Autotuning Symbolic Optimization Fabrics for Trajectory Generation

Max Spahn* and Javier Alonso-Mora*

***Abstract*— In this paper, we present an automated parameter optimization method for trajectory generation. We formulate parameter optimization as a constrained optimization problem that can be effectively solved using Bayesian optimization. While the approach is generic to any trajectory generation method, we showcase it using optimization fabrics. Optimization fabrics are a geometric trajectory generation method based on non-Riemannian geometry. By symbolically pre-solving the structure of the tree of fabrics, we obtain a parameterized trajectory generator, called symbolic fabrics. We show that autotuned symbolic fabrics reach expert-level performance in a few trials. Additionally, we show that tuning transfers across different robots, motion planning problems and between simulation and real world. Finally, we qualitatively showcase that the framework could be used for coupled mobile manipulation.**

Code github.com/maxspahn/optuna_fabrics

I. Introduction

Mobile manipulation is the field of robotics concerned with highly capable robots characterized by their locomotion and manipulation ability. Such robots are getting ever more attention as they will be deployed to human-shared environments, like households or warehouses. In such dynamic environments, fast trajectory generation is crucial to avoid collisions and react quickly to changing goal definitions.

Trajectory generation is often addressed by solving an optimization problem that consists of a scalar objective function – the dynamics or transition function – and several constraints. As the degrees of freedom and number of constraints increase, solving that problem in real-time becomes challenging. This is especially limiting in the case of mobile manipulation [1]. Optimization fabrics represent a different approach to the problem, as they formulate trajectory generation as the shortest-geodesic-problem in a manifold of the configuration space [2].

With optimization fabrics, different components, or desired behaviors, such as collision avoidance and joint limit avoidance, are combined using Riemannian metrics. As the structure of the resulting trajectory generation methods remains unchanged across all time steps, it can be composed before runtime, thus saving computational costs during executing. Optimization fabrics, but also their predecessor Riemannian Motion Policies (RMPs), have shown impressive results for several manipulator applications, including dynamic and crowded environments [3]–[5]. However, despite their theoretical properties of inherent collision avoidance and convergence, these methods require expertise and intuition to tune individual components to generate smooth and well-behaving trajectories.

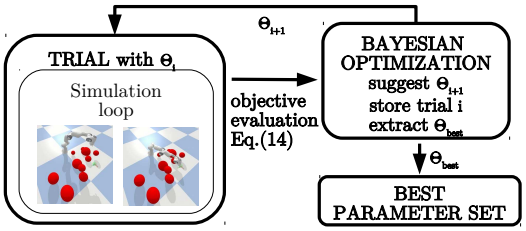


Fig. 1: Overview of one trial in the tuning pipeline for symbolic optimization fabrics. The objective function is evaluated after an entire trial run is simulated. Using Bayesian optimization, a new parameter set is suggested based on the history of trials. The best parameter set is extracted from all trials.

Contributions: To address this issue, we formulate optimization fabrics as a **symbolic trajectory generation** method. Precisely, the combination of the individual components (joint limit avoidance, goal reaching, collision avoidance, etc.) is performed in a parameterized way before runtime. Separating composition and evaluation allows for changing the individual parameters at runtime while achieving low computational costs. Additionally, this allows formulating parameter-tuning as a constrained optimization problem. Solving this problem effectively **automates the tuning process** systematically using Bayesian optimization. We show that automated tuning requires only few trials to achieve similar performance to an expert in the field, and systematically outperforms a randomized parameter setting. Moreover, we show that one parameter tuning generalizes across different robots, to some extent, across different tasks and between simulation and real world. Finally, we demonstrate how **coupled mobile manipulation** with a differential drive can be achieved using autotuned optimization fabrics for in-store order-picking integrating visual servoing.

II. Related Works

A. Geometric control for trajectory generation

Operational space control was the first control method that imposed a desired dynamical system onto a robotic system [6], [7]. The concept was an important step toward naturally controlling kinematically redundant robots. The concept was formalized in the field of geometric control, where the study of differential geometry leads to stable and converging behavior under geometric conditions [8]. More recently, RMPs for manipulation tasks offered a highly reactive trajectory generation method [9], [10]. This method achieves

*This research was supported by Ahold Delhaize. All content represents the opinion of the author(s), which is not necessarily shared or endorsed by their respective employers and/or sponsors.

The authors are with the Department of Cognitive Robotics, Delft University of Technology, 2628 CD, Delft, The Netherlands {m.spahn, j.alonsomora}@tudelft.nl

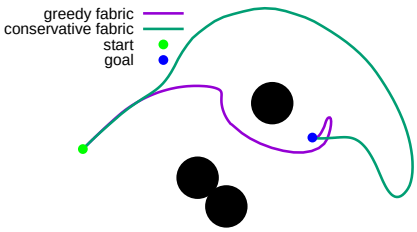


Fig. 2: Two different parameter sets for optimization fabrics given the same problem. While the greedy tuning is more aggressive (purple), the more conservative tuning results in a smoother trajectory (green).

composable behavior by introducing a split between the importance metric and the forcing term. Using the *pullback* and *pushforward* operator to change between manifolds of the configuration space, individual components, such as collision avoidance and goal attraction, can be designed iteratively. However, RMPs require intuition and experience when being designed, and convergence can only be proven conditionally [11]. Later, optimization fabrics were introduced that are able to completely decouple importance metrics and the defining geometry. Under simple construction rules for these two components, convergence can be easily guaranteed [4], [11]–[13]. In our prior work on optimization fabrics, the framework was first applied to mobile manipulation and generalized to more dynamic environments. [5].

B. Autotuning for trajectory generation

Autotuning can be beneficial for trajectory generation when using model predictive control. In [14], an autotuned model predictive controller has outperformed a manual tuned controller of the same kind by 25%. Jointly optimizing parameters and the model of the controller, *AutoMPC* showed the benefit of parameter tuning in the context of simultaneous system identification and control [15]. These methods are explicitly formulated for model predictive control and do not transfer easily to other trajectory generation methods. In contrast, we propose a generic parameter optimization approach to trajectory generation.

C. Hyperparameter tuning in machine learning

Within the machine-learning field, hyperparameter tuning has shown to be highly important for all different kinds of applications [16]–[18]. Parameter optimization aims to minimize training costs while achieving the best possible performance. Hyperparameter tuning is most valuable in extremely costly applications such as reinforcement learning [19]. Generally, two different search algorithms have been investigated: grid search and random search [20]. Current state-of-the-art methods for parameter search are based on random search with a Bayesian optimizer [18], [21]. While the machine-learning community has largely agreed on the importance of parameter tuning, systematic tuning of trajectory generation methods are not well established. In this paper, we showcase, with the example of optimization fabrics, how important parameter tuning is and how trajectory generation can benefit from it.

III. Overview

In this paper, we first recall very briefly the theory of optimization fabrics and the steps to use it for trajectory generation (Section IV). Then, we formulate optimization fabrics as a symbolic trajectory generator, so that combining of individual components is only performed once (Section V). Then, we formulate parameter tuning for trajectory generation as a constrained optimization problem and propose Bayesian optimization for effective autotuning (Section VI). As an example, we apply this autotuning to symbolic optimization fabrics (Section VII), but it is generally independent of the trajectory generator at hand.

IV. Background

In this section, we very briefly introduce the concepts required for trajectory generation with optimization fabrics. For a more in-depth introduction to optimization fabrics and its foundations in differential geometry, the reader is referred to [3], [5], [11].

A. Configurations and task variables

We denote $\boldsymbol{q} \in \mathcal{Q} \subset \mathbb{R}^n$ a configuration of the robot with n its degrees of freedom; $\mathcal{Q}$ is the configuration space of the generalized coordinates of the system. Generally, $\boldsymbol{q}(t)$ defines the robot's configuration at time t, so that $\dot{\boldsymbol{q}}$, $\ddot{\boldsymbol{q}}$ define the instantaneous derivatives of the robot's configuration. Similarly, we assume that there is a set of task variables $\boldsymbol{x}_j \in \mathcal{X}_j \subset \mathbb{R}^{m_j}$ with variable dimension $m_j \leq n$. The task space $\mathcal{X}_j$ defines an arbitrary manifold of the configuration space $\mathcal{Q}$ in which a robotic task can be represented. Further, we assume that there is a differential map $\phi_j : \mathbb{R}^n \to \mathbb{R}^{m_j}$ that relates the configuration space to the j^{th} task space. For example, when a task variable is defined as the end-effector position, then ϕ_j is the positional part of the forward kinematics. On the other hand, if a task variable is defined to be the joint position, then ϕ_j is the identity function. In the following, we drop the subscript j in most cases for readability when the context is clear.

We assume that ϕ is in $\mathcal{C}^1$ so that the Jacobian is defined as

$$\boldsymbol{J}_\phi = \frac{\partial \phi}{\partial \boldsymbol{q}} \in \mathcal{R}^{m \times n}, \tag{1}$$

or $\boldsymbol{J}_\phi = \partial_{\boldsymbol{q}} \phi$ for short. Thus, we can write the total time derivatives of $\boldsymbol{x}$ as $\dot{\boldsymbol{x}} = \boldsymbol{J}_\phi \dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{x}} = \boldsymbol{J}_\phi \ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}_\phi \dot{\boldsymbol{q}}$.

B. Spectral semi-sprays

Inspired by simple mechanics (e.g., the simple pendulum), the framework of optimization fabrics designs motion policies as second-order dynamical systems $\ddot{\boldsymbol{x}} = \pi(\boldsymbol{x}, \dot{\boldsymbol{x}})$ [11], [22]. The motion policy is defined by the differential equation $\boldsymbol{M}\ddot{\boldsymbol{x}} + \boldsymbol{f} = 0$, where $\boldsymbol{M}(\boldsymbol{x}, \dot{\boldsymbol{x}})$ and $\boldsymbol{f}(\boldsymbol{x}, \dot{\boldsymbol{x}})$ are functions of position and velocity. Besides, $\boldsymbol{M}$ is symmetric and invertible. We denote such systems as $\mathcal{S} = (\boldsymbol{M}, \boldsymbol{f})_{\mathcal{X}}$ and refer to them as *spectral semi-sprays*, or *specs* for short. When the space of the task variable is clear from the context, we drop the subscript.

C. Operations on specs

Complex trajectory generation is composed of multiple components, such as collision avoidance, joint limits avoidance, etc. The power of optimization fabrics lies in the metric-weighted sum to combine multiple components from different manifolds. These operations are derived from operations on specs and are briefly recalled here.

Given a differential map $\phi : \mathcal{Q} \to \mathcal{X}$ and a spec $(\boldsymbol{M}, \boldsymbol{f})_{\mathcal{X}}$, the *pullback* is defined as

$$\text{pull}_{\phi}(\boldsymbol{M}, \boldsymbol{f})_{\mathcal{X}} = \left(\boldsymbol{J}_{\phi}^{T} \boldsymbol{M} \boldsymbol{J}_{\phi}, \boldsymbol{J}_{\phi}^{T}(\boldsymbol{f} + \dot{\boldsymbol{J}}_{\phi}\dot{\boldsymbol{q}})\right)_{Q}. \quad (2)$$

The pullback allows converting between two distinct manifolds (e.g. a spec could be defined in the robot's workspace and pulled into the robot's configuration space using the pullback with ϕ being the forward kinematics).

For two specs, $\mathcal{S}_1 = (\boldsymbol{M}_1, \boldsymbol{f}_1)_{\mathcal{X}}$ and $\mathcal{S}_2 = (\boldsymbol{M}_2, \boldsymbol{f}_2)_{\mathcal{X}}$, their *summation* is defined by:

$$\mathcal{S}_1 + \mathcal{S}_2 = (\boldsymbol{M}_1 + \boldsymbol{M}_2, \boldsymbol{f}_1 + \boldsymbol{f}_2)_{\mathcal{X}}. \quad (3)$$

Additionally, a spec can be *energized* by a Lagrangian energy. Effectively, this equips the spec with a metric. Specifically, given a spec of form $\mathcal{S}_{\boldsymbol{h}} = (\boldsymbol{I}, \boldsymbol{h})$ and an energy Lagrangian $\mathcal{L}_e$ with the derived equations of motion $\boldsymbol{M}_{\mathcal{L}_e}\ddot{\boldsymbol{x}} + \boldsymbol{f}_{\mathcal{L}_e} = 0$, we can define the operation

$$\begin{aligned} S_{\boldsymbol{h}}^{\mathcal{L}_e} &= \text{energize}_{\mathcal{L}_e}\{S_{\boldsymbol{h}}\} \\ &= (\boldsymbol{M}_{\mathcal{L}_e}, \boldsymbol{f}_{\mathcal{L}_e} + \boldsymbol{P}_{\mathcal{L}_e}[\boldsymbol{M}_{\mathcal{L}_e}\boldsymbol{h} - \boldsymbol{f}_{\mathcal{L}_e}]), \end{aligned} \quad (4)$$

where $\boldsymbol{P}_{\mathcal{L}_e} = \boldsymbol{M}_{\mathcal{L}_e}\left(\boldsymbol{M}_{\mathcal{L}_e}^{-1} - \frac{\dot{\boldsymbol{x}}\dot{\boldsymbol{x}}^T}{\dot{\boldsymbol{x}}^T \boldsymbol{M}_{\mathcal{L}_e}\dot{\boldsymbol{x}}}\right)$ is an orthogonal projector. The resulting spec is an *energized spec* and we call the operation *energization*.

With spectral semi-sprays and the presented operations, avoidance behavior, such as joint limit avoidance, collision avoidance or self-collision avoidance, can be realized.

D. Optimization fabrics

In the previous subsection, we explained how different avoidance behaviors can be combined. Spectral semi-sprays can additionally be *forced* by a potential, denoted as the *forced variant* of form $\mathcal{S}_{\psi} = (\boldsymbol{M}, \boldsymbol{f} + \partial_{\boldsymbol{x}}\psi)$. This forcing term clearly changes the behavior of the system. Optimization fabrics introduce construction rules to make sure that the solution path $\boldsymbol{x}(t)$ of $\mathcal{S}_{\psi}$ converges towards the minimum of $\psi(\boldsymbol{x})$. Then, the potential is designed in such a way that its minimum represents a goal state of the motion planning problem.

First, the initial spec that represents an avoidance component is written in the form $\ddot{\boldsymbol{x}} + \boldsymbol{h}(\boldsymbol{x}, \dot{\boldsymbol{x}}) = 0$, such that $\boldsymbol{h}$ is *homogeneous of degree 2*: $\boldsymbol{h}(\boldsymbol{x}, \alpha\dot{\boldsymbol{x}}) = \alpha^2\boldsymbol{h}(\boldsymbol{x}, \dot{\boldsymbol{x}})$ (**Creation**). Secondly, the geometry is energized (Eq. (4)) with a Finsler structure [11, Definition 5.4] (**Energization**). The property of homogeneity of degree 2 and the energization with the Finsler structure guarantees, according to [11, Theorem 4.29], that the energized spec forms a *frictionless fabric*. A frictionless fabric is defined to optimize the forcing potential ψ when being damped by a positive definite damping term [11, Definition 4.4]. Thirdly, all avoidance components are combined in the configuration space of the robot using the pullback and summation operation (**Combination**). Note, that both operations are closed under the algebra designed by these operations, i.e. every pulled optimization fabric or the sum of two optimization fabrics is, itself, an optimization fabric. In the last step, the combined spec is forced by the potential ψ with the desired minimum and damped with a positive definite damping term (**Forcing**). This resulting system of form $\boldsymbol{M}\ddot{\boldsymbol{q}} + \boldsymbol{f}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \partial_{\boldsymbol{q}}\psi + \beta\dot{\boldsymbol{q}} = 0$ is solved to obtain the trajectory generation policy in acceleration form $\ddot{\boldsymbol{q}} = \pi(\boldsymbol{q}, \dot{\boldsymbol{q}})$.

V. Symbolic fabrics

A trajectory generator that is based on optimization fabrics is composed of several components, such as collision avoidance, joint limit avoidance, goal attraction, etc. Each component contributes to the resulting optimization fabric through the metric-weighted summation that creates the tree of fabrics. The trajectory generator is parameterized by the individual terms of the components. Here, we lay out the parameterization for collision avoidance, joint limit avoidance, self-collision avoidance, and speed-control. In our framework, the tree of fabrics is generated before runtime as a symbolic expression, to which the parameters are set at runtime. Note that the approach of symbolic pre-solving results in much higher planning frequencies. In the following, we explain the individual parameters that we exposed symbolically. The form of the individual terms is adapted from [3], [4], [11] but written in a symbolic form.

a) Basic inertia: The final tree of fabrics is equipped with a basic inertia metric that indicates how reactive the entire motion is. This basic inertia metric is derived from the symbolic Finsler structure: $\mathcal{L}_e = 0.5 m_{\text{base}} \dot{\boldsymbol{q}}^T \boldsymbol{I} \dot{\boldsymbol{q}}$.

b) Collision avoidance: For collision avoidance, the task manifold $\mathcal{X}$ is defined by the distance function between an obstacle and a robot link. The differential map used is defined as

$$\phi_{\text{i}}(\boldsymbol{q}) = \frac{\|\text{fk}_{\text{i}}(\boldsymbol{q}) - \boldsymbol{x}_{\text{obst}}\|}{r_{\text{obst}} + r_{\text{i}}} - 1,$$

where $\text{fk}_{\text{i}}(\boldsymbol{q})$ is the positional forward kinematics for link i in a configuration $\boldsymbol{q}$, r_{obst} and r_{i} are the radii of the englobing spheres for the obstacle and the link respectively. While this mapping between configuration space and task manifold is different for each obstacle and each collision link of the robot, the geometry and metric are the same for all of them. For the geometry $\ddot{\boldsymbol{x}} + \boldsymbol{h}(\boldsymbol{x}, \dot{\boldsymbol{x}}) = \boldsymbol{0}$, we use the parameterized forcing term

$$\boldsymbol{h}(\boldsymbol{x}, \dot{\boldsymbol{x}}) = \frac{-k_{\text{geo,col}}}{\boldsymbol{x}^{\beta_{\text{geo,col}}}}\dot{\boldsymbol{x}}^2, \quad (5)$$

where $k_{\text{geo,col}}$ and $\beta_{\text{geo,col}}$ are parameters of the trajectory generator. Generally, we use k and β for proportional parameters and exponential parameters. The Finsler structure for collision avoidance is parameterized as

$$\mathcal{L}_e(\boldsymbol{x}, \dot{\boldsymbol{x}}) = \frac{k_{\text{fin,col}}}{\boldsymbol{x}^{\beta_{\text{fin,col}}}}\left(-0.5(\text{sgn}(\dot{\boldsymbol{x}}) - 1)\right)\dot{\boldsymbol{x}}^2, \quad (6)$$

where $\text{sgn}(\dot{\boldsymbol{x}})$ is the signum-operator returning the sign of $\dot{\boldsymbol{x}}$.

c) Self-collision avoidance: For self-collision avoidance, the task manifold $\mathcal{X}$ is defined similarly to collision avoidance:

$$\phi_{i,j} = \frac{\|\text{fk}_i(\boldsymbol{q}) - \text{fk}_j(\boldsymbol{q})\|}{r_i + r_j} - 1,$$

where $\text{fk}_i(\boldsymbol{q})$ and $\text{fk}_j(\boldsymbol{q})$ are the positional forward kinematics of the two links for a self-collision pair and r_i and r_j are the radii for both englobing spheres. The geometries are defined analogously

$$\boldsymbol{h}(\boldsymbol{x}, \dot{\boldsymbol{x}}) = \frac{-k_{\text{geo,self}}}{\boldsymbol{x}^{\beta_{\text{geo,self}}}} \dot{\boldsymbol{x}}^2. \tag{7}$$

The Finsler structure for collision avoidance is parameterized as

$$\mathcal{L}_e(\boldsymbol{x}, \dot{\boldsymbol{x}}) = \frac{k_{\text{fin,self}}}{\boldsymbol{x}^{\beta_{\text{fin,self}}}} \left(-0.5(\text{sgn}(\dot{\boldsymbol{x}}) - 1)\right) \dot{\boldsymbol{x}}^2. \tag{8}$$

d) Joint limit avoidance: For joint-limit avoidance, two simple differential maps denoting the distance to the joint limits are used, specifically

$$\phi_{\text{limit,i,lower}}(\boldsymbol{q}) = \boldsymbol{q}_i - \boldsymbol{q}_{\text{min,i}}, \forall i \in (1, \ldots, n)$$
$$\phi_{\text{limit,i,upper}}(\boldsymbol{q}) = \boldsymbol{q}_{\text{max,i}} - \boldsymbol{q}_i, \forall i \in (1, \ldots, n).$$

Similar to collision avoidance, we use the parameterized forcing term

$$\boldsymbol{h}(\boldsymbol{x}, \dot{\boldsymbol{x}}) = \frac{-k_{\text{geo,limit}}}{\boldsymbol{x}^{\beta_{\text{geo,limit}}}} \dot{\boldsymbol{x}}^2 \tag{9}$$

and the Finsler structure

$$\mathcal{L}_e(\boldsymbol{x}, \dot{\boldsymbol{x}}) = \frac{k_{\text{fin,limit}}}{\boldsymbol{x}^{\beta_{\text{fin,limit}}}} \left(-0.5(\text{sgn}(\dot{\boldsymbol{x}}) - 1)\right) \dot{\boldsymbol{x}}^2. \tag{10}$$

e) Speed control: As the root of the tree of fabrics is a frictionless fabric, it only converges if damped [11]. Constant damping is sufficient to achieve the theoretical properties that are needed for trajectory generation. However, [3], [11], [23] proposed enhanced damping under the name of *speedcontrol*. We employ the same damping strategy while adding parameterization. The technique is based on a dynamic damping modification based on the distance to the goal. Specifically, the final optimization fabric is damped according to

$$\ddot{\boldsymbol{q}} = -\boldsymbol{h}_2 - \boldsymbol{M}^{-1} \partial_{\boldsymbol{q}} \boldsymbol{\psi} + \alpha_{\text{ex}} \dot{\boldsymbol{q}} - \beta \dot{\boldsymbol{q}},$$

where $\boldsymbol{h}_2$ is the sum of all pulled forcing terms, $\boldsymbol{M}$ is the sum of all metrics of the individual geometries, $\partial_{\boldsymbol{q}} \boldsymbol{\psi}$ is the goal attraction term pulled in the configuration space, α_{ex} is a weighted sum of α^0_{ex} that maintains constant execution energy without goal attraction and $\alpha^{\boldsymbol{\psi}}_{\text{ex}}$ that maintains constant execution energy with goal attraction:

$$\alpha_{\text{ex}} = \boldsymbol{s}_\eta(\mathcal{L}_{ex}) \alpha^0_{\text{ex}} + (1 - \boldsymbol{s}_\eta(\mathcal{L}_{ex})) \alpha^{\boldsymbol{\psi}}_{\text{ex}}.$$

Then, β is the damping term, computed as:

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

where $\boldsymbol{B}_{\text{max}}$ and $\boldsymbol{B}_{\text{min}}$ are the upper and lower damping values and $\alpha_{\mathcal{L}_e}$ is the energization coefficient maintaining constant system energy (not execution energy) without goal attraction. The switching functions $\boldsymbol{s}_\beta(\boldsymbol{q})$, $\boldsymbol{s}_\eta(\boldsymbol{q})$ are further parameterized as

$$\boldsymbol{s}_\beta(\boldsymbol{q}) = 0.5(\tanh -\alpha_\beta(\|\boldsymbol{q}\| - r_{shift})) + 1$$
$$\boldsymbol{s}_\eta(\mathcal{L}_{ex}) = 0.5(\tanh(-0.5 \mathcal{L}_{ex}(1 - v_{ex}) - 0.5) + 1),$$

where r_{shift} determines the distance to the goal at which the switch between $\boldsymbol{B}_{\text{min}}$ and $\boldsymbol{B}_{\text{max}}$ occurs, α_β is the steepness of that switching, $\mathcal{L}_{ex}$ is the user-defined execution energy (usually a simple kinetic energy in joint space) and v_{ex} is the execution energy factor, i.e. it determines the desired speed of motion. For a detailed discussion on speed control with optimization fabrics, we refer to previous works on optimization fabrics [3], [11].

We group all parameters resulting from the symbolic fabrics defined here into a vector of parameters $\boldsymbol{\Theta}$. All parameters are listed in Table I.

VI. Parameter tuning as an optimization problem

We define parameter tuning as a constrained optimization problem:

$$\boldsymbol{\Theta}^* = \arg\min_{\boldsymbol{\Theta}} c(\boldsymbol{\Theta}), \text{ s.t } \boldsymbol{\Theta}_{\text{min}} < \boldsymbol{\Theta} < \boldsymbol{\Theta}_{\text{max}}, \tag{11}$$

where $\boldsymbol{\Theta}_{\text{max}}$ and $\boldsymbol{\Theta}_{\text{min}}$ are the upper and lower bounds of the parameters. The objective $c(\boldsymbol{\Theta})$ is a function of the parameters specifying the tree of fabrics and can be evaluated after one trajectory planning problem has finished. We call the evaluation of one parameter set a *trial*. Next, we propose an objective function that is flexible as different scenarios may require different parameter tuning.

A. Objective

The objective function $c(\boldsymbol{\Theta})$ is a weighted sum of several metrics, that are invariant to the robot:

$$c(\boldsymbol{\Theta}) = w_{\text{distance}} c_{\text{distance}} + w_{\text{path}} c_{\text{path}} + w_{\text{clearance}} c_{\text{clearance}}. \tag{12}$$

c_{distance} accounts for the normalized, summed distance to the goal over one trial and is defined as

$$c_{\text{distance}} = \frac{\sum_{i=0}^{T} \|\boldsymbol{x}_i - \boldsymbol{x}_{\text{goal}}\|}{\|\boldsymbol{x}_0 - \boldsymbol{x}_{\text{goal}}\|}, \tag{13}$$

where $i \in [0, T]$ are the discretized time steps and $\boldsymbol{x}_{\text{goal}}$ is the goal of the motion planning problem. c_{path} accounts for the normalized path length over one trial and is defined as

$$c_{\text{path}} = \frac{\sum_{i=1}^{T} \|\boldsymbol{x}_i - \boldsymbol{x}_{i-1}\|}{\|\boldsymbol{x}_0 - \boldsymbol{x}_{\text{goal}}\|}. \tag{14}$$

$c_{\text{clearance}}$ accounts for the average clearance to obstacles over one trial and is defined as

$$c_{\text{clearance}} = \frac{1}{T} \sum_{i=1}^{T} \min_{o^j} \left\| \boldsymbol{x}_i - o^j_i) \right\|, \tag{15}$$

where o^j_i is the position of obstacle j at time step i. Each of these terms is evaluated after an entire trial that was obtained by a specific set of parameters.

B. Bayesian optimization

In the tuning phase, the problem specification for the investigated scenario, e.g., the goal and obstacle positions, across all trials during tuning remains the same while $\boldsymbol{\Theta}$ are optimized according to the objective. To solve the Bayesian optimization we employ the *Tree-structured Parzen Estimator* as it has shown improved performance over grid-search and conventional random search in machine learning applications [21], [24]. To deploy this technique we used

Algorithm 1: Autotuning for trajectory generators

Formulate trajectory generator with parameters $\boldsymbol{\Theta}$
Define parameter space by $\boldsymbol{\Theta}_{\text{min}}, \boldsymbol{\Theta}_{\text{max}}$
Formulate objective $c(\boldsymbol{\Theta})$
Initialize objective function estimate $\tilde{c}(\boldsymbol{\Theta})$
for *i = 0 to N* **do**
  Suggest parameter $\boldsymbol{\Theta}_i$ based on $\tilde{c}(\boldsymbol{\Theta})$
  for *t = 0 to T* **do**
    Compute action with parameter set $\boldsymbol{\Theta}_i$
    Apply action to robot
    Store observation relevant for metrics
  end
  Evaluate $c(\boldsymbol{\Theta}_i)$
  Update $\tilde{c}(\boldsymbol{\Theta})$
end
Extract the best parameter set $\boldsymbol{\Theta}_{\text{best}}$

Parameter	boundaries	type	distribution	manual
m_{base}	$[0, 1]$	float	uniform	0.2
$k_{\text{geo,col}}$	$[0.01, 1]$	float	log	0.03
$k_{\text{geo,limit}}$	$[0.01, 1]$	float	log	0.3
$k_{\text{geo,self}}$	$[0.01, 1]$	float	log	0.03
$k_{\text{fin,col}}$	$[0.01, 1]$	float	log	0.03
$k_{\text{fin,limit}}$	$[0.01, 1]$	float	log	0.05
$k_{\text{fin,self}}$	$[0.01, 1]$	float	log	0.03
$\beta_{\text{geo,col}}$	$[1, 5]$	int	uniform	3
$\beta_{\text{geo,limit}}$	$[1, 5]$	int	uniform	2
$\beta_{\text{geo,self}}$	$[1, 5]$	int	uniform	3
$\beta_{\text{fin,col}}$	$[1, 5]$	int	uniform	3
$\beta_{\text{fin,limit}}$	$[1, 5]$	int	uniform	3
$\beta_{\text{fin,self}}$	$[1, 5]$	int	uniform	3
α_β	$[0, 1]$	float	uniform	0.5
$\boldsymbol{B}_{\text{min}}$	$[0, 1]$	float	uniform	0.01
$\boldsymbol{B}_{\text{max}}$	$[5, 20]$	float	uniform	6.5
r_{shift}	[0.01, 0.1]	float	uniform	0.05
v_{ex}	[1.0, 30]	float	uniform	15.0

TABLE I: Search space for parameters. Some parameters are restricted to integers, and for some a log-distribution is applied.

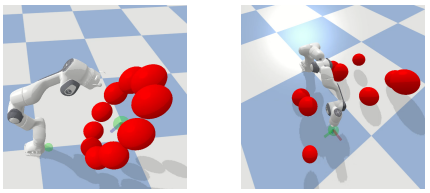
Optuna, a hyperparameter optimization framework initially designed for machine learning applications [18]. The general setup for one trial is shown in Fig. 1 and the procedure is summarized in Algorithm 1.

VII. Experimental Results

We showcase our parameter optimization method for symbolic fabrics. The search space for the parameters is summarized in Table I. We first analyze the importance of tuning for optimization fabrics on the performance of trajectory generation. Then, we investigate how tuned parameters can be transferred across different robots (Section VII-C), different scenarios (Section VII-D), and between simulation and real world (Section VII-E).

A. Experimental setup

The method was tested in simulation and in the real world on a Panda robot and a mobile manipulator composed of a Clearpath Boxer and a Panda robot. The simulation uses the pybullet physics engine with an interface through OpenAI-gym [25]. The different motion planning goals evaluated



(a) *reaching-in-ring* (b) *reaching-on-table*

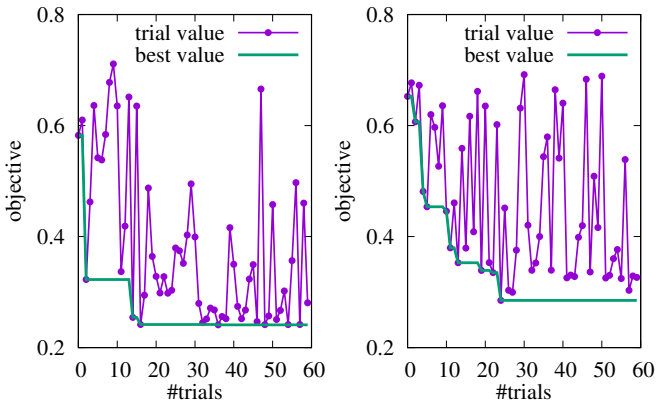


Fig. 4: Optimization history for simulation (left) and real world (right) for panda robot in *reaching-in-ring* scenario.

in this paper are: (a) reaching an end-effector pose inside a ring of obstacles (Fig. 3a) (similar to the experiment in [3]) and (b) reaching an end-effector pose above a surface with random obstacles (Fig. 3b). The two scenarios will be referred to as *reaching-in-ring* and *reaching-on-table*, see Section VII-A. Unless stated otherwise, the weights are set to $w_{\text{path}} = 0.1, w_{\text{clearance}} = 0.2, w_{\text{distance}} = 0.7$. We also use this weighted sum as the performance metric. While these weights are chosen arbitrarily in this work to demonstrate the usefulness of autotuning, they should be derived from a human evaluator in a more realistic scenario. We refer with *manual* to an expert-tuning, see Table I for specific parameters. During testing, the trial was randomized with changing obstacles and goals. For autotuning on the robotic arms, we consistently used $N = 60$ trials, although the best parameter set is usually reached earlier, see Fig. 4.

B. Importance of tuning

We compare the autotuned parameters with seven random parameter sets from the search space and a manually tuned parameter set that we obtained through expertise in previous works like [5]. In this experiment, tuning and testing are performed on the test scenario *reaching-in-ring*. Tuning is crucial for optimization fabrics, as the performance with a random parameter set cannot compete with tuning, Fig. 5. This result was expected and should only demonstrate that the right parameter set is required to deploy this method. Autotuned parameters reach a similar performance to the expert. This result highlights the importance of tuning for optimization fabrics and shows that autotuning is an effective way to obtain parameter sets for novice users of optimization fabrics.

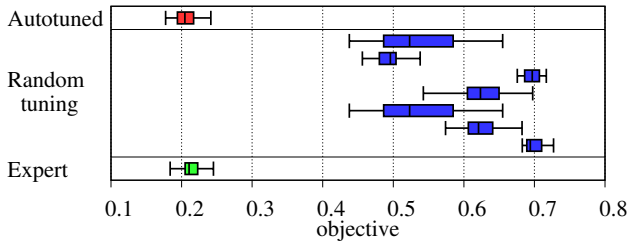


Fig. 5: Evaluation for scenario *reaching-in-ring* autotuned parameters and compared to random parameter selection and manual tuning. Autotuning is able to systematically outperform random parameter sets and reach expert level tuning.

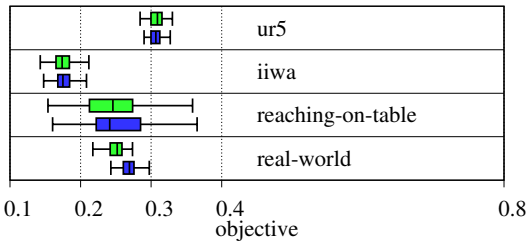


Fig. 6: The autotuned for the panda robot in simulation for the *reaching-in-ring* scenario on modified scenarios (blue) is compared to autotuned parameter sets obtained on these scenarios directly (green). Exchanging the robot (ur5, iiwa) and changing the scenario (*reaching-on-table*) results in a very small loss in performance, while the loss is higher when parameters are transferred between simulation and real world (real-world).

C. Cross validation: Transfer across robots

Without any retuning, we deploy the symbolic optimization fabrics planner tuned on the Panda robot on two other robots with similar specifications (Kuka LBR IIwa 7, Universal Robot UR5) and compare the performance with tuning performed on the respective robot. Specifically, we do not change the leaf geometries and energies but change differential maps according to relevant collision links on the robot at hand. From Fig. 6, we conclude that tuning is independent of the robot. This can be explained by the fact, that optimization fabrics are a purely geometric approach to trajectory generation and the different dimension of the robots do not change the dynamical system enforced onto the robot.

D. Cross Validation: Transfer across scenarios

In the third experiment, we evaluate how well an autotuned parameter set transfers to a different scenario. In the specific example, we use the tuning obtained from the *reaching-in-ring* case and test it on *reaching-on-table*. Performance can be transferred smoothly if the objective remains the same, see Fig. 6. However, note that different scenario might require generally slower motion because of a more crowded environment. Such a step would require to retune the parameters according to the new objective.

Fig. 7: Trajectory generation with optimization fabrics for mobile manipulator using visual serving for product picking.

E. Cross Validation: Transfer real world

As optimization fabrics are a geometric method [3], they should be independent of the robot embodiment. Relying on the low-level controller. In this paper, we investigate how the performance is affected by the transfer from the simulation environment to the real world. Performance benefits from tuning in the real world highlight that low-level controller differences affect the behavior, see Fig. 6. Specifically, the accumulated distance to the goal is increased (0.14m tuned in the real world vs 0.16m tuned in simulation) when tuning is transferred between simulation and real world. Thus, there is added value in tuning in the real world. Our framework offers to quickly tune fabrics in the real-world using the *fabrics-ros-bridge*. With relaxed performance requirements, it is sufficient to tune in simulation.

F. Cross Validation: Transfer mobile manipulator

Finally, we qualitatively test the performance of the tuning method on a real mobile manipulator with 10 degrees of freedom. After only $N = 30$ trials, the robot was able to perform coupled mobile manipulation based on a visual servoing approach [26]. Symbolic optimization fabrics are especially suited for visual servoing as their symbolic character allows them to constantly update the position of the goal. A video of this experiment is attached to the paper.

VIII. Conclusion

We formulated parameter tuning for trajectory generation as a constrained optimization problem. Additionally, we introduced symbolic optimization fabrics that implement optimization fabrics in a parameterized way, for which the general structure is pre-solved. The trajectory generator obtained with this technique is parameterized and achieves low computational costs at runtime. We showed that parameter tuning for symbolic optimization fabrics can be effectively solved using Bayesian optimization. Additionally, we have shown that the tuning generalized across different robots, tasks, and between simulation and the real world. Finally, we qualitatively demonstrated that the method applies to mobile manipulators. While we aim at developing a method-agnostic autotuning framework for motion generation, symbolic optimization fabrics were selected as an example in this work.

REFERENCES

[1] M. Spahn, B. Brito, and J. Alonso-Mora, "Coupled mobile manipulation via trajectory optimization with free space decomposition," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 12 759–12 765.

[2] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," in *International Journal of Robotics Research*, vol. 30, no. 7, jun 2011, pp. 846–894. [Online]. Available: http://journals.sagepub.com/doi/10.1177/0278364911406761

[3] K. Van Wyk, M. Xie, A. Li, M. A. Rana, B. Babich, B. Peele, Q. Wan, I. Akinola, B. Sundaralingam, D. Fox, B. Boots, and N. D. Ratliff, "Geometric fabrics: Generalizing classical mechanics to capture the physics of behavior," number: arXiv:2109.10443. [Online]. Available: http://arxiv.org/abs/2109.10443

[4] M. Xie, K. Van Wyk, A. Li, M. A. Rana, Q. Wan, D. Fox, B. Boots, and N. Ratliff, "Geometric fabrics for the acceleration-based design of robotic motion," number: arXiv:2010.14750. [Online]. Available: http://arxiv.org/abs/2010.14750

[5] M. Spahn, M. Wisse, and J. Alonso-Mora, "Dynamic optimization fabrics for motion generation," number: arXiv:2205.08454. [Online]. Available: http://arxiv.org/abs/2205.08454

[6] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Proceedings - IEEE International Conference on Robotics and Automation*. Institute of Electrical and Electronics Engineers Inc., 1985, pp. 500–505.

[7] ——, "A Unified Approach for Motion and Force Control of Robot Manipulators: The Operational Space Formulation," *IEEE Journal on Robotics and Automation*, vol. 3, no. 1, pp. 43–53, 1987.

[8] F. Bullo and A. D. Lewis, *Geometric control of mechanical systems: modeling, analysis, and design for simple mechanical control systems*. Springer, 2019, vol. 49.

[9] N. D. Ratliff, J. Issac, D. Kappler, S. Birchfield, and D. Fox, "Riemannian motion policies," number: arXiv:1801.02854. [Online]. Available: http://arxiv.org/abs/1801.02854

[10] C.-A. Cheng, M. Mukadam, J. Issac, S. Birchfield, D. Fox, B. Boots, and N. Ratliff, "RMPflow: A computational graph for automatic motion policy generation," number: arXiv:1811.07049. [Online]. Available: http://arxiv.org/abs/1811.07049

[11] N. D. Ratliff, K. Van Wyk, M. Xie, A. Li, and M. A. Rana, "Optimization fabrics," number: arXiv:2008.02399. [Online]. Available: http://arxiv.org/abs/2008.02399

[12] A. Li, C.-A. Cheng, M. A. Rana, M. Xie, K. Van Wyk, N. Ratliff, and B. Boots, "RMP2: A structured composable policy class for robot learning," number: arXiv:2103.05922. [Online]. Available: http://arxiv.org/abs/2103.05922

[16] L. Yang and A. Shami, "On hyperparameter optimization of machine learning algorithms: Theory and practice," *Neurocomputing*, vol. 415, pp. 295–316, 2020.

[13] X. Meng, N. Ratliff, Y. Xiang, and D. Fox, "Neural autonomous navigation with riemannian motion policy," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8860–8866.

[14] A. Loquercio, A. Saviolo, and D. Scaramuzza, "Autotune: Controller tuning for high-speed flight," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4432–4439, 2022.

[15] W. Edwards, G. Tang, G. Mamakoukas, T. Murphey, and K. Hauser, "Automatic tuning for data-driven model predictive control," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 7379–7385. [Online]. Available: https://ieeexplore.ieee.org/document/9562025/

[17] F. Hutter, L. Kotthoff, and J. Vanschoren, Eds., *Automated Machine Learning: Methods, Systems, Challenges*, ser. The Springer Series on Challenges in Machine Learning. Springer International Publishing. [Online]. Available: http://link.springer.com/10.1007/978-3-030-05318-5

[18] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 2623–2631.

[19] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[20] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization." *Journal of machine learning research*, vol. 13, no. 2, 2012.

[21] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," *Advances in neural information processing systems*, vol. 24, 2011.

[22] C.-A. Cheng, M. Mukadam, J. Issac, S. Birchfield, D. Fox, B. Boots, and N. Ratliff, "RMPflow: A geometric framework for generation of multi-task motion policies," number: arXiv:2007.14256. [Online]. Available: http://arxiv.org/abs/2007.14256

[23] N. D. Ratliff, K. Van Wyk, M. Xie, A. Li, and M. A. Rana, "Generalized nonlinear and finsler geometry for robotics," number: arXiv:2010.14745. [Online]. Available: http://arxiv.org/abs/2010.14745

[24] R. Turner, D. Eriksson, M. McCourt, J. Kiili, E. Laaksonen, Z. Xu, and I. Guyon, "Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020," in *NeurIPS 2020 Competition and Demonstration Track*. PMLR, 2021, pp. 3–26.

[25] M. Spahn, "Urdf-Environment," https://github.com/maxspahn/gym_envs_urdf, 2022. [Online]. Available: https://github.com/maxspahn/gym_envs_urdf

[26] F. Chaumette, S. Hutchinson, and P. Corke, "Visual servoing," in *Springer Handbook of Robotics*. Springer, 2016, pp. 841–866.