do**
      if $\boldsymbol{q}_{i,h}$ *satisfies (5.28a)* **then**
        $K \leftarrow K + 1$
        $k^* = \min_k ||\boldsymbol{q}_{i,h} - \hat{\boldsymbol{q}}_k||_2$
        $(\hat{\boldsymbol{q}}_K, \boldsymbol{\mu}_K, \boldsymbol{\Sigma}_K) \leftarrow (\boldsymbol{q}_{i,h}, \boldsymbol{\mu}_{k^*}, \boldsymbol{\Sigma}_{k^*})$
  /* Find lowest cost */
  $i^* = \min_i c_i$
  /* Update robot state */
  $\dot{\boldsymbol{q}}_{s-1} = \dot{\boldsymbol{q}}_{i^*,0}$
  $\boldsymbol{q}_s = \boldsymbol{q}_{s-1} + \dot{\boldsymbol{q}}_{s-1}\Delta\tau$
  $s = s + 1$
return $\{\boldsymbol{q}_s, \dot{\boldsymbol{q}}_s\}_{s=1..S}, \{\hat{\boldsymbol{g}}_k, \hat{\boldsymbol{q}}_k\}_{k=1..K}$

Bibliography

Ajoudani, A., Lee, J., Rocchi, A., Ferrati, M., Hoffman, E. M., Settimi, A., Caldwell, D. G., Bicchi, A., and Tsagarakis, N. G. (2014). A manipulation framework for compliant humanoid coman: Application to a valve turning task. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 664–670.

Amato, N. and Wu, Y. (1996). A randomized roadmap method for path and manipulation planning. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 1, pages 113–120 vol.1.

Bhardwaj, M., Sundaralingam, B., Mousavian, A., Ratliff, N. D., Fox, D., Ramos, F., and Boots, B. (2022). STORM: An integrated framework for fast joint-space model-predictive control for reactive manipulation. In Faust, A., Hsu, D., and Neumann, G., editors, *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 750–759. PMLR.

Billard, A., Mirrazavi, S., and Figueroa, N. (2022). *Learning for Adaptive and Reactive Robot Control: A Dynamical Systems Approach*. Intelligent Robotics and Autonomous Agents series. MIT Press.

Brock, O. and Khatib, O. (2002). Elastic strips: A framework for motion generation in human environments. *The International Journal of Robotics Research*, 21(12):1031–1052.

Chen, B., Kwiatkowski, R., Vondrick, C., and Lipson, H. (2022). Fully body visual self-modeling of robot morphologies. *Science Robotics*, 7(68):eabn1944.

Connolly, C. I. and Grupen, R. A. (1993). The applications of harmonic functions to robotics. *J. Field Robotics*, 10:931–946.

Dahiya, R. S., Mittendorfer, P., Valle, M., Cheng, G., and Lumelsky, V. J. (2013). Directions toward effective utilization of tactile skin: A review. *IEEE Sensors Journal*, 13(11):4121–4138.

Danielczuk, M., Mousavian, A., Eppner, C., and Fox, D. (2021). Object rearrangement using learned implicit collision functions. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6010–6017.

Das, N. and Yip, M. (2020). Learning-based proxy collision detection for robot motion planning applications. *IEEE Transactions on Robotics*, 36(4):1096–1114.

De Luca, A., Albu-Schaffer, A., Haddadin, S., and Hirzinger, G. (2006). Collision detection and safe reaction with the dlr-iii lightweight manipulator arm. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1623–1630.

De Santis, A., Albu-Schaffer, A., Ott, C., Siciliano, B., and Hirzinger, G. (2007). The skeleton algorithm for self-collision avoidance of a humanoid manipulator. In *2007 IEEE/ASME international conference on advanced intelligent mechatronics*, pages 1–6.

Dietrich, A., Wimböck, T., Täubig, H., Albu-Schäffer, A., and Hirzinger, G. (2011). Extensions to reactive self-collision avoidance for torque and position controlled humanoids. In *2011 IEEE International Conference on Robotics and Automation*, pages 3455–3462.

Erez, T., Lowrey, K., Tassa, Y., Kumar, V., Kolev, S., and Todorov, E. (2013). An integrated system for real-time model predictive control of humanoid robots. In *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 292–299.

Ericson, C. (2004). *Real-Time Collision Detection*. CRC Press, Inc., USA.

Escande, A., Miossec, S., Benallegue, M., and Kheddar, A. (2014). A strictly convex hull for computing proximity distances with continuous gradients. *IEEE Transactions on Robotics*, 30(3):666–678.

Escande, A., Miossec, S., and Kheddar, A. (2007). Continuous gradient proximity distance for humanoids free-collision optimized-postures. In *2007 7th IEEE-RAS International Conference on Humanoid Robots*, pages 188–195.

Fang, C., Rocchi, A., Hoffman, E. M., Tsagarakis, N. G., and Caldwell, D. G. (2015). Efficient self-collision avoidance based on focus of interest for humanoid robots. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 1060–1066.

Faraji, S. and Ijspeert, A. J. (2017). Singularity-tolerant inverse kinematics for bipedal robots: An efficient use of computational power to reduce energy consumption. *IEEE Robotics and Automation Letters*, 2(2):1132–1139.

Faraji, S., Pouya, S., Atkeson, C. G., and Ijspeert, A. J. (2014). Versatile and robust 3d walking with a simulated humanoid robot (Atlas): A model predictive control approach. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1943–1950.

Feder, H. and Slotine, J.-J. (1997). Real-time path planning using harmonic potentials in dynamic environments. In *Proceedings of International Conference on Robotics and Automation*, volume 1, pages 874–881 vol.1.

Figueroa, N. and Billard, A. (2017). Learning complex manipulation tasks from heterogeneous and unstructured demonstrations. In *Proceedings of Workshop on Synergies between Learning and Interaction*.

Figueroa, N. and Billard, A. (2018). A physically-consistent bayesian non-parametric mixture model for dynamical system learning. In Billard, A., Dragan, A., Peters, J., and Morimoto, J., editors, *Proceedings of The 2nd Conference on Robot Learning*, volume 87 of *Proceedings of Machine Learning Research*, pages 927–946. PMLR.

Figueroa, N., Faraji, S., Koptev, M., and Billard, A. (2020). A dynamical system approach for adaptive grasping, navigation and co-manipulation with humanoid robots. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE.

Figueroa, N., Mirrazavi Salehian, S. S., and Billard, A. (2018). Multi-arm self-collision avoidance: A sparse solution for a big data problem. *Proceedings of the Third Machine Learning in Planning and Control of Robot Motion (MLPC) Workshop.*, page 6.

Frasch, J. V., Gray, A., Zanon, M., Ferreau, H. J., Sager, S., Borrelli, F., and Diehl, M. (2013). An auto-generated nonlinear mpc algorithm for real-time obstacle avoidance of ground vehicles. In *2013 European Control Conference (ECC)*, pages 4136–4141.

Frisch, D. (2022). Distance between point and triangulated surface. https://www.mathworks.com/matlabcentral/fileexchange/52882-point2trimesh-distance-between-point-and-triangulated-surface. [Online; accessed June 3, 2022].

Gilbert, E., Johnson, D., and Keerthi, S. (1988). A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation*, 4(2):193–203.

Guadarrama-Olvera, J. R., Dean-Leon, E., Bergner, F., and Cheng, G. (2019). Pressure-driven body compliance using robot skin. *IEEE Robotics and Automation Letters*, 4(4):4418–4423.

Haddadin, S., De Luca, A., and Albu-Schäffer, A. (2017). Robot collisions: A survey on detection, isolation, and identification. *IEEE Transactions on Robotics*, 33(6):1292–1312.

Harrison, N., Liu, W., Jang, I., Carrasco, J., Herrmann, G., and Sykes, N. (2020). A comparative study for obstacle avoidance inverse kinematics: Null-space based vs. optimisation-based. In Mohammad, A., Dong, X., and Russo, M., editors, *Towards Autonomous Robotic Systems*, pages 147–158, Cham. Springer International Publishing.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107.

Head, H. and Holmes, G. (1912). Researches into sensory disturbances from cerebral lesions. *The Lancet*, 179(4612):144–152. Originally published as Volume 1, Issue 4612.

Hintjens, P. (2013). *ZeroMQ: messaging for many applications*. O'Reilly Media, Inc.

Hoffmann, M., Marques, H., Arieta, A., Sumioka, H., Lungarella, M., and Pfeifer, R. (2010). Body schema in robotics: A review. *IEEE Transactions on Autonomous Mental Development*, 2(4):304–324.

Huber, L., Billard, A., and Slotine, J.-J. E. (2019). Avoidance of convex and concave obstacles with convergence ensured through contraction. *IEEE Robotics and Automation Letters*, 4:1462–1469.

Huber, L., Slotine, J.-J., and Billard, A. (2022). Avoiding dense and dynamic obstacles in enclosed spaces: Application to moving in crowds. *IEEE Transactions on Robotics*, 38(5):3113–3132.

Joachims, T. and Yu, C.-N. (2009a). Sparse kernel SVMs via cutting-plane training. *Machine Learning*, 76:179–193.

Joachims, T. and Yu, C.-N. J. (2009b). Sparse kernel svms via cutting-plane training. *Machine Learning*, 76:179–193.

Kalakrishnan, M., Chitta, S., Theodorou, E., Pastor, P., and Schaal, S. (2011). Stomp: Stochastic trajectory optimization for motion planning. In *2011 IEEE International Conference on Robotics and Automation*, pages 4569–4574.

Karaman, S., Walter, M. R., Perez, A., Frazzoli, E., and Teller, S. (2011). Anytime motion planning using the RRT*. In *2011 IEEE International Conference on Robotics and Automation*, pages 1478–1483.

Kavraki, L. E., Svestka, P., Latombe, J. C., and Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580.

Kew, C., Ichter, B. A., Bandari, M., Lee, E., and Faust, A. (2020). Neural collision clearance estimator for batched motion planning. In *The 14th International Workshop on the Algorithmic Foundations of Robotics (WAFR)*.

Khalil, H. (2002). *Nonlinear Systems*. Pearson Education. Prentice Hall.

Khansari-Zadeh, S. M. and Billard, A. (2011). Learning stable nonlinear dynamical systems with gaussian mixture models. *IEEE Transactions on Robotics*, 27(5):943–957.

Khansari-Zadeh, S. M. and Billard, A. (2012a). A dynamical system approach to realtime obstacle avoidance. *Autonomous Robots*, 32(4):433–454.

Khansari-Zadeh, S. M. and Billard, A. (2012b). Realtime avoidance of fast moving objects: A dynamical system-based approach. In *Workshop on Robot Motion Planning: Online, Reactive, and in Real-time*.

Khansari-Zadeh, S. M., Kronander, K., and Billard, A. (2012). Learning to play minigolf: A dynamical system-based approach. *Advanced Robotics*, 26(17):27. 1967–1993.

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1):90–98.

Khoramshahi, M., Henriks, G., Naef, A., Salehian, S. S. M., Kim, J., and Billard, A. (2020). Arm-hand motion-force coordination for physical interactions with non-flat surfaces using dynamical systems: Toward compliant robotic massage. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4724–4730.

Kim, S., Shukla, A., and Billard, A. (2014). Catching objects in flight. *IEEE Transactions on Robotics*, 30(5):1049–1065.

Kingston, Z., Moll, M., and Kavraki, L. E. (2018). Sampling-based methods for motion planning with constraints. *Annual review of control, robotics, and autonomous systems*, 1:159–185.

Kingston, Z., Moll, M., and Kavraki, L. E. (2019). Exploring implicit spaces for constrained sampling-based planning. *The International Journal of Robotics Research*, 38(10-11):1151–1178.

Koenemann, J., Del Prete, A., Tassa, Y., Todorov, E., Stasse, O., Bennewitz, M., and Mansard, N. (2015). Whole-body model-predictive control applied to the hrp-2 humanoid. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3346–3351. IEEE.

Koptev, M., Figueroa, N., and Billard, A. (2021). Real-time self-collision avoidance in joint space for humanoid robots. *IEEE Robotics and Automation Letters*, 6(2):1240–1247.

Koptev, M., Figueroa, N., and Billard, A. (2023). Neural joint space implicit signed distance functions for reactive robot manipulator control. *IEEE Robotics and Automation Letters*, 8(2):480–487.

Kot, T., Wierbica, R., Oščádal, P., Spurný, T., and Bobovský, Z. (2022). Using elastic bands for collision avoidance in collaborative robotics. *IEEE Access*, 10:106972–106987.

Kronander, K., Khansari, M., and Billard, A. (2015). Incremental motion learning with locally modulated dynamical systems. *Robotics and Autonomous Systems*, 70:52–62.

Kuffner, J. J. and LaValle, S. M. (2000). Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. IEEE International Conference on Robotics and Automation.*, volume 2, pages 995–1001 vol.2.

LaValle, S. M. (1998). Rapidly-exploring random trees: a new tool for path planning. *TR 98-11, Computer Science Dept., Iowa State University*.

LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press.

Li, S., Figueroa, N., Shah, A., and Shah, J. A. (2021). Provably Safe and Efficient Motion Planning with Uncertain Human Dynamics. In *Proceedings of Robotics: Science and Systems*, Virtual.

Liu, P., Zhang, K., Tateo, D., Jauhri, S., Peters, J., and Chalvatzaki, G. (2022). Regularized deep signed distance fields for reactive motion generation. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6673–6680. IEEE.

Luna, R., Moll, M., Badger, J., and Kavraki, L. E. (2020). A scalable motion planner for high-dimensional kinematic systems. *The International Journal of Robotics Research*, 39(4):361–388.

Mattingley, J. and Boyd, S. (2012). CVXGEN: A code generator for embedded convex optimization. *Optimization and Engineering*, 13.

Metta, G., Natale, L., Nori, F., Sandini, G., Vernon, D., Fadiga, L., von Hofsten, C., Rosander, K., Lopes, M., Santos-Victor, J., Bernardino, A., and Montesano, L. (2010). The iCub humanoid robot: An open-systems platform for research in cognitive development. *Neural Networks*, 23(8):1125–1134.

Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., and Ng, R. (2020). Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*.

Miller, K. S. (1981). On the inverse of the sum of matrices. *Mathematics Magazine*, 54(2):67–72.

Mirtich, B. (1998). V-clip: Fast and robust polyhedral collision detection. *ACM Trans. Graph.*, 17(3):177–208.

Montaut, L., Le Lidec, Q., Petrik, V., Sivic, J., and Carpentier, J. (2022). Collision detection accelerated: An optimization perspective. In *Robotics: Science and Systems*.

Nava, G., Pucci, D., Guedelha, N., Traversaro, S., Romano, F., Dafarra, S., and Nori, F. (2017). Modeling and control of humanoid robots in dynamic environments: iCub balancing on a seesaw. In *2017 IEEE-RAS International Conference on Humanoid Robotics*, pages 263–270.

Pan, J. and Manocha, D. (2016). Fast probabilistic collision checking for sampling-based motion planning using locality-sensitive hashing. *The International Journal of Robotics Research*, 35(12):1477–1496.

Park, J. J., Florence, P., Straub, J., Newcombe, R., and Lovegrove, S. (2019). Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Quinlan, S. and Khatib, O. (1993). Elastic bands: connecting path planning and control. In *[1993] Proceedings IEEE International Conference on Robotics and Automation*, pages 802–807 vol.2.

Quiroz-Omaña, J. J. and Adorno, B. V. (2019). Whole-body control with (self) collision avoidance using vector field inequalities. *IEEE Robotics and Automation Letters*, 4(4):4048–4053.

Rakita, D., Mutlu, B., and Gleicher, M. (2018). Relaxedik: Real-time synthesis of accurate and feasible robot arm motion. In Kress-Gazit, H., Srinivasa, S. S., Howard, T., and Atanasov, N., editors, *Robotics: Science and Systems XIV, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, June 26-30, 2018*.

Ratliff, N., Zucker, M., Bagnell, J. A., and Srinivasa, S. (2009). CHOMP: Gradient optimization techniques for efficient motion planning. In *Proceedings of the 2009 IEEE International Conference on Robotics and Automation*, ICRA'09, page 4030–4035. IEEE Press.

Richter, S., Jones, C. N., and Morari, M. (2012). Computational complexity certification for real-time mpc with input constraints based on the fast gradient method. *IEEE Transactions on Automatic Control*, 57(6):1391–1403.

Rimon, E. and Koditschek, D. (1992). Exact robot navigation using artificial potential functions. *IEEE Transactions on Robotics and Automation*, 8(5):501–518.

Rimon, E. and Koditschek, D. E. (1988). Exact robot navigation using cost functions: the case of distinct spherical boundaries in en. In *Proceedings. 1988 IEEE International Conference on Robotics and Automation*, pages 1791–1796 vol.3.

Sacks, J. and Boots, B. (2022). Learning sampling distributions for model predictive control. In *6th Annual Conference on Robot Learning*.

Salehian, S. S. M., Figueroa, N., and Billard, A. (2018a). A unified framework for coordinated multi-arm motion planning. *The International Journal of Robotics Research*, 37(10):1205–1232.

Salehian, S. S. M., Khoramshahi, M., and Billard, A. (2016). A dynamical system approach for softly catching a flying object: Theory and experiment. *IEEE Transactions on Robotics*, 32(2):462–471.

Salehian, S. S. M., N.Figueroa, and Billard, A. (2018b). A unified framework for coordinated multi-arm motion planning. *The International Journal of Robotics Research*, 37(10):1205–1232.

Schmitz, A., Maiolino, P., Maggiali, M., Natale, L., Cannata, G., and Metta, G. (2011). Methods and technologies for the implementation of large-scale robot tactile sensors. *IEEE Transactions on Robotics*, 27:389–400.

Schölkopf, B. and Smola, A. (2002). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, USA.

Schulman, J., Duan, Y., Ho, J., Lee, A., Awwal, I., Bradlow, H., Pan, J., Patil, S., Goldberg, K., and Abbeel, P. (2014). Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, 33(9):1251–1270.

Schwienbacher, M., Buschmann, T., Lohmeier, S., Favot, V., and Ulbrich, H. (2011). Self-collision avoidance and angular momentum compensation for a biped humanoid robot. In *2011 IEEE International Conference on Robotics and Automation*, pages 581–586.

Shavit, Y., Figueroa, N., Salehian, S. S. M., and Billard, A. (2018). Learning augmented joint-space task-oriented dynamical systems: A linear parameter varying and synergetic control approach. *IEEE Robotics and Automation Letters*, 3(3):2718–2725.

Shukla, A. and Billard, A. (2012). Augmented-svm: Automatic space partitioning for combining multiple non-linear dynamics. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 1016–1024. Curran Associates, Inc.

Spong, M., Hutchinson, S., and Vidyasagar, M. (2020). *Robot Modeling and Control*. Wiley.

Stasse, O., Escande, A., Mansard, N., Miossec, S., Evrard, P., and Kheddar, A. (2008). Real-time (self)-collision avoidance task on a HRP-2 humanoid robot. In *2008 IEEE International Conference on Robotics and Automation*, pages 3200–3205.

Sugiura, H., Gienger, M., Janssen, H., and Goerick, C. (2006). Real-time self collision avoidance for humanoids by means of nullspace criteria and task intervals. In *2006 6th IEEE-RAS International Conference on Humanoid Robots*, pages 575–580.

Sugiura, H., Gienger, M., Janssen, H., and Goerick, C. (2007). Real-time collision avoidance with whole body motion control for humanoid robots. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2053–2058.

Taylor, A., Singletary, A., Yue, Y., and Ames, A. (2020). Learning for safety-critical control with control barrier functions. In *Learning for Dynamics and Control*, pages 708–717. PMLR.

Vahrenkamp, N., Asfour, T., Metta, G., Sandini, G., and Dillmann, R. (2012). Manipulability analysis. In *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, pages 568–573.

Wang, Y., Figueroa, N., Li, S., Shah, A., and Shah, J. (2022). Temporal logic imitation: Learning plan-satisficing motion policies from demonstrations. In *6th Annual Conference on Robot Learning*.

Webb, D. J. and van den Berg, J. (2013). Kinodynamic rrt*: Asymptotically optimal motion planning for robots with linear dynamics. In *2013 IEEE International Conference on Robotics and Automation*, pages 5054–5061.

Wen, Z., Shi, J., Li, Q., He, B., and Chen, J. (2018). ThunderSVM: A fast SVM library on GPUs and CPUs. *Journal of Machine Learning Research*, 19:797–801.

White, J., Jay, D., Wang, T., and Hubicki, C. (2022). Avoiding dynamic obstacles with real-time motion planning using quadratic programming for varied locomotion modes. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 13626–13633.

Williams, G., Aldrich, A., and Theodorou, E. A. (2017). Model predictive path integral control: From theory to parallel computation. *Journal of Guidance, Control, and Dynamics*, 40(2):344–357.

Yoon, H.-J., Tao, C., Kim, H., Hovakimyan, N., and Voulgaris, P. (2022). Sampling complexity of path integral methods for trajectory optimization. In *2022 American Control Conference (ACC)*, pages 3482–3487.

Zhi, Y., Das, N., and Yip, M. C. (2021). Diffco: Auto-differentiable proxy collision detection with multi-class labels for safety-aware trajectory optimization. *CoRR*, abs/2102.07413.

Zimmermann, S., Busenhart, M., Huber, S., Poranne, R., and Coros, S. (2022). Differentiable collision avoidance using collision primitives.

Zucker, M., Ratliff, N., Dragan, A. D., Pivtoraiko, M., Klingensmith, M., Dellin, C. M., Bagnell, J. A., and Srinivasa, S. S. (2013). Chomp: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research*, 32(9-10):1164–1193.

Mikhail Koptev

E-mail: *mikhail.koptev@epfl.ch, mkoptev@pm.me*

Education

2019 - 2023	**École Polytechnique Fédérale de Lausanne** (EPFL), Lausanne, Switzerland Ph. D., Machine Learning and Robotics Thesis: *"Implicit Distance Functions: Learning and Applications in Robotics"*
2016 - 2018	**Moscow Institute of Physics and Technology** (MIPT), Moscow, Russia M. Sc., Applied Mathematics and Control GPA: 4.8 of 5 (with Honors) Thesis: *"Design, Deployment and Keeping of Nanosatellite-Based Tetrahedral Highly Elliptical Orbit Formation"*
2012 - 2016	**Bauman Moscow State Technical University** (BMSTU), Moscow, Russia B. Sc., Applied Mathematics GPA: 4.81 of 5 (with Honors)

Work Experience

2019 - 2023	**Doctoral Assistant** EPFL Learning Algorithms and Systems Lab (LASA) Lausanne, Switzerland
2018 - 2019	**Intern** EPFL Space Engineering Center (eSpace) Lausanne, Switzerland
2014 - 2018	**Research Assistant** Spaceflight Mechanics and Control Department, Keldysh Institute of Applied Mathematics (KIAM) Moscow, Russia

Journal Publications

1. Mikhail Koptev, Nadia Figueroa, and Aude Billard (2023b). "Reactive Collision-Free Motion Generation in Joint Space via Dynamical Systems and Sampling-Based MPC". in: *The International Journal of Robotics Research (IJRR)*. Under Review.

2. Mikhail Koptev, Nadia Figueroa, and Aude Billard (2023a). "Neural Joint Space Implicit Signed Distance Functions for Reactive Robot Manipulator Control". In: *IEEE Robotics and Automation Letters* 8.2, pp. 480–487. DOI: 10.1109/LRA.2022.3227860

3. Mikhail Koptev, Nadia Figueroa, and Aude Billard (2021). "Real-Time Self-Collision Avoidance in Joint Space for Humanoid Robots". In: *IEEE Robotics and Automation Letters* 6.2, pp. 1240–1247. DOI: 10.1109/LRA.2021.3057024

4. Michael D. Koptev, Sergey P. Trofimov, and Mikhail Yu. Ovchinnikov (2019). "Design and deployment of a tetrahedral formation with passive deputy nanosatellites for magnetospheric studies". In: *Advances in Space Research* 63.12, pp. 3953–3964. DOI: https://doi.org/10.1016/j.asr.2019.03.007