# Direct Sensor Integration for Optimization Fabrics

Caspar van Venrooij

TU Delft

# Direct Sensor Integration for Optimization Fabrics

by

Caspar van Venrooij

in partial fulfillment of the requirements for the degree of

**Master of Science**
in Robotics

at the Delft University of Technology,
to be defended on Friday, May 26, 2023, at 2:00 PM.

| | | |
|---|---|---|
| Thesis committee: | Dr. Javier Alonso-Mora | TU Delft, supervisor |
| | Max Spahn | TU Delft, daily supervisor |
| | Dr. Carlos Hernandez Corbato | TU Delft |
| Duration: | January 18, 2022 - May 26, 2023 | |
| Student number: | 4389581 | |

An electronic version of this thesis is available at `https://repository.tudelft.nl/`.

TU Delft

# Abstract

Autonomous robots hold great potential for positive impacts on society by applying them to tasks that are hazardous, repetitive, or complex and difficult for humans to perform. To achieve these tasks, autonomous robots require the ability to perceive environmental changes and create corresponding motion plans, which involve a combination of perception and motion planning techniques. Building a perception pipeline that can detect all relevant details in a dynamic environment is challenging and computationally expensive. To address this issue, the raw data output of a distance-measuring sensor can be used as a perception pipeline directly, transferring most of the computational load to the motion planner. However, most motion planning techniques cannot handle this high computational load. The motion planning technique optimization fabrics offers a promising solution, which utilizes a combination of differential equations to design robot behavior with extremely low computational complexity.

This thesis proposes a method for local motion planning that combines the low computational complexity of optimization fabrics with direct sensor integration. Our goal is to develop a method that enables autonomous robots to perceive and respond to changes in their environment quickly and efficiently. Our method, called direct sensor integrated (DSI) optimization fabrics, utilizes collision-avoidance differential equations generated for each raw data point collected from a distance measuring sensor. We adapted the regular optimization fabrics method to incorporate sensor data directly, and our analytical analysis shows that direct sensor integration does not require scaling adjustments to the regular collision-avoidance differential equations. By combining optimization fabrics with direct sensor integration, DSI optimization fabrics offers a promising solution to local motion planning. This method can enable autonomous robots to handle tasks that are challenging for humans to perform without needing a complex perception pipeline.

We conducted several experiments to assess our method, utilizing a simulated LiDAR-equipped point robot. First, we show empirically that an adjustment is required to properly scale the collision-avoidance differential equations, resulting in similar collision-avoidance behavior across sensor resolution. Second, we demonstrate that DSI optimization fabrics is feasible regarding computational complexity, achieving a frequency of $23$ Hz even with the maximum sensor resolution of 2048 LiDAR rays. Third, we demonstrate the effect of varying the sensor resolution on performance in multiple goal-reaching scenarios. We measure performance by monitoring the time-to-goal, total path length, minimum clearance from obstacles, and success rates. Fourth, we compare our method to regular optimization fabrics with a simulated perception pipeline in a scenario with static obstacles and a scenario with dynamic obstacles. In both scenarios, we show comparable performance regarding time-to-goal, total path length, minimum clearance from obstacles, and success rates. Finally, we showcase a real-world application of our method.

# Nomenclature

| Abbreviation | Definition |
|---|---|
| 2R robot | Two-revolute joints robot |
| DSI | Direct Sensor Integrated |
| DOF | Degrees Of Freedom |
| Fabric | Optimization Fabric |
| GC | Geometric Control |
| HD2 | Positively homogeneous of degree 2 in velocities |
| LiDAR | LIght Detection And Ranging of Laser Imaging Detection And Ranging |
| MPC | Model Predictive Control |
| OSC | Operational Space Control |
| PRM | Probabilistic Roadmap |
| RC | Radio Controlled |
| RGB | Red Green Blue |
| RMP | Riemannian Motion Policies |
| RRT | Rapidly-exploring Random Tree |
| Spec | Spectral semi-spray |

| Symbol | Definition |
|---|---|
| $\phi$ | Differential map |
| $\boldsymbol{\psi}$ | Optimization fabric forcing potential |
| $\boldsymbol{A}$ | RMP Riemannian metric |
| $\boldsymbol{B}$ | Damping matrix |
| $\boldsymbol{C}$ | An n-dimensional configuration space |
| $\boldsymbol{f}$ | RMP motion policy |
| $f$ | A map that maps a region on a manifold to a Euclidean space as $f : M \to X$ |
| $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$ | State transition function: $\boldsymbol{x}' = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$ |
| $\boldsymbol{f}(\boldsymbol{x}, \dot{\boldsymbol{x}})$ | Optimization fabric function of position and velocity |
| $G$ | Riemannian metric representing kinetic energy of a system |
| $\boldsymbol{h}_2$ | Geometry generator |
| $\boldsymbol{J}$ | Jacobian matrix |
| $\dot{\boldsymbol{J}}$ | Time derivative of the Jacobian matrix |
| $\mathcal{L}_e$ | Finsler energy |
| $\mathcal{L}_g$ | Finsler structure |
| $M$ | A manifold |
| $\boldsymbol{M}(\boldsymbol{x}, \dot{\boldsymbol{x}})$ | Metric tensor of a Langrangian |
| $\boldsymbol{O}$ | Set of obstacles inside the state space |
| $\mathcal{O}$ | Big O notation |
| $\mathcal{Q}$ | Configuration manifold |
| $\boldsymbol{q}$ | Robot position in the configuration space |
| $\dot{\boldsymbol{q}}$ | Robot velocity in the configuration space |
| $\ddot{\boldsymbol{q}}$ | Robot acceleration in the configuration space |
| $\mathcal{S}$ | Spectral semi-spray |
| $\mathcal{T}$ | An n-dimensional task space |
| $TM$ | Tangent bundle |
| $T_xM$ | The tangent space |
| $\boldsymbol{U}(x)$ | Action space, set of all possible actions at state $\boldsymbol{x}$ |

| Symbol | Definition |
|---|---|
| $\boldsymbol{u}$ | An action that changes a state into a new state $\boldsymbol{x}'$ |
| $X$ | An Euclidean space |
| $\mathcal{X}$ | State space, set of all feasible states $\boldsymbol{x}$ |
| $\mathcal{X}_{free}$ | The free state space: $\mathcal{X} \setminus \mathcal{XO}$ |
| $\mathcal{XO}$ | Combination of all obstacles in the state space |
| $\boldsymbol{x}$ | System position in the state space |
| $\dot{\boldsymbol{x}}$ | System velocity in the state space |
| $\ddot{\boldsymbol{x}}$ | System acceleration in the state space |

# Contents

# List of Figures

# 1 Introduction

For decades, robots have been widely used in various industries, including automotive, food, and electronics, to perform complex, repetitive, and dangerous tasks. Robots are increasingly utilized in non-industrial applications, such as healthcare, households, and supermarkets. The main challenge in robotics is to create autonomous robots capable of executing complex tasks while ensuring safety. Motion planning is a critical aspect of robotics that involves finding a valid plan for moving the robot to a goal location or configuration. A general distinction can be made between global and local motion planning methods. Global motion planning methods build a general path toward the goal, while local motion planning methods adjust the plan reactively using a perception pipeline to perceive environmental changes.

## 1.1. Motivation

To improve productivity while maintaining safety, robots require fast and reactive behavior, which involves a combination of a fast local motion planning method and a fast perception pipeline. The control frequency of the system is limited by either the motion planner or the perception pipeline. Therefore, a fast and reactive robot demands a fast local motion planner and a fast perception pipeline. A perception pipeline that processes sensor data to detect all obstacles and relevant environment details is challenging to build and computationally expensive, reducing the robot's reactivity. While using sensor data directly without any processing would be the most straightforward perception pipeline, the raw output is typically too intricate and dense for a local motion planner to handle. Consequently, more data must be processed by the local motion planner, leading to increased computation time and less responsive robot behavior.

While computation time harms optimization-based methods used for local motion planning, such as Model Predictive Control (MPC), geometric methods are typically faster. Optimization fabrics is a recent geometric approach to local motion planning that utilizes second-order differential equations to design robot behavior. Compared to other local motion planning techniques, optimization fabrics is computationally efficient, making them faster. By directly integrating sensor measurements into optimization fabrics, the robot can take unforeseen obstacles into account without the need for a complex detection process or a perception pipeline, thereby reducing computational costs and improving the reactivity of the robot.

## 1.2. Research question

This thesis explores how sensor data can be directly incorporated into optimization fabrics. The main research question is *"How can sensor data be directly integrated into optimization fabrics?"*. The thesis investigates the feasibility of directly integrating sensor data into optimization fabrics for reactive robot control and studies how the performance of optimization fabrics changes with the complexity of sensor data. Additionally, it compares DSI optimization fabrics' performance to regular optimization fabrics that rely on a perfect simulated perception pipeline, assuming knowledge of all obstacle locations. The

thesis also provides insight into the scaling of optimization fabrics with direct sensor integration. A scaling factor adjusts the equations to maintain consistent collision-avoidance behavior across different sensor resolutions. Finally, the viability of direct sensor-integrated optimization fabrics in a practical scenario is demonstrated through a real-world experiment.

## 1.3. Contribution

The main contribution of this thesis is developing and evaluating DSI optimization fabrics for local motion planning in robotics. This thesis investigates the feasibility and performance of integrating sensor data directly into optimization fabrics for local motion planning. A comparison is made between DSI and regular optimization fabrics where the locations of obstacles are assumed to be known. The thesis analyzes the collision-avoidance equations in optimization fabrics and their sensitivity to sensor resolution. Through this analysis, it was found that no scaling is required in a simplified scenario. However, in simulation, empirical findings revealed that a scaling factor is necessary to adjust the collision-avoidance equations to maintain consistent collision-avoidance behavior across different sensor resolutions. Finally, the viability of DSI optimization fabrics in a practical scenario is demonstrated through a real-world experiment, showing the potential of this approach to improve the safety and efficiency of robotic systems.

## 1.4. Structure

The remaining chapters are structured as follows. Chapter 2 provides an overview of related work on motion planning and perception. Chapter 3 presents relevant robotics notations and the theory of optimization fabrics. The method of DSI optimization fabrics is described in detail in Chapter 4. The experiments and results are presented in Chapter 5. Finally, Chapter 6 concludes the thesis and suggests directions for future work.

# 2

# Related Work

This chapter offers an overview of related work on motion planning and perception. Section 2.1 outlines the motion planning problem and briefly overviews its two interconnected phases: local and global. The related works presented in sections 2.2 and 2.3 cover local motion planning techniques for reactive control and work on perception, respectively.

## 2.1. Motion planning

Motion planning is a subfield of robotics that regards the problem of finding motions to move the robot to a goal location or goal configuration collision-free by using planning algorithms. Path planning is a subproblem of motion planning and only concerns finding a collision-free path between two states without considering whether this path is dynamically feasible for a robot [24]. Motion planning concerns finding a motion path under certain constraints, e.g., joint limits and torque limits. In general, a feedback controller is used to follow the motion path obtained from the motion planner. A general distinction between global and local techniques is made in this chapter. However, many techniques also overlap the two domains. This chapter briefly overviews the differences and describes the more widely used global and local motion planning methods.

### 2.1.1. Notations for motion planning

This section gives an overview of basic notations of motion planning as found in [19]. Vectors are denoted in bold lower case, and matrices in bold upper case. A state-space model of a physical system and its environment can be constructed by discretizing it. A single state $\boldsymbol{x}$ represents the state of a physical system and its environment at a specific time step, and the set of all feasible states is called the state space $\mathcal{X}$. Information from the physical system or environment irrelevant to motion planning should be excluded from the state space to lower the computational complexity of algorithms using the state space. An action, $\boldsymbol{u}$, changes the current state into a new state $\boldsymbol{x}'$ from the state space by a state transition function $\boldsymbol{f}$. The new state can be expressed with the state transition function $\boldsymbol{x}' = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$ [19]. $\boldsymbol{U}(x)$ is defined as the action space which represents the set of possible actions at state $\boldsymbol{x}$, the combination of all possible actions is defined by the following set [19]:

$$\boldsymbol{U} = \bigcup_{\boldsymbol{x} \in \mathcal{X}} \boldsymbol{U}(x)$$

Obstacles inside the state space are represented with the set $\boldsymbol{O}$ and the combination of all obstacles in the state space as $\mathcal{X}\boldsymbol{O}$. The free state-space is defined as $\mathcal{X}_{free} = \mathcal{X} \setminus \mathcal{X}\boldsymbol{O}$ [19].

A motion planning problem can be formulated as follows [19] [16]:

- A state space $\mathcal{X}$ consisting of all possible states $\boldsymbol{x}$
- An obstacle free state space $\mathcal{X}_{free} = \mathcal{X} \setminus \mathcal{X}\boldsymbol{O}$

- A state transition function $\boldsymbol{f}$, with as input a state $\boldsymbol{x} \in \mathcal{X}$ and an action $\boldsymbol{u} \in \boldsymbol{U}(\boldsymbol{x})$, and as output a new state $\boldsymbol{x}'$ derived as $\boldsymbol{x}' = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$
- An initial state $\boldsymbol{x}_i \in \mathcal{X}$ and a goal state $\boldsymbol{x}_g \in \mathcal{X}$
- A continuous function $\sigma$, that takes a mapping parameter $\tau$ to a curve in $\mathcal{X}_{free}$ for all $\tau \in [0, 1]$
- $\sigma : [0, 1] \mapsto \mathcal{X}_{free}$ with:
- $\sigma(0) = \boldsymbol{x}_i, \sigma(1) = \boldsymbol{x}_g, \sigma(\tau) \in \mathcal{X}_{free}$

### 2.1.2. Methods

Various motion planning techniques exist, and a general distinction can be made between global or offline motion planning and local or online motion planning techniques. Global motion planning techniques compute a *global* motion plan towards a goal state. Local motion planning techniques plan motions *locally* along a global motion path obtained from a global motion planner. A commonly used strategy in motion planning is then to compute a global motion plan *offline* once or periodically and a local motion plan *online*, resulting in a motion planner that can reactively plan along a global plan. A local motion planner is helpful in the case of a dynamic environment with moving obstacles. The location of the obstacles in all coming states cannot be known beforehand. Therefore it is impossible to consider the obstacles in the global motion plan. The local motion planner will deviate from the global motion plan to reactively avoid the obstacles. Also, high dimensionality and a high environment resolution will result in increased computational complexity of the global motion planning technique. A motion planning technique's increased computational complexity will require more time to execute and often more computational space to store it. Therefore, it can be beneficial to keep the resolution of the environment lower to obtain better global motion planning results while smoothing the global motion plan again with the local motion planner, which can have a higher resolution for the local details of the environment. Figure 2.1 shows a simplified workflow for a combined global and local motion planner. The global motion planner has as inputs the initial state $\boldsymbol{x}_i$ and the goal state $\boldsymbol{x}_g$, and computes a global motion plan $\sigma(\tau)$ as output. The local motion planner is part of an iterative process that takes as inputs the global motion plan and the new state $\boldsymbol{x}'$ and returns an action $\boldsymbol{u}$ for the system in the real world. The new state $\boldsymbol{x}'$ results from the system interacting with the real world under the influence of action $\boldsymbol{u}$. This iterative process continues until the goal state is reached or until it fails to do so. The local motion planner is executed online while the states are transitioning, and the global motion planner is computed offline beforehand. In the case of failure, the global motion planner can replan a global motion plan from the new initial state.

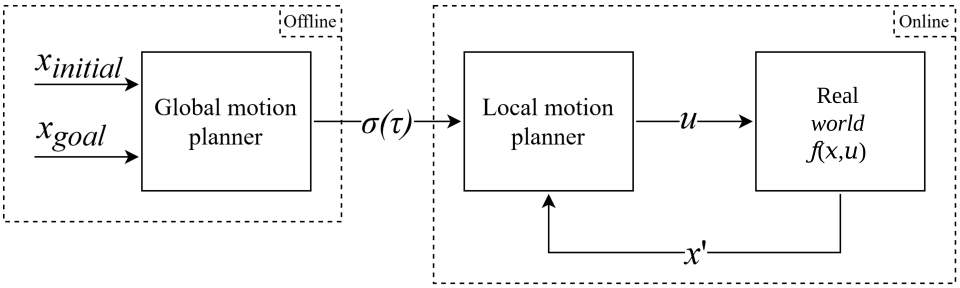



**Figure 2.1:** Simplified workflow of a combined local and global motion planner. The global motion planner computes an offline global motion plan based on the initial and goal state. The local planner iteratively computes an action online based on the new state obtained from the real world.

#### Global motion planning

Most offline or global motion planning techniques use a discretized model of the environment or world [24]. By defining the world with a graph or grid, graph theory and graph-search methods can be used to find a motion path on the graph [19]. A graph consists of vertices (nodes) and edges between the nodes. The nodes correspond to discretized locations in the real world, with edges connecting the nodes. The edges can be directed or undirected and are weighted optionally. Graph search methods find the globally lowest-cost connection between an initial and goal node [19]. Various graph-search

algorithms exist, e.g., forward search, breadth or depth-first, Dijkstra, Bellman-Ford, A* [19]. All algorithms have specific properties in terms of optimality and time complexity. Graph-search methods for motion planning are limited by their time complexity, especially as the state-space grows in complexity or resolution, and limits the applicability to low-dimensional environments [24].

Sampling-based motion planners are another method for global motion planning and can solve high-dimensional motion planning problems [24]. Instead of building a graph or grid for the whole environment in one go, sampling-based methods take (non)-uniform samples in the state-space [19]. The samples are verified to be collision-free, and the distance between the collision-free samples and the previously obtained collision-free samples is determined. By doing this, a graph is built consisting of nodes and edges representing viable motions [19]. Sampling-based methods can be split into single-query and multi-query methods [19]. Multi-query methods, e.g., probabilistic roadmap (PRM) [17], construct a global graph and use graph search methods to find a path connecting a starting and goal node on the graph. Multi-query methods are limited by dimensionality as the time complexity rises with the size of the graph. Single-query methods, e.g., rapidly-exploring random tree (RRT) [20], create a graph towards a specific goal location. A tree is incrementally grown, rooted at the initial position towards a goal position. Samples that are in a collision are not added to the graph. Once a solution is found, the method builds further and returns the shortest path. RRT is not optimal; there is an asymptotically optimal variant (RRT*) [16]. The single-query sampling-based motion planners can solve high-dimensional motion planning problems [19]. Still, the solutions might not be smooth, finding solutions can be slow, and since it is a single query method, a new tree has to be built for every new goal [19] [24].

**Local motion planning**

Local or online motion planning is used to smooth out a global motion plan or to consider dynamic environments, which is essential for reactive control. As described in the previous section, most global motion planner techniques are constrained by computational complexity due to the environment's high dimensionality and high resolution. Global motion planners can reliably find global motion plans by simplifying the environment. However, these motion plans can be rough and will not result in smooth robot behavior when followed. A local motion planner can smooth out the global motion plan. There are various methods to achieve reactive behavior, and a general distinction in local motion planning can be made between virtual potential field methods and nonlinear optimization methods.

Virtual potential field methods use force fields that push or pull a robot towards or away from specific locations in the state space, such as a goal or obstacle [24]. Attractive potential fields can pull the robot toward a goal location or intermediate locations of the global motion plan towards a goal. In contrast, repulsive potential fields can push the robot away from obstacles. This method has a relatively uncomplicated implementation and is fast [24]. However, the robot might get stuck in the local minima of the potential function [24]. Also, positive and negative potential fields can cancel each other out before the robot reaches the goal [24] [34].

Optimization techniques can also be used for local motion planning. A local motion planning problem can be converted to an optimization problem by defining a cost function and constraints [24]. The cost function assigns a cost value based on a given input. In motion planning, it is commonly defined as the distance to the goal, with a higher cost for longer distances. Similarly, it considers the distance to obstacles, with a lower cost assigned for greater distances. The inputs are the control inputs and a dynamic model of a system together with the state space. Constraints define limits on the control inputs that must be satisfied. Various optimization algorithms or techniques can then be used to minimize or maximize this cost function and, in the case of motion planning, find control inputs that minimize the cost. Model predictive control (MPC) [26] is a process control method with optimization techniques. MPC has been successfully applied to various fields in the past decades [12], including robotics [7]. MPC is an iterative optimization method that requires a model of the system, a cost function, and an optimization algorithm to minimize the cost function. With MPC, the cost function is defined not only for the next state but for several states in the future as the receding horizon. MPC optimizes the cost function defined for the horizon or future states and uses the control input found in the first step to control the system; this is done iteratively for the next steps or states while the horizon keeps receding. MPC requires a sufficiently accurate system model and extensive controller design [12]. Instead of analyzing a control system on stability and guarantees after it has been built, MPC can provide theoretical guarantees by design [12]. The computational complexity of MPC grows with the number of recorded data points [12].

The solving time of MPC increases sharply by including more obstacles [41]; this limits the number of obstacles that can be included while still maintaining reactive control.

### 2.1.3. Summary

For a robot to perform a specific action, control inputs are needed for the robot's actuators. The control inputs move the robot from an initial position toward a goal position. Motion planning calculates a plan or path that moves the robot to the desired position. Typically, a global motion planner is used to construct a global path toward the goal location, and a local motion planner is used to smooth out the global plan or to deviate from this path in case of unforeseen obstacles. Global motion planning techniques are often based on graph search methods and sampling-based methods. Virtual potential fields and optimization techniques are two techniques that can be used for local motion planning. Virtual potential fields are uncomplicated to implement and fast but are prone to local minima, and the potential fields can conflict with each other. MPC is a popular optimization-based technique employed for reactive control in various domains. While MPC offers theoretical guarantees, its practical application is constrained by its computational complexity.

## 2.2. Reactive control

This section outlines four local motion planning techniques used for reactive robot control. The techniques include Operational Space Control (OSC), Geometric Control (GC), Riemannian Motion Policies (RMP), and Optimization Fabrics (fabrics). The first three serve as the foundation for optimization fabrics. In sections 2.2.1, 2.2.2, 2.2.3, and 2.2.4, the fundamental concepts, applications, and limitations of each technique are explained in detail.

### 2.2.1. Operational space control

The operational space (OSC) framework was first introduced in 1987 by Khatib [18]. It proposed a method for controlling manipulator systems with dynamic end-effector behavior in the operational (task) space. OSC allows a manipulator to be controlled as an operational space control system instead of a joint space control system. This is achieved by transforming the equation of motion from the operational space to the joint space for the controller. This technique is beneficial when a task is better defined in the operational rather than the configuration space. This is often the case for many desired robot behaviors, such as controlling the end-effector. In OSC, the desired joint space velocity, acceleration, and torque are calculated from the desired end-effector space velocity, acceleration, and force [29]. The equations of motion relate the velocity, acceleration, and forces or torques of a robotic system in both joint and configuration space. In OSC, the equations of motion in both operational and joint space are based on classical mechanical systems and are derived using Lagrangian mechanics based on the kinetic and potential energy of the system [18].

To control a manipulator using OSC, a function for the desired acceleration in operational space is defined using the current and goal end-effector position and velocity. Calculating the torque in configuration space is possible using the desired acceleration function and the equation of motion. Another layer of control is added in OSC for nonlinear dynamic decoupling of the end-effector, making it a two-level control system. Besides controlling the end-effector towards a goal location, more behavior is needed for reactive control, such as collision avoidance. However, the original framework does not describe combining multiple motion policies. A variant does [44]; this variant uses Bayesian techniques to fuse numerous motion policies, similar to Bayesian sensor fusing. Still, it is limiting and not well-defined in the literature. The amount of motion policies that can be combined in OSC is limited by the number of degrees of freedom of the robot [5].

#### Applications

The original framework [18] is developed to be used for manipulators. The OSC framework is implemented using the manipulator programming system COSMOS (Control in Operational Space of a Manipulator-with-Obstacles System) using a PUMA 560 manipulator [18], a robot with six degrees of freedom. Experiments have been conducted by performing basic assembly operations and with square wave force inputs. The framework shows a high-level dynamic rate of $100$ Hz and effective elimination of bounces at contact with stiff surfaces. The square wave force input experiments showed responses with $0.02$ s rise time and less than $12$ percent steady force errors. However, extensive results are not

given. For redundant manipulator systems, only the equations of motion are defined, without experiments or results shown.

In [29], an extensCalculatingal evaluation of various variants of OSC is given. Three main variants are considered, force-based control as in the original framework [18] with extensions by the same author [8], velocity-based control [28] and acceleration-based control [13]. All control variants are evaluated for multiple simple motion tasks using a seven-degrees-of-freedom Sarcos Master Arm robot. Regarding theoretical complexity, the velocity-based controller is the simplest to model and implement, followed by the acceleration-based controller and the most complex variant, the force-based controller. The force-based controller performs almost perfectly in simulations. However, it scores low in the experimental evaluations. Compared to the other two methods, the force-based controller requires more accurate model identification, and the performance deteriorates quite heavily under modeling errors. The velocity-based controller is relatively straightforward to implement and achieves good tracking performance in simulations and real-world experiments. However, the controller lacks the complexity to model certain desired behaviors. Lastly, the acceleration-based controller showed the overall best task performance in simulations and real-world experiments, especially in the most demanding experiments. A simplified version of this controller had the overall best performance, ease of parameter tuning, and general robustness and compliance [29]. Regarding computational complexity, the acceleration-based and velocity-based controllers are less expensive to compute compared to the force-based controller. The computational complexity of the force-based controller is $\mathcal{O}(n^2)$, while for the other controllers, it is $\mathcal{O}(n)$.

**Limitations**
These extensions improve the original framework. However, OSC is still limited as a reactive control technique. The applications are primarily concerned with end-effector tasks defined in the task space. Besides these end-effector tasks, reactive control requires avoiding obstacles and, thus, defining more than one motion policy. While combining multiple motion policies is possible with OSC, it is limited by the number of degrees of freedom of the robot [5] and is not well-defined in the literature.

In OSC, the task spaces are assumed to be Euclidean, which limits the direct consideration of non-Euclidean spaces [5]. While Euclidean space is suitable for tasks such as reaching a goal, obstacles can cause the space to become curved and non-Euclidean. Consequently, it becomes necessary to define collision-avoiding behavior within this non-Euclidean space.

### 2.2.2. Geometric control

Geometric control is a technique to model, analyze and control mechanical systems. Operational space control can be seen as a subset of geometric control [34], although geometric control was developed mostly independent of OSC [5]. The book *Geometric control of mechanical systems* [3] can be viewed as a primary reference for geometric control. This book considers geometric control for *simple* mechanical systems. In this context, *simple* relates to mechanical systems described as a class of Lagrangian systems, with the Lagrangian defined as the difference between kinetic and potential energy. These types of *simple* systems are also referred to as *natural* systems. To control these *simple* mechanical systems, geometric control utilizes the mathematical overlap, described with differential geometry, between the fields of nonlinear control and classical mechanical systems.

Differential geometry is a branch of mathematics that deals with the geometry of smooth shapes and spaces. It uses techniques from differential calculus, linear algebra, and other mathematical tools to study these smooth shapes and spaces [3]. Smooth shapes and spaces mean these are continuous and differentiable [3]. This branch of mathematics calculates equations between the configuration space and task space in robotics control. Geometric control is a generalization of operational space control (OSC), as it allows for swapping between spaces using forward and inverse kinematics as a mapping between spaces. The primary object of differential geometry is a manifold [3]. In geometric control, the manifold describes the different spaces or shapes, e.g., configuration and task space. An n-dimensional manifold $M$ is a space that locally resembles an n-dimensional Euclidean space. Locally around a point on the manifold, defined as the neighborhood, we can find an n-dimensional Euclidean space homeomorphic to this neighborhood region. Homeomorphic means that there is a continuous function and inverse function that maps between the two spaces that preserve the properties of both

spaces. This mapping between a section of a manifold $M$ and a Euclidean space $X$ is done with what is called a map or chart $f$, defined as $f : M \to X$.

Other important notions from differential geometry are the tangent vector, tangent space $T_xM$, and tangent bundle $TM$. These describe the direction and velocity of smooth curves moving through a point on the manifold. At a point $x$ on the manifold, we can define a set of all smooth curves that move through this point. A tangent vector is defined for every smooth curve at point $x$; this tangent vector is tangent to the specific smooth curve, showing the direction or derivative of this curve at point $x$. The combination of all tangent vectors, for all smooth curves at a single point $x$, is the tangent space $T_xM$ [21]. Lastly, the tangent bundle is the disjoint union of all tangent spaces $T_xM$ [21]. The tangent bundle $TM$ is also a manifold, consisting of all the possible velocities of all points on a manifold. So, at a certain point $x$ on a manifold, multiple tangent vectors are defined for multiple smooth curves moving through $x$. The combination of these vectors is the tangent space $T_xM$, and the combination of all tangent spaces is called the tangent bundle $TM$ [21]. Using the tangent space, we can define a Riemannian metric for a manifold. A Riemannian metric is a metric tensor with some extra properties [2]. A metric tensor takes the tangent space as input, resulting in a scalar. A Riemannian metric takes the positive-definite inner product of the tangent space $T_xM \times T_xM$ [2] [3]. A manifold with a Riemannian metric is called a Riemannian manifold [2]. In geometric control, a kinetic energy metric is used for the Riemannian metric of a Riemannian manifold [3]. A Riemannian manifold consists of the pair $\mathcal{Q}, G$, with $\mathcal{Q}$ the configuration manifold and $G$ the Riemannian metric representing the kinetic energy of the system [3]. The *simple* mechanical systems with a Riemannian metric based on the kinetic energy are observed to evolve as geodesics across a Riemannian manifold [5]. In geometric control, behavior is shaped by controlling a system to behave as a virtual classical mechanical system, as in OSC [34]. The virtual classical mechanical system is modeled using Lagrangian mechanics, which are based on the system's kinetic and potential energy representations.

To recap, geometric control generalizes OSC and uses concepts from differential geometry, nonlinear control, and classical mechanical systems to create a virtual mechanical system to define geometries or curves in the configuration manifold, which can be used as a motion policy for the controller of a robot.

**Applications**
Geometric control is defined more broadly as compared to OSC. Geometric control can be used for any system, described using *simple* mechanical systems. This includes manipulators but also other types of robots and systems. Both fully actuated and underactuated systems are described in [3]. However, experiments or simulations of geometric control with robots are lacking.

**Limitations**
One main limitation of geometric control is that the task priority weights are defined as positive-definite matrices and are position dependent only [5], not velocity dependent. Therefore, it cannot express the priorities for different directions at a specific location. This lack of directionality is problematic since it can be crucial for more advanced control situations, especially collision avoidance.

Directionality is essential for collision avoidance since the position relative to an obstacle gives only partial information. An obstacle can move towards a robot or away from it while being in the same location. The velocity of an obstacle can be used to deduce the obstacle's direction of motion. With the obstacle's direction of motion available, it is possible to perceive whether the obstacle is moving towards or away from the robot, which is crucial for the system's motion policies. The simple example in figure 2.2 shows a scenario where the directionality would provide helpful information for the behavior of a two-revolute joint (2R) robot. Both obstacles are equally close to the robot's end-effector, but the robot is at a greater risk of collision with the upper obstacle since it is in motion toward the robot itself.

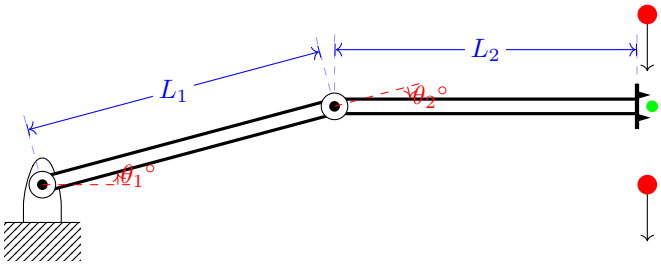



**Figure 2.2:** 2R robot example with two obstacles (red). The obstacle above the robot moves toward the end effector, and the obstacle below the robot moves away from the end effector. The obstacles are equally close to the robot, but the robot is at greater risk of collision with the upper obstacle since it is moving toward the robot. In this example, the directionality of the obstacles provides useful information for motion planning.

### 2.2.3. Riemannian Motion Policies

OSC and geometric control are limiting, with missing directionality and no implicit consideration of non-Euclidean geometries. These limitations lead to the inability of these techniques to combine multiple local motion policies effectively. Riemannian Motion Policies (RMP) are introduced [36] [4] as a continuation of these techniques. RMPs effectively combine multiple motion policies by introducing solutions to the directionality and implicit consideration of non-Euclidean geometry limitations. Each RMP is paired with a Riemannian metric which defines the importance of a policy with directionality; it considers both position and velocity. To combine multiple policies into one, we must be able to define how they contribute individually and how to transform them between spaces.

A RMP is defined as the tuple $(\boldsymbol{f}, \boldsymbol{A})_{\mathcal{X}}$, with a motion policy $\boldsymbol{f}$ and a Riemannian metric $\boldsymbol{A}$ defined in the space $\mathcal{X}$. The motion policy $\boldsymbol{f}$ maps position and velocity to an acceleration, which is the desired acceleration for the robot's low-level controllers. Like OSC and geometric control, the RMPs can be transformed between spaces using pullback and push-forward operations. RMPs also introduce an operation to obtain the metric-weighted average of multiple RMPs [36]. For now, we will not describe these operations here further, as these are similar to the operations used in optimization fabrics in the following chapter.

To generate motions for a robot, the following is done with RMPs. A single RMP is created in the corresponding task space for each desired subtask or sub-behavior for the robot, such as reaching for a target or collision avoidance. The RMPs are combined into a single RMP in the configuration space using the pullback and summation operations. Using the RMP in the configuration space, the motion policy can be used as a desired acceleration for a low-level controller of the robot.

As the RMPs are defined separately, they can be computed in parallel. This is useful since this makes it possible to compute RMPs that need a lower control frequency but have a higher complexity offboard and compute RMPs that require a higher control frequency and are less complex onboard. By parallelization and separation of computation, improved computational performance is possible [36]. In the paper, RMPflow [4], a computational graph is introduced, the RMP-tree. On this tree, the various RMPs are defined as leaf nodes. The tree's root node combines the nodes in the configuration space. Figure 2.3 shows three RMP-tree examples with increasing complexity from left to right. The root node is shown at the bottom and denoted as $\boldsymbol{C}$, as this node is defined in the configuration space. The leaf nodes are represented by $\mathcal{T}$, and these are described in the task space. The pullback and summation operations can be used to pull the leaf nodes, defined in the task space, back to the configuration space and combine them into one motion policy.

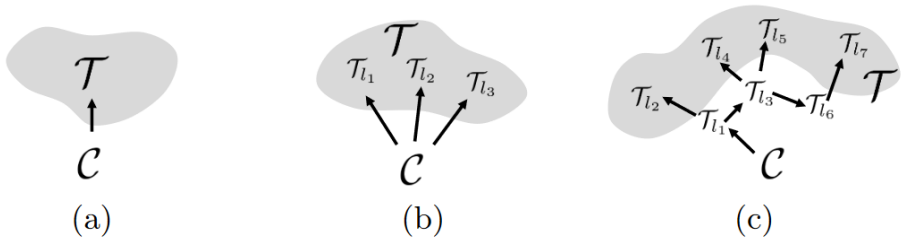



**Figure 2.3:** RMP-tree structured task maps showing common examples in increasing complexity from left to right. The nodes on the graphs are manifolds, and the edges are transformations. The root node is defined in the configuration space $\boldsymbol{C}$, and the leaf nodes are in the task space $\mathcal{T}$. For each subtask, a task space leaf node is made. From RMPflow: A Computational Graph for Automatic Motion Policy Generation, fig. 1. [4]

**Applications**

In the original paper introducing RMP [36], two simulation experiments are performed using a seven-degrees-of-freedom manipulator. In the first experiment, three cluttered environments with four cylindrical obstacles each are generated. A distribution of goal points on the other side of the obstacles is chosen. RMP is compared to a general OSC approach, wherein multiple policies are transformed using pseudoinverses and superimposed for a single motion policy. The general OSC approach shows conflicting behavior; often, the policies fight each other and result in a collision with obstacles. Also, complexity is missing to balance stability and achieve the task properly. RMP showed natural, predictable, and smooth motion for each task.

In the second experiment [36], four cluttered environments with four cylindrical obstacles each are generated. Configurations are manually designed to create environments where the robot must navigate the obstacles in various ways, e.g., between and around obstacles. In this experiment and the previous one, a function or heuristic is used for the RMP that must move toward a goal. The second experiment compares two types of these heuristics, a simple and a more sophisticated. The simple one is already quite robust, but the sophisticated one performs slightly better under challenging environments [36], such as in a larger workspace or long-range navigation.

The RMPflow paper [4] performs simulation and real-world experiments. The simulation is performed with a modeled seven-degrees-of-freedom ABB YuMi manipulator. The manipulators must reach a goal location with obstacles in between. RMPflow is compared with OSC. OSC fails to match the speed and precision of RMPflow. The real-world experiments are performed with the dual arm manipulation robots; both arms have seven degrees of freedom, the ABB YuMi and the Rethink Baxter, with integrated vision. RMP performed exceptionally well and showed generalization between the two robots; no extensive results were revealed.

Another application uses RMP [25] to generate smooth collision-avoiding behavior for a small RC car. The car is equipped with an RGB camera. Distance information is needed from these images to steer the vehicle, but depth cannot be directly obtained from an RGB camera. Three deep neural network methods are introduced and compared to overcome this limitation. The methods are predicting depth, directly predicting controls, and predicting whole RMPs. Using these methods, the RC car is navigated autonomously through a hallway with obstacles in simulations and real-world experiments. The simulations show that the predicting RMP method achieves the highest goal completion and lowest chance of collision. The methods generalized poorly in the real-world experiments; manual fine-tuning was first needed using real-world images. After fine-tuning, the predicting RMP method showed the best goal-reaching and collision-avoiding behavior. Although the predicting RMP method has a high chance of reaching the goal and avoiding obstacles, it is still a local policy, and there is no guarantee that it will converge to a desired local minimum. Therefore, the authors state that using a neural planner for a global plan might be more suitable than using the neural RMP for handling the vehicle dynamics and reactive control to avoid obstacles [25].

**Limitations**

In [4], stability is analyzed for RMP, but formal stability guarantees are lacking. To create a stable class of RMP, experience is needed; therefore, they can be dangerous for inexperienced users [34].

Only a subset of the possible RMPs that can be designed are stable; however, knowing why this is the case or how to create stable variants requires knowledge of RMP and geometric control. These are strict requirements and limit RMP's usability to those who know how to design them. Creating stable RMP requires knowledge of the design principles, parameter tuning techniques, and damping heuristics [34]. Users that do not know to design stable variants can create unsafe applications. With RMP, the priorities are directly coupled with the motion policies. This direct coupling limits expressibility since it is impossible to create a priority metric and motion policy separately.

### 2.2.4. Optimization fabrics

Optimization fabrics [34] is a local motion planning or reactive control technique. Whereas geometric control is, a generalization of OSC and RMP is a class of geometric control. Optimization fabrics are a generalization of RMP and geometric control. It continues on the theory and concepts of RMP but improves upon the lack of stability and makes the theory more accessible and less dangerous for inexperienced users. The motion policy is a desired acceleration obtained from a second-order differential equation. Optimization fabrics combine multiple second-order differential equations into one. The multiple separate second-order differential equations define a desired subtask behavior, e.g., collision avoidance, redundancy resolution, joint limit avoidance, and goal guiding. These separate equations can then be combined into one by using a metric that defines how important a subtask behavior is. The modularity makes them easier to build, use, maintain, and calculate in parallel, resulting in a quick execution time.

**Positively Homogeneous of degree 2**

A critical property used for optimization fabrics functions is positively homogeneous of degree 2 (HD2) in velocities. This property assures path consistency for optimization fabrics and is crucial for designing behavior independent of the robot's velocity [34]. A function is HD2 in velocities if and only if:

The value of the function, acceleration in this case, is scaled by $\alpha^2$ if the velocity is scaled by some scalar $\alpha > 0$.

The property of HD2 relates the velocity and acceleration of a function and results in path consistency for trajectories computed with optimization fabric using these functions. Path consistency means that the paths generated from these functions are unique for initial values [34]. This path consistency property due to HD2 can be described by looking at the units of velocity and acceleration, $\frac{m}{s}$ and $\frac{m}{s^2}$. Velocity is related to time by the power of one, and acceleration is related to time by the power of two. If the velocity of a trajectory is scaled by some scalar $\alpha$, acceleration must be scaled by the same scalar squared $\alpha^2$. By doing this, the trajectory will remain equal; the only difference is a varying velocity along the trajectory. Choosing differential equations for which this property holds can be advantageous, as this will result in the consistency of paths generated by these differential equations. If an optimization fabrics function is HD2 and therefore has path consistency, the system will always follow the same trajectory if starting from the same starting conditions. It is possible to change velocity along these trajectories, speeding up and slowing down. It is impossible to change the velocity or acceleration along different directions than the direction of motion.

The result is a geometric fabric wherein behavior and priority are uncoupled. The behavior can be designed using the geometry generator, and the priority metric follows from the Lagrangian energy. By bending, a new geometry is obtained that preserves its original HD2 geometry and energy. The HD2 geometry follows the desired geometry behavior and gains a Finsler geometry's energy metric for prioritization [34].

**Applications**

To use optimization fabrics for reactive robot control, separate components must be designed for the desired behaviors. All components are built using two parts, the Finsler energy and a policy. The two most essential components are goal-reaching and collision-avoiding behaviors. A component uses a potential attractor function for the robot to reach a target. The collision-avoiding behavior component is designed with a collision-avoiding geometry generator. Both components are paired with a Finsler energy.

In [36], two experiments are performed. One with a simple two-dimensional point mass system and one with a two-dimensional planar manipulator. The point mass system is used to demonstrate and

validate the theoretical properties such as path consistency, energization commuting with the pullback, and other fabric operations. The planar manipulator experiments are similar to the point mass system experiments. The desired behavior for the arm is built from geometric fabrics to move to random goals in the environment continuously. The robot reaches all the goals and shows the desired behavior. Similar planar manipulator experiments are performed in [33]. General Lagrangian fabrics are compared to geometric fabrics in [47] using planar manipulator experiments. It is shown that the general Lagrangian fabrics have issues with reaching the target due to fighting potentials, whereas the geometric fabrics reach the target.

Simulations with a seven-degrees-of-freedom Franka arm are performed in [47]. In the experiment, there are six reachable bins near the arm. Geometric fabrics make the arm reach into and out of the bins. The results show natural and successful behavior. To test generalization, two experiments were performed. First, 90 problems starting from the same configuration were performed with a 96.7% success rate. After rotating the starting configuration 30 degrees, the same experiments were performed, now with a 92.2% success rate. By additional tuning, a success rate of 100% was obtained.

In [46], more experiments are performed with the seven-degrees-of-freedom Franka arm. A geometric fabric is designed once and tested with multiple problems, reaching for a target with varying obstacles. The geometric fabric is compared to the current best-performing RMP. It can be seen that RMP performs worse due to conflicting behaviors. Moving towards obstacles, the RMP keeps a more considerable unneeded distance; in some cases, this results in the RMP being unable to reach the target location. In experiments with moving obstacles, RMP and geometric fabric are only given the position of the obstacles. The RMP reaches 11 of the 19 targets and has a collision rate of 8.18%, while the geometric fabric reaches 16 targets and has a collision rate of 0.4%.

The functions used in fabrics have to be designed by hand. There has been some research into learning the functions used in RMP [32] [27] [22], which can improve the standard functions or make it more straightforward for inexperienced users to design them for a specific scenario by training. The work [40] presents an automated parameter optimization method for trajectory generation using optimization fabrics.

**Limitations**
Optimization fabrics have no collision avoidance guarantees. At every time step, a new acceleration is computed, which can take obstacles into account. The acceleration that can be given to the robot's controllers is limited by the maximum physical acceleration of a robot. If an obstacle can move with a higher acceleration than the robot, the robot cannot avoid it. Techniques like MPC can predict future states with the receding horizon. If one of these future states shows the robot in collision with an obstacle, actions can be taken in advance, i.e., shutting down the robot. By doing this, MPC can guarantee that the robot will not dynamically collide with an obstacle, which is essential for human and robot interaction. Collision avoidance guarantees can be added to optimization fabrics by including some prediction horizon that checks for collisions. However, this is not inherently included in the theory and would result in higher computational complexity.

All applications given in section 2.2.4 assume that all obstacles' positions and sizes are known. In real-world scenarios, however, these must first be obtained. A perception pipeline is required to obtain information about the environment and obstacles. Different kinds of perception pipelines exist, which remain to be tested in combination with optimization fabrics. Most functions used in optimization fabrics are designed by hand, which limits inexperienced users to using the standard functions, which might be suboptimal, or designing the functions themselves. The theory of optimization fabrics improved significantly upon the older techniques of RMP and geometric control in terms of designing them for inexperienced users. By following the standard concepts introduced, a new user can design stable behavior, which is still far from straightforward. As fabrics are a local technique without a prediction horizon, as in MPC, the obtained trajectory might lead the robot toward a local minimum.

### 2.2.5. Summary
OSC uses a relationship between the forces acting on the robot in the task space and the torques acting on the robot in the configuration space for end-effector control of a manipulator. Using the relationship, tasks can be modeled and calculated using Lagrangian mechanics in the task space, where it would be

logical to perform a specific task and then relate it to torques in the configuration space, which can be used to control the robot. Geometric control introduced a similar method to control robots or mechanical systems. The method is a generalization of OSC where any robot modeled as a *simple* mechanical system can be controlled. *Simple* relates to a class of mechanical systems that can be described using Lagrangian mechanics, which are purely based on potential and kinetic energy. Mapping between OSC's task space and configuration space is generalized as a mapping between spaces or manifolds in differential geometry. In doing so, the method obtains notions and concepts from differential geometry that benefit robot control. A metric is obtained that states the importance of a specific task, which makes it possible to construct behavior built from multiple tasks. However, it is impossible to include velocity information in this metric, and therefore, geometric control cannot take directionality into account, which can result in conflicting behaviors. RMP is introduced as an extension and continuation of OSC and geometric control. It takes the general concepts from both methods to create a motion policy that can be built from multiple separate desired behaviors or tasks. By also including velocity information in the metric, directionality can be considered. RMP also introduces a computational graph, the RMP tree, and specific operations, RMP algebra. The RMP tree can create a motion policy at the root from multiple separate RMPs at the leaves. The RMP algebra is then used to combine the leaves into the root; the output of this root is an acceleration that can be used as an input for the robot's low-level controllers. RMP still lacks formal stability guarantees and can be dangerous for inexperienced users. Fabrics are a reactive control or local motion planning technique. They are built from second-order differential equations known as spectral semi-sprays or specs. Similar to RMP, behavior can be built by combining multiple specs into one spec. From this resulting spec, acceleration can be computed, which is used as the input for the robot's low-level controllers. Spec algebra operations can combine the various specs and map them between task spaces and the configuration space across a transform tree. The homogeneous of degree 2 in velocities (HD2) property is critical for geometric generators and geometric fabrics; the property ensures path consistency. Finsler fabrics are introduced as a generalization of Lagrangian fabrics, where the Lagrangian is a Finsler energy. A geometric fabric is a geometry generator energized with Finsler energy. With this, the design of behavior and priority is uncoupled. A speed control term can be added to the fabric to attain a desired Euclidean speed. Experiments with geometric fabrics show natural and successful target-reaching behavior while avoiding obstacles. Optimization fabrics have better task success rates and collision rate performance than RMP. Optimization fabrics do not guarantee collision avoidance. To be used for reactive control, optimization fabrics require the location and size information from obstacles; this can be complicated in real-world scenarios and has not been researched yet in combination with optimization fabrics.

## 2.3. Perception

Autonomous robotics requires knowledge of the surrounding environment to locate obstacles and the goal location. More specifically, the local motion planning algorithms, such as optimization fabrics or MPC, require a constant differentiable function of the distance to obstacles relative to the robot. All published research on optimization fabrics assumes that this distance function is given for obstacles. However, practical usage of optimization fabrics for reactive control would eventually need perception to obtain this distance function from real-world obstacles. The perception pipeline obtains information about the surrounding environment by interpreting sensor data. A perception pipeline consists of sensors and algorithms that transform raw sensor data of the environment into information that can be utilized for motion planning and control. Proprioceptive sensors measure the robot's internal state, typically with encoders [38]. Exteroceptive sensors measure information from the surrounding environment [38], such as distances to obstacles. Exteroceptive sensors commonly used for distance measurements include sonars, depth cameras, and, more recently, LiDARs [10]. LiDAR and sonar sensors use the time-of-flight principle with light and sound pulses. Pulses of light or sound are sent out, and a distance can be calculated by measuring the time until the pulses return. Depth cameras use the stereo matching technique, which estimates a depth map from a disparity found between matching pixels obtained from two slightly shifted images [30]. Additionally, depth cameras can include an extra sensor that projects light patterns upon the scene, typically infrared [38]. The relationship between the known emitted light pattern, and the curved light pattern on the scene's surface is used to obtain depth information [30]. This further improves the quality of the depth map obtained from depth cameras, specifically on relatively flat surfaces in the environment [38].

The raw data obtained from these sensors typically require preprocessing as the raw data can be corrupted by noise and is too dense for the planning algorithm, which results in higher solving times, or using the raw data directly in the planning algorithm is impractical. Different forms of preprocessing result in various data representations, such as a raw sensor or point data representations, grid-based representations, and feature or object representations [38]. The preprocessing methods and their representations vary in terms of computational complexity and computational space. Raw sensor representations require a low preprocessing level and therefore have low computational complexity. However, the computational space needed to store the data is high. Grid-based representations require more computational complexity and less computational space. Methods to obtain feature or object representations are highly computationally complex but require less computational space to be stored.

Perception pipelines to obtain raw data or point cloud and grid-based representation have a relatively low computational complexity [37]. These methods provide the locations of the environment as sets of data points in space and can be used to create grid maps based on occupancy [14]. Depending on the sensor's resolution, point clouds or grid maps can become extremely dense. The Intel RealSense D435 depth camera has a depth output resolution of up to $1290$ x $720$ with frame rates up to $90$ per second [15]. These techniques have relatively low computational complexity but result in more data being processed by the motion planner. The computational complexity of MPC increases with the number of data points, resulting in more constraints to be solved by the optimization problem [12], and the number of obstacles included increases the solving time heavily [41]. Highly complex perception pipelines, often using statistical or machine learning techniques, use more preprocessing to provide obstacles' locations, shapes, and sizes [38]. Machine or computer vision techniques use the data obtained from one or more cameras and extract numerical features of the environment [6]. In general, these techniques have a high computational complexity, and the representations obtained require less computational space than the raw sensor data. Since obstacle detection with highly complex perception pipelines is challenging and often impractical or impossible, and the output of a low-complexity perception pipeline is often too dense, methods to obtain feature-based representations can be used. These methods combine multiple data points into planes or shapes [23], resulting in a less dense output. In [41], a method based on [23] generates a set of convex polyhedrons around the robot's links, which are modeled as ellipsoids, from the 3D sensor data. The polyhedrons represent the free space and are built using simplified descriptions of the obstacles' point clouds as planes. The solving times are unaffected by using this method compared to representing every obstacle as a sphere.

Deciding which perception pipeline is optimal for a robot use case depends on the sensors and the motion planning algorithm used, besides the type of robot and the surrounding environment. Practical considerations concerning the type of sensor to be used include the minimal distance required, the required resolution and accuracy, the materials and condition of the surrounding environment, space and power available to mount and power the sensor, and the costs [10]. Besides the practical considerations for selecting a sensor or perception pipeline, there exists a relationship between the perception pipeline and the local motion planner that we can utilize. Perception of the environment is challenging, and various methods vary in complexity. Local motion planning techniques and the perception pipeline are connected, as the output of the perception pipeline is an input for the local motion planner for reactive motion control. Local motion planning techniques also vary in their computational complexity. Techniques with relatively high computational complexity cannot maintain reactive control when the complex output of a low computationally complex perception pipeline is used, e.g., MPC with a point cloud representation [41]. These techniques benefit significantly from using a more complex perception pipeline, simplifying the output, e.g., MPC with convex polyhedron representation [41]. However, designing and utilizing a complex perception pipeline is challenging and requires additional computation time.

# 3

# Background

In this chapter, we provide background information and context for our method, which will be introduced in the following chapter. Section 3.1 describes general robotics notations used to model robotic systems. Section 3.2 describes the method of optimization fabrics with basic notations and theory. Examples of optimization fabrics for reactive control are given in 3.3.

## 3.1. Notations for robotic systems

This section describes basic concepts and notations for robotic systems. Robots are typically modeled as a mechanical combination of links and joints and typically controlled as a joint space control system, as the actuators are on the joints. The configuration $\boldsymbol{q} \in \mathcal{Q} \subset \mathbb{R}^n$ of a robot uses a set of $n$ variables to describe the position of the whole robotic system, consisting of the links and joints. The velocities and accelerations of the robotic system are given by the first and second derivative of the configuration with respect to time, $\dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}$, using Newton's notation or the dot notation to denote differentiation of the variable with respect to time. The minimum number of variables $n$ needed to describe a robot's configuration entirely is equal to the number of degrees of freedom (DOF) of a robot. The configuration space $\mathcal{Q}$ is an n-dimensional space containing all possible configurations; a single configuration can be viewed as a single point in this space. The task space $\mathcal{X}$ uses another set of variables; the task space coordinates $\boldsymbol{x} \in \mathcal{X} \subset \mathbb{R}^m$, with $m \leq n$, to describe positions of a robotic task. For example, the Cartesian coordinates in the Euclidean plane can be used as task space coordinates. The velocities and acceleration in the task space are given by the derivatives of the task space coordinates, $\dot{\boldsymbol{x}}, \ddot{\boldsymbol{x}}$.

The end effector of a robot is a location on the robot where a tool can be attached to the end of a robot's link, optionally on a joint, to have an effect on or to interact with the environment. Task space coordinates are used to describe the location of the end-effector of a robot, and configuration space coordinates are used to describe the robot's configuration. Forward kinematics of a robot can be used to calculate the position of the end-effector $\boldsymbol{x}$ from the robot's configuration coordinates $\boldsymbol{q}$ using basic trigonometry. Inverse kinematics of a robot is used to calculate the reverse situation, the robot's configuration coordinates $\boldsymbol{q}$ from the position of the end-effector $\boldsymbol{x}$. Forward kinematics has a unique solution, whereas inverse kinematics can result in multiple solutions. Basic trigonometry can be used to calculate the inverse kinematics, or it can be approximated using numerical analysis methods such as the Newton-Raphson method [43] [24].

Using forward and inverse kinematics, positions can be transformed between the task and configuration space. Using the Jacobian matrix can do the same for velocities and accelerations, see eq. 3.2 and eq. 3.3 respectively. The Jacobian matrix consists of the first-order partial derivatives of a function, and in dynamics, it is used to relate between $\boldsymbol{q}$ and $\boldsymbol{x}$; the Jacobian is described with the following equation:

$$\boldsymbol{J} = \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{q}} \tag{3.1}$$

By rewriting the general Jacobian using the chain rule and rearranging it, an equation relating $\dot{\boldsymbol{x}}$, $\dot{\boldsymbol{q}}$, and the Jacobian $\boldsymbol{J}$ are found. By differentiating this equation with respect to time, another equation is found, relating $\ddot{\boldsymbol{x}}$, $\ddot{\boldsymbol{q}}$, and $\boldsymbol{J}$.

$$\dot{\boldsymbol{x}} = \boldsymbol{J}\dot{\boldsymbol{q}} \tag{3.2}$$

$$\ddot{\boldsymbol{x}} = \dot{\boldsymbol{J}}\dot{\boldsymbol{q}} + \boldsymbol{J}\ddot{\boldsymbol{q}} \tag{3.3}$$

## 3.2. Notations for optimization fabrics

This section describes general notations and concepts of optimization fabrics. Section 3.3 gives an example of using a two-dimensional point mass to use the concepts and notations for reactive control.

### 3.2.1. Spectral semi-spray

The building blocks of fabrics are a class of differential equations called spectral semi-sprays (specs). Specs are described as the pair $\mathcal{S} = (\boldsymbol{M}, \boldsymbol{f})_{\mathcal{X}}$, with the corresponding differential equation $\boldsymbol{M}\ddot{\boldsymbol{x}} + \boldsymbol{f} = \boldsymbol{0}$. $\boldsymbol{M}(\boldsymbol{x}, \dot{\boldsymbol{x}})$ is a symmetric and invertible function of position and velocity. $\boldsymbol{f}(\boldsymbol{x}, \dot{\boldsymbol{x}})$ is also a function of position and velocity. Specs have a natural form $(\boldsymbol{M}, \boldsymbol{f})$ and a canonical form $(\boldsymbol{M}, \boldsymbol{M}^{-1}\boldsymbol{f})$. The canonical form is also called the policy form, although then notated as $[\boldsymbol{M}, \boldsymbol{\pi}]$ with $\boldsymbol{\pi} = -\boldsymbol{M}^{-1}\boldsymbol{f}$ and the corresponding differential equation as $\ddot{\boldsymbol{x}} = \boldsymbol{\pi}(\boldsymbol{x}, \dot{\boldsymbol{x}})$.

### 3.2.2. Forced spec

By adding the gradient of a potential function $\partial_{\boldsymbol{x}}\psi$, a spec is forced towards the minimum of a potential function. A forced spec is defined as:

$$\boldsymbol{M}\ddot{\boldsymbol{x}} + \boldsymbol{f} + \partial_{\boldsymbol{x}}\psi = \boldsymbol{0} \tag{3.4}$$

When a forced spec converges to a local minimum of the potential function $\psi$, it is said to be *optimizing*, and the spec forms an *optimization fabric*. An arbitrarily forced spec is unlikely to converge automatically [34] without damping. A spec that converges after adding damping is called a *frictionless fabric*. A damped forced spec is described as:

$$\boldsymbol{M}\ddot{\boldsymbol{x}} + \boldsymbol{f} + \partial_{\boldsymbol{x}}\psi + \boldsymbol{B}\dot{\boldsymbol{x}} = \boldsymbol{0} \tag{3.5}$$

### 3.2.3. Spec algebra

The spec algebra consists of the spec $\mathcal{S}$, pullback, and summation operations. These are used to combine specs across multiple spaces. A spec $\mathcal{S}$ can be expressed on any manifold. Using the Jacobian, the pullback operation transforms a spec expressed on the codomain $\mathcal{X}$ to the configuration space domain $\mathcal{Q}$. With the differentiable map $\phi$ defined as $\phi : \mathcal{Q} \to \mathcal{X}$ and $\boldsymbol{x} = \phi(\boldsymbol{q})$, the pullback operation is defined as [34]:

$$\text{pull}_{\phi}(\boldsymbol{M}, \boldsymbol{f})_{\mathcal{X}} = (\boldsymbol{J}^T\boldsymbol{M}\boldsymbol{J}, \boldsymbol{J}^T(\boldsymbol{f} + \boldsymbol{M}\dot{\boldsymbol{J}}\dot{\boldsymbol{q}}))_{\mathcal{Q}} \tag{3.6}$$

When multiple specs are expressed on the same domain, they can be added together into one spec using the summation operation [34]:

$$(\boldsymbol{M}_1, \boldsymbol{f}_1)_{\mathcal{X}} + (\boldsymbol{M}_2, \boldsymbol{f}_2)_{\mathcal{X}} = (\boldsymbol{M}_1 + \boldsymbol{M}_2, \boldsymbol{f}_1 + \boldsymbol{f}_2)_{\mathcal{X}} \tag{3.7}$$

### 3.2.4. Spec tree structure

A transform tree is used to construct a combination of specs; it consists of one root node and multiple leaf nodes. At the root node of this tree is the combined spec expressed in the configuration space domain $\mathcal{Q}$. At the tree's leaf nodes are a variable amount of specs expressed in different spaces defined with their differentiable maps. The $m$ amount of differentiable maps are defined as: $\phi_i : \mathcal{Q} \to \mathcal{X}_i, i = 1, ..., m$. The tree structure of fabrics is similar to that of RMPs; figure 2.3 shows three examples. The resultant spec at the root node can then be defined by pulling the specs from their respective spaces

into the configuration space and taking the metric weighted average of the individual specs using the summation operation [34]:

$$\sum_{i=1}^{m} \text{pull}_{\phi_i}(\boldsymbol{M}_i, \boldsymbol{f}_i)_{\mathcal{X}} = \left( \sum_i \boldsymbol{J}_i^T \boldsymbol{M}_i \boldsymbol{J}_i, \sum_i \boldsymbol{J}_i^T (\boldsymbol{f}_i + \boldsymbol{M}_i \dot{\boldsymbol{J}}_i \dot{\boldsymbol{q}}) \right)_{\mathcal{Q}} \tag{3.8}$$

### 3.2.5. Geometry generator

A spec that is positively homogeneous of degree 2 in the velocities is called a geometry generator. The HD2 property enforces that the integral curves are path consistent [46]. A geometry generator is expressed in canonical form as follows:

$$\ddot{\boldsymbol{x}} + \boldsymbol{h}_2(\boldsymbol{x}, \dot{\boldsymbol{x}}) = \boldsymbol{0} \tag{3.9}$$

**Finsler fabrics**

Finsler fabrics are defined using the property of HD2 and classical mechanics. So far, a spec is defined as $\boldsymbol{M}\ddot{\boldsymbol{x}} + \boldsymbol{f} = \boldsymbol{0}$. In geometric control, systems are defined using *simple* classical mechanical systems based on Lagrangian mechanics. For Finsler fabrics, the same is done while changing the default Lagrangian used, adding specific properties to it. In simple classical mechanical systems, the Lagrangian function is defined as the difference between the kinetic and potential energy of a system. Applying the Euler-Lagrange equation to this Lagrangian results in the following equation of motion and equations for $\boldsymbol{M}$ and $\boldsymbol{f}$ [47]:

$$\boldsymbol{M}(\boldsymbol{x}, \dot{\boldsymbol{x}})\ddot{\boldsymbol{x}} + \boldsymbol{f} = \boldsymbol{0} \tag{3.10}$$

$$\partial^2_{\dot{\boldsymbol{x}}\dot{\boldsymbol{x}}}\mathcal{L}\ddot{\boldsymbol{x}} + \partial_{\dot{\boldsymbol{x}}\boldsymbol{x}}\mathcal{L}\dot{\boldsymbol{x}} - \partial_x\mathcal{L} = \boldsymbol{0} \tag{3.11}$$

$$\boldsymbol{M}(\boldsymbol{x}, \dot{\boldsymbol{x}}) = \partial^2_{\dot{\boldsymbol{x}}\dot{\boldsymbol{x}}}\mathcal{L}; \boldsymbol{f} = \partial_{\dot{\boldsymbol{x}}\boldsymbol{x}}\mathcal{L}\dot{\boldsymbol{x}} - \partial_x\mathcal{L} \tag{3.12}$$

A Finsler fabric is a type of fabric with a special type of Lagrangian. The Lagrangian is defined to be a Finsler energy $\mathcal{L}_e$, which is a special case of kinetic energy from classical mechanics [47] [34]. A Finsler energy is a Lagrangian with the following properties [47] [46]:

1. Positivity: $\mathcal{L}_e \geq \boldsymbol{0}$ with equality only for $\dot{\boldsymbol{x}} = \boldsymbol{0}$
2. Homogeneity: $\mathcal{L}_e$ is positively homogeneous of degree 2 in $\dot{\boldsymbol{x}}$
3. Energy tensor invertibility: $\boldsymbol{M}_e = \partial^2_{\dot{\boldsymbol{x}}\dot{\boldsymbol{x}}}\mathcal{L}_e$ is invertible

The Finsler energy is based on a Finsler structure $\mathcal{L}_g$ as $\mathcal{L}_e = \frac{1}{2}\mathcal{L}_g^2$, the second property only requires $\mathcal{L}_g$ to be HD1, as this equation still gets squared, making $\mathcal{L}_e$ HD2. $\mathcal{L}_e$ is known as the energy form of $\mathcal{L}_g$ [35]. A Finsler geometry is a nonlinear geometry defined by the equations of motion of the Finsler energy, whose geometric equation is given by the equation of motion of the Finsler structure [33]. The energy of a Lagrangian is called the Hamiltonian $\mathcal{H}_e$, and in the case of Finsler energy, the Hamiltonian is the Lagrangian itself. This energy will be used for analyzing stability using energy conservation.

### 3.2.6. Geometric fabric

This section combines geometry generators and Finsler fabrics to form geometric fabrics. This is done by adding an energization term to the geometry generator differential equation, this term bends the Finsler geometry to match the geometry generator's geometry without affecting the energy [35] [33]. The energization or bending term performs no work on the system, a zero work modification, and therefore does not affect the system's energy.

Suppose a geometry generator's equation given by $\ddot{\boldsymbol{x}} + \boldsymbol{h}_2(\boldsymbol{x}, \dot{\boldsymbol{x}}) = \boldsymbol{0}$ and any Finsler energy Lagrangian $\mathcal{L}_e$ with its corresponding equations of motion $\boldsymbol{M}_e\ddot{\boldsymbol{x}} + \boldsymbol{f}_e = \boldsymbol{0}$ and energy $\mathcal{H}_e$. An energization or bending term $\alpha_{\mathcal{H}e}$ can be added to the geometry generator, $\ddot{\boldsymbol{x}} + \boldsymbol{h}_2(\boldsymbol{x}, \dot{\boldsymbol{x}}) + \alpha_{\mathcal{H}e}\dot{\boldsymbol{x}} = \boldsymbol{0}$. This equation is energy conserving and only affects the acceleration along the direction of motion by using the following $\alpha_{\mathcal{H}e}$ term [36].

$$\alpha_{\mathcal{H}e} = -(\dot{\boldsymbol{x}}^T \boldsymbol{M}_e \dot{\boldsymbol{x}})^{-1} \dot{\boldsymbol{x}}^T [\boldsymbol{M}_e \boldsymbol{h} - \boldsymbol{f}_e] \tag{3.13}$$

The energization transform commutes with the pullback, meaning a spec can be energized before or after pulling it back to the configuration space. This means that a transform tree can be populated with multiple leaves containing a geometry generator and a Finsler energy, energize them all at the leaves, pull them back to the root, and take the weighted metric average. Pulling the original geometry generators back first is also possible, summing and energizing them at the root.

### 3.2.7. Speed control

A geometric fabric's energization term only affects the velocity in the direction of motion as it attempts to conserve the Finsler energy. This results in non-constant Euclidean speed. A speed control term can be added to obtain desired speed behavior. This is an additional term that is added to the differential equation [36].

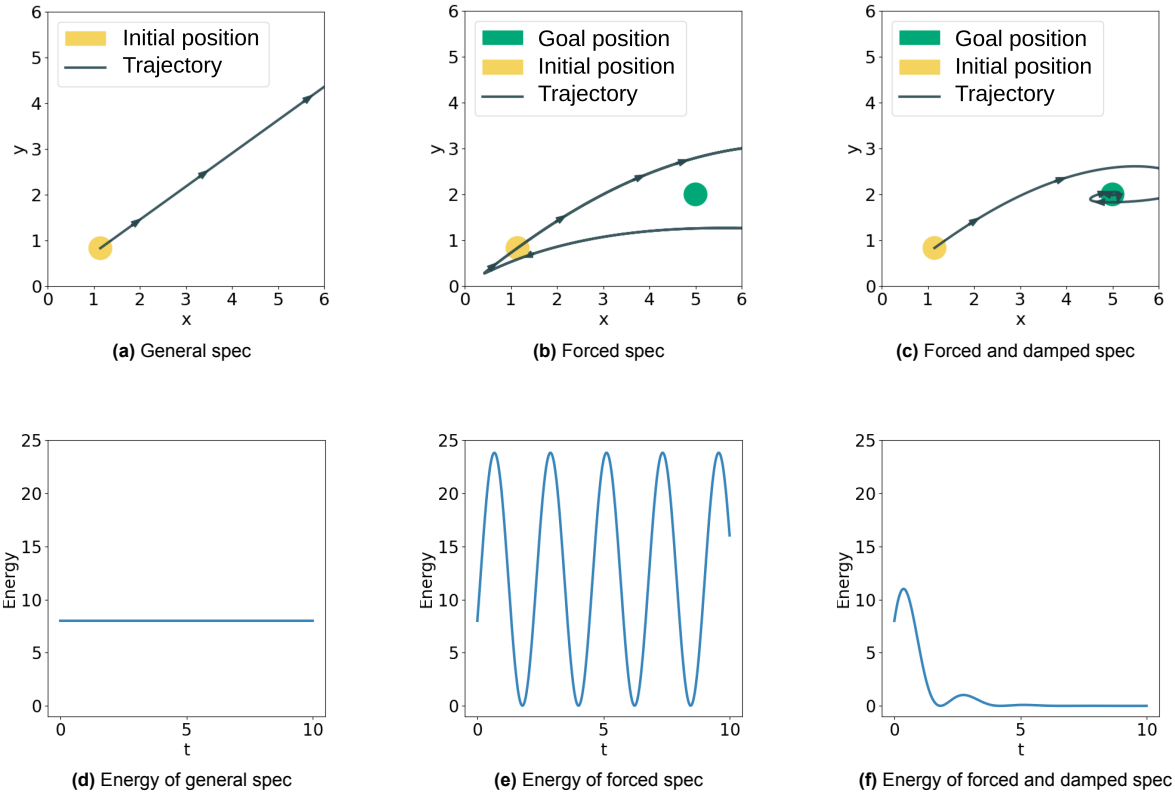
## 3.3. Reactive control example

This section describes how optimization fabrics can be used for reactive control or motion planning with a simple two-dimensional point mass as an example robot. The point mass can move in any direction in the two-dimensional plane. The first example in 3.3.1 shows the behavior of a general spec and how it changes when forced and damped. The second example in 3.3.2 shows the collision-avoiding behavior of a geometry generator and a geometric fabric. The third example in 3.3.3 combines the former two to create reactive motion. The acceleration obtained from the specs in all examples is used directly to control the point mass. In all examples, the figures show the trajectory of the point mass and the system's energy.

### 3.3.1. General spec, forced spec, and forced damped spec

A simple example of a forcing potential is a potential pulling the position of the end-effector $\boldsymbol{q}_{end-effector}$ of the point mass robot towards a goal configuration $\boldsymbol{q}_{goal}$, both defined in the configuration space $\mathcal{Q}$. Here, the point mass itself is the end-effector. A useful potential function can be defined using the squared 2-norm $\psi = ||\boldsymbol{q}_{end-effector} - \boldsymbol{q}_{goal}||_2^2$. The Finsler energy $\mathcal{L}_e$ used is defined as $\mathcal{L}_e = \frac{1}{2}\mathcal{L}_g^2 = \frac{1}{2}||\dot{\boldsymbol{q}}||_2^2$ and the Finsler structure as the 2-norm of $\dot{\boldsymbol{q}}$ as $\mathcal{L}_g = ||\dot{\boldsymbol{q}}||_2$. The derivative of the potential function and the functions for $\boldsymbol{M}$ and $\boldsymbol{f}$ form the forced spec equation $\boldsymbol{M}\ddot{\boldsymbol{x}} + \boldsymbol{f} + \partial_{\boldsymbol{x}}\psi + \boldsymbol{B}\dot{\boldsymbol{x}} = \boldsymbol{0}$. The $\boldsymbol{B}$ term is for optional damping; in this example, it is defined as an identity matrix.

Figure 3.1 shows three trajectories on the top row and the corresponding energy plots on the bottom. Figure 3.1a shows the trajectory of a spec without the forcing potential and the damping $\boldsymbol{M}\ddot{\boldsymbol{x}} + \boldsymbol{f} = \boldsymbol{0}$, the point mass moves towards its initial direction, and the energy remains constant as shown in figure 3.1d. A forcing potential is added to the spec to form a forced spec $\boldsymbol{M}\ddot{\boldsymbol{x}} + \boldsymbol{f} + \partial_{\boldsymbol{x}}\psi = \boldsymbol{0}$ as shown in figure 3.1b, the point mass moves towards the goal but fails to reach it due to overshooting around it. The system's energy is shown in figure 3.1e; it increases and decreases but cannot stabilize. The point mass can converge to the goal location by adding damping to the spec, as shown in figure 3.1c. Figure 3.1f shows the energy bleeding out.




**(a)** General spec **(b)** Forced spec **(c)** Forced and damped spec

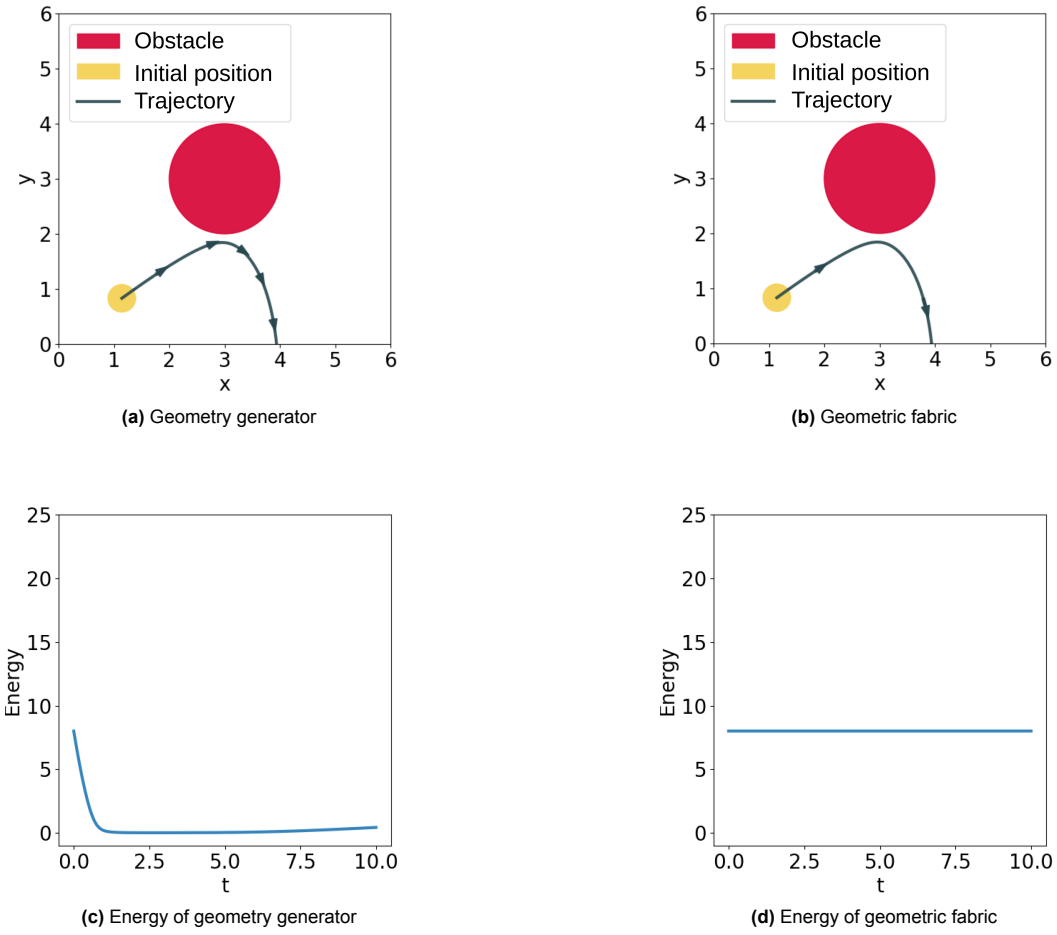
**(d)** Energy of general spec **(e)** Energy of forced spec **(f)** Energy of forced and damped spec

**Figure 3.1:** Trajectories of two-dimensional point mass based on three different specs on the top row and the corresponding energies on the bottom row. Left shows a general spec that starts at an initial position with an initial direction; the point mass will move to the top right indefinitely. The middle shows a spec forced towards a goal location using a forcing potential; the point mass overshoots and cannot converge toward the goal location. The right shows a forced spec with added damping; due to the damping, the point mass can converge to the goal location after initially overshooting. The energy is a non-physical scalar. Hence no units are shown.

### 3.3.2. Collision-avoiding geometry generator and geometric fabric

A geometry generator can be used to create collision-avoiding behavior for the point mass example. The generator can be defined in the same manner as with the previous potential function. A simple circular collision avoiding geometry generator in the task space $\mathcal{X}$ can be defined using the following function for $\boldsymbol{h}_2(\boldsymbol{x}, \dot{\boldsymbol{x}}) = \frac{||\dot{\boldsymbol{x}}||_2^2}{||\boldsymbol{x}-\boldsymbol{x}_{obst}||_2^2 - r_{obst}}$. Forming the following geometry generator $\ddot{\boldsymbol{x}} + \boldsymbol{h}_2(\boldsymbol{x}, \dot{\boldsymbol{x}}) = \boldsymbol{0}$. Energizing this geometry generator with a Finsler energy results in a geometric fabric. The Finsler energy $\mathcal{L}_e$ used is defined as $\mathcal{L}_e = \frac{1}{2}\mathcal{L}_g^2 = \frac{1}{2}||\dot{\boldsymbol{q}}||_2^2$ and the Finsler structure as the 2-norm of $\dot{\boldsymbol{q}}$ as $\mathcal{L}_g = ||\dot{\boldsymbol{q}}||_2$.

Figure 3.2a shows the trajectory of the geometry generator; the point mass moves towards the obstacle but deflects when close. Figure 3.2b shows the geometric fabric, which is the geometry generator energized. The energization results in an energy-conserving fabric; figure 3.2d shows the energy is now constant. The geometry generator on the left slows down fast near the obstacle and speeds up after the obstacle. The geometric fabric on the right shows more constant velocity through the whole trajectory while remaining on the identical trajectory of the geometry generator due to the HD2 property.




**(a)** Geometry generator

**(b)** Geometric fabric

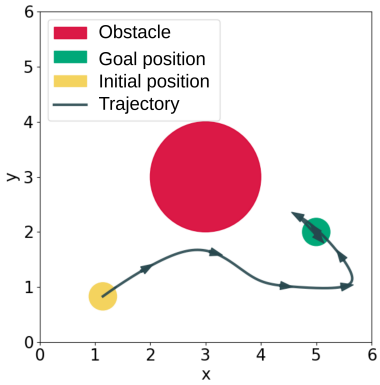
**(c)** Energy of geometry generator

**(d)** Energy of geometric fabric

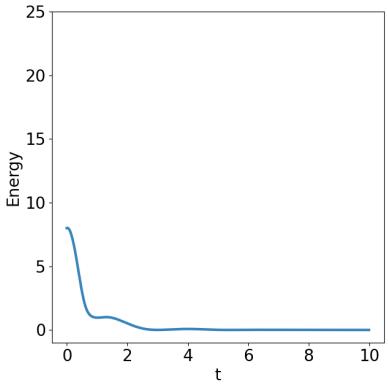
**Figure 3.2:** Trajectories of two-dimensional point mass based on two different specs on the top row, the bottom row shows the corresponding energies. To the left shows a collision-avoiding geometry generator, the point mass moves toward the obstacle and deflects to avoid it while slowing down. The right shows a geometric fabric, the geometry generator energized with Finsler energy. The point mass of the geometric fabric shows the same trajectory due to the HD2 property but has a different velocity due to conserving the Finsler energy; it shows a more constant velocity which can be seen with the arrows. The arrows show the direction of motion at certain time intervals. On the left, the arrows show a slowing down of the point mass movement due to the obstacle. The energy is a non-physical scalar. Hence no units are shown.

### 3.3.3. Summation and pullback operations

The collision-avoiding behavior and goal-reaching behavior specs need to be added together to obtain a spec with reactive behavior. The pullback and summation operations can be used to pull back and add together the forcing potential spec and the obstacle geometric fabric. The specs can be combined with the summation operation; the summation operation returns the metric-weighted average of the two accelerations. Together, the two specs form a new spec that exhibits the behavior of both specs, as shown in figure 3.3a. Figure 3.3b shows that the system's total energy bleeds out due to the forced and damped spec. The spec avoids the obstacle while converging towards the goal configuration.



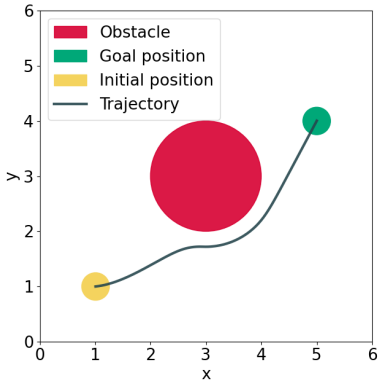

**(a)** Combined damped forced spec and geometric fabric



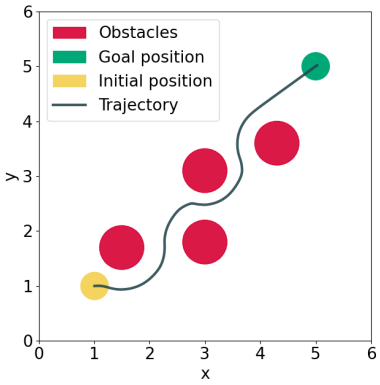

**(b)** Energy of combined damped forced spec and geometric fabric

**Figure 3.3:** On the left shows a two-dimensional point mass trajectory based on the combination of two specs, a damped forced spec, and a collision-avoiding geometric fabric. The point mass initially moves towards the obstacle but is pushed away from it while converging towards the goal afterward. The right shows the corresponding energy. The energy is a non-physical scalar. Hence no units are shown.

The example in Figure 3.3 shows how the two behaviors interact to obtain the desired combined behavior. However, the trajectory depicted in the figure is not optimal since reaching the goal is longer than necessary. By tuning the variables, obtaining an improved trajectory with a shorter path length is possible. Figure 3.4 displays a tuned combined spec in a scenario with one obstacle and another scenario with four obstacles.




**(a)** Combined and tuned damped forced spec and geometric fabric with one obstacle




**(b)** Combined and tuned damped forced spec and geometric fabric with four obstacles

**Figure 3.4:** Two trajectories of a two-dimensional point mass. The combined spec is tuned for a trajectory with minimal path length.

# 4

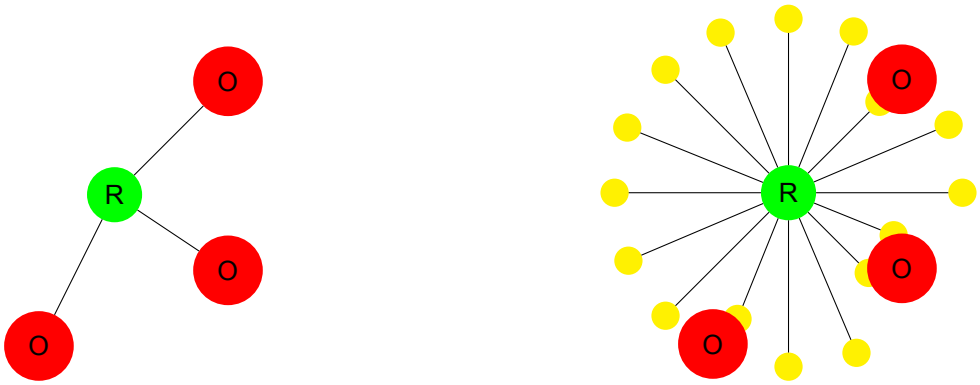
# Methods

The chapter describes the novel approach of direct sensor integrated (DSI) optimization fabrics for local motion planning. Section 4.1 presents the process of using DSI optimization fabrics for motion generation and the adjustments required to incorporate the sensors directly. Section 4.2 comprehensively describes each step involved in the DSI optimization fabrics process. To explore the need for scaling the collision-avoidance equations with an increasing number of obstacles, an analytical comparison is conducted in Section 4.3.

## 4.1. Method overview

We propose a novel approach for local motion planning, direct sensor integrated (DSI) optimization fabrics, as an alternative to traditional optimization fabrics. Local motion planning for reactive robot control requires the motion planner to perceive obstacles, traditional optimization fabrics would do so by having a perception pipeline providing the position and size of the obstacles. DSI optimization fabrics utilize the relative positions obtained from a sensor directly as the input for collision-avoidance equations. An example with a point robot surrounded by three robots is shown in Fig. 4.1; on the left, the robot obtains the obstacle information from a perception pipeline, and on the right, 16 LiDAR rays are used to obtain relative positions, which are then modeled as obstacles.

DSI optimization fabrics adjust the process of designing and obtaining motions of regular optimization fabrics. First, we provide a high-level summary of the main modifications in our approach compared to regular optimization fabrics, as described in [34]. The process of using regular optimization fabrics for motion generation is described below on the left (based on [34]), and the process of motion generation with DSI optimization fabrics is shown below on the right, with the key modifications highlighted in **bold font**. Subsequently, we present a detailed explanation of each step involved in the DSI process and delve deeper into the key modifications.




**(a)** Regular optimization fabrics

**(b)** DSI optimization fabrics with 16 LiDAR rays

**Figure 4.1:** Point robot (green and denoted with letter *R*) in a scenario with three obstacles (red and denoted with letter *O*). The lines show the LiDAR rays; at every endpoint of a LiDAR ray, a small obstacle is designed for the optimization fabric (shown as the small yellow spheres). Left (a) shows regular optimization fabrics, and right (b) shows direct sensor integrated (DSI) optimization fabrics.

*Motion generation w. regular optimization fabrics:*

1. Composition of optimization fabric:
   (a) Design components for desired behaviors, such as collision avoidance and goal guiding, and set these as leaf nodes.
2. Computing motion policy at runtime:
   (a) Obtain all obstacle sizes and shapes from a perception pipeline (red spheres in Fig. 4.1a).
   (b) Update the leaf nodes with the current state and obstacle information.
   (c) Pull the leaf nodes back to the configuration space of the root node using the pullback operation.
   (d) Sum the pulled-back leaf nodes at the root using the summation operation.
   (e) Resolve the root node for a motion policy.
   (f) Feed the motion policy as input for a robot's low-level controllers.

*Motion generation w. DSI optimization fabrics:*

1. Composition of optimization fabric:
   (a) Design components for desired behaviors, such as collision avoidance and goal guiding, and set these as leaf nodes [4.2.1].
   (b) Pull the leaf nodes back to the configuration space of the root node using the pullback operation [4.2.2].
   (c) Sum the pulled-back leaf nodes at the root using the summation operation and create a **symbolic root node**, allowing runtime parameter adjusting [4.2.3].
2. Computing motion policy at runtime [4.2.4]:
   (a) **Obtain raw sensor data output** and convert it to **relative positions** (yellow spheres in Fig. 4.1b).
   (b) **Update** the **symbolic root node** with the current state and the **relative positions**.
   (c) Resolve the root node for a motion policy.
   (d) Feed the motion policy as input for a robot's low-level controllers.

There are three main modifications between the two processes. First, the DSI optimization fabric is entirely composed before runtime using symbolic fabrics [40], which allows us to create placeholder collision-avoiding components. Second, the placeholder collision-avoiding components are designed with DSI in mind, creating a component for every relative position instead of a component for every real obstacle. Last, the relative positions are obtained with a basic conversion of raw sensor data output, removing the need for a perception pipeline that detects all obstacles.

## 4.2. Method description

This section provides a detailed description of each step in our method, including the design of fabric components, the utilization of symbolic fabrics, and the creation of collision-avoiding components.

### 4.2.1. Designing fabric components

Optimization fabric components are designed for specific robot behaviors. A tree structure of these components as leaf nodes are built to create an optimization fabrics tree, and adjusting a single component requires this optimization fabrics tree structure to be rebuilt. Changing components is necessary to tune behavior or to adjust collision-avoiding components. Whenever a perception pipeline perceives a new obstacle, a collision-avoiding component has to be created, and the optimization fabrics tree structure is modified. This is typically not an issue since optimization fabrics are computationally efficient. However, DSI optimization fabrics require new collision-avoiding components for all *new* obstacles at every time step, which would result in rebuilding the entire optimization fabric tree at every time step with potentially hundreds or thousands of these components.

We prevent this by utilizing **symbolic fabrics**, which allows us to adjust individual parameters at runtime, allowing for higher planning frequencies [40]. The optimization fabric tree is built before runtime as a symbolic expression, and sensor data is directly integrated into the expression at runtime. We utilize symbolic optimization fabric components for base inertia, collision avoidance, goal guiding, and speed control to design reactive robot behavior. These components are described below.

*a) Basic inertia:* The root node is designed using a base inertia geometry as a starting point, which includes a non-zero energy priority tensor to ensure that the system has a mass at all moments while all other parts of the optimization fabric have zero priority. The base inertia geometry, denoted as $\boldsymbol{h}_{2b}(\boldsymbol{x}, \dot{\boldsymbol{x}})$, is defined using an energy tensor $\boldsymbol{M}_{eb}$, based on a Finsler energy $\mathcal{L}_{eb}$, given by:

$$\boldsymbol{h}_{2b}(\boldsymbol{x}, \dot{\boldsymbol{x}}) = 0 \tag{4.1}$$

$$\boldsymbol{M}_{eb} = \partial^2_{\dot{\boldsymbol{q}}\dot{\boldsymbol{q}}} \mathcal{L}_{eb} \tag{4.2}$$

$$\mathcal{L}_{eb} = \frac{\lambda_g}{2\dot{\boldsymbol{q}}^T \dot{\boldsymbol{q}}} \tag{4.3}$$

*b) Collision avoidance:* Symbolic fabrics are a key component of our approach as they enable the creation of collision-avoiding components for each obstacle before runtime. The number of components required depends on the sensor resolution, with a placeholder component created for each data point generated by the sensor. In our method, obstacles are represented as small spheres since they are easily defined mathematically and generalize well with other distance-providing sensors. The sphere's position is described by the distance the sensor provides, while the radius is arbitrarily set to a value that acts as an additional safety boundary, providing more space around the real-world obstacle.

We construct a collision-avoidance component by employing a differential map described by Equation 4.4. This map describes the distance between the robot and an obstacle and allows us to generate a potential function, as specified in Equation 4.5. Subsequently, utilizing the potential function, we establish a collision-avoidance geometry, denoted by Equation 4.6.

$$\boldsymbol{x}_{obstacle} = \phi(\boldsymbol{q}) = \frac{||\boldsymbol{q}_{robot} - \boldsymbol{q}_{obstacle}||}{r_{robot} + r_{obstacle}} - 1 \tag{4.4}$$

$$\boldsymbol{\psi}_o(\boldsymbol{x}, \dot{\boldsymbol{x}}) = \frac{1}{\boldsymbol{x}_o} \tag{4.5}$$

$$\boldsymbol{h}_{2o}(\boldsymbol{x}, \dot{\boldsymbol{x}}) = \lambda_o \dot{\boldsymbol{x}}^2 \partial_{\boldsymbol{x}} \boldsymbol{\psi}_o \tag{4.6}$$

To ensure appropriate prioritization of this geometry, an associated energy tensor, represented by Equation 4.7, is introduced. This energy tensor is based on the Finsler energy expression defined in Equation 4.8.

$$\boldsymbol{M}_{eo} = \partial^2_{\dot{\boldsymbol{x}}\dot{\boldsymbol{x}}} \mathcal{L}_{eo} \tag{4.7}$$

$$\mathcal{L}_{eo} = \frac{\dot{\boldsymbol{x}}^2}{2\boldsymbol{x}} \tag{4.8}$$

Both $\boldsymbol{q}$ and $\boldsymbol{q}_{obs}$ are defined and kept symbolic until runtime. At runtime, the pulled back and weighted symbolic root node, containing the obstacle components, is given the configuration state and the absolute positions of the sphere obstacles.

*c) Goal guiding:* A goal-guiding component is introduced to facilitate goal-reaching behavior. In this context, we incorporate a goal-reaching geometry accompanied by a priority metric derived from a Finsler energy. The potential function for goal-reaching behavior is directly obtained from the differential map described by Equation 4.9, which characterizes the distance between the robot and the desired goal. By leveraging this potential function, we establish a goal-reaching geometry denoted by Equation 4.10. To ensure appropriate prioritization of this geometry, an associated energy tensor, represented by Equation 4.11, is introduced. This energy tensor is based on the Finsler energy expression given in Equation 4.12.

$$\psi_g(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{x}_g = |\boldsymbol{q}_{robot} - \boldsymbol{q}_{goal}| \tag{4.9}$$

$$\boldsymbol{h}_{2g}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \lambda_g \dot{\boldsymbol{x}}_g^2 \partial_{\boldsymbol{q}} \psi_g \tag{4.10}$$

$$\boldsymbol{M}_{eg} = \partial^2_{\dot{\boldsymbol{x}}\dot{\boldsymbol{x}}} \mathcal{L}_{eg} \tag{4.11}$$

$$\mathcal{L}_{eg} = \dot{\boldsymbol{x}}_g^2 \boldsymbol{x}_g \tag{4.12}$$

*d) Speed control:* Combining all of the above components will result in robot behavior that only converges when damped and usually exhibits non-constant end-effector speed [36]. Converging robot behavior is achievable with constant damping; however, enhanced damping known as *speed control* [36] is preferred. Speed control ensures convergence and enables us to control the desired speed behavior. We use the symbolic speed control described in [40]. After combining and weighing all the fabric components, the motion policy $\ddot{\boldsymbol{q}}$ can be computed. With speed control, we add the speed control term $\ddot{\boldsymbol{q}}_s$ to the motion policy. The speed control term is defined as

$$\ddot{\boldsymbol{q}}_s = \alpha_{ex} \dot{\boldsymbol{q}} - \beta \dot{\boldsymbol{q}} \tag{4.13}$$

The term $\alpha_{ex}$ maintains constant execution energy and consists of the weighted sum of two parts. One part with goal attraction $\alpha^{\psi}_{ex}$, and one part without goal attraction $\alpha^{0}_{ex}$. It is computed with the following equation

$$\alpha_{ex} = \boldsymbol{s}_{\eta}(\mathcal{L}_{ex}) \alpha^0_{ex} + (1 - \boldsymbol{s}_{\eta}(\mathcal{L}_{ex}))\, \alpha_{ex} \tag{4.14}$$

The damping term $\beta$ is computed as

$$\beta = \boldsymbol{s}_{\beta}(\boldsymbol{q}) \boldsymbol{B}_{max} + \boldsymbol{B}_{min} + \mathsf{max}(0, \alpha_{ex} - \alpha_{\mathcal{L}_e}) \tag{4.15}$$

With $\boldsymbol{B}_{max}$ and $\boldsymbol{B}_{max}$ as the upper and lower damping values and $\alpha_{\mathcal{L}_e}$ as the constant system energy coefficient. $\boldsymbol{s}_{\beta}$ and $\boldsymbol{s}_{\eta}$ are switching functions, parameterized with the following equations [40]

$$\boldsymbol{s}_{\beta}(\boldsymbol{q}) = 0.5\,(\mathsf{tanh} - \alpha_{\beta}(||\boldsymbol{q}|| - r_{shift})) + 1 \tag{4.16}$$

$$\boldsymbol{s}_{\eta}(\mathcal{L}_{ex}) = 0.5\,(\mathsf{tanh}(-0.5\mathcal{L}_{ex}(1 - v_{ex}) - 0.5) + 1) \tag{4.17}$$

With the variable $r_{shift}$, we can control at what distance to the goal the maximal damping $\beta_{max}$ is enabled. With another variable $\alpha_\beta$, we control the rate of change of this transition from minimal damping $\beta_{min}$. The term $\mathcal{L}_{ex}$ is the execution energy typically designed as a simple kinetic energy. We can provide a desired velocity with the variable $v_{ex}$. See [34] for a detailed description of speed control or [40] for more information about symbolic speed control.

### 4.2.2. Pullback operation

Using the Jacobian, the pullback operation transforms a spec expressed on the codomain $\mathcal{X}$ to the configuration space domain $\mathcal{Q}$. With the differentiable map $\phi$ defined as $\phi : \mathcal{Q} \to \mathcal{X}$ and $\boldsymbol{x} = \phi(\boldsymbol{q})$, the pullback operation is defined as [34]

$$\text{pull}_\phi(\boldsymbol{M}, \boldsymbol{f})_\mathcal{X} = (\boldsymbol{J}^T \boldsymbol{M} \boldsymbol{J}, \boldsymbol{J}^T(\boldsymbol{f} + \boldsymbol{M}\dot{\boldsymbol{J}}\dot{\boldsymbol{q}}))_\mathcal{Q} \tag{4.18}$$

### 4.2.3. Summation and symbolic root node

*a) Summation:* Multiple components expressed on the same domain, which is achieved with the pullback operation, are added together into one component using the summation operation [34]

$$(\boldsymbol{M}_1, \boldsymbol{f}_1)_\mathcal{X} + (\boldsymbol{M}_2, \boldsymbol{f}_2)_\mathcal{X} = (\boldsymbol{M}_1 + \boldsymbol{M}_2, \boldsymbol{f}_1 + \boldsymbol{f}_2)_\mathcal{X} \tag{4.19}$$

*b) Symbolic root node:* The DSI optimization fabric used in the experiments consists of several components, including base inertia, goal-guiding, collision-avoiding, and speed control components, as described in the first step. Finally, we describe the pulled-back and weighted symbolic root node, combining all the behavior components into one node, which can be used as the motion policy.

*c) Combined and weighted symbolic root node:* All geometries are pulled back to the root configuration space and combined using the priority metrics. The Lagrangian energies are summed as

$$\mathcal{L}_e = \mathcal{L}_{eb} + \mathcal{L}_{eg} + \sum_{o}^{n\_rays} \mathcal{L}_{eo} \tag{4.20}$$

Then, using the summed Lagrangian energy, the system metric is calculated as

$$\boldsymbol{M}_e = \partial^2_{\dot{\boldsymbol{x}}\dot{\boldsymbol{x}}} \mathcal{L}_e \tag{4.21}$$

With $n\_rays$ being the number of LiDAR rays. Using the system's combined metric, the total weighted geometry can be calculated as follows

$$\ddot{\boldsymbol{q}} = -\boldsymbol{M}_e^{-1} \left( \boldsymbol{M}_{eb}\boldsymbol{h}_{2b} - \left( \boldsymbol{J}^T \boldsymbol{M}_{eg} \dot{\boldsymbol{J}}\dot{\boldsymbol{q}} + \boldsymbol{J}^T \boldsymbol{M}_{eg} \boldsymbol{h}_{2g} \right) - \left( \sum_{o}^{n\_rays} \boldsymbol{J}^T \boldsymbol{M}_{eo} \dot{\boldsymbol{J}}\dot{\boldsymbol{q}} + \boldsymbol{J}^T \boldsymbol{M}_{eo} \boldsymbol{h}_{2o} \right) \right) \tag{4.22}$$

### 4.2.4. Computing motion policy at runtime

The process of computing a motion policy at runtime using DSI optimization fabrics is depicted in Fig. 4.2. The top part of the figure illustrates the steps described in the previous section. The bottom part of the figure outlines the process of computing a motion policy at runtime, which consists of three main steps.

First, the raw data output from a LiDAR sensor is converted and provided as input to the symbolic root node. This step ensures the appropriate integration of sensor information into the optimization framework. Conversion of the raw sensor data is required as the symbolic components designed for collision-avoiding behavior need to be provided with the absolute positions of the obstacles. With DSI optimization fabrics, we utilize the raw output of a LiDAR sensor, which are measured distances ($\boldsymbol{x}_{obstacle}$) at certain angles ($\alpha_{obstacle}$). We use the distance and angle to compute a relative position for each LiDAR ray endpoint; by adding the absolute position of the robot, we obtain the absolute LiDAR ray obstacles positions, these are the input for $\boldsymbol{q}_{obstacle}$ in Eq. 4.4. For example, in Fig. 4.1b, the LiDAR sensor would output the raw data containing $16$ angles and $16$ distances. Using basic trigonometry as in Eq. 4.23, we calculate the absolute positions of the $16$ yellow sphere obstacles. These positions are

fed into the symbolic root node, shown in Fig. 4.2 by the arrow connecting the LiDAR and symbolic root node.

$$\boldsymbol{q}_{obstacle} = \begin{bmatrix} cos(\alpha_{obstacle}) \\ sin(\alpha_{obstacle}) \end{bmatrix} \boldsymbol{x}_{obstacle} + \boldsymbol{q}_{robot} \tag{4.23}$$

Second, the symbolic root node is updated with the current configuration of the robot. Additionally, the symbolic goal node variables within the root node can be adjusted anytime, allowing for dynamic modifications to the robot's goal configuration. Finally, the symbolic root node is resolved to generate a motion policy, which is then transferred to the robot for execution.

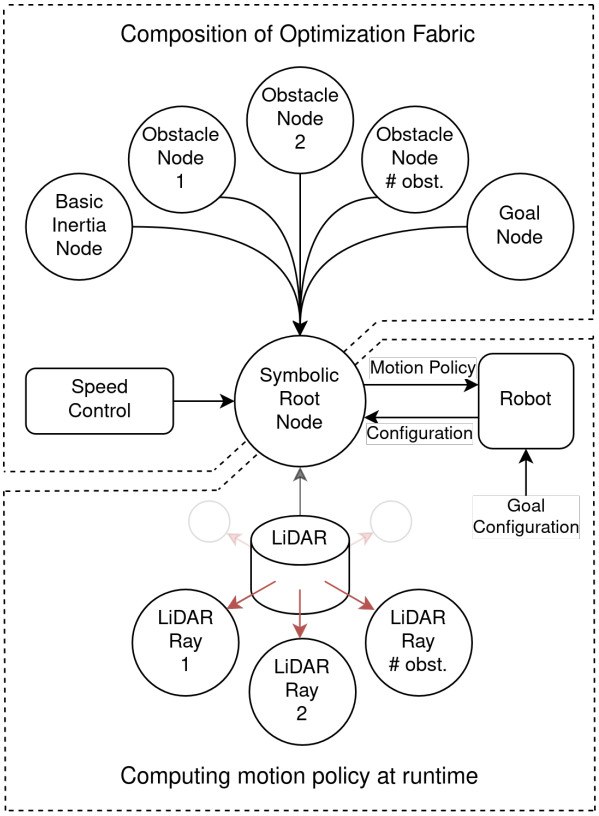
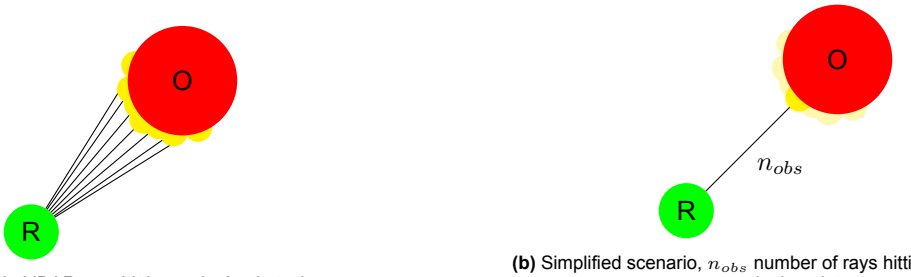



**Figure 4.2:** Overview of the motion generation process with DSI optimization fabrics.

## 4.3. Scaling of DSI optimization fabrics

The proposed method is an alternative to formally detecting obstacles and using that information to create collision-avoidance components with corresponding energies. Our method represents the real physical obstacles by multiple small surface spheres with a small radius. This results in multiple optimization fabric components for one real physical obstacle instead of only one. This is a fundamental adjustment to the equations used for motion generation, so the resulting behavior is possibly affected. Using DSI or regular optimization fabrics should result in similar behavior in similar scenarios. The multiple collision-avoidance components need to be adjusted accordingly to obtain similar behavior.

We propose a simplified scenario to investigate the analytical influence of utilizing multiple collision-avoidance components for one obstacle. Fig. 4.3a depicts a scenario with a point robot and a single obstacle using direct sensor integration. Depending on the sensor resolution, multiple LiDAR rays hit the obstacle, and these relative positions are used for collision-avoidance components. We simplify this scenario by bundling all the LiDAR rays hitting the obstacle, creating multiple collision-avoidance components at a single relative position, as shown in Fig. 4.3b.




**(a)** Multiple LiDAR rays hitting a single obstacle

**(b)** Simplified scenario, $n_{obs}$ number of rays hitting the obstacle at a single point

**Figure 4.3:** Point robot (green and denoted with letter *R*) in a scenario with a single obstacle (red and denoted with letter *O*). The lines show the LiDAR rays; at every endpoint of a LiDAR ray, a small obstacle is designed for the optimization fabric (shown as the small yellow spheres). Left (a) shows how DSI optimization fabrics perceive the obstacle. The right (b) shows the scenario simplified with all rays perceiving the obstacle at a single point.

The simplification results in multiple similar collision-avoidance components. Since the components are identical, the summation can be simplified by multiplying the component by the number of rays instead of creating one component for each separate ray and summing the components. We create a single collision-avoidance component with a geometry and an energy, as we did similarly in the previous section.

First, we create the collision-avoidance geometry using the collision-avoidance differential map $\boldsymbol{x}_o$ defined in Eq. 4.4. This is the basic geometry used for all obstacles.

$$\boldsymbol{h}_{2o}(\boldsymbol{x}, \dot{\boldsymbol{x}}) = \lambda_o \dot{\boldsymbol{x}}^2 \partial_{\boldsymbol{x}} \frac{1}{\boldsymbol{x}_o} \tag{4.24}$$

Then, we design a Lagrangian energy and a metric to weigh the single components.

$$\mathcal{L}_{eo} = \frac{\dot{\boldsymbol{x}}^2}{2\boldsymbol{x}} \tag{4.25}$$

$$\boldsymbol{M}_{eo} = \partial^2_{\dot{\boldsymbol{x}}\dot{\boldsymbol{x}}} \mathcal{L}_{eo} \tag{4.26}$$

The total Lagrangian energy is a summation of all energies; in the simplified case, the summation can be replaced by a multiplication of the energy by the total number of obstacles or rays.

$$\mathcal{L}_e = n_{obs} \mathcal{L}_{eo} \tag{4.27}$$

$$\boldsymbol{M}_e = \partial^2_{\dot{\boldsymbol{x}}\dot{\boldsymbol{x}}} \mathcal{L}_e \tag{4.28}$$

$$\boldsymbol{M}_e = n_{obs} \partial^2_{\dot{\boldsymbol{x}}\dot{\boldsymbol{x}}} \mathcal{L}_{eo} \tag{4.29}$$

The motion policy $\ddot{\boldsymbol{q}}$ can be derived using the above equations. The summation of the pulled-back collision-avoidance components can again be replaced by a simple multiplication of the term by the total number of obstacles or rays. The ratio of the total weighted system motion policy with $n_{obs}$ collision-avoidance components over 1 collision-avoidance component is given by:

$$\frac{\ddot{\boldsymbol{q}}_{n_{obs}}}{\ddot{\boldsymbol{q}}_{1_{obs}}} = \frac{-\left(n_{obs} \partial^2_{\dot{\boldsymbol{x}}\dot{\boldsymbol{x}}} \mathcal{L}_{eo}\right)^{-1} \left(n_{obs} \left(-\boldsymbol{J}^T \boldsymbol{M}_{eo} \dot{\boldsymbol{J}} \dot{\boldsymbol{q}} - \boldsymbol{J}^T \boldsymbol{M}_{eo} \boldsymbol{h}_{2o}\right)\right)}{-\left(\partial^2_{\dot{\boldsymbol{x}}\dot{\boldsymbol{x}}} \mathcal{L}_{eo}\right)^{-1} \left(-\boldsymbol{J}^T \boldsymbol{M}_{eo} \dot{\boldsymbol{J}} \dot{\boldsymbol{q}} - \boldsymbol{J}^T \boldsymbol{M}_{eo} \boldsymbol{h}_{2o}\right)} = 1 \tag{4.30}$$

This simplified scenario, with only collision-avoidance components and every obstacle having a similar position, results in a motion policy ratio of $1$. The inverse of the summed metric is multiplied by the pulled-back and summed geometry; the number of obstacles or rays scales both, but the effect on the motion policy is canceled out due to the inverse multiplication.

# 5

# Experiments & Results

This chapter evaluates the proposed DSI optimization fabrics method through a series of experiments. The first three experiments are conducted in a simulated environment using a point robot equipped with a LiDAR sensor, while the fourth experiment involves a qualitative study using a real-world robot with an actual LiDAR sensor.

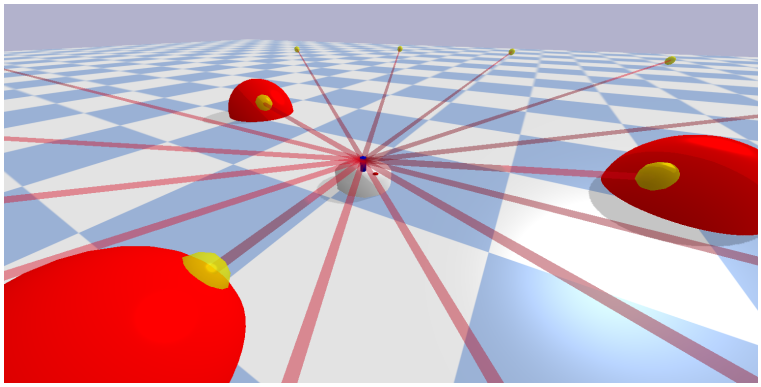
The experiment setup is described in detail in Section 5.1, providing insights into the configurations and conditions employed. In Section 5.1.1, we introduce the performance metrics utilized to assess the effectiveness and efficiency of the method. Section 5.1.2 formulates a hypothesis based on the research question of the thesis, which serves as the foundation for the subsequent experiments.

The first experiment, presented in Section 5.2, aims to empirically investigate the scaling behavior of the collision-avoidance components by comparing the scaled and unscaled methods. Experiment 2, discussed in Section 5.3.2, focuses on examining the influence of the number of LiDAR rays through an ablation study. In the third set of experiments, presented in Section 5.4, we compare the performance of the direct sensor integrated optimization fabrics to regular optimization fabrics, where the obstacle locations are known. Finally, in Experiment 4 (Section 5.5), we demonstrate a real-world application of direct sensor integrated fabrics.

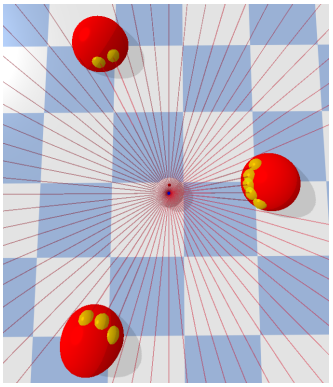
Throughout these experiments, we aim to validate the efficacy and applicability of the DSI optimization fabrics method, providing valuable insights and practical implications for its potential use in various robotic applications.

## 5.1. Experiment setup

The simulation environment employs a point robot with a 2-dimensional LiDAR sensor, shown in Fig. 5.1. The point robot is a 2-degree-of-freedom holonomic robot, and the LiDAR sensor is mounted on top of the robot. The LiDAR sensor provides a distance and angle measurement for each ray; these are translated into positions relative to the robot. The relative positions are modeled as small sphere obstacles with a radius of $r = 0.1$ m, visible as the yellow spheres in Fig. 5.1. As mentioned in 4.2.1, we want the radius to be small. To be consistent, we selected $0.1$ m for all experiments. Setting the radius to a higher value would increase the safety margin, which we do not desire for these experiments as we are deliberately testing the limits of the method.



**(a)** Side view of the simulation environment



**(b)** Top view of the simulation environment

**Figure 5.1:** Simulation environment examples: (a) side view and (b) top view. The white disk in the center represents the point robot. Obstacles are depicted as red spheres, and LiDAR rays as red horizontal cylinders. LiDAR detections are shown as yellow spheres.

All experiments are performed with $N = 30$ randomized runs. The performance metrics described in 5.1.1 are used to assess the performance in all experiments. The randomization and environment details for each scenario are described in the corresponding sections. Each method is evaluated with identical randomized runs, and the performance metrics are only assessed for successful runs that reach the goal without collision. The performance metrics for composition and solving time are evaluated only in the first scenario of experiment 2 (5.3.1), as subsequent experiments do not impact these metrics.

### 5.1.1. Performance metrics

This study uses a set of performance metrics to evaluate the method. Minimal clearance is measured to track the minimum distance between the robot and obstacles using the following equation with $n$ as the total number of obstacles and $T$ as the total number of simulation time steps.

$$\begin{aligned} \textbf{Minimal Clearance} = \min_{\substack{\boldsymbol{q}_d \in \mathcal{Q}_d \\ \boldsymbol{q}_{obs} \in \mathcal{Q}_{obs}}} \{||\boldsymbol{q}_d - \boldsymbol{q}_{obs}|| - (r_{robot} + r_{obs})\} \\ \mathcal{Q}_d = \{\boldsymbol{q}_0, \ldots, \boldsymbol{q}_T\} \\ \mathcal{Q}_{obs} = \{\boldsymbol{q}_{obs_1}, \ldots, \boldsymbol{q}_{obs_n}\} \end{aligned} \tag{5.1}$$

Time-to-goal and path length are measured to compare the total time to reach the goal and the trajectory length with the following two equations. The $t_{goal\ reached}$ variable is the time at which the robot has reached the goal, $t_{start}$ the time at the start of the simulation, and $T$ is the total number of simulation time steps again.

$$\begin{aligned} \textbf{Time-to-Goal} &= t_{goal\ reached} - t_{start} \\ \textbf{Path Length} &= \sum_{d=1}^{T} ||\boldsymbol{q}_d - \boldsymbol{q}_{d-1}|| \end{aligned} \tag{5.2}$$

Success, failure-to-reach, and collision rates are tracked to assess if the method can reach the goal, becomes stuck in a local minimum, or collides with an obstacle. If the robot has not reached the goal in time, a value of $-2$ is returned; if the clearance is lower than $0$, a value of $-1$ is returned; otherwise, we return $1$ and consider it a successful run.

$$\textbf{Succes} = \begin{cases} -2, & \text{if the goal is not reached} \\ -1, & \text{if } \textbf{Minimal Clearance} < 0 \\ 1, & \text{otherwise} \end{cases} \tag{5.3}$$

Composition time and solving time are measured to analyze the influence of increasing the number of LiDAR rays on time required to build the optimization fabric and the time needed to solve it at runtime. The composition and solving time correspond to steps one and two in the motion generation process described above.

### 5.1.2. Hypothesis

The method introduced in this thesis utilizes optimization fabrics with direct sensor integration. We first want to understand how direct sensor integration impacts the method of optimization fabrics. After that, we want to evaluate how our method compares to the baseline method of regular optimization fabrics with a perfect perception pipeline. The hypothesis introduced serves as the foundation for the experiments conducted in the following chapter; it is based on several key statements:

1. Using a varying number of LiDAR rays will result in similar collision-avoidance behavior due to the scaling ratio of the motion policy.
2. Direct integration of sensor data into optimization fabrics for reactive robot control is feasible in terms of computational complexity and performance metrics.
3. An increase in the number of LiDAR rays will lead to an improvement in performance metrics, e.g., higher success ratio, shorter path length, and less time-to-goal.
4. Direct sensor integrated fabrics are expected to show worse performance, e.g., lower success ratio, longer path length, and more time-to-goal, compared to regular optimization fabrics with a perfect perception pipeline.

In the upcoming experiments, the hypothesis will be verified. The first statement is tested in experiment 1 (5.2). Experiment 2 in 5.3.2 tests the second and third parts of the hypothesis through an ablation study. The comparison between DSI optimization fabrics and regular optimization fabrics is performed in Experiment 3 in 5.4; this experiment will examine the last statement of the hypothesis.

## 5.2. Experiment 1: Scaling factor

In this experiment, we will verify our theoretical finding from section 4.3 by comparing direct sensor-integrated optimization fabrics with and without a scaling factor. We introduce a scaling factor that adjusts $\lambda_o$ inside the collision-avoidance geometry (4.24) by dividing it by the total number of LiDAR rays as follows: $\boldsymbol{h}_{2o}(\boldsymbol{x}, \dot{\boldsymbol{x}}) = \frac{\lambda_o}{n_{rays}} \dot{\boldsymbol{x}}^2 \partial_{\boldsymbol{x}} \frac{1}{\boldsymbol{x}_o}$. The robot is placed at a distance of $3.5$ m from an obstacle with a radius of $r = 0.5$ m and an initial velocity of $0.3$ m/s toward the obstacle. No goal-guiding geometry is used; the robot moves toward the obstacle due to the initial velocity. An example of the setup can be seen in Fig. 5.2. The following number of LiDAR rays are tested: 1, 32, 64, 128, and 256. The clearance metric is used to assess how the scaling factor influences performance. This experiment tests the first statement of the hypothesis, which states that using no scaling creates similar collision-avoiding behavior with a varying number of LiDAR rays.

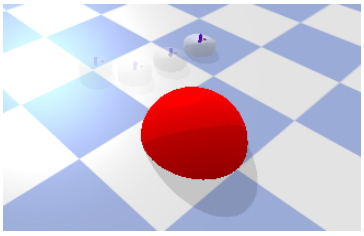
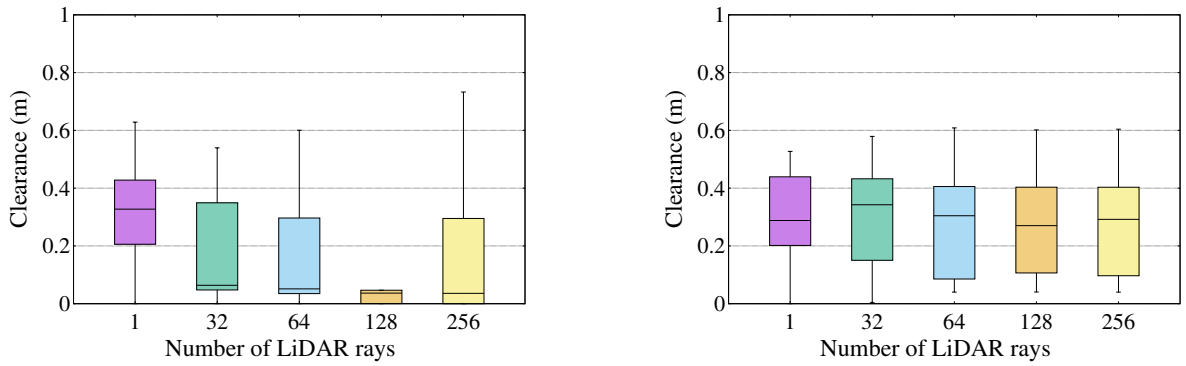


**Figure 5.2:** Setup for experiment 1.

### 5.2.1. Results




**(a)** Clearance evaluation without scaling factor

**(b)** Clearance evaluation with scaling factor

**Figure 5.3:** Clearance without scaling factor (a) and with (b)

The clearance metric results for this experiment are shown in Fig. 5.3.
Both (a) and (b) show the clearance metric for a varying number of LiDAR rays, (a) without any scaling and (b) with a scaling factor.

### 5.2.2. Discussion

The results have not been able to confirm the first statement of the hypothesis. Using no scaling results in unpredictable behavior, as seen in the clearance metric. Utilizing a scaling factor that scales the collision-avoidance components by the total number of LiDAR rays creates stable behavior independent of the sensor resolution; using a scaling factor results in a similar clearance for all LiDAR rays used. In all subsequent experiments, this scaling factor will be utilized.

## 5.3. Experiment 2: Ablation study on sensor resolution

In this ablation study, we assess the viability of our approach and investigate the impact of the number of LiDAR rays on performance, testing the second and third statements of the hypothesis. A total of three different and randomized scenarios are used. First, all three experiment scenarios are described in more detail. Second, the results of the first scenario are presented. Third, the combined results of the second and third scenarios are presented, and finally, the results are discussed.

### 5.3.1. Scenario A: Room environment

The robot is tested in a room scenario with the following number of LiDAR rays: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, and 2048. A total of $N = 30$ trials is conducted for each number of rays. The robot starts at one end of the room and must reach the goal at the opposite end at a distance of $5 \sim 7$ m while navigating around ten randomly placed spherical obstacles with a radius of $0.4$ m. An example of the setup can be seen in Fig. 5.4.

**Figure 5.4:** Setup for experiment 2A. The initial position is shown on the bottom, and the goal position is shown on the top left as the green sphere. In between are ten randomly placed spherical obstacles.

### 5.3.2. Scenario B: Corridor environment with global planner

In this scenario, we focus on a reduced set of LiDAR rays, specifically 4, 16, 64, 256, and 512, compared to the first experiment. The reason for choosing a reduced set of LiDAR rays is that this and subsequent experiments do not explore the composition and solving time aspects. Preliminary results have indicated that a low number of LiDAR rays is inadequate for achieving the goal, and increasing the number beyond 512 does not significantly impact the performance metrics. Hence, we select the previous set for further investigation.

In this scenario, we opt for a corridor environment to introduce a more significant challenge for our method. The corridor setup ensures no randomized runs where the robot can easily reach the goal without obstacles. Three obstacles, each with a radius of $r = 0.4$ m, are randomly placed within separate corridor segments. Figure 5.5 provides an example of the experimental setup.

To guide the robot through the corridor, we employ a global motion planner in the form of a simple spline modeled along the corridor. This global planner, depicted by the green line in the visualization, does not consider the presence of obstacles.

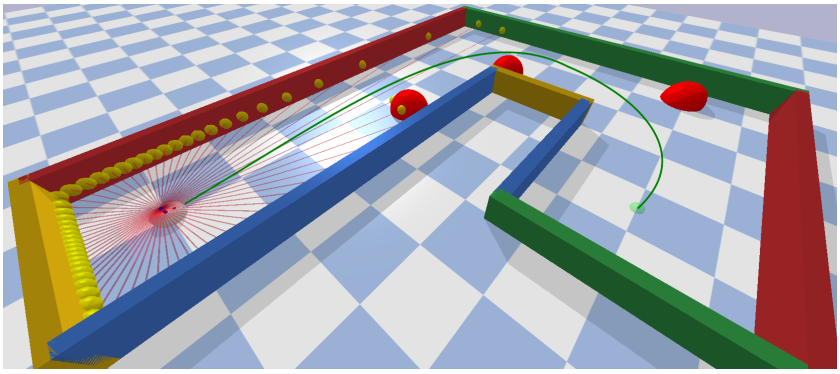


**Figure 5.5:** Setup for experiment 2B. The initial position is shown on the left, and the goal position is shown on the right as the green sphere. In between are three randomly placed spherical obstacles, in each segment one. The spline used as a global planner is shown by the green line.

### 5.3.3. Scenario C: Obstacle size

In this experiment, the impact of obstacle size is examined. The previous scenario changed the setup, but the obstacle shape and type remained the same. To assess performance with a different kind of obstacle, relatively small cylinder obstacles are used in this scenario. The setup consists of a single

corridor with 16 randomly placed cylinder obstacles with a radius of $r = 0.025$ m between the starting and goal positions with a total distance of $15$ m. No global motion planner is used since the environment consists only of a single corridor. An example of the experimental setup is shown in Fig. 5.6.

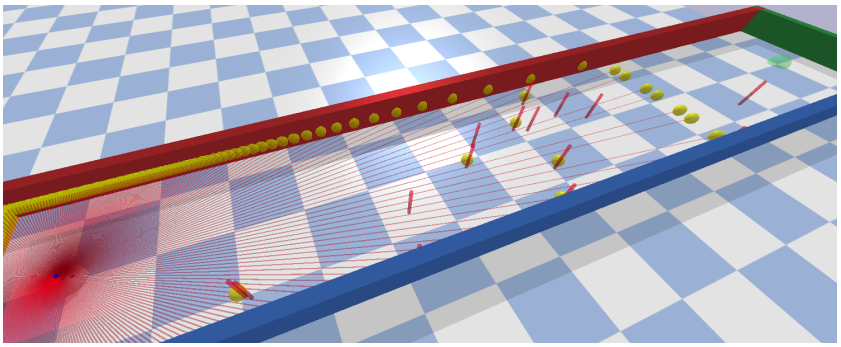


**Figure 5.6:** Setup for experiment 2C. The initial position is shown in the bottom left, and the goal position is shown in the top right as the green sphere. In between are 16 randomly placed cylindrical obstacles.

### 5.3.4. Results scenario A

The composition and solving time are fundamental aspects of the DSI motion generation process. The composition time represents the duration required to entirely compose the optimization fabric before runtime, while the solving time refers to the time necessary to compute a motion policy at runtime.

The evaluation of composition time differed from other performance metrics due to the nature of the optimization fabric, which was composed once and used for all runs. Consequently, there was only one composition time per ray for evaluation. We repeated the composition process four more times to obtain a reliable estimate and calculated the mean composition time. The results, depicted in the left section of Fig. 5.7, indicate that the composition of the fabric with a single LiDAR ray takes approximately $0.06$ s. However, this time increases to $1330$ s (equivalent to $22$ minutes) exponentially when employing 2048 rays.

The evaluation of solving time followed a different approach. We computed the mean solving time per trial and then determined the average solving time across all trials for a specific number of LiDAR rays. The resulting mean solving time, shown in the right section of Fig. 5.7, exhibits a linear increase with the number of LiDAR rays. The mean solving time remains below $0.01$ s for up to 512 LiDAR rays. With the maximum number of LiDAR rays, an average solving time of $0.043$ s (corresponding to a frequency of $23$ Hz) is achieved.

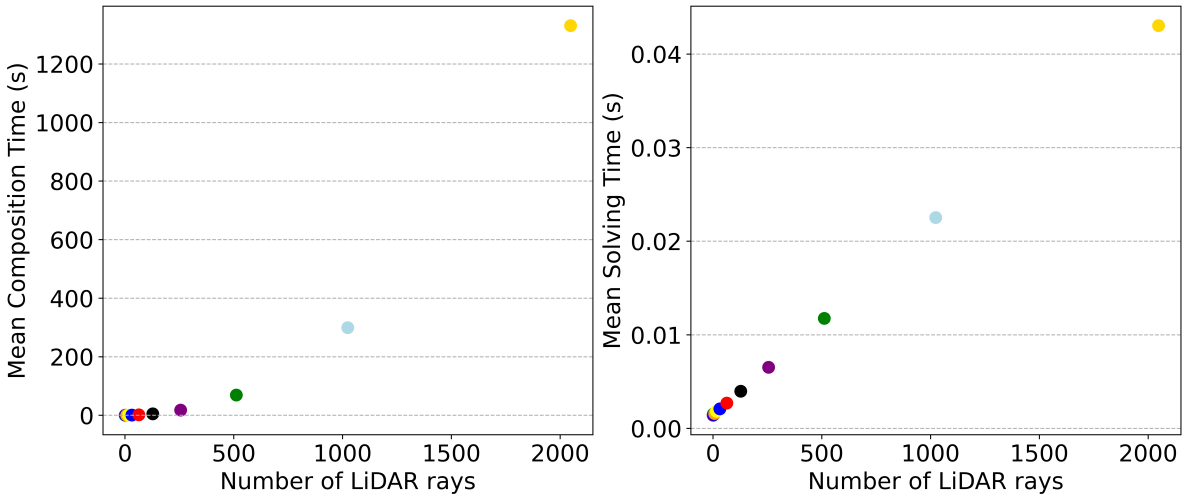
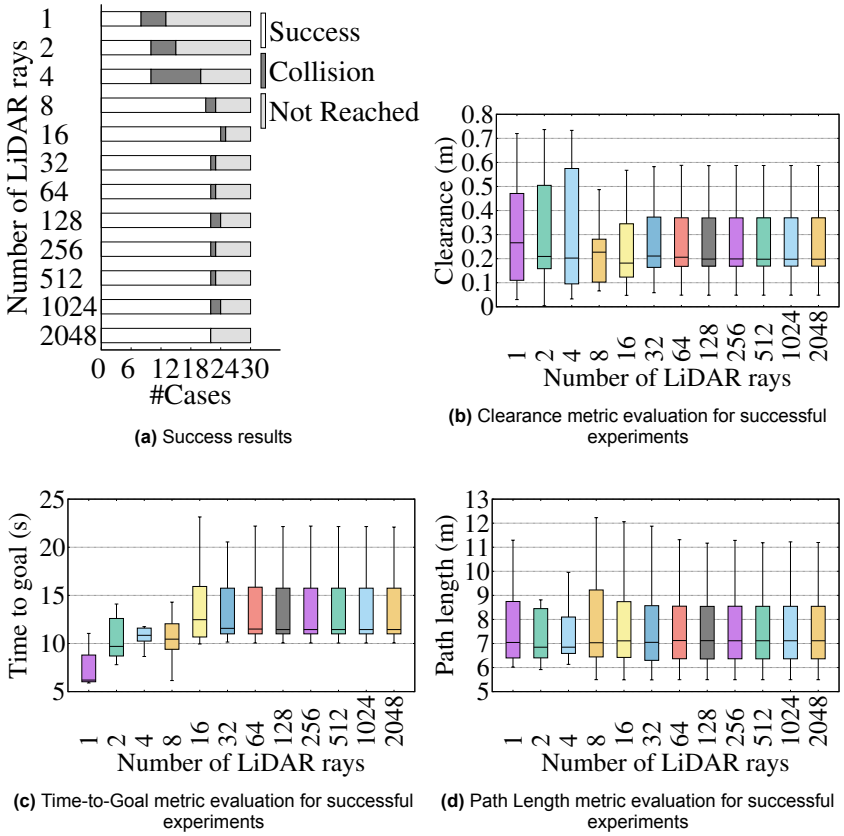



**Figure 5.7:** The mean composition time (left) and mean solving time (right) in seconds, with the number of LiDAR rays displayed on the x-axes.

Figure 5.8 presents the performance metrics for scenario A. The figure in the top row on the left illustrates the ratio of successful, not reached, and collided runs for each configuration of LiDAR rays. This provides insights into the effectiveness of different ray configurations in achieving the goal. The minimum clearance between the robot and all obstacles is depicted on the top row on the right. The bottom row of the figure displays the time-to-goal and path-length performance metrics. These metrics provide information on the efficiency and effectiveness of the robot's navigation in terms of reaching the goal and the distance traveled along the path, respectively.




**(a)** Success results

**(b)** Clearance metric evaluation for successful experiments

**(c)** Time-to-Goal metric evaluation for successful experiments

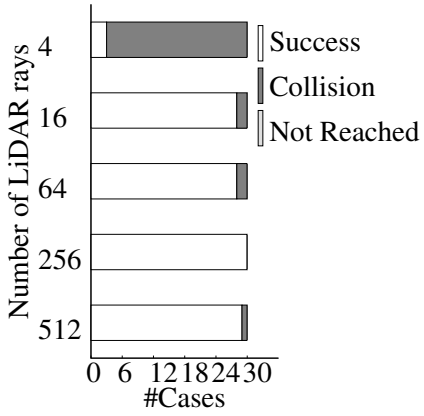
**(d)** Path Length metric evaluation for successful experiments

**Figure 5.8:** Metrics evaluation for scenario A (room).

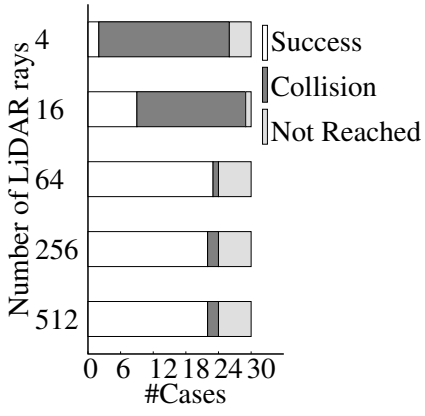
### 5.3.5. Results scenario B & C

Fig. 5.9 displays the performance metrics for scenarios B and C. Scenarios B and C are shown in the left and right columns, respectively. Each row shows a performance metric. The first row displays the ratio of successful, not reached, and collided runs for each number of LiDAR rays used. The minimum clearance between the robot and all obstacles can be seen in the second row. The third and final rows show the time-to-goal and path-length performance metrics.

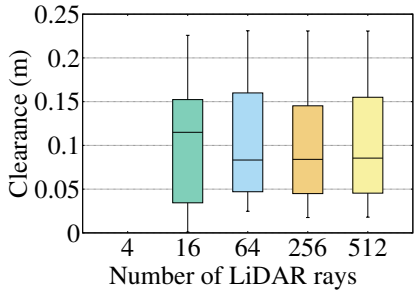
Due to the low success rate with four rays, no data is available for other performance metrics in scenarios B and C. Also, with 16 rays, the success rate is low; hence, the performance metrics are evaluated on fewer cases.



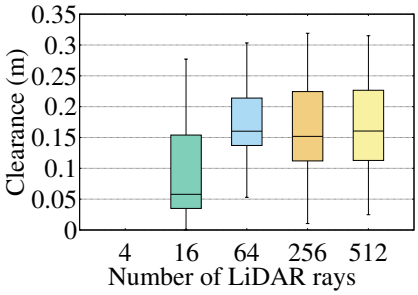

**(a)** Success results



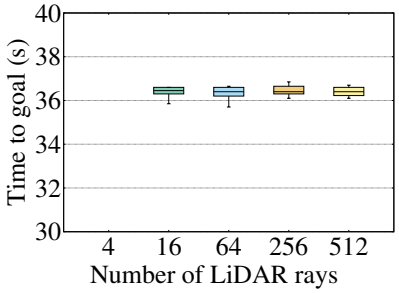

**(b)** Success results



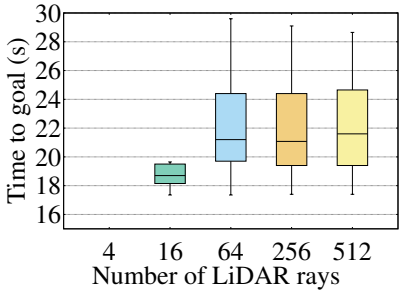

**(c)** Clearance metric evaluation for successful experiments



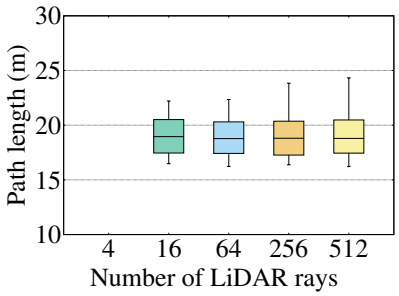

**(d)** Clearance metric evaluation for successful experiments



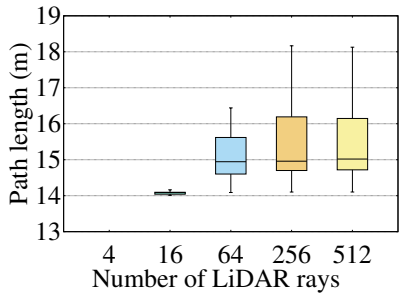

**(e)** Time-to-Goal metric evaluation for successful experiments




**(f)** Time-to-Goal metric evaluation for successful experiments




**(g)** Path Length metric evaluation for successful experiments



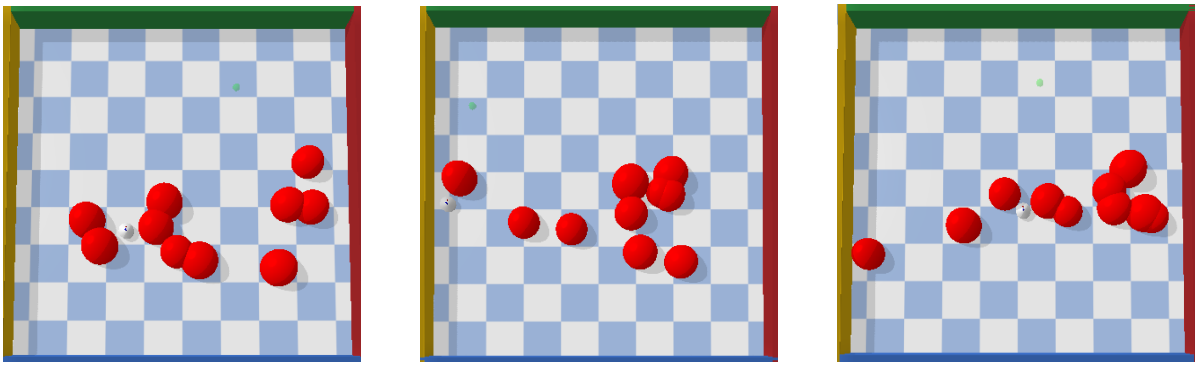

**(h)** Path Length metric evaluation for successful experiments

**Figure 5.9:** Metrics evaluation for scenarios B and C. The left column shows results for scenario B (corridor with a global planner), and the right column for scenario C (obstacle size).

### 5.3.6. Discussion

The results indicate that direct sensor integration for optimization fabrics is a feasible method in terms of solving time. A frequency of $23$ Hz was achieved even with the maximum number of rays (2048), which is a higher sensor resolution than required in most use cases. However, it should be noted that robots with more degrees of freedom will result in increased solving times. Although the composition time increases with the number of rays, it is possible to compose the optimization fabric once and save it for future use.

The experiments demonstrate that the minimally required number of LiDAR rays to achieve a high success rate depends on the complexity of the scenario. In scenario A, the robot has a low success rate with less than 16 rays. Using between 16 to 2048 rays, the success rate remains constant at an average of 23 successful runs out of 30, resulting in a success rate of $0.77$. Fig. 5.10 illustrates three instances of failure using 128 LiDAR rays. These failures occur due to local minima created by narrow passages in the environment. When moving toward a narrow passage, the robot either decelerates too excessively to reach the goal on time, gets stuck, or collides. The second scenario shows similar success performance; using only four rays, the robot had a success rate of $0.1$. With 16 rays or more, the success rate is high, averaging $0.96$. In scenario C, using four rays, the robot has a success rate of $0.07$; this improves to $0.3$ with 16 rays and reaches $0.7$ with 64 rays or more. Utilizing less than 32 rays provides insufficient information from the environment, as demonstrated by Fig. 5.1a, which shows a point robot with 16 rays. This sensor resolution is sufficient to detect the presence of obstacles of this size, but significant parts of the obstacles remain unknown to the robot. In scenario C, using less than 32 rays even results in not detecting some obstacles at all. However, scenario C also shows that DSI optimization fabrics can generalize over different types of obstacles, which regular optimization fabrics cannot do without fundamentally adjusting the design.



**(a)** Too slow through passage **(b)** Stuck in local minimum **(c)** Collision in local minimum

**Figure 5.10:** Three failure examples from scenario A with 128 LiDAR rays.

The success rate in all scenarios with less than 16 rays is low; this influences the other metrics as these are only evaluated for successful runs. In most cases, the runs are successful because these randomized environments make the goal easy to reach. Therefore, the time-to-goal and path length metrics show low values. In scenario B, due to using a global motion planner in this experiment, the time-to-goal and path length metrics are not showing a performance increase since the robot is already capable of following the global motion path with 16 rays. Therefore, these two metrics are influenced mainly by the path and time parametrization of the global motion plan itself. The local motion planner can slightly increase or decrease the path length or the time-to-goal, but the effect in this experiment is minimal. The results of using smaller obstacles emphasize the need for a sufficient number of LiDAR rays. In scenario B, 16 rays were adequate, but in scenario C, a success ratio of only $0.3$ was achieved with 16 rays.

The experiments verify the second statement of the hypothesis, the direct sensor integrated optimization fabrics method is feasible for reactive control in terms of computational complexity and performance. A high solving frequency is obtainable even with a high sensor resolution. Using sufficient LiDAR rays results in satisfactory performance for reactive robot control. The results indicate that the number of LiDAR rays required depends on the complexity of the environment. Utilizing more than the minimally

required number of LiDAR rays does not necessarily improve clearance, time-to-goal, and path length performance. Therefore, the third statement of the hypothesis is confirmed but with a restriction.

## 5.4. Experiment 3: DSI fabrics vs. regular fabrics

In this experiment, we compare the performance of our direct sensor integrated (DSI) optimization fabrics to regular optimization fabrics, verifying the last statement of the hypothesis. Based on the positive performance observed in previous experiments, we have chosen to utilize the DSI fabric method with 64 rays. The regular optimization fabric is given obstacle information such as position and size at each time step through a simulated perception pipeline. The following two sections describe two different randomized scenarios in more detail. After that, the combined results are presented together with a discussion.

### 5.4.1. Scenario A: Static obstacles

This scenario consists of a plane with six randomly placed static sphere obstacles, with a radius of $r = 0.4$ m, between the starting and goal positions, with a total distance of $14$ m. Walls were not used for practical reasons, as the regular fabric method would need to be enhanced to handle such obstacles. Thus, the two methods are compared using sphere obstacles in the environment. An example of the experimental setup is shown in Fig. 5.11.

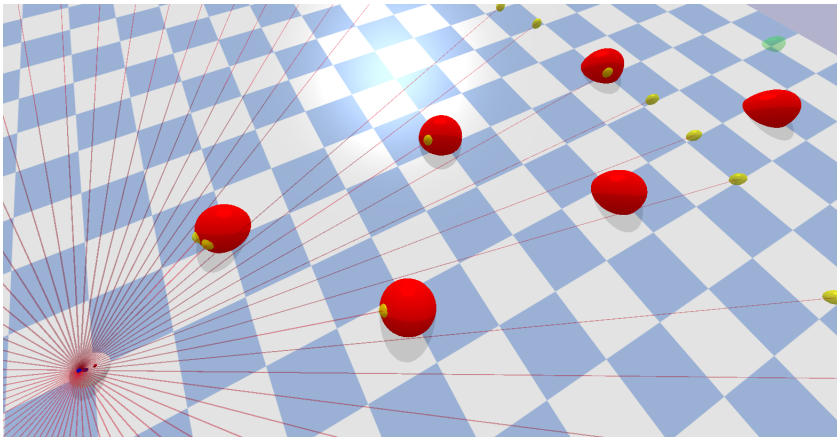


**Figure 5.11:** Setup for experiment 3A. The initial position is shown in the bottom left, and the goal position is shown in the top right as the green sphere. In between are six randomly placed obstacles.

### 5.4.2. Scenario B: Moving obstacles

This scenario features ten randomly placed moving spheres with a radius of $r = 0.4$ m; the position and velocity are randomized. An example of the experimental setup can be found in Fig. 5.12. The robot is initially assigned a velocity of $0.1$ m/s towards the goal position to prevent collisions with nearby moving obstacles at the start. This non-zero velocity ensures that the robot can safely navigate the environment without encountering immediate crashes with obstacles nearby and moving toward the robot at the beginning of the experiment.

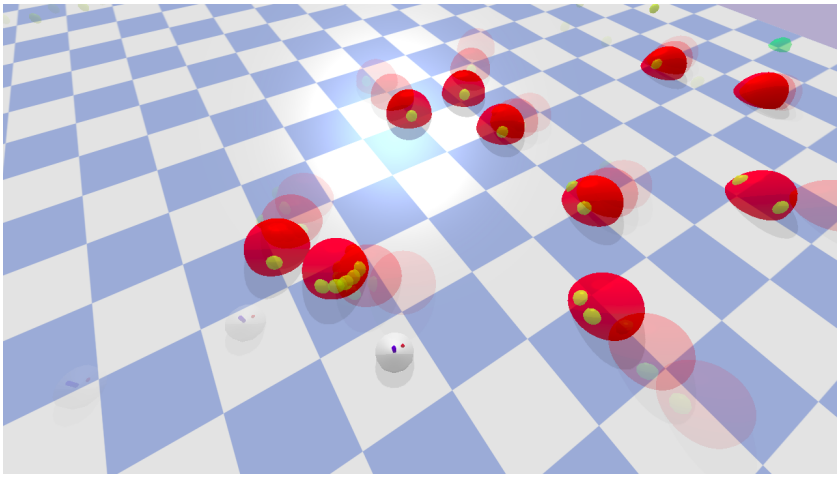


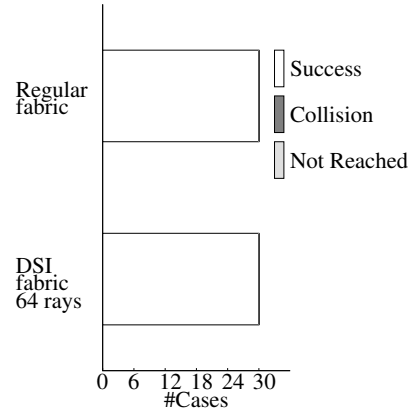
**Figure 5.12:** Setup for experiment 3B. The initial position is shown in the bottom left, and the goal position is shown in the top right as the green sphere. In between are ten randomly placed moving obstacles. Previous positions of the robot, moving obstacles, and detections are shown by transparent visualizations.

### 5.4.3. Results

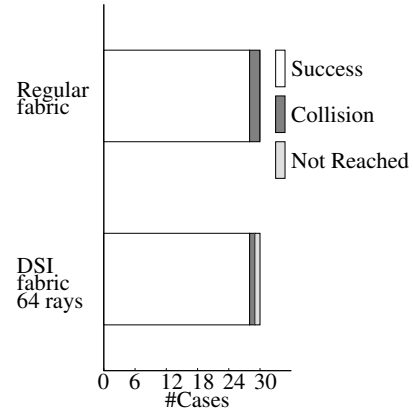
Figure 5.13 provides a comprehensive overview of the performance metrics for the two experiments. The left and right columns represent the first and second scenarios, respectively, while each row corresponds to a specific performance metric.

In the first scenario, both methods achieved a 100% success ratio in all 30 trials, as depicted in Figure 5.13a. The DSI method exhibited a slightly lower clearance than the regular fabric, as illustrated in Figure 5.13c. The time-to-goal metric (Figure 5.13e) showed minimal differences between the two methods. Furthermore, both methods demonstrated comparable path lengths, as shown in Figure 5.13g.

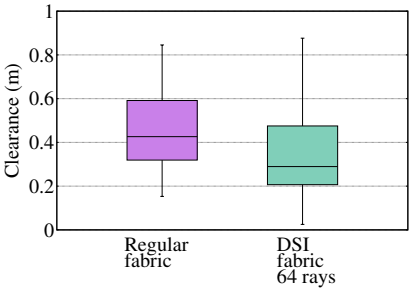
Both methods achieved a success ratio of $0.93$ in the second scenario, as depicted in Figure 5.13b. Notably, the DSI method maintained a higher clearance than the regular fabric in the presence of moving obstacles, as highlighted in Figure 5.13d. The time-to-goal and path length metrics exhibited similar performance between both methods, as shown in Figure 5.13f and Figure 5.13h, respectively.



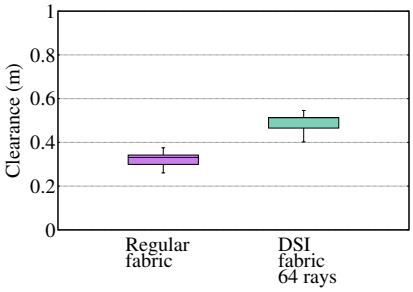

**(a)** Success results



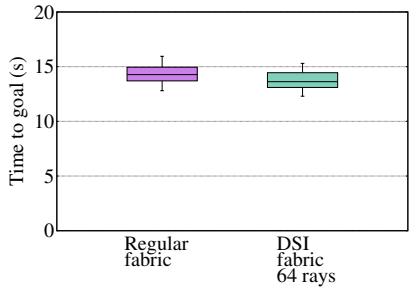

**(b)** Success results



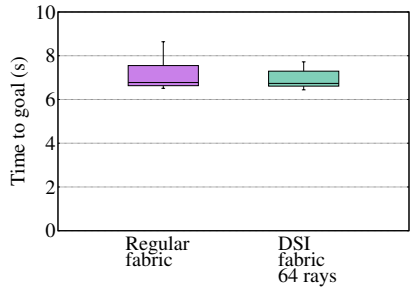

**(c)** Clearance metric evaluation for successful experiments



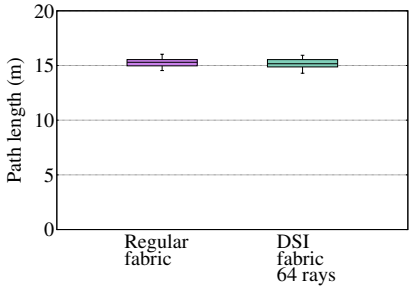

**(d)** Clearance metric evaluation for successful experiments



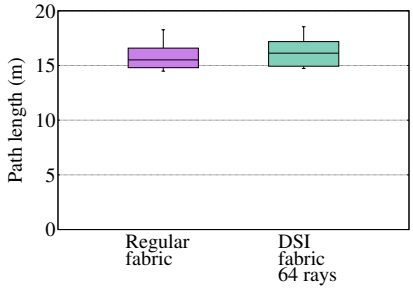

**(e)** Time-to-Goal metric evaluation for successful experiments




**(f)** Time-to-Goal metric evaluation for successful experiments




**(g)** Path Length metric evaluation for successful experiments

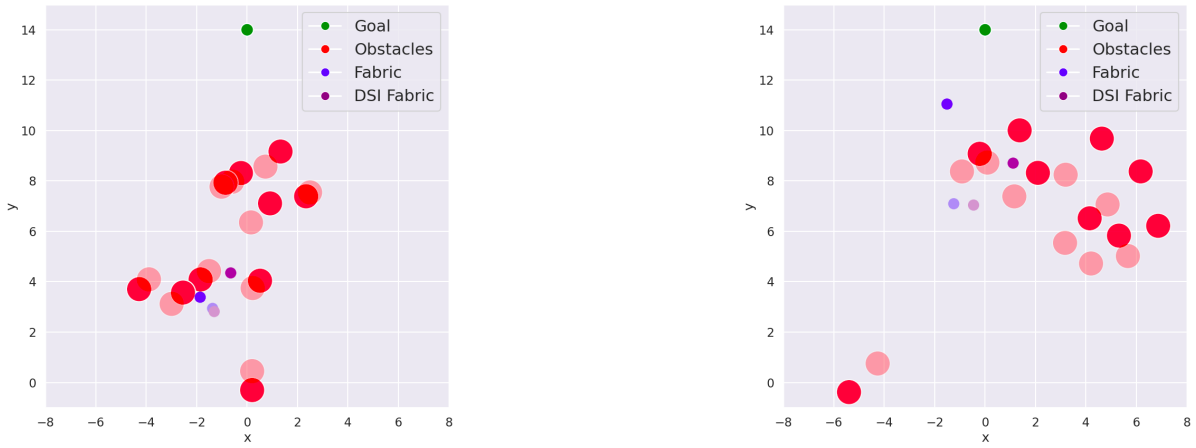



**(h)** Path Length metric evaluation for successful experiments

**Figure 5.13:** Metrics evaluation for experiment 3; comparing regular fabrics with direct sensor integrated (DSI) fabrics utilizing 64 LiDAR rays. The left column shows results for scenario A with static obstacles, and the right column shows results for scenario B with moving obstacles.

### 5.4.4. Discussion

In the first scenario, both methods demonstrate successful goal achievement in all trials. However, in the second scenario, where moving obstacles are present, both methods exhibit a slightly lower success ratio than the previous scenario. Out of the total of 60 runs, there were four failure cases. The figures below illustrate these failure instances. Each figure provides a top view of the scenario, including the goal, obstacles, and robots. The movement of robots and obstacles is represented by displaying their previous positions in lighter colors. The first figure (Figure 5.14a) depicts a scenario where the regular fabric fails due to a collision while the DSI fabric successfully reaches the goal. In the second figure (Figure 5.14b), the regular fabric reaches the goal while the DSI fabric fails to reach it in time. The DSI fabric moves toward the top right, with three obstacles moving similarly. Although the robot does not collide, it moves along with the obstacles until the maximum time limit is reached. In this case, the robot should have moved backward, highlighting an example where a local minimum can result in a longer trajectory than necessary. This problem can be addressed by implementing a periodic global planner. The last figure (Figure 5.15) demonstrates a scenario where both the regular and DSI fabric fail due to a collision with an obstacle. In all collision cases, the robot slows down as the obstacles approach. However, as the robot's velocity decreases, the collision-avoidance components have a diminished effect on the robot's behavior. These specific scenarios show that the robot reacts too slowly to the obstacles. This issue arises from the current implementation of optimization fabrics, which do not consider obstacle dynamics. In the existing approach, the position of obstacles at each time step is used to compute a motion policy. When the robot's velocity is very low, the collision-avoiding component activates too late. An extension of optimization fabrics called dynamic fabrics [42] addresses this problem by incorporating the dynamics of the environment. This extension considers the velocity of the obstacles, offering a solution to the collision examples mentioned above.




**(a)** Regular fabric collision with two obstacles

**(b)** DSI fabric failure to reach the goal

**Figure 5.14:** Top view of failures from experiment 3 with moving obstacles of regular fabric (a) and DSI fabric (b).

Both methods continue to demonstrate comparable performance metrics. The regular optimization fabric exhibits a slightly higher clearance in the first scenario. In contrast, in the presence of moving obstacles, the DSI fabric maintains a higher clearance than the regular fabric. These differences in clearance are minor and can be attributed to the tuning of both methods. Both methods perform similarly regarding the time-to-goal and path length metrics in both scenarios. However, the DSI fabric method shows a slightly lower time-to-goal and higher path length when moving obstacles are present. As a result, the DSI fabric method achieves a higher average velocity. The collision-avoidance components have a more significant impact with increased velocity, leading to a higher clearance.

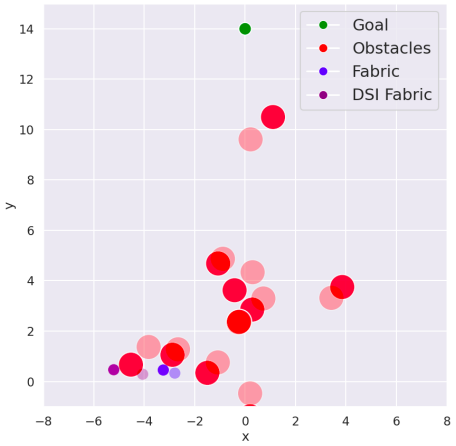



**Figure 5.15:** Top view of the regular and DSI fabric collision.

## 5.5. Experiment 4: Real-world showcase

In this experiment, we present a real-world showcase of our method and discuss the observed performance. By experimenting with a real-world setting, we aim to assess our method's practical viability and effectiveness in addressing the challenges and complexities encountered outside of simulation environments. Sim-to-real transfer, which involves transitioning from simulated scenarios to real-world implementation, is often challenging in robotics research. However, one advantage of optimization fabrics is that sim-to-real transfer is easily facilitated, allowing for a smoother integration of the method into real-world applications. Apart from the transfer of the optimization fabrics method, additional challenges are associated with the sim-to-real transfer, which will be discussed in the final section.

### 5.5.1. Experiment Setup

The experiment involved testing our method on a holonomic ground robot equipped with a LiDAR sensor [39]. The setup is illustrated in Figure 5.16. We utilized the ROS framework [31] to integrate the LiDAR sensor with our DSI fabric method. Similar to the simulation experiments, we selected a subset of rays from the LiDAR sensor for integration with the DSI fabric. We tested with 49, 90, and 190 rays for this particular experiment, but the results presented below are based on utilizing 49 rays.

**Figure 5.16:** Setup for the real-world experiment in the retail store environment lab of AIRLab Delft [1], showing the ground robot to the right and two random obstacles in the middle.

To facilitate localization, we employed the LiDAR sensor to create a map of the environment using the gmapping ROS package [11]. The map was then used by the AMCL (Adaptive Monte Carlo Localization) algorithm [9] to localize the robot within the map together with odometry data based on the wheels encoder data, calculated using [24]. Our system's complete source code and implementation details will be publicly available for further reference and reproducibility [45].

### 5.5.2. Results

Figure 5.17 showcases the robot's motion during the real-world experiment in one scenario. The scenario demonstrates a simple showcase of our method with two obstacles between the robot and the goal location. The figure illustrates the robot's ability to navigate the environment using our DSI fabric method. Despite not having an entire perception pipeline, the robot successfully avoids collision using only the raw sensor data output from the LiDAR sensor.

**Figure 5.17:** Motion of the robot during the real-world experiment in an environment with two obstacles. The robot's initial location is shown at the bottom, while the goal location is positioned at the top right.

### 5.5.3. Discussion

The discussion focuses on qualitative observations from the robot's motion during the real-world showcase and testing. Comparing the observed behavior in the real-world experiment and testing with the results obtained from simulation experiments, we find that tuning the fabric parameters remains crucial for achieving desired performance. However, due to the real-time nature of the experiments, parameter tuning is a slower process than in simulation. As a result, we have not yet achieved the same level of observed performance as in the simulation experiments.

In real-world experiments, several factors influence the performance of our method. One significant factor is the presence of noise introduced by the localization methods. Unlike the simulation environment, where perfect knowledge of the robot's position and velocity is available, real-world experiments heavily rely on localization techniques that introduce noise. While this noise has minimal impact on the collision-avoidance components, as the obstacle positions are relative to the robot's position, it can affect other crucial aspects of our method, including goal guidance components and speed control.

To address the challenge of providing accurate velocity commands to the robot, we encountered difficulties due to noisy localization. While the motion policy we provide to the robot is in the form of acceleration, the robot requires velocity commands for control. Since obtaining a stable estimate of the robot's velocity was challenging, we opted to multiply the motion policy by the time step and provide it as the velocity command. Improving the accuracy of the robot's position and velocity estimates would enable us to provide the integrated motion policy to the robot, which is expected to enhance overall performance.

Additionally, the LiDAR sensor data in real-world experiments introduces additional noise that can impact the performance of the collision-avoidance components. However, despite the presence of noise, it is evident that the overall performance can still mostly be improved through further fine-tuning of the optimization fabric parameters, enhancing localization techniques, and carefully selecting the appropriate number of LiDAR rays.

# 6

# Conclusion & Future work

This chapter marks the conclusion of the thesis, consolidating the key findings and contributions made throughout the research. The study has introduced and evaluated the direct sensor integrated (DSI) optimization fabrics method, showcasing its effectiveness in motion generation and collision avoidance. The results obtained from the experiments have provided valuable insights into the performance of the DSI method under various scenarios and conditions. Based on these findings, there are several potential areas for future research and development.

## 6.1. Conclusion

In this thesis, we have adjusted the method of optimization fabrics to incorporate sensor data directly. We show that our proposed method of DSI optimization fabrics is feasible in terms of computational complexity and performance in various goal-reaching tasks. Our analytical analysis has shown that direct sensor integration does not require scaling adjustments to the regular collision-avoidance equations. Our method eliminates the need for a complex perception pipeline while maintaining comparable performance to the regular optimization fabrics approach that utilizes a perception pipeline.

We have performed multiple simulation experiments to verify the theoretical findings. First, we showed that a scaling factor is required to adjust the collision-avoidance equations correctly. The empirically found scaling factor has been used in all subsequent experiments. The second set of experiments has shown that it is possible to directly integrate sensor data into optimization fabrics regarding computational complexity, achieving a frequency of $23$ Hz even with the maximum sensor resolution of 2048 LiDAR rays. The results also indicate that the performance metrics of time-to-goal, total path length, minimal clearance from obstacles, and success rate do not necessarily improve with an increase in the number of LiDAR rays. Increasing the number of LiDAR rays up to a certain amount is crucial for the robot to detect all obstacles and their shapes; increasing the number of rays past the minimally required does not increase performance any further. The third set of experiments demonstrated that our method achieved comparable performance with the regular optimization fabrics method using a simulated perception pipeline that detected all obstacles. Contrary to initial expectations, our method demonstrated comparable performance to the regular optimization fabric, defying the assumption of potential underperformance. The performance metrics revealed minimal disparities between the two approaches, with the most notable distinction observed in the clearance metric. In the scenario with static obstacles, the regular method demonstrates higher clearance, while in the scenario with moving obstacles, the DSI method shows a higher clearance. These findings highlight the robustness and effectiveness of our approach, as it successfully maintained comparable performance to the established method while adapting to dynamic environments.

Our research has addressed the question of how to integrate sensor data into optimization fabrics directly. Through our proposed method, which involves designing separate collision-avoiding behavior components for every sensor data point and tuning the combined optimization fabric with a scaling factor dependent on the sensor resolution, we have shown that it is possible to achieve comparable

performance to regular optimization fabrics method without requiring a complex perception pipeline.

The main contribution of this thesis is introducing a novel method that uses optimization fabrics combined with direct sensor integration. Specifically, this thesis has made the following contributions. First, it presents an approach to adjust the optimization fabrics method to incorporate sensor data for collision avoidance directly. Second, it proposes an empirically derived scaling factor for the collision-avoidance equations. Third, the feasibility of DSI optimization fabrics is demonstrated through multiple simulation experiments. In summary, our work provides a practical solution to the challenge of integrating perception and motion planning, which can benefit a wide range of robotic applications in dynamic environments.

## 6.2. Future work

While this study has provided valuable insights into the performance and capabilities of the DSI optimization fabric method, several opportunities for future work can further enhance its applicability and effectiveness.

### 6.2.1. Increased complexity environment & robot

One direction for future research is to investigate the application of the DSI method in different real-world environments and with robotic platforms of increased complexity. For example, testing the method with non-holonomic robots with more degrees of freedom would present a more challenging and realistic scenario. This would allow a more thorough evaluation of the method's capabilities and limitations in complex and dynamic environments. Furthermore, to improve the performance of the DSI method in real-world experiments, it is essential to address the impact of localization noise to enhance the accuracy of the robot's position and velocity estimates. Fine-tuning the optimization fabric parameters specifically for real-world scenarios and investigating different configurations of LiDAR rays can also contribute to better collision-avoidance performance.

### 6.2.2. Higher resolution LiDAR

In this study, a two-dimensional LiDAR sensor was used for perception. Exploring using higher resolution LiDAR sensors, such as three-dimensional LiDAR, can enable the method to handle more complex environments and provide more detailed perception information. This can lead to improved motion planning and collision avoidance capabilities.

### 6.2.3. Processing of LiDAR data

While the proposed method aims to eliminate the need for a traditional perception pipeline, future research could explore incorporating a simplified perception pipeline that processes LiDAR data to provide additional context and information. This could strike a balance between the simplicity of the DSI approach and the benefits of processed sensor data, potentially improving the overall performance and adaptability of the system.

### 6.2.4. Other sensor types

Expanding the integration of additional sensors and sensor modalities can enhance the perception capabilities of the system. Exploring the fusion of LiDAR data with other sensor types, such as depth cameras, can provide complementary information and improve the robustness and reliability of the method in different scenarios.

### 6.2.5. Dynamic fabrics

Future research can focus on exploring the combination of dynamic fabrics ([42]), which consider the dynamics of the environment and moving obstacles, with direct sensor integration. This integration can significantly enhance the method's ability to handle dynamic scenarios and improve the efficiency of collision avoidance. The experiments conducted with moving obstacles have highlighted failure cases that could have been prevented by incorporating dynamic fabrics. The method can effectively address these challenges by developing and implementing dynamic fabrics in conjunction with direct sensor integration and ensure safer and more robust robot navigation in dynamic environments.

### 6.2.6. Tuning of the method

The tuning of optimization fabrics is a crucial aspect that directly impacts the method's performance. Currently, the tuning process requires expert knowledge and manual parameter adjustments. However, future research can explore automated tuning approaches to streamline and improve the tuning process. One promising area is auto-tuning techniques, as demonstrated in work referenced as [40]. By leveraging auto-tuning, the method's parameters can be automatically adjusted based on performance metrics and objective functions, leading to more efficient and effective collision avoidance behavior. Applying auto-tuning techniques can significantly reduce the burden of manual tuning and potentially uncover optimal parameter settings that may not be easily identifiable through manual methods. This future research direction can contribute to the scalability and adaptability of the method, making it more accessible and applicable in various robotic applications.

By pursuing these future research directions, the proposed DSI optimization fabrics method can continue to evolve and contribute to the advancement of robotic motion planning and control, enabling robots to navigate complex environments with improved safety and efficiency.

## 6.3. Insights and Reflections

Throughout this research, I have gained valuable knowledge and insights that have deepened my understanding of robotics and motion planning. This research journey has been a transformative learning experience, broadening my perspective and enhancing my skills in the field.

One of the most significant learning experiences throughout my research journey has been the deep understanding and application of the optimization fabrics technique. Exploring and implementing this method has given me invaluable insights into its principles, functionalities, and practical implications. Through rigorous study and experimentation, I have gained a comprehensive understanding of how optimization fabrics can be leveraged for motion planning and collision avoidance in robotics.

Another key lesson is the opportunities presented by integrating sensor data directly into optimization fabrics. This approach simplifies the overall system architecture and enhances the real-time performance and efficiency of collision avoidance. The empirical exploration and analysis conducted during this thesis have highlighted the critical role of carefully selecting and tuning the optimization fabrics' parameters for optimal performance.

Furthermore, this research has underscored the significance of considering the limitations and trade-offs associated with different sensor configurations. The simulations provided valuable insights into the method's behavior in controlled environments, while the real-world experiments highlighted the challenges and differences between simulation and reality. Understanding these differences has emphasized the need for robustness and adaptability in the proposed method when deployed in practical scenarios. Beyond the technical aspects, this thesis has also fostered my skills in experimental design, data analysis, and critical thinking. The collaborative work with my research colleagues and the guidance of my thesis supervisors have provided me with invaluable insights into the research process and robotics.

# References


[1] *AI for Retail (AIR) Lab Delft*. https://icai.ai/airlab-delft/. Accessed: May, 2023.

[2] William M. Boothby. *An Introduction to Differentiable Manifolds and Riemannian Geometry*. Academic Press New York San Francisco London 1975, 1975.

[3] Francesco Bullo and Andrew D. Lewis. *Geometric Control of Mechanical Systems*. Vol. 49. Texts in Applied Mathematics. New York-Heidelberg-Berlin: Springer Verlag, 2004. ISBN: 0-387-22195-6.

[4] Ching-An Cheng et al. *RMPflow : A Computational Graph for Automatic Motion Policy Generation*. 2018.

[5] Ching-An Cheng et al. *RMPflow: A Geometric Framework for Generation of Multi-Task Motion Policies*. 2020.

[6] Peter Corke. *Robotics, Vision and Control - Fundamental Algorithms in MATLAB®*. Vol. 73. Springer Tracts in Advanced Robotics. Springer, 2011, pp. 1–495. ISBN: 978-3-642-20143-1. URL: http://dblp.uni-trier.de/db/series/star/index.html#Corke11.

[7] Mark L. Darby and Michael Nikolaou. "MPC: Current practice and challenges". In: *Control Engineering Practice* 20.4 (2012). Special Section: IFAC Symposium on Advanced Control of Chemical Processes - ADCHEM 2009, pp. 328–342. ISSN: 0967-0661. DOI: https://doi.org/10.1016/j.conengprac.2011.12.004. URL: https://www.sciencedirect.com/science/article/pii/S0967066111002528.

[8] Roy Featherstone and Oussama Khatib. "Load Independence of the Dynamically Consistent Inverse of the Jacobian Matrix". In: *International Journal of Robotics Research* 16.2 (1997), pp. 168–170. URL: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.949.7988&rep=rep1&type=pdf.

[9] Dieter Fox et al. "Monte Carlo Localization: Efficient Position Estimation for Mobile Robots". In: Jan. 1999, pp. 343–349.

[10] Jacob Fraden. *Handbook of Modern Sensors*. Available at https://doi.org/10.1007/978-1-4419-6466-3. Springer New York, NY, 2010.

[11] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. "Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling". In: *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*. 2005.

[12] Lukas Hewing et al. "Learning-Based Model Predictive Control: Toward Safe Learning in Control". In: *Annual Review of Control, Robotics, and Autonomous Systems* 3 (1 May 2020), pp. 269–296. ISSN: 2573-5144. DOI: 10.1146/ANNUREV-CONTROL-090419-075625.

[13] J.M. Hollerbach et al. "Redundancy Resolution of Manipulators through Torque Optimization". In: *IEEE Journal of Robotics and Automation* RA-3.4 (1987), pp. 308–316. URL: https://dspace.mit.edu/handle/1721.1/5607.

[14] Armin Hornung et al. "OctoMap: an efficient probabilistic 3D mapping framework based on octrees." In: *Auton. Robots* 34.3 (2013), pp. 189–206. URL: http://dblp.uni-trier.de/db/journals/arobots/arobots34.html#HornungWBSB13.

[15] *Intel RealSense Product Family D400 Series Datasheet*. 337029-013. Rev. 013. Intel Corporation. Apr. 2022.

[16] Sertac Karaman and Emilio Frazzoli. "Sampling-based Algorithms for Optimal Motion Planning". In: *CoRR* abs/1105.1186 (2011). arXiv: 1105.1186. URL: http://arxiv.org/abs/1105.1186.

[17] L.E. Kavraki et al. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces". In: *IEEE Transactions on Robotics and Automation* 12.4 (1996), pp. 566–580. DOI: 10.1109/70.508439.

[18] Oussama Khatib. “A Unified Approach for Motion and Force Control of Robot Manipulators: The Operational Space Formulation”. In: *IEEE Journal of Robotics and Automation* VOL. RA-3 (NO. 1 1987), pp. 43–53.

[19] S. M. LaValle. *Planning Algorithms*. Available at http://planning.cs.uiuc.edu/. Cambridge, U.K.: Cambridge University Press, 2006.

[20] Steven M. LaValle. “Rapidly-exploring random trees : a new tool for path planning”. In: *The annual research report* (1998).

[21] John M. Lee. *Introduction to Smooth Manifolds, Graduate Texts in Mathematics 218*. Second Edition. Springer Science+Business Media New York 2013, 2000. DOI: DOI10.1007/978-1-4419-9982-5.

[22] Anqi Li et al. “RMP2: A Structured Composable Policy Class for Robot Learning”. In: Robotics: Science and Systems Foundation, June 2021. ISBN: 9780992374778. DOI: 10.15607/rss.2021.xvii.092.

[23] Sikang Liu et al. “Planning Dynamically Feasible Trajectories for Quadrotors Using Safe Flight Corridors in 3-D Complex Environments”. In: *IEEE Robotics and Automation Letters* 2.3 (2017), pp. 1688–1695. DOI: 10.1109/LRA.2017.2663526.

[24] Kevin M. Lynch and Frank C. Park. *Modern Robotics: Mechanics, Planning, and Control*. 1st. USA: Cambridge University Press, 2017. ISBN: 1107156300.

[25] Xiangyun Meng et al. *Neural Autonomous Navigation with Riemannian Motion Policy*. 2019.

[26] Manfred Morari and Jay H. Lee. “Model predictive control: past, present and future”. In: *Computers  Chemical Engineering* 23.4 (1999), pp. 667–682. ISSN: 0098-1354. DOI: https://doi.org/10.1016/S0098-1354(98)00301-9. URL: https://www.sciencedirect.com/science/article/pii/S0098135498003019.

[27] Mustafa Mukadam et al. “Riemannian Motion Policy Fusion through Learnable Lyapunov Function Reshaping”. In: *3rd Conference on Robot Learning (CoRL 2019)* (Oct. 2019). URL: http://arxiv.org/abs/1910.02646.

[28] Yoshihiko Nakamura and Hideo Hanafusa. “Optimal Redundancy Control of Robot Manipulators”. In: *The International Journal of Robotics Research* 6.1 (1987), pp. 32–42. DOI: 10.1177/027836498700600103. eprint: https://doi.org/10.1177/027836498700600103. URL: https://doi.org/10.1177/027836498700600103.

[29] Jun Nakanishi et al. “Operational Space Control: A Theoretical and Empirical Comparison”. In: *The International Journal of Robotics Research* 27.6 (2008), pp. 737–757. DOI: 10.1177/0278364908091463. eprint: https://doi.org/10.1177/0278364908091463. URL: https://doi.org/10.1177/0278364908091463.

[30] Matteo Poggi et al. *On the confidence of stereo matching in a deep-learning era: a quantitative evaluation*. 2021. DOI: 10.48550/ARXIV.2101.00431. URL: https://arxiv.org/abs/2101.00431.

[31] M. Quigley et al. “ROS: an open-source Robot Operating System”. In: *International Conference on Robotics and Automation*. Open-Source Software workshop. 2009.

[32] M Asif Rana et al. “Learning Reactive Motion Policies in Multiple Task Spaces from Human Demonstrations”. In: *3rd Conference on Robot Learning (CoRL 2019)* (2019).

[33] Nathan D. Ratliff et al. *Generalized Nonlinear and Finsler Geometry for Robotics*. 2021.

[34] Nathan D. Ratliff et al. *Optimization Fabrics*. 2020.

[35] Nathan D. Ratliff et al. *Optimization Fabrics for Behavioral Design*. 2021.

[36] Nathan D. Ratliff et al. *Riemannian Motion Policies*. 2018.

[37] Radu Bogdan Rusu and Steve Cousins. “3D is here: Point Cloud Library (PCL)”. In: *2011 IEEE International Conference on Robotics and Automation*. 2011, pp. 1–4. DOI: 10.1109/ICRA.2011.5980567.

[38] Bruno Siciliano and Oussama Khatib. *Springer Handbook of Robotics*. Springer, Berlin, Heidelberg, 2008. DOI: https://doi.org/10.1007/978-3-540-30301-5.

[39] Slamtec. *Slamtec RPLIDAR A2*. https://www.slamtec.com/en/Lidar/A2Spec. Accessed 2023.

[40] Max Spahn and Javier Alonso-Mora. “Autotuning Symbolic Optimization Fabrics for Trajectory Generation”. In: Feb. 2023.

[41] Max Spahn, Bruno Brito, and Javier Alonso-Mora. “Coupled Mobile Manipulation via Trajectory Optimization with Free Space Decomposition”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 12759–12765. DOI: 10.1109/ICRA48506.2021.9561821.

[42] Max Spahn, Martijn Wisse, and Javier Alonso-Mora. “Dynamic Optimization Fabrics for Motion Generation”. In: *IEEE Transactions on Robotics* (2023), pp. 1–16. DOI: 10.1109/TRO.2023.3255587.

[43] Endre Süli and David F. Mayers. *An Introduction to Numerical Analysis*. Cambridge University Press, 2003. DOI: 10.1017/CBO9780511801181.

[44] Marc Toussaint and Christian Goerick. “A Bayesian view on motor control and planning”. In: *From Motor to Interaction Learning in Robots* (2010).

[45] Caspar van Venrooij. *DSI Fabrics ROS*. https://github.com/casparvv/dsi_fabrics_ros. Accessed: May, 2023. 2023.

[46] Karl Van Wyk et al. “Geometric Fabrics: Generalizing Classical Mechanics to Capture the Physics of Behavior”. In: (Sept. 2022). URL: http://arxiv.org/abs/2109.10443.

[47] Mandy Xie et al. *Geometric Fabrics for the Acceleration-based Design of Robotic Motion*. 2021.