2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

Global and Reactive Motion Generation with Geometric Fabric Command Sequences

Weiming Zhi[1,2,3], Iretiayo Akinola[1], Karl Van Wyk[1], Nathan D. Ratliff[1], Fabio Ramos[1,2]

2023 IEEE International Conference on Robotics and Automation (ICRA) | 979-8-3503-2365-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICRA48891.2023.10160965

Abstract—**Motion generation seeks to produce safe and feasible robot motion from start to goal. Various tools at different levels of granularity have been developed. On one extreme, sampling-based motion planners focus on completeness – a solution, if it exists, would eventually be found. However, produced paths are often of low quality, and contain superfluous motion. On the other, reactive methods optimise the immediate cost to obtain the next controls, producing smooth and legible motion that can quickly adapt to perturbations, uncertainties, and changes in the environment. However, reactive methods are highly local, and often produce motion that become trapped in non-convex regions of the environment. This paper contributes, *Geometric Fabric Command Sequences*, a method that lies in the middle ground. It can produce globally optimal motion that is smooth and intuitive, while being also reactive. We model motion via a reactive *Geometric Fabric* policy that ingests a sequence of attractor states, or *commands*, and then apply global optimisation over the space of commands. We postulate that solutions for different problems and scenes are highly transferable when conditioned on environmental features. Therefore, an implicit generative model is trained on solutions from optimisation and environment features in a self-supervised manner. That is, faced with multiple motion generation problems, the learning and optimisation are contained within the same loop: the optimisation generates labels for learning, while the learning improves the optimisation for the next problem, which in turn provides higher quality labels. We empirically validate our method in both simulation and on a real-world 6-DOF JACO arm.**

I. Introduction

Motion generation is a central problem in robotics. It is concerned with finding collision-free and executable trajectories from start to goal. Generating robot motions in dynamic environments is particularly challenging: the motion needs to be reactive to changes, and also be intuitive to any spectating humans. Recent reactive approaches to motion generation, such as Riemannian Motion Policies [1] and Geometric Fabrics [2], produce local policies to efficiently generate smooth and natural motions. These approaches produce a policy that find the instantaneous optimal controls, and allow for perturbations and deviations in the environment. However, these locally optimal solutions are often globally sub-optimal, resulting in robots getting stuck in local minima. Sampling-based planners, such as RRTs [3], can find global solutions, but often produce non-intuitive paths, and typically require re-planning to handle deviations and local changes in the environment.

In this work, we introduce *Geometric Fabric Command Sequences* (GFCS), a new approach to motion generation that produces globally feasible solutions while retaining local reactivity. In broad strokes, our method solves the global motion generating problem by optimising over a sequence of attractor states, which we call *commands*, for a *Geometric Fabric* policy [2]. Geometric Fabrics produce local motion by combining decoupled dynamical systems, which each represent an individual behaviour, such as goal reaching, obstacle avoidance, and joint limit handling. We propose to directly apply global optimisation over the commands. Furthermore, we hypothesise the solutions of the optimisation are transferable over different problem setups. We take a self-supervised approach and learn, over a dataset of motion generation problems and solutions, to warm-start the optimisation procedure. Specifically, we use an implicit generative model to progressively learn to recommend candidate solutions, as more problems are solved, with solutions of past optimisation runs used as training labels. This allows for faster and higher quality solutions as more problems are solved. Eventually, the implicit model itself can accurately generate optimal solutions, without further optimisation.

Concretely, we contribute: (1) *Geometric Fabric Command Sequences* (GFCS), a formulation of global and reactive motion generation as optimisation over sequentially connected commands, inputted to *Geometric Fabrics* [2]; (2) *Self-supervised GFCS*, a learning approach to warm-start the optimisation, using self-labelled data. Thus, improving the speed and quality of motion generation.

II. Related Work

Reactive Motion Generation: Our method is built atop reactive methods for motion generation. Reactive methods take into account the local environment and the current robot state to compute the optimal immediate next control. Methods in this category include [1, 2, 4, 5]. These approaches are able to react to changes in the environment, perturbations, and sensor uncertainty, as the instantaneous best control is found at every timestep. However, these methods can often get stuck in local minima when the workspace geometry is non-convex. **Probabilistic Complete Motion Planning:** Probabilistic Complete Motion Planning lies on the other end of the spectrum, where planners aim to find a path (represented as waypoints) from start to goal configuration, by iteratively sampling and maintaining a graph structure. As the number of sample points approaches infinity, the probability of finding a path, if it exists, tends to one. Representative approaches in this category include PRMs [6], RRTs [3], as well as subsequent variants such as RRT* [7], BIT [8]. Like our proposed method, these motion planners aim to find a globally optimal solution

Correspondence to: W. Zhi, wzhi@andrew.cmu.edu.
[1] NVIDIA, USA
[2] School of Computer Science, the University of Sydney, Australia
[3] Robotics Institute, Carnegie Mellon University, Pittsburgh, USA

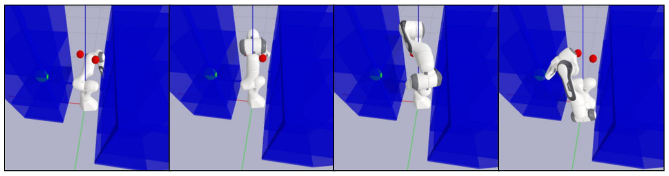
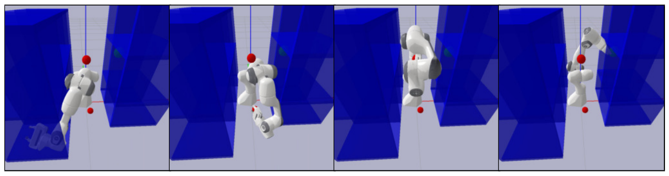



Fig. 1: Two examples of a manipulator sandwiched between two cupboards, reaching from one to the goal (green sphere) in another. The environment is tightly-packed. Local reactive methods fall into local minima, while our approach finds a sequence of *commands* to produce reactive motion to the goal. The end-effector coordinates of the command states are shown in red spheres.

rather than immediate optimal controls. However, methods in this category cannot reactively adapt to changes, and often produce non-smooth, non-intuitive motion with drastic swings. **Learning to Plan:** Learning approaches to aid planning have recently gained popularity. Machine learning models can be used to memorise solutions of planning, by training over a large dataset containing problem setups, and with corresponding labelled solutions. During inference, the learning model can speed up or improve planning by suggesting promising candidates to use in the downstream planning algorithm. Motion planning networks [9] condition on the problem to iteratively predict deterministic waypoints. Further work in trajectory or sample generation take a probabilistic approach and draw samples from mixture [10], normalising flow [11], or variational autoencoder models [12]. Similarly, our method also applies learning to aid motion generation. However, we tackle this in a self-supervised learning manner, where learning and motion generation is combined into the same loop. We apply an implicit generative model capable of flexibly modelling unnormalised multi-modal distributions.

III. Preliminaries: Geometric Fabrics

Geometric Fabrics (GFs) are the central building block of our method, and we retain the machinery of GFs. For completeness, here we give an intuitive introduction to GFs, with further details and theoretical analysis available in [2].

Fabric terms: Geometric fabrics (GFs) modularise robot motion by representing each individual local behaviour as position and velocity-dependent second order dynamical systems, referred to as *fabric terms*. Examples of these local behaviour include: accelerating towards a specified goal; reactively avoiding an obstacle by getting "repelled" away; avoiding the robot joint limits. Each of these behaviours are defined independently, and may be defined in different task spaces. For example, obstacle avoidance is defined with respect to body points on the robot, while joint limits are defined in the configuration space. These fabric terms can then be fused into a single system. We denote the position in some task space as $\mathbf{x}$, its velocity and acceleration as $\dot{\mathbf{x}}$, $\ddot{\mathbf{x}}$. We denote a configuration belonging to a d-dimensional C-space $\mathcal{Q} \subseteq \mathbb{R}^d$ as $\mathbf{q} \in \mathcal{Q}$, its time derivatives as $\dot{\mathbf{q}} \in \mathbb{R}^d$, $\ddot{\mathbf{q}} \in \mathbb{R}^d$. Formally, each fabric term, defining a specific behaviour, is defined by a pair $(\mathcal{L}_e, \pi)$, where $\mathcal{L}_e(\mathbf{x}, \dot{\mathbf{x}}) \to \mathbb{R}$ is a *Finsler energy* [13], $\ddot{\mathbf{x}} = \pi(\mathbf{x}, \dot{\mathbf{x}})$ is a policy, and $\mathbf{M} = \partial^2_{\dot{\mathbf{x}}\dot{\mathbf{x}}}\mathcal{L}_e$ is a priority metric. The policy is restricted to define a collection of velocity-independent paths. An $\texttt{Energize}_{\mathcal{L}_e}(\pi)$ operator (eq. 8 in [2]) is used to compute the execution velocity. We can then focus our efforts on designing pairs of $(\mathcal{L}_e, \pi)$ for each behaviour. Various design strategies can be found in Section C of [2]. Next, we aim to fuse these GF terms.

Fusion: Fabric terms defined in different task spaces can be fused, by first finding the configuration space coordinates of each term, in a *transform tree* [14] manner. Suppose, from each fabric term, we have a set of N_{ft} triples, $\{(\phi_i, \mathbf{M}_i, \ddot{\mathbf{x}}_i)\}_{i=1}^{N_{ft}}$, where ϕ_i maps a $\mathbf{q}$ to task space coordinates $\mathbf{x}_i$. The combined acceleration, $\ddot{\mathbf{q}}$, is given by: $\ddot{\mathbf{q}} = (\sum_{i=1}^{N_{ft}} \mathbf{J}_i^\top \mathbf{M}_i \mathbf{J}_i)^{-1} (\sum_{i=1}^{N_{ft}} \mathbf{J}_i^\top \mathbf{M}_i (\ddot{\mathbf{x}}_i - \dot{\mathbf{J}}_i \dot{\mathbf{q}}))$, where $\mathbf{J}_i$ is the Jacobian of ϕ_i, $\dot{\mathbf{J}}_i$ its time derivative.

IV. Self-supervised Learning of Geometric Fabric Command Sequences

We shall first formulate motion generation as a global optimisation over a sequence of *commands* (section IV-A). Then, we develop a generative model to *learn* to optimise (section IV-B), and a self-supervised learning framework that encompasses both optimisation and learning (section IV-C).

A. Global Motion Generation as Geometric Fabrics Command Optimisation

Let us consider how Geometric Fabrics can be used to generate locally optimal motion trajectories. A combined Geometric Fabric, $\ddot{\mathbf{q}} = f(\mathbf{q}, \dot{\mathbf{q}}, |\mathbf{q}_0, \dot{\mathbf{q}}_0, \mathbf{q}_g, E, C)$, can be viewed as a policy that maps from $\mathbf{q}$ and $\dot{\mathbf{q}}$ to $\ddot{\mathbf{q}}$. It is also dependent on problem-specific parameters: the start configuration $\mathbf{q}_0 \in \mathcal{Q}$ and velocity $\dot{\mathbf{q}}_0 \in \mathbb{R}^d$, goal $\mathbf{q}_g \in \mathcal{Q}$, environment E, and a set of robot-specific configuration parameters C. Trajectories can be obtained by a double integrator:

$$\begin{bmatrix} \dot{\mathbf{q}}_t \\ \mathbf{q}_t \end{bmatrix} = \int_0^t \begin{bmatrix} f(\mathbf{q}_u, \dot{\mathbf{q}}_u, |\mathbf{q}_0, \dot{\mathbf{q}}_0, \mathbf{q}_g, E, C) \\ \dot{\mathbf{q}}_u \end{bmatrix} \mathrm{d}u + \begin{bmatrix} \dot{\mathbf{q}}_0 \\ \mathbf{q}_0 \end{bmatrix}. \quad (1)$$

We define $\texttt{rollout}_\texttt{t}$ as an operation that finds the time steps used, $t^* \in \mathbb{R}^+$, to roll-out near the goal,

$$t^* = \texttt{rollout}_\texttt{t}(\mathbf{q}_0, \dot{\mathbf{q}}_0, \mathbf{q}_g, E, C) = \min\{\min(t), t_{max}\}, \quad (2)$$

$$\text{s.t. } ||\mathbf{q}_t - \mathbf{q}_g|| < \epsilon, \quad (3)$$

where ϵ denotes some tolerance. In practice $\mathbf{q}_t$ is obtained via numerical integration, such as Euler's method, of eq. (1), with a time step budget of $t_{max} \in \mathbb{R}^+$. Instead of defining a single goal, we inject global behaviour into the policy by stringing together a sequence of attraction states, or commands $\mathbf{q}_g^i$, while sharing local collision-avoidance and joint-limit handling fabric terms. We define the Geometric Fabrics Command Optimisation Problem (GFCOP), where we have a sequence of $n+1$ commands with different goal configurations, and optimise over the goals of the first n commands. The final command is specified as the final global goal. Specifically,

$$[\text{GFCOP}]: \min_{\mathbf{q}_g^1 \cdots \mathbf{q}_g^n} \sum_{i=1}^{n+1} t_i^* + p\mathbf{1} \tag{4}$$

$$\text{s.t. } t_i^* = \texttt{rollout}_\texttt{t}(\mathbf{q}_0^i, \dot{\mathbf{q}}_0^i, \mathbf{q}_g^i, E, C), \text{for } i=1,\ldots,n+1 \tag{5}$$

$$\mathbf{q}_0^1 = \mathbf{q}_0,\ \dot{\mathbf{q}}_0^1 = \dot{\mathbf{q}}_0,\ \mathbf{q}_g^n = \mathbf{q}_g, \tag{6}$$

$$\mathbf{q}_0^i = \mathbf{q}_{t_{i-1}^*}^{i-1},\ \dot{\mathbf{q}}_0^i = \dot{\mathbf{q}}_{t_{i-1}^*}^{i-1},\ \text{ for } i = 2, \ldots, n+1, \tag{7}$$

$$\mathbf{1} = \{0, \text{ if } t_{n+1}^* < t_{max}; 1, \text{ otherwise.}\} \tag{8}$$

where the cost (eq. (4)) is defined as the sum of the integration times exhausted until we reach our goal. Note that superscripts indicate the command index, while the subscript specifies the integration steps, i.e. $\mathbf{q}_0^i$ indicates the initial configuration at the i^{th} command. If the roll-out towards the final command in the sequence cannot reach our goal, then a large penalty of $p \in \mathbb{R}$ is applied. This is controlled by the indicator $\mathbf{1} \in \{0, 1\}$, as defined in eq. (8). Equation (6) designates that the start configuration and its velocity, matches given initial conditions $\mathbf{q}_0$ and $\dot{\mathbf{q}}_0$, and the final command is also the specified global goal, $\mathbf{q}_g$. Equation (7) enforces continuity, specifying that the initial conditions when integrating towards a command, shall be the terminating configuration and velocity of that of the previous. Constraints eqs. (5) to (7) are satisfied by running the roll-outs in a sequential manner, using the terminal conditions of the former integration as the initial conditions of the next.

The underlying local Geometric Fabric, given by f in eq. (1), performs local obstacle avoidance, self collision avoidance, joint coordinate and velocity limit handling. This allows the GFCOP to be concisely defined. Additionally, as local avoidance fabric terms are repulsion-based, motion slows down as the robot approaches obstacles, resulting in larger $\texttt{rollout}_\texttt{t}$ values. Therefore, solutions of GFCOP tend to smoothly move around obstacles, and not scrape past them.

The defined GFCOP is discontinuous and non-convex. We can apply the black-box optimiser *Covariance Matrix Adaptation Evolution Strategy* (CMA-ES) [15], which iteratively updates a multivariate Gaussian (MVG) sampling distribution over our command decision variables, $\mathbf{q}_g^1 \ldots \mathbf{q}_g^n$, after ranking the computed cost of previously evaluated candidates. The probability of CMA-ES drawing samples over the entire domain is non-zero, and has been shown to efficiently find globally optimal solutions [16, 17]. We denote each solution, $\mathbf{y}_i = [\mathbf{q}_g^{1\top}, \ldots, \mathbf{q}_g^{n\top}]_i^\top$, as a sequentially concatenated vector of the command goals. In each iteration, we sample a batch candidates from the MVG, compute the cost of the candidates in parallel, and then update the sampling distribution. Details of CMA-ES can be found in the tutorial [15]. Although CMA-ES finds globally optimal solutions, under a fixed time budget, the quality and speed of the optimisation is impacted by the initial batch of samples. Vanilla CMA-ES simply initialises a MVG based on random guesses of the mean and covariance, or simply sample uniformly. This motivates us to imbue prior knowledge into the optimisation and warm-start [18] the process by learning to sample.

B. Learning to Sample for Optimisation

We postulate that the optimal motions across problem setups with similar start, $\mathbf{q}_0$ $\dot{\mathbf{q}}_0$, goal, $\mathbf{q}_g$, and environments, E, are similar, and that we can *learn* to generate promising candidates from the optimisation solutions in alternative problem setups. We can use the generated candidates to warm-start CMA-ES, or directly use the candidate solution with the lowest cost as an approximate solution. Next, we define operations needed for the training, and outline the training, sampling and encoding used to construct the generative model.

Encoding a GFCOP: We require a fixed length context vector, $\mathbf{x}^c$, to condition on. The initial configuration, its velocity, and the goal are all fixed in size. We aim to find a fixed-sized encoding for the environment E. Here, we assume that each environment is represented by a point set $\mathcal{S} = \{\mathbf{s}_i\}_{i=1}^{N_E}$ of size N_E, where $\mathbf{s}_i \in \mathbb{R}^3$ are coordinate points in Euclidean space. This represents surfaces of objects, and can be obtained from an RGBD camera. The size of the point set N_E varies with each environment. We take a reference point-set approach [19] and compute the minimum Euclidean distance between the point set, and a set of fixed N_R reference coordinates $\widehat{\mathcal{S}} = \{\widehat{\mathbf{s}}_i\}_{i=1}^{N_R}$, $\mathbf{d}_E(\mathcal{S}, \widehat{\mathcal{S}}) := \big[\min_{\mathbf{s}\in\mathcal{S}} ||\mathbf{s} - \widehat{\mathbf{s}}_1||_2,\ \min_{\mathbf{s}\in\mathcal{S}} ||\mathbf{s} - \widehat{\mathbf{s}}_2||_2,\ \ldots,\ \min_{\mathbf{s}\in\mathcal{S}} ||\mathbf{s} - \widehat{\mathbf{s}}_{N_R}||_2\big]$. This provides us a fixed length (of length N_R) vector representation for environments with point sets of different sizes. The reference points are laid out in a lattice, with equal distance. If the size of the fixed length vector is excessively large, its dimensions are further reduced using an autoencoder, to obtain encoding $\widehat{\mathbf{d}}_E$. We concatenate $\mathbf{q}_0$, $\dot{\mathbf{q}}_0$, $\mathbf{q}_g$, and $\widehat{\mathbf{d}}_E$ to produce the input $\mathbf{x}^c$.

Solutions of GFCOP: Let us define the operation $\texttt{SolveGFCOP}(\mathbf{q}_0, \dot{\mathbf{q}}_0, \mathbf{q}_g, E | f_{sampler})$ to solve a GFCOP problem for a specific initial configuration, velocity, goal, environment, and return the top-m candidate solutions. A detailed algorithm is shown in algorithm 1. A sampler $f_{sampler}$ is passed on to provide the initial samples, and warm-start CMA-ES. We keep track of candidate solutions and their cost. For each call of SolveGFCOP, we obtain a set of $\{\mathbf{y}_i\}_{i=1}^m$ of m solutions, where $\mathbf{y}_i = [\mathbf{q}_g^{1\top}, \ldots, \mathbf{q}_g^{n\top}]_i^\top$, is the sequentially concatenated vector of the command goals. Note that the GFCOP is also dependent on other inputs, such as the length of the sequence of commands, $n+1$, as well as the robot-specific parameters, C. However, we shall drop the explicit dependence on these, as they are held constant during learning, i.e. learning is conducted over command sequences of the same length and on the same robot.

Training: Implicit energy-based models (EBMs) [20] excel at learning complex *unnormalised* distributions, where only sampling and not density estimation is performed using the model. Additionally, we can efficiently extract out a batch of candidate solutions from the EBM, as required for the warm-starting of CMA-ES. Here, we learn an energy function $E_\theta(\mathbf{x}^c, \mathbf{y}) \rightarrow \mathbb{R}$, mapping from a (context, target) pair to an energy value, by contrasting positive and negative samples, where positive samples have high energy and negative samples have low energy values. In our setup, the context is an encoding of the problem, while the target is the optimised command sequence. Suppose we have a dataset of positive examples $D^{pos} = \{\mathbf{x}_j^c, \mathbf{y}_j^*\}_{j=1}^{N_{pos}}$. Each $\mathbf{x}_j^c$ encodes the inputs to SolveGFCOP, and each $\mathbf{y}_j^*$ is outputted from SolveGFCOP. For each positive example $\mathbf{y}_j^*$, we actively generate N_{neg} negative examples $\{\widehat{\mathbf{y}}_j^i\}_{i=1}^{N_{neg}}$. Then, we can construct a InfoNCE-like loss [20, 21]:

$$\mathcal{L} = \sum_{j=1}^{N_{pos}} -\log \left\{ \frac{e^{-E_\theta(\mathbf{x}_j^c, \mathbf{y}_j^*)}}{e^{-E_\theta(\mathbf{x}_j^c, \mathbf{y}_j^*)} + \sum_{k=1}^{N_{neg}} e^{-E_\theta(\mathbf{x}_j^c, \widehat{\mathbf{y}}_j^k)}} \right\}. \quad (9)$$

The energy function E_θ is modelled by a neural network with parameters θ, which can be trained via stochastic gradient descent and its variants.

Generating Samples with Stochastic Gradient Langevin Dynamics: Stochastic gradient Langevin dynamics (SGLD) [22] is a technique to obtain a full posterior distribution from some energy function E_θ, rather than a single maximum *a posteriori* mode. This allows us to draw samples from the energy function to: (1) provide challenging negative examples to contrast against [23]; (2) warm-start CMA-ES. Given a trained energy function E_θ, and an context encoding $\mathbf{x}^c$, we randomly initialise n_p particles $\mathbf{y}_1^0, \ldots, \mathbf{y}_{n_p}^0 \sim \mathcal{U}(\mathbf{q}^{up}, \mathbf{q}^{low})$ from a uniform distribution, where $\mathbf{q}^{up}, \mathbf{q}^{low}$ are the upper and lower joint limits. The particles are updated according to:

$$\Delta \mathbf{y}_i^{k+1} = \mathbf{y}_i^k + \frac{\sigma_k^2}{2}(\nabla_{\mathbf{y}_i} E_\theta(\mathbf{x}^c, \mathbf{y}_i^k)) + \eta^k, \quad (10)$$

$$\text{where } \eta \sim \mathcal{N}(0, \sigma_k^2), \text{for } i = 0, \ldots, n_p. \quad (11)$$

Additionally, we decrease σ^2 according to the scheduler $\sigma_k^2 = (1+k)^{-0.5}$. After a maximum number of gradient ascent iterations has been exhausted, we take the updated particles and either select them as negative examples during training, or use them as solutions during inference.

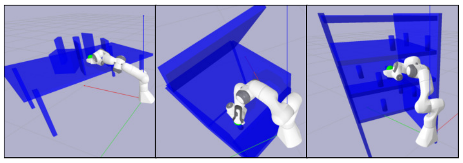
C. Self-supervision: Optimisation and Learning in the Loop

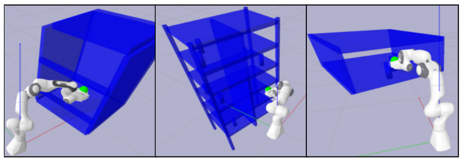
A key insight is that the optimisation outlined in section IV-A and the learning outlined in section IV-B are complimentary. Specifically, as more GFCOPs are solved, the more self-labelled training data is available for our implicit EBM. The EBM is trained by eq. (9), with negative samples generated from SGLD. On the other hand, as our EBM is progressively more well-trained, the quality of its generated candidate solutions also improve. This section brings both the optimisation and learning into a unified loop, and outlines a self-supervised learning framework. Following the definition in [24], in this context, self-supervision refers to the absence of external supervision signal (i.e. no human labelling) [24]. Algorithm 2 outlines the self-supervised GFCS learning framework. We are assumed to have a set of N_{prob} motion generation problems, $\{\mathbf{q}_0^n, \dot{\mathbf{q}}_0^n, \mathbf{q}_g^n, E^n\}_{n=1}^{N_{prob}}$. To self-generate data, we iteratively add solutions of SolveGFCOP to a buffer D^{buffer} of some maximum length; we train our EBM, E_θ, on the data in the buffer, after every batch, of size n_{train}, of problems have been solved; E_θ is then used as the sampler $f_{sampler}$ for future calls of SolveGFCOP.

Algorithm 1: SolveGFCOP with warm-started CMA-ES

```
input  : $\mathbf{q}_0$,$\dot{\mathbf{q}}_0$, $\mathbf{q}_g$, $E$, $f_{sampler}$, $C$, number of commands
         in sequence $n+1$, iterations $N_{iters}$, samples
         batch-size $N_{samp}$, $M$ large penalty for failure.
$\mathbf{y}_1, \ldots, \mathbf{y}_{N_{samp}} \sim f_{sampler}$ ;  // Draw candidate solutions from $f_{sampler}$ to warm-start
AllCandidatesWithCosts ← {} ;  // Empty set for all solutions with costs
for ( $j = 1$; $j <= N_{iters}$; $j = j + 1$ ) {
    BatchCandidatesWithCosts ← {} ;  // Empty set for solutions with costs
    ▷ Compute costs for each candidate in parallel;
    Do in Parallel for each ( $\mathbf{y}$ in $[\mathbf{y}_1, \ldots, \mathbf{y}_{N_{samp}}]$ ) {
        $\mathbf{q}_g^1, \ldots, \dot{\mathbf{q}}_g^n \leftarrow \mathbf{y}$ ;  // Unpack $\mathbf{y}$ to get candidate solution goals
        $\mathbf{q}_g^{n+1} \leftarrow \mathbf{q}_g$ ;  // Set command goal as the global goal
        $\mathbf{q}_0^1 \leftarrow \mathbf{q}_0$, $\dot{\mathbf{q}}_0^1 \leftarrow \dot{\mathbf{q}}_0$ ;  // Set Initial configuration and velocity
        Cost ← 0;  // Initialise the cost of a candidate solution
        for ( $i = 1$; $i <= n + 1$; $i = i + 1$ ) {
            $t_i^* \leftarrow$ rollout$_t$($\mathbf{q}_0^i, \dot{\mathbf{q}}_0^i, \mathbf{q}_g^i, E, C$) ;  // Roll-out the $i^{th}$ command in sequence
            Cost ← Cost + $t_i^*$
        }
        if $t^* < t_{max}$ then $\mathbb{1} \leftarrow 0$ else $\mathbb{1} \leftarrow 1$;
        Cost ← Cost + $M\mathbb{1}$ ;  // If we don't reach before time-out, apply penalty
        BatchCandidatesWithCosts.insert({$\mathbf{y}$, Cost});
        AllCandidatesWithCosts.insert({$\mathbf{y}$, Cost});
    }
    ▷ Update CMA-ES and sample the next batch;
    CMA-ES.update(BatchCandidatesWithCosts) ;  // Update CME-ES with new costs
    $\mathbf{y}_1, \ldots, \mathbf{y}_{N_{samp}} \sim$ CMA-ES.sampler();  // Sample next batch with CMA-ES sampler
}
top-m ← GetTopMByCost(AllCandidatesWithCosts) ;  // Sort by cost and get lowest $m$ candidates
output : top-m
```

Fig. 2: Examples problems of different classes from [25]


(a) Table under (b) Box (c) Tall shelf


(d) Cage (e) Thin shelf (f) Single shelf

V. Experimental Evaluation

We empirically investigate the performance of self-supervised GFCS to reliably generate global and reactive motion in various complex environments, both on a simulation FRANKA Panda and on a real-world JACO manipulator. We evaluate 6 different classes of benchmark problems from [25] (shown in fig. 2), each class contains 100 environments

Algorithm 2: Self-supervised Geometric Fabric Command Sequence Learning Framework

```
input : A set of $N_{prob}$ motion generation problems
        $\{\mathbf{q}_0^n, \dot{\mathbf{q}}_0^n, \mathbf{q}_g^n, E^n\}_{n=1}^{N_{prob}}$; an untrained implicit model
        $E_\theta(\mathbf{x}, \mathbf{y}) \to \mathbb{R}$, $N_{neg}$, $\mathbf{q}^{up}$, $\mathbf{q}^{low}$, $\mathcal{D}^{buffer}$.
$f_{sampler} \leftarrow \mathcal{U}(\mathbf{q}^{up}, \mathbf{q}^{low})$ ;   // Uniformly initialise sampler
for ( $n = 1$; $n <= N_{prob}$; $n = n + 1$ ) {
    ▷ Self-generate positive examples via optimisation
    $\mathcal{D}^{pos} \leftarrow$ SolveGFCOP($\mathbf{q}_0^n, \dot{\mathbf{q}}_0^n, \mathbf{q}_g^n, E^n | f_{sampler}$) ;
      // Solve for top-$m$ solutions
    $\mathcal{D}^{buffer}$.insert($\mathcal{D}^{pos}$) ;   // Add positive data into buffer
    $\mathcal{D}^{buffer} \leftarrow$ MaintainBuffer($\mathcal{D}^{buffer}$) ;   // Maintain the
      length of buffer
    $\mathcal{D}^{neg} \leftarrow \{\}$ ;   // Initialise empty set for negatives
    for ( $j = 1$; $j <=$ length($\mathcal{D}^{buffer}$); $j = j + 1$ ) {
        $\mathbf{y}_1^0, \ldots, \mathbf{y}_{N_{neg}}^0 \sim \mathcal{U}(\mathbf{q}^{up}, \mathbf{q}^{low})$;
        $\{\hat{\mathbf{y}}_j^i\}_{i=1}^{N_{neg}} \leftarrow$ SGLD($\mathbf{y}_1^0, \ldots, \mathbf{y}_{N_{neg}}^0 | E_\theta$) ;   // Run
          SGLD using eq. (11) on $E_\theta$
        $\mathcal{D}^{neg}$.insert($\{\hat{\mathbf{y}}_j^i\}_{i=1}^{N_{neg}}$) ; // Insert into set of negatives
    }
    $\mathbf{x} \leftarrow$ encode($\mathbf{q}_0^n, \dot{\mathbf{q}}_0^n, \mathbf{q}_g^n, E^n$) ;   // Encode problem.
    ▷ Train on self-generated data, and update sampler
    if $n \% n_{train} == 0$ then
        $E_\theta$.train($\mathcal{D}^{buffer}, \mathcal{D}^{neg}$) ;   // train $E_\theta$ with eq. (9),
          after $n_{train}$ solves
    end
    $f_{sampler} \leftarrow$ SGLD($\mathbf{y}_1^0, \mathbf{y}_2^0 \ldots | E_\theta$),  where
    $\mathbf{y}_1^0, \mathbf{y}_2^0 \ldots \sim \mathcal{U}(\mathbf{q}^{up}, \mathbf{q}^{low})$ ;   // Update the sampler as
      SGLD to the trained $E_\theta$
}
output: Trained $E_\theta$
```

with different start/goals. For each class, we use 50 problems for self-supervised training, and report our evaluation on the remaining 50 test problems. The Geometric Fabric terms follow the original implementation from [2]. When rolling out trajectories, we set the time budget per command, t_{max}, to 7s, and integrate with step-size 0.01. We iterate through all the training problems twice, and add the top-4 solutions into the buffer for training, and the buffer is maintained to have 200 solutions. Training on solutions occurs after every 25 problems have been solved, that is $n_{train} = 25$. During training, to encode the environment, we lay out fixed reference point in intervals of 0.15. The neural network used to approximate E_θ contains 3 hidden layers of width 800, with ReLU activations. We use a machine with an Intel Core i7 CPU, RTX3080 Ti GPU and 32 GB of RAM to conduct experiments.

We report on the following metrics: *% Success*: the percentage of problems a solution is found; *Times*: the time taken to generate the trajectories; *EE length*: the length of the end-effector trajectory; *C-space length*: the length of the C-space trajectory; *EE-LDJ*: The log-dimensionless jerk (LDJ) [26] of the end-effector trajectory; *C-space LDJ*: The LDJ [26] of the C-space trajectory. *EE length* reflects trajectory legibility, with a low length indicating that little superfluous motion is present. LDJ reflects the trajectory smoothness, where closer to zero indicates a smoother trajectory. Additional videos of the experiments can be found in the supplementary video.

Global Solutions and Motion Trajectory Quality: We hypothesise that our method, like local motion generation approaches, is able to generate legible motion trajectories, without superfluous end-effector motion, while capable of handling non-convexities in the task space to generate globally feasible motion. We observe that a GFCS that is very short is sufficient to generate complex non-local behaviour. We compare GFCS, of length two, against: (1) *Geometric Fabrics (GFs)*, a reactive method capable of producing high quality motion trajectories; (2) Bi-directional Expansive Space Trees (Bi-EST) [27], a bi-direction sampling-based motion planner (3) Batch Informed Trees (BIT*) [8], an asymptotically optimal sampling-based planner. We use the Bi-EST and BIT* implementations in the Open Motion Planning Library (OMPL) [28]. As our method can continuously optimise and refine the solution, we evaluate our method after the first and fifth batches of samples have been evaluated. Likewise, BIT* is asymptotically optimal and will continuously refine the solution. We evaluate BIT* with a budget of 1.2 and 6 seconds. This gives BIT* at least as much time our method takes to evaluate the first and fifth batch.

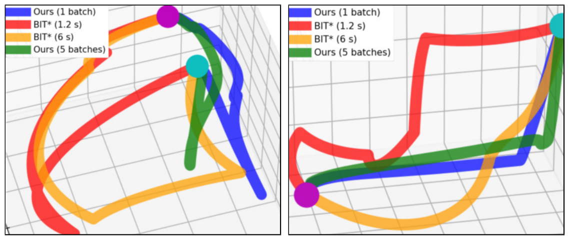


Fig. 3: Examples of end-effector trajectories, from start (cyan) to goal (magenta), in “Table under” (Left) and “Tall shelf” (Right). GFCS consistently produces shorter and smoother motion. BIT*(1.2s) given in red, BIT*(6s) in yellow, ours (1 batch) in blue, and ours (5 batch) in green.

We tabulate the results over the six classes of benchmark environments in table I. We observe that the local GF approach fails to solve many of the problems, particularly in the “table under” class, where the manipulator is tasked with starting from under a table and reaching a pose on a table. Completing such a task requires the manipulator to exhibit global behaviour and escape from a local minimum. On the other hand, GFCS are able to generate motion which escape from local minima. Bi-EST can typically find feasible solutions quickly, but produce undesirable trajectories, as measured by our metrics of trajectory quality. We also observe that the “cage” problems require the manipulator to traverse a narrow passage, through the bars of a cage, to reach inside. Sampling-based motion planning methods struggle at this: Bi-ESTs become inefficient are slower than local approaches to find a feasible trajectory, while BIT* simply cannot find a feasible solution for most of the “cage” problems. Whereas the reactive GF and GFCS can efficiently find the solutions. Additionally, GFs and GFCS produce trajectories with shorter and smoother end-effector motion than sampling-based planners. End-effector trajectories in two example problems are shown in fig. 3.

The Effect of Learning: Our implicit generative model learns to produce good samples for GFCS to optimise over. Here, we investigate the performance improvements provided by our learnt model. We observe that the learned generative model is able to suggest high quality candidate to the

TABLE I: Evaluated metrics (mean $\pm$ std) for motion generated in multiple benchmark environments

(a) Evaluation in the "Table Under" problems

	% Success	EE len	C-space len	Times (s)	EE-LDJ	C-space-LDJ
GF	0	NA	NA	NA	NA	NA
Bi-EST	100	8.24 ± 3.2	23.69 ± 9.2	0.028 ± 0.04	-11.09 ± 1.7	-10.58 ± 1.3
BIT* (1.2s)	100	3.93 ± 0.9	7.71 ± 2.8	NA	-7.86 ± 1.3	-7.34 ± 0.9
BIT* (6s)	100	3.71 ± 1.0	7.12 ± 2.8	NA	-7.95 ± 1.0	-7.13 ± 1.1
Ours (1 b)	96	1.43 ± 0.5	9.35 ± 2.4	0.96 ± 0.05	-7.45 ± 1.8	-7.51 ± 2.0
Ours (5 b)	98	1.32 ± 0.3	8.65 ± 2.0	5.29 ± 0.4	-7.05 ± 1.6	-7.09 ± 1.8

(b) Evaluation in the "Box" problems

	% Success	EE len	C-space len	Times (s)	EE-LDJ	C-space-LDJ
GF	100	0.92 ± 0.07	5.25 ± 0.6	0.066 ± 0.01	-5.32 ± 0.9	-5.57 ± 1.5
Bi-EST	100	5.52 ± 3.3	19.68 ± 9.3	0.030 ± 0.03	-10.55 ± 1.8	-10.36 ± 1.3
BIT* (1.2s)	92	1.97 ± 0.7	8.21 ± 2.2	NA	-8.99 ± 1.1	-8.30 ± 0.8
BIT* (6s)	100	1.84 ± 0.6	7.39 ± 2.3	NA	-8.69 ± 1.0	-7.99 ± 1.0
Ours (1 b)	100	1.00 ± 0.1	5.43 ± 0.8	0.74 ± 0.03	-5.84 ± 1.1	-5.55 ± 1.5
Ours (5 b)	100	0.97 ± 0.1	5.23 ± 0.6	4.01 ± 0.2	-5.89 ± 1.6	-5.40 ± 1.6

(c) Evaluation in the "Tall shelf" problems

	% Success	EE len	C-space len	Times (s)	EE-LDJ	C-space-LDJ
GF	88	0.700 ± 0.2	6.51 ± 1.1	0.1 ± 0.03	-4.97 ± 1.3	-6.03 ± 1.9
Bi-EST	100	4.57 ± 2.6	14.01 ± 5.8	0.036 ± 0.08	-9.42 ± 2.2	-9.37 ± 1.5
BIT* (1.2s)	90	1.73 ± 0.7	5.78 ± 1.2	NA	-6.80 ± 2.4	-5.63 ± 2.4
BIT* (6s)	94	1.60 ± 0.6	5.72 ± 1.3	NA	-6.94 ± 2.4	-5.75 ± 2.5
Ours (1 b)	100	1.02 ± 0.3	5.89 ± 1.0	0.75 ± 0.04	-6.39 ± 1.3	-6.02 ± 1.2
Ours (5 b)	100	1.00 ± 0.2	5.74 ± 0.9	3.69 ± 0.25	-6.19 ± 1.4	-5.76 ± 1.3

(d) Evaluation in the "Cage" problems

	% Success	EE len	C-space len	Times (s)	EE-LDJ	C-space-LDJ
GF	88	0.66 ± 0.1	7.87 ± 0.8	0.16 ± 0.02	-5.3 ± 1.4	-7.02 ± 0.8
Bi-EST	100	10.67 ± 5.2	37.3 ± 15.4	1.99 ± 2.1	-12.57 ± 1.4	-12.02 ± 1.4
BIT* (1.2s)	2	NA	NA	NA	NA	NA
BIT* (6s)	2	NA	NA	NA	NA	NA
Ours (1 b)	98	0.75 ± 0.1	6.91 ± 0.9	0.82 ± 0.04	-7.27 ± 2.2	-7.97 ± 1.7
Ours (5 b)	100	0.73 ± 0.1	6.31 ± 0.9	4.06 ± 0.2	-7.46 ± 1.7	-7.94 ± 1.4

(e) Evaluation in the "Thin shelf" problems

	% Success	EE len	C-space len	Times (s)	EE-LDJ	C-space-LDJ
GF	80	0.68 ± 0.2	6.49 ± 1.4	0.11 ± 0.02	-6.06 ± 2.3	-6.56 ± 1.2
Bi-EST	100	5.11 ± 2.4	16.50 ± 7.1	0.033 ± 0.04	-9.89 ± 1.7	-9.86 ± 1.5
BIT* (1.2s)	86	1.89 ± 0.7	6.16 ± 1.2	NA	-8.04 ± 1.4	-6.79 ± 1.3
BIT* (6s)	92	1.66 ± 0.6	6.01 ± 1.1	NA	-8.51 ± 1.2	-6.70 ± 1.1
Ours (1 b)	100	0.93 ± 0.2	5.64 ± 0.8	0.82 ± 0.04	-7.32 ± 1.7	-7.05 ± 1.4
Ours (5 b)	100	0.92 ± 0.2	5.38 ± 0.8	4.33 ± 0.4	-7.38 ± 1.4	-6.68 ± 1.4

(f) Evaluation in the "Single shelf" problems

	% Success	EE len	C-space len	Times (s)	EE-LDJ	C-space-LDJ
GF	94	0.69 ± 0.2	6.51 ± 1.46	0.084 ± 0.02	-5.11 ± 1.8	-5.78 ± 1.4
Bi-EST	100	5.64 ± 2.8	16.26 ± 7.2	0.066 ± 0.13	-10.04 ± 1.8	-9.89 ± 1.6
BIT* (1.2s)	86	1.51 ± 0.6	5.79 ± 1.7	NA	-6.75 ± 2.8	-5.38 ± 2.7
BIT* (6s)	94	1.61 ± 0.8	5.48 ± 1.5	NA	-6.88 ± 2.9	-5.26 ± 2.7
Ours (1 b)	100	0.90 ± 0.2	5.79 ± 1.2	0.73 ± 0.03	-6.97 ± 1.4	-6.65 ± 1.3
Ours (5 b)	100	0.91 ± 0.2	5.68 ± 1.1	3.62 ± 0.3	-7.03 ± 1.5	-6.64 ± 1.2

TABLE II: First batch motion by uniformly sampling, against by a learned model, on "Table under" problems.

	% Success	EE len	C-space len	EE-LDJ	C-space-LDJ
No learning	90	1.85 ± 0.5	13.53 ± 2.9	-8.61 ± 1.6	-8.76 ± 1.4
Learning	**96**	$\mathbf{1.43 \pm 0.5}$	$\mathbf{9.35 \pm 2.4}$	$\mathbf{-7.45 \pm 1.8}$	$\mathbf{-7.51 \pm 2.0}$

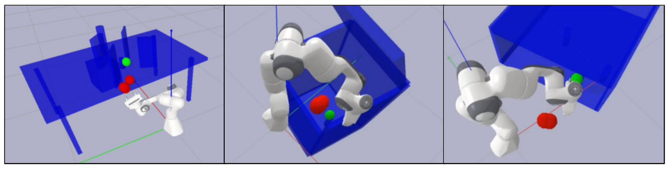


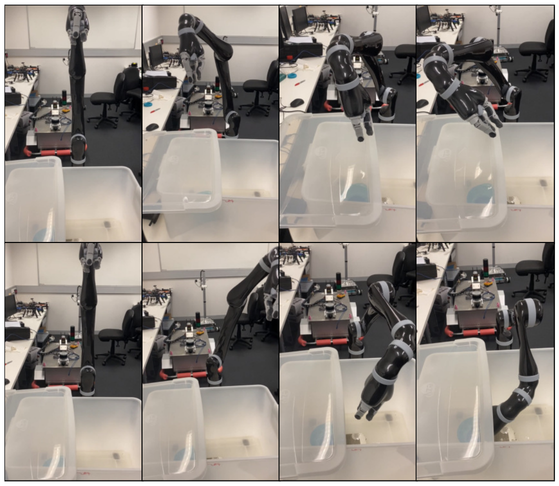
Fig. 4: After obtaining a solution to the goal (in green), we add new obstacles (in red), GFCS reactively avoid the new obstacles without re-optimising. Examples shown in supplementary video.

global optimiser. We investigate the performance of our learnt generative model compared against drawing from a uniform distribution, on "table under" problems, which in particular requires global solutions. The results in Table II show that using the learned sampler gives rises to more successful solutions, and higher quality motion trajectories.

Reactive Motion Generation: Geometric Fabric Sequences inherit reactive behaviour from Geometric Fabrics, while finding global solutions. Specifically, after commands in the GFCS has been optimised, upon encountering previously unseen obstacles, local collision avoidance fabric terms will execute local avoidance. We can study this property on the "table under", "box", and "single shelf" problem setups. We randomly generate new obstacles, blocking the previously solved motion trajectory, and check whether the motion can instantaneously adapt to the new obstacles. Examples are illustrated in fig. 4, and shown in the supplementary video. We observe that GFCS can consistently generate collision-free trajectories which avoids the new obstacles.

Execution on a Real Robot: To evaluate the robustness of global motion found from GFCS, we generate motion on a real-world JACO manipulator to reach a goal in a half-covered box. Qualitative results from pure GF and GFCS are shown in fig. 5, where local avoidance by GF is insufficient to reach the goal, and the manipulator stabilises on the cover. GFCS avoids the local minimum, and reaches into the box towards the goal. Results shown in supplementary video.

Fig. 5: Geometric Fabrics (4 figs on the top) perform local avoidance and fails to reach the blue goal. GFCS (4 figs on the bottom) performs global optimisation and finds a non-local solution.



VI. Conclusions

We present Geometric Fabric Command Sequences, a novel approach to generate global and reactive motion. This is achieved by running global optimisation over selected parameters of sequentially joined command parameters for *Geometric Fabrics*. The optimisation can then be sped-up and improved by transferring knowledge from solving similar problems. We introduce a self-supervised framework, where an implicit generative model iteratively learns to provide informed candidate solutions to the optimisation. We validate our approach on a range of problem classes, both in simulation and on a real-world JACO manipulator.

References

[1] N. D. Ratliff, J. Issac, D. Kappler, S. Birchfield, and D. Fox, “Riemannian motion policies,” 2018.

[2] 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,” *IEEE Robotics and Automation Letters*, 2022.

[3] S. LaValle and J. Kuffner, “Randomized kinodynamic planning,” in *Proceedings 1999 IEEE International Conference on Robotics and Automation*, 1999.

[4] O. Khatib, “Real-time obstacle avoidance for manipulators and mobile robots,” in *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, 1985.

[5] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, “Dynamical movement primitives: Learning attractor models for motor behaviors,” *Neural Computation*, 2013.

[6] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *IEEE Transactions on Robotics and Automation*, 1996.

[7] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *The International Journal of Robotics Research*, 2011.

[8] J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa, “Batch informed trees (bit*): Informed asymptotically optimal anytime search,” *The International Journal of Robotics Research*, vol. 39, no. 5, 2020.

[9] A. H. Qureshi, M. J. Bency, and M. C. Yip, “Motion planning networks,” *2019 International Conference on Robotics and Automation (ICRA)*, 2019.

[10] W. Zhi, T. Lai, L. Ott, and F. Ramos, “Trajectory generation in new environments from past experiences,” in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

[11] T. Lai, W. Zhi, T. Hermans, and F. Ramos, “Parallelised diffeomorphic sampling-based motion planning,” in *Conference on Robot Learning (CoRL)*, 2021.

[12] B. Ichter, J. Harrison, and M. Pavone, “Learning sampling distributions for robot motion planning,” *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[13] N. D. Ratliff, K. V. Wyk, M. Xie, A. Li, and M. A. Rana, “Generalized nonlinear and finsler geometry for robotics,” *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[14] C.-A. Cheng, M. Mukadam, J. Issac, S. Birchfield, D. Fox, B. Boots, and N. D. Ratliff *ArXiv*, vol. abs/1811.07049, 2018.

[15] N. Hansen, “The CMA evolution strategy: A tutorial,” *CoRR*, 2016.

[16] N. Hansen and S. Kern, “Evaluating the cma evolution strategy on multimodal test functions,” 01 2004.

[17] N. Hansen, A. Auger, R. Ros, S. Finck, and P. Pošík, “Comparing results of 31 algorithms from the black-box optimization benchmarking bbob-2009,” The Genetic and Evolutionary Computation Conference, 2010.

[18] M. Nomura, S. Watanabe, Y. Akimoto, Y. Ozaki, and M. Onishi, “Warm starting cma-es for hyperparameter optimization,” in *The AAAI Conference on Artificial Intelligence*, 2021.

[19] Z. Yang, S. Liu, H. Hu, L. Wang, and S. Lin, “Reppoints: Point set representation for object detection,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[20] P. Florence, C. Lynch, A. Zeng, O. Ramirez, A. Wahid, L. Downs, A. Wong, J. Lee, I. Mordatch, and J. Tompson, “Implicit behavioral cloning,” in *Conference on Robot Learning (CoRL)*, 2021.

[21] A. van den Oord, Y. Li, and O. Vinyals, “Representation learning with contrastive predictive coding,” *CoRR*, 2018.

[22] M. Welling and Y. W. Teh, “Bayesian learning via stochastic gradient langevin dynamics,” in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, 2011.

[23] Y. Du and I. Mordatch, “Implicit generation and modeling with energy based models,” in *Advances in Neural Information Processing Systems*, 2019.

[24] M. Nava, J. Guzzi, R. O. Chavez-Garcia, L. M. Gambardella, and A. Giusti, “Learning long-range perception using self-supervision from short-range sensors and odometry,” *IEEE Robotics and Automation Letters*, 2019.

[25] C. Chamzas, C. Quintero-Peña, Z. K. Kingston, A. Orthey, D. Rakita, M. Gleicher, M. Toussaint, and L. E. Kavraki, “Motionbenchmaker: A tool to generate and benchmark motion planning datasets,” *IEEE Robotics and Automation Letters*, 2022.

[26] N. Hogan and D. Sternad, “Sensitivity of smoothness measures to movement duration, amplitude, and arrests,” *Journal of Motor Behavior*, vol. 41, no. 6, 2009.

[27] D. Hsu, J.-C. Latombe, and R. Motwani, “Path planning in expansive configuration spaces,” in *Proceedings of International Conference on Robotics and Automation*, 1997.

[28] I. A. Şucan, M. Moll, and L. E. Kavraki, “The Open Motion Planning Library,” *IEEE Robotics & Automation Magazine*, pp. 72–82, December 2012.