# Learning Advanced Manipulation Skills on the SE(3) Manifold with Stability Guarantees

**Erlernen fortgeschrittener Manipulationsfähigkeiten auf Basis von Lie-Gruppen mit Stabilitätsgarantie**

Bachelor thesis in the department of Computer Science by Kilian Feess
Date of submission: July 18, 2024

1. Review: Ph.D Davide Tateo
2. Review: M.Sc Junning Huang
3. Review: Prof. Ph.D Jan Peters

Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

Computer Science Department

## Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 APB TU Darmstadt

Hiermit erkläre ich, Kilian Feess, dass ich die vorliegende Arbeit gemäß § 22 Abs. 7 APB der TU Darmstadt selbstständig, ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Ich habe mit Ausnahme der zitierten Literatur und anderer in der Arbeit genannter Quellen keine fremden Hilfsmittel benutzt. Die von mir bei der Anfertigung dieser wissenschaftlichen Arbeit wörtlich oder inhaltlich benutzte Literatur und alle anderen Quellen habe ich im Text deutlich gekennzeichnet und gesondert aufgeführt. Dies gilt auch für Quellen oder Hilfsmittel aus dem Internet.

Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 18. Juli 2024

K. Feess

# Abstract

Disaster scenarios like nuclear reactor accidents, bomb defusal, or accidents at chemical facilities often disallow on-site inspection by humans. In the last decades, robotics have moved into the spotlight to perform manipulation and navigation tasks in these environments. Currently, robots are controlled remotely most of the time, putting pressure on the operators who are controlling the robot in these stressful conditions. Autonomous assistance functions that help with navigation and environment manipulation are useful since they allow the operator to focus on crucial mission objectives. Task-based autonomous support of this kind is only sparsely available on a few robotics platforms. Since movements in disastrous environments carry innate risks, these control policies need to guarantee stability, and the robot needs to converge to the nominal motion even when external environmental disturbances are present.

We extend the LieFlows method which uses stable vector fields on Lie groups to represent robot motions. We introduce two new motion types that we embed into the method, to learn two new tasks commonly needed in these emergency response settings. These motion types are developed to represent precise positioning and cyclic motions, we use them for opening doors and swiping a swab on a table to take a sample. The trained policies are executed and evaluated in simulation. We integrate the swiping policy into a graphical user interface for easy access by teleoperators and deploy it on the real robot.

# Zusammenfassung

Katastrophenszenarien wie Kernreaktorunfälle, Bombenentschärfung oder Unfälle in Chemieanlagen lassen eine Inspektion vor Ort durch Menschen oft nicht zu. In den letzten Jahrzehnten ist die Robotik ins Rampenlicht gerückt, um Manipulations- und Navigationsaufgaben in solchen Umgebungen zu übernehmen. Gegenwärtig werden Roboter meistens ferngesteuert. Das erhöht den Druck auf die Bediener, die den Roboter unter diesen stressvollen Bedingungen steuern. Autonome Assistenzfunktionen, die bei der Navigation und der Umgebungsmanipulation helfen, sind nützlich, da sie es dem Bediener ermöglichen, sich auf die wichtigsten Missionsziele zu konzentrieren, sind allerdings nur auf wenigen Robotikplattformen verfügbar. Da Robotereinsatz in diesen Katastrophenszenarien riskant ist, müssen autonome Steuerungsmethoden Stabilität garantieren, und der Roboter muss selbst bei externen Störungen aus der Umgebung zur nominalen Bewegung zurückkehren.

Wir erweitern die LieFlows-Methode, die stabile Vektorfelder auf Lie-Gruppen als Repräsentation für Roboterbewegungen verwendet. Wir führen zwei neue Bewegungstypen ein, die wir in die Methode einbetten. Damit trainieren wir zwei neue Bewegungsroutinen, die in diesen Notfallsituationen häufig benötigt werden. Diese Bewegungstypen modellieren präzise Positionierung und die Ausführung zyklischer Bewegungen. Wir wenden sie an um das Öffnen von Türen und das Wischen eines Tupfers auf einem Tisch, um eine Probe zu entnehmen zu erlernen.

Die trainierten Strategien werden in Simulation ausgeführt und ausgewertet. Wir integrieren das Kontrollschema zur Probenentnahme durch wischen in eine grafische Benutzeroberfläche, um den Teleoperatoren die Anwendung zu erleichtern und führen die steuerung auf dem Roboter aus.

# Contents

# 1 Introduction

Teleoperated robots are used in scenarios where human access to a site is deemed too dangerous. These can be rescue missions, explosive ordnance disposal, or inspections after accidents at industrial facilities or power plants. Robotics have been used in nuclear disasters, at least since the Chernobyl nuclear reactor catastrophe of 1986 [2]. Robot operators have to deal with complex problems in these situations. While working on the main objective, like radiation source locating, they have to overcome obstacles and do routine tasks that contribute to the exhaustion of the operator.

Autonomous or semi-autonomous assistance functions can reduce the operator's workload by automating repeating movement tasks. These assistance functions can be the automation of navigation tasks or locally defined manipulation procedures like taking a sample by picking up objects or swiping a surface with a swab. Currently, advanced robotic control policies that take care of these tasks are, at best, partially available on the robots used in these scenarios. One notable example of such an assistance function is the Spot robot by Boston Dynamics, which includes a routine to open doors[6]. The robotics systems typically used by special task forces are usually explicitly designed for teleoperation. They often need these advanced assistance features. The Telemax robot, a prominent model used in Germany, shows the capabilities of robotic technology in these scenarios. Manufactured by Telerob, this robot is highly appreciated for its manipulation tasks, thanks to its robust and precise arm attached to a stable base. Its performance in these tasks instills confidence in the potential of robotic assistance functions. We use this robot to test the application of our method in realistic scenarios.

Since these robots often operate in very hazardous environments, limiting the risk of further deterioration of the situation is necessary. On the one hand, the robot should not worsen the contamination by changing the environment, for example, by knocking over containers, on the other hand, the robot should stay operable so it does not block the way for further missions. This necessitates stable control policies, which make the execution of autonomous tasks predictable. Since these movements are carried out in a

three-dimensional cartesian environment where orientation is essential, representations that include position and orientation into a single pose are needed. Control policies need to be defined concerning this representation, and their execution must be stable. Therefore, representing the task space in the $\mathrm{SE}(3)$ manifold satisfies the requirements of the pose, and the stable control policy is defined by a stable vector field defined on this manifold.

Defining these manipulation tasks with a handcrafted algorithm is difficult. The algorithmic representation must be adapted and hand-tuned from scratch by an expert for each task. Representing these movements may require complex logic and tracking of the robot's state to decide which motion to execute. Guaranteeing stability of the motion is difficult in this context. Machine learning enables data-driven execution of motor skills that supersede the need for algorithmic movement definition. Behavioral cloning, a machine learning technique that relies on the invaluable expertise of human demonstrations, uses these demonstrations as the basis to learn actions to execute depending on the current state of the robot, resulting in a control policy that can be applied to robot motion. Since complex sequences of movements can be hard and laborious to define programmatically, imitation learning simplifies capturing the essence of a movement into a computable policy. Inducing domain-specific structure into the learning process can ensure stability and significantly aid the generalization of the model. LieFlows is a method that guarantees stable robot policies using a vector field.

In contrast to movement and locomotion tasks, the barrier of entry for the practical deployment of autonomous assistance functions is lower for manipulation tasks that are temporally and spatially contained. These functions can be exposed as discrete triggerable modules to the operator of the robot, providing a flexible system where the manipulation can still be done by hand using the joystick interface if automatic execution fails. Nevertheless, having access to these assistance functions is useful anyway. Not every operator is able to do these tasks, so in addition to saving time, assistance functions make it possible for less skilled users to navigate uncertain environments.

LieFlows defines a vector field in task space used with an operational space controller. Defining the action in task space is useful, as it allows us to acquire demonstrations of the task space representation. This is advantageous since this movement description is independent of any specific robot or kinematic chain. Deployment profits from this, too. Trained skills can be re-used plug-and-play by running the task space policy on a different robot using a robot-specific operational space or nullspace control scheme. The fact that the movements are independent of a particular robot enables the demonstration


Figure 1.1: ANYmal by ANYbotics (left) and Telemax by Telerob (right)
Picture taken at the training scenario from Kerntechnische Hilfsdienst GmbH during the final KIARA project meeting

generation by humans using motion-tracking markers. Any other robot can also capture demonstration trajectories.

Robots like the telemax deployed for exploration and manipulation in a training setting can also capture examples. A skilled operator controlling the robot by teleoperation can also generate demonstrations. The resulting data set can be used to train an autonomous skill, making demonstration acquisition particularly easy.

For the training and operation of our skills, we assume sufficiently precise perception is available and focus on representing the pure manipulation task. The tasks are learned as stable vector fields, which provide stability guarantees, inherent reactivity, and good generalization.

## 1.1 Goal of the Thesis

In this thesis we extend the LieFlows method to find a good policy architecture for precise movements and cyclic motions, we describe LieFlows in chapter 3. LieFlows is based on representing movement by stable vector fields that are created by deforming a movement primitive in a latent space to the task space by a diffeomorphic mapping that is based on Lie groups. We apply this imitation learning scheme to two different manipulation tasks that are common in typical search and rescue scenarios where robots are deployed to learn these manipulation tasks from data.

First, we learn a movement to take a sample on a surface by swiping in an oval planar shape. We provide a simple user interface to the operator so that sample-taking can be automated by clicking on a surface inside the 3D representation of the robot's environment. The robot will then execute the trained skill in an automated way. Since this type of movement requires a steady height above the surface, which is difficult to maintain by manual control, executing the task with an autonomic routine speeds up taking samples from the environment significantly.

The second skill we train addresses the task of automated door opening. Robot operators typically take around 25 minutes[1] to open a door manually. This shows, that opening a door with a robot remotely is extremely difficult. It is apparent that automation can greatly reduce the time needed to execute these tasks as well as spare the users from executing those movements manually by using the joystick input.

In order to enable learning to open doors, we have to enable motions that allow positioning the robot to a target pose precisely. To learn swiping, we extend the method to use limit cycles as movement primitives. We parametrize the movement primitives by neural networks predicting the movement speeds. Those two movement types are described in the methodology section (chapter 4). In chapter 5 we apply these learned tasks to the Telemax robot in simulation, as well as on the real system.

[1] I learned this from the teleoperators using remote-controlled robots in nuclear accidents to navigate and observe contaminated spaces

# 2 Related Work

Currently, the focus for manipulation tasks has shifted from classical trajectory descriptions to methods driven by machine learning. Imitation learning is a class of techniques used to directly learn a policy from demonstration movements, enabling the utilization of expert demonstrations to describe a task[19]. Humans quickly learn to execute movements based on simple observations, suggesting using a similar approach for machine learning. In recent years several different imitation learning techniques have emerged. One of the most basic techniques is behavioral cloning[3], which directly learns the policy by state action pairs using supervised training. Since the policy is simply trained on the demonstration data, the performance of the trained policy can suffer from a range of problems. It is not robust against disturbances and errors, potentially resulting in poor generalization, due to small errors adding up, causing the trajectory to stray from the state space area covered by the demonstrations. Therefore behavioral cloning has been extended, one method to do this is by augmenting the dataset during training querying an expert. The policy is trained on the demonstrations then the resulting policy is evaluated using the expert. these results are iteratively added to the training dataset and training is repeated, resulting in an iterative process. Several methods have been proposed doing this, like the DAgger (Dataset Aggregation) method[17] or Direct Policy Learning (DPL). Other approaches try to model the expert's reward function like inverse reinforcement learning (IRL)[1].

There are two fundamental classes of imitation learning methods. The training architecture can be *offline* or *online*. Offline methods base the training on a previously captured data set. Online methods incorporate interaction with an expert or the environment during the training. This makes offline methods generally simpler since the process of generating the demonstrations can be completely separated from the training. Online methods, however, are intertwined with the domain expert or the environment, which can make the training more difficult or slow.

Imitation learning methods can be based on task space or joint space demonstrations. Joint space methods offer a more application-specific definition of the movement since every joint angle is precisely defined by the captured demonstration trajectory. However, acquiring training data is more difficult, since it can only be generated with access to an identical kinematic system. If demonstrations should be taken from moving the robot directly, the robot needs to have a compliant controller as well, so that an expert can move the manipulator according to the task. Capturing a teleoperated movement is also possible. Task space demonstrations are more versatile as they intrinsically are not dependent on any kinematic chain. Therefore capturing examples can be done using a variety of different methods. Task-space demonstrations exceed the possibilities set out by joint space demonstrations. We gain the possibility to acquire demonstrations by teleoperated and manually guided demonstrations, it's also possible to use motion capture systems or robots with different kinematic chains. This independence of physical layout makes it possible to quickly transfer policies trained in task space between different robots using operational space control.

Movement primitives are another common technical foundation used for robotic manipulation. Parametrizing these enables the construction of complex movements using these dynamic systems as building blocks. This lays the foundation of motor primitives theory[12], whose premise is that complex movements are decomposable into distinct atomic movements. In recent years it has become clear, that motor actions of large animals can be understood as a combination of primitive building blocks[9]. This makes representing movements with motor primitives an obvious choice for robotics. Different mathematical constructs have been used to model these primitives, from dynamical systems and statistical models (ProMP and Kernelized Movement Primitives) to Task parametrized Gaussian mixture models (TP-GMM)[4][25]. Dynamic movement primitives are using dynamical systems as a basis to model movements. The basis of this technique is laid by simple dynamic systems like attractors[2] or limit cycles. Dynamic movement primitives (DMP) [18] span a vector field that defines the movement of the policy through the state space by assigning velocities to each point in the state space. Integrating and using these velocities as a control input results in a reactive policy controlling robotic movements.

Behavioral cloning techniques have been augmented by supplementing them with dynamic movement primitives. The movement primitives live in a latent space representation. Task space movements are created by combining these two building blocks with a mapping that morphs the latent space into the task space. The movement primitives can be understood

[2]This can be understood as using singularities in the dynamic systems like sources and sinks as movement features

as inductive biases to the learning architecture to impose guarantees on the resulting policy. For control policies defined in task space, the mathematical representation of rotation is an important component of the control strategy, since it imposes the inherent properties of the representation onto the control behavior. Multiple representations have been used in robotics from Euler angles to rotation matrices and quaternions. Euler angles are easily understandable since humans have an intuitive understanding of rotation by axes. They have unfavorable characteristics disincentivizing the use for robot control, however, since they are susceptible to gimbal locks[3] as well as ambiguity[24]. Quaternions and rotation matrices provide more attractive properties since they are unambiguous. Rotation matrices are defined in the special orthogonal lie group $\mathrm{SO}(3)$ describing any rotations in three-dimensional space. Defining a diffeomorphic mapping on these representations is difficult, however, since not every combination of values constitutes correct quaternions or rotation matrices. Additionally, ensuring stability guarantees poses difficulty since the mapping is not defined in an Euclidean space. LieFlows[22] solve this issue by defining the mapping function connecting the latent dynamics to the task space with Lie groups, enabling the training of robot policies with stability guarantees this way. We extend the method of LieFlows, which previously has been deployed to do robot control in joint space[23] and task space[22]. We adapt the architecture to enable precise point-to-point motions as well as cyclic motions.

[3] Singularities in the rotation angle representation

# 3 Foundations

This thesis uses the term *skill* to describe a movement executing a specific spatial manipulation task. This movement is defined as a vector field on the tangent space of a Lie group, generated by morphing the dynamics of a latent space by using neural ODEs as a diffeomorphism. A *task* can be composed of several skills.

## 3.1 Stable Vector Fields

Vector fields are a functional mapping of points in a manifold to vectors, these vectors are called tangent vectors. Vector fields are commonly used in robotics to assign actions to a state space representation. In our method, stable vector fields are generated by nonlinear dynamical systems that are stable in terms of Lyapunov. We define basic vector fields on the $\mathrm{SE}(3)$ manifold that describe circular or linear motions. Combining these vector fields with invertible mappings forms the basis for more complex movements. Ensuring stability by carefully constraining the mapping enables us to guarantee convergence to the terminal motion. This destination can either be a singular point or a limit cycle. Stable vector fields form the basis of our technique.

Using stable vector fields on the $\mathrm{SE}(3)$ Lie group creates stable, smooth, and reactive control policies. A reactive control scheme makes the movement execution robust against disturbances. If the robot is controlled by a compliant controller, users can move the arm without the quality of the control policy deteriorating, which results in very robust movements. The policy adapts the actions to the new position, and the vector field pushes the robot state toward the trained movement pattern.

### 3.1.1 Dynamic Systems as Movement Primitives

We use dynamical systems as movement primitives to describe motion on a vector field. Dynamical systems are generally described like this

$$\dot{\mathbf{x}} = f(\mathbf{x}, t) \quad \mathbf{x}(t_0) = \mathbf{x}_0$$

We define $\mathbf{x}$ as part of a cartesian vector space.

$$\mathbf{x} \in \mathbb{R}^n$$

It is possible to incorporate components that can be trained using gradient descent since dynamic systems are differentiable. This is elaborated in subsection 4.1.3 and subsection 4.2.2 in the methodology of the thesis.

#### Lyapunov Stability

Consider a dynamical system described by an ordinary differential equation (ODE).

$$\dot{\mathbf{x}} = f(t, \mathbf{x}), \quad \mathbf{x}(t_0) = \mathbf{x}_0, \quad \mathbf{x} \in \mathbb{R}^n \tag{3.1}$$

According to the Picard-Lindelöf theorem the ODE $f(t, \mathbf{x})$ has a unique solution if Lipschitz continuous with respect to $\mathbf{x}$ and continuous with respect to $t$. Each point $\mathbf{x}^*$ that satisfies $f(\mathbf{x}^*, t) = 0$ is called an *equilibrium point* of 3.1. An equilibrium point is *locally stable* if initial conditions near $\mathbf{x}^*$ do not diverge and stay near $\mathbf{x}^*$. This formalizes the notion of *Lyapunov stability*. By shifting the coordinate system so that $\mathbf{x}^* = 0$ we define an equilibrium point $\mathbf{x}^* = 0$ stable with respect to Lyapunov at $t_0$ if

$$\forall\, \varepsilon > 0 \quad \exists\, \delta > 0 : \qquad \|\mathbf{x}(t_0)|| < \delta \implies \|\mathbf{x}(t)\| < \varepsilon \tag{3.2}$$

Intuitively, this says, that trajectories can not move away from the equilibrium point according to the amount of coordinates[16].

#### Asymptotic Stability

An equilibrium point $\mathbf{x}^* = 0$ is considered *asymptotically stable*[16] at $t_0$ if

1. the point is stable in terms of Lyapunov
2. $\mathbf{x}^*$ is locally attractive: $\exists\, \delta : \|x(t_0)\| < \delta \implies \lim_{t\to\infty} x(t) = 0$

#### Stability of Cyclic Motions

On limit cycles, the notion of Lyapunov stability does not hold. Limit cycles carry innate notions of stability, however. Limit cycles are called stable if trajectories generated from neighboring initial conditions converge to the limit cycle. Therefore stable limit cycles are *attractors*.

## 3.2 Lie Theory

Lie theory has been named after Sophus Lie, who developed this methodology to solve differential equations using symmetries, inspired by Galois theory which addressed algebraic equations[20]. Besides differential equations, its usefulness reaches many different branches of mathematics, physics, and engineering. Lie theory has been applied to geometrical problems early. In recent years, however, Lie theory has become increasingly popular in robotics and is used as a favorable representation of task space in robotic applications[21].

### 3.2.1 Lie Groups

A Lie group is a mathematical *group* as well as a *differentiable manifold*. A mathematical group $(\mathcal{G}, \star)$ is a set $\mathcal{G}$ with an operator $\star$ satisfying the group axioms [21].

We require for all $x, y, z \in \mathcal{G}$:

| | |
|---|---|
| closure under $\star$ | $x \star y \in \mathcal{G}$ |
| existence of identity $\epsilon$ | $\epsilon \star x = x \star \epsilon$ |
| existence of inverse element | $x^{-1} \star x = x \star x^{-1} = \epsilon$ |
| associativity | $(x \star y) \star z = x \star (y \star z)$ |

A differentiable manifold is a manifold that is smooth, with smoothness referring to the property of being locally similar to an Euclidean vector space. This allows us to use calculus. Lie groups have a tangent space, which is modeled by the Lie algebra. The logarithmic map relates lie groups to lie algebras, and the exponential map does the inverse. We describe them in detail in subsection 3.2.2.

**The Lie Group** $\mathrm{SO}(3)$

$$\mathrm{SO}(3) = \left\{ \mathbf{R} \;\middle|\; \mathbf{R} \in \mathbb{R}^{3\times 3}, \mathbf{R}\mathbf{R}^\intercal = \mathbf{R}^\intercal\mathbf{R} = \mathbf{I}, \det(\mathbf{R}) = 1 \right\} \tag{3.3}$$

The Lie group $\mathrm{SO}(3)$ is the *special orthogonal group*. It is defined by an orthogonal three-by-three matrix with determinant one. This class of matrices can be used to express any rotation in three-dimensional space. Its tangent vector spaces can be expressed as values in $\mathbb{R}^3$.

**The Lie Group** $\mathrm{SE}(3)$

$$\mathrm{SE}(3) = \left\{ \mathbf{A} \;\middle|\; \mathbf{A} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^\intercal & 1 \end{bmatrix}, \mathbf{R} \in \mathbb{R}^{3\times 3}, \mathbf{t} \in \mathbb{R}^3, \mathbf{R}\mathbf{R}^\intercal = \mathbf{R}^\intercal\mathbf{R} = \mathbf{I}, \det(\mathbf{R}) = 1 \right\} \tag{3.4}$$

The Lie group $\mathrm{SE}(3)$ consists of homogenous transformation matrices. It consists of a rotation matrix $\mathbf{R} \in \mathrm{SO}(3)$ and a position vector $\mathbf{t}$ represented using homogeneous coordinates. This group is called the *special Euclidean group*. Transformations in $\mathrm{SE}(3)$ are a subgroup of affine transformations preserving angles. Operations in $\mathrm{SE}(3)$ are describing rigid motion. Its tangent vector spaces can be expressed as values in $\mathbb{R}^6$.

$\mathrm{SE}(3)$ and $\mathrm{SO}(3)$ satisfy the group axioms under multiplication. Since multiplication is a continuous function of two matrices, $\mathrm{SO}(3)$ and $\mathrm{SE}(3)$ are differentiable manifolds.

### 3.2.2 The Tangent Space of a Lie Group

We write the *tangent space* of a Lie group $\mathcal{M}$ in the point $\mathbf{x}$ as $T_\mathbf{x}\mathcal{M}$. The tangent space at $\mathbf{x}$ is called the Lie algebra of $\mathcal{M}$ and spans the local tangent space. We call this $T_\mathbf{x}\mathcal{M}$. Lie algebras can have complex structures like skew-symmetrical matrices. Since representing the tangent space of a Lie group using an Euclidean coordinate system $\mathbb{R}^m$ is intuitive, we focus on describing Lie theory using this representation. This is possible since each Lie algebra has an isomorphic cartesian vector space description[21]. We skip the mathematically rigorous definition of Lie algebras and their mapping to the tangent vector space for the sake of briefness and simplicity. Intuitively a tangent space can be thought of as imposing Euclidean coordinates onto its manifold (Figure 3.1).

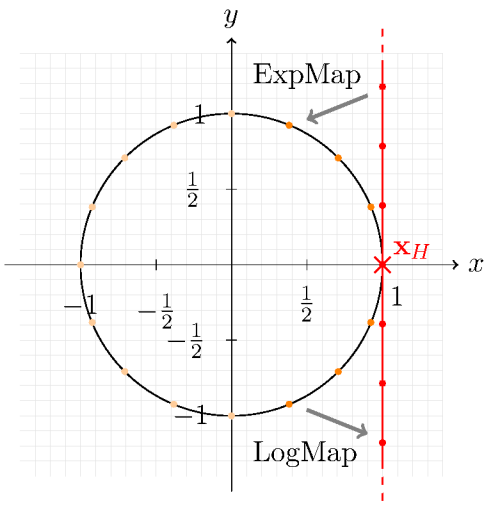


Figure 3.1: Coordinates in orange on the manifold of the unit circle $\mathcal{S}^1$ at $\mathbf{x}_H$ and in red on the tangent space. The tangent space spans the interval $(-\pi, \pi)$.

Since representing coordinates on an Euclidean vector space $\mathbb{R}^m$ simplifies mathematical operations Lie theory simplifies the numerical representation of three dimensional, homogenous coordinates. Calculations, done in one coordinate system can be transferred to different coordinate systems if the coordinate systems can be differentiably transformed into each other. Transformations obeying this property are called *diffeomorphisms*.

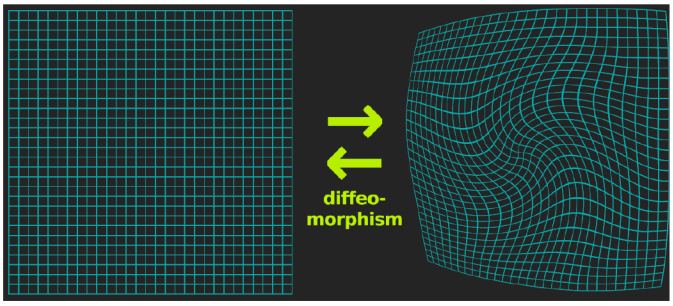


Figure 3.2: Simple illustration showing the intuition behind diffeomorphisms

### 3.2.3 Diffeomorphisms

A diffeomorphism is a bijective mapping between two manifolds $\mathcal{M}$ and $\mathcal{N}$, that is continuously differentiable in both directions. Figure 3.2 shows an intuitive description of a diffeomorphism. A function $f : \mathcal{M} \to \mathcal{N}$ is *diffeomorphic*, if it is bijective and continuously differentiable, while its inverse function $f^{-1} : \mathcal{N} \to \mathcal{M}$ is also continuously differentiable.

### 3.2.4 ExpMap and LogMap

The exponential map moves coordinates from a tangent space to the Lie group $\mathcal{M}$.

$$\text{ExpMap} : \mathbb{R}^m \longrightarrow \mathcal{M} \tag{3.5}$$

The logarithmic map moves elements of the Lie group to coordinates in the tangent space.

$$\text{LogMap} : \mathcal{M} \longrightarrow \mathbb{R}^m \tag{3.6}$$

These exponential and logarithmic maps are defined with respect to a point $\mathbf{x} \in \mathcal{M}$, at which the tangent space $T_{\mathbf{x}}\mathcal{M}$ is created.

## 3.3 Learning Stable Vector Fields using LieFlows

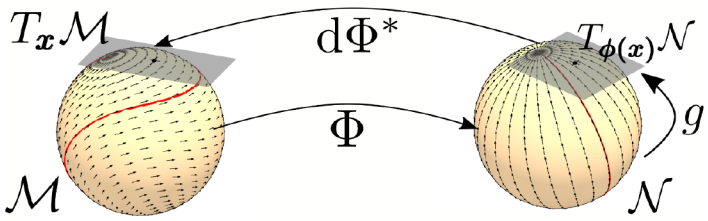


Figure 3.3: Mapping a stable vector field to task space using the diffeomorphic mapping $\boldsymbol{\Phi}$.
Graphics taken from [22]

The LieFlows method learns stable vector fields using diffeomorphisms on Lie groups. Diffeomorphism-based stable vector field methods are based on a movement primitive represented in a latent space, that is deformed and projected back to task space using a diffeomorphic function $\Phi$ (Figure 3.3). Since the movement primitive is represented by a dynamic system in latent space, we also call it the *latent dynamics*. We describe it in subsection 3.3.2. Complex movements are represented by the shape of $\Phi$. In order to train policies based on this architecture, practical learning-based methods model $\Phi$ by an invertible neural network [23][22][7], which is trained using gradient descent. LieFlows represents these invertible networks with neural ODEs, described in detail in subsection 3.3.1.

The task space and latent space are modeled with identical Lie groups and called $\mathcal{M}$ and $\mathcal{N}$. This paper applies the method to three-dimensional poses, we use the Lie group $\mathrm{SE}(3)$ to represent them. The latent space movement primitive is modeled by stable differential equations and projects the calculated velocity as a stable vector field $g : \mathcal{N} \to T\mathcal{N}$ to the latent tangent space. Points from the task space $\mathcal{M}$ are mapped to the latent space using the diffeomorphism $\Phi : \mathcal{M} \to \mathcal{N}$. To correctly resemble a complex movement, $\Phi$, and the latent dynamics need to be chosen carefully. Since we ultimately want to calculate the vector field in task space, $\Phi$ must be invertible. A given mapping $\Phi$ between $\mathcal{M}$ and $\mathcal{N}$ has a corresponding linear map between the tangent spaces of $\mathcal{M}$ and $\mathcal{N}$.

$$\Phi_x \ : \ T_{\mathbf{x}}\mathcal{M} \longrightarrow T_{\Phi}(x)\mathcal{N} \tag{3.7}$$

With $g(\mathbf{x})$ cleverly designed, defining the dynamical system as the $\mathrm{LogMap}$ of the latent space representation at $\Phi(\mathbf{x})$, the reverse mapping is used to get the morphed dynamics in task space. We call this the *pullback operation*[4]:

$$\mathrm{d}\Phi^*_{\mathbf{x}} : \ T_{\Phi}(x)\mathcal{N} \longrightarrow T_{\mathbf{x}}\mathcal{M} \tag{3.8}$$

$$\dot{\mathbf{x}} = \mathrm{d}\Phi^*_x(g(\Phi(\mathbf{x}))) \tag{3.9}$$

Note that the pullback operation is in tangent space representation, this only works if $\Phi$ is diffeomorphic.

Defining mappings on manifolds like $\mathrm{SE}(3)$ directly can result in unstable policies, since definitions of diffeomorphic functions on non-euclidean manifolds are difficult. Stable vector field approaches usually achieve stability guarantees by defining $\Phi$ on Euclidean spaces.

$$\Phi : \mathbb{R}^n \to \mathbb{R}^m \tag{3.10}$$

[4]This is the reason why invertibility is not sufficient and $\Phi$ needs to be diffeomorphic

Euclidean spaces do not represent orientations, however.

The LieFlows method models the diffeomorphism $\Phi$ with Lie groups since they allow representing rotations in the tangent space as an Euclidean space. The mapping can then be described with a diffeomorphic function $f_\theta$ which is much simpler to define since it operates on Euclidean spaces.

We describe how this method can be applied to the $\mathrm{SE}(3)$ lie group. The diffeomorphism must be defined in parts since a Lie group is topologically different from its tangent space, making the definition of a diffeomorphism by a single function impossible.

$$\Phi(\mathbf{x}) = \begin{cases} \mathrm{ExpMap}(f_\theta(\mathrm{LogMap}(\mathbf{x}))) & \text{if } \mathbf{x} \in U_{\mathcal{M}} \\ \mathbf{x} & \text{if } \mathbf{x} \in \mathcal{M} \setminus U_{\mathcal{M}} \end{cases} \tag{3.11}$$

Points of the Lie group that are unambiguous under the coordinate map are assigned to $U_{\mathcal{M}}$, the rest are in $\mathcal{M} \setminus U_{\mathcal{M}}$. Under the condition that $f_\theta(x)$ is diffeomorphic $\mathrm{ExpMap}(f_\theta(\mathrm{LogMap}(\mathbf{x})))$ also is, since $\mathrm{ExpMap}$ and $\mathrm{LogMap}$ are diffeomorphic functions[14]. The identity mapping of elements $x \in \mathcal{M} \setminus U_{\mathcal{M}}$ to $y \in \mathcal{N} \setminus U_{\mathcal{N}}$ is also diffeomorphic. Although $\Phi$ is diffeomorphic in parts, continuity and differentiability at the boundaries are required, too. We require

$$\begin{aligned} \Phi(\mathbf{x}) &= \mathrm{ExpMap}(f_\theta(\mathrm{LogMap}(\mathbf{x}))) \\ \mathrm{ExpMap}(\mathrm{LogMap}(\mathbf{x})) &= \mathbf{x} \end{aligned} \tag{3.12}$$

when $x$ approaches the boundaries $\mathcal{M} \setminus U_{\mathcal{M}}$. This way, values in $U_{\mathcal{M}}$ nearing the boundaries will approach identity mapping. Section 5.2 and section 5.3 show how this is implemented in practice.

$$\begin{aligned} \mathrm{LogMap} : U_{\mathcal{M}} &\longrightarrow \hat{U}_{\mathcal{M}}, & \hat{\mathbf{x}} &\in \hat{U}_{\mathcal{M}} \subseteq T_{\mathbf{x}_H}\mathcal{M} \\ f_\theta : \hat{U}_{\mathcal{M}} &\longrightarrow \hat{U}_{\mathcal{N}}, & \hat{\mathbf{z}} &\in \hat{U}_{\mathcal{M}} \subseteq f_\theta \circ T_{\mathbf{x}_H}\mathcal{M} \\ \mathrm{ExpMap} : \hat{U}_{\mathcal{N}} &\longrightarrow U_{\mathcal{N}}, & \mathbf{z} &\in \hat{U}_{\mathcal{N}} \subseteq \mathcal{N} \setminus U_{\mathcal{N}} \end{aligned} \tag{3.13}$$

The sets $\hat{U}_{\mathcal{N}}$ and $\hat{U}_{\mathcal{M}}$ are called the *first cover of the tangent space*.

### 3.3.1 Modeling $f_\theta$ with Bounded Flows

In the LieFlows method $f_\theta(\mathbf{x})$ is modeled with neural ODEs. Neural ODEs are invertible neural networks, that are used to model diffeomorphisms by taking a point $\mathbf{x}$ and setting

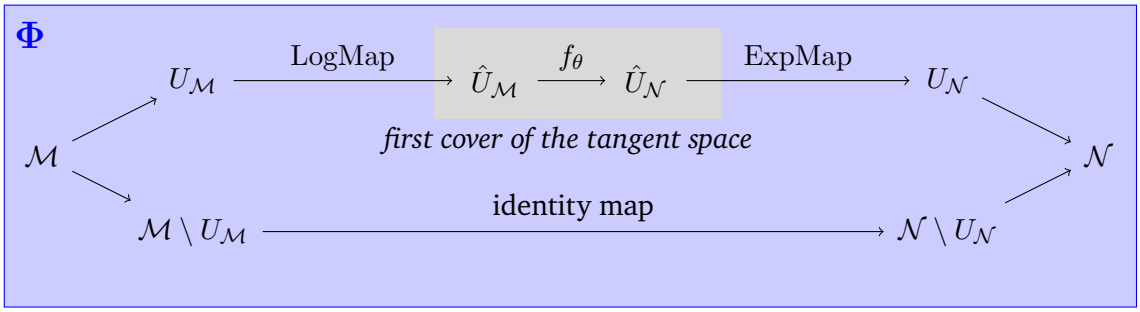


Figure 3.4: Modeling $\Phi : \mathcal{M} \to \mathcal{N}$ with Lie groups. This enables us to define the diffeomorphism $f_\theta$ on Euclidean manifolds with stability guarantees.

it as the initial point of the dynamic system $\mathbf{x}(t_0)$. Movement on the vector field encodes the diffeomorphic map[5]. This movement is calculated by forward Euler integration.

$$\hat{\mathbf{z}} \; = f_\theta(\hat{\mathbf{x}}) = \hat{\mathbf{x}} + \int_0^1 \alpha(\hat{\mathbf{x}})\psi(\hat{\mathbf{x}}) \tag{3.14}$$

where

$\alpha(\mathbf{x})$ the *constraint* or *scaling function*, forcing continuity at the boundary of $U_\mathcal{M}$

$\psi(\mathbf{x})$ arbitrary Lipschitz continuous vector field

The scaling function $\alpha(\mathbf{x})$ prevents $f_\theta$ from moving values out of the set of the image and preimage and vice versa. This is done by choosing the scaling function $\alpha(\mathbf{x}) = 0$ for $\mathbf{x}$ near values in $\mathcal{M} \setminus U_\mathcal{M}$, this way the ODE representing $f_\theta$ becomes the identity at the boundaries and diffeomorphic in between. The vector field $\psi(\mathbf{x})$ is represented with a parametrized neural network trained to learn the mapping.

[5]Important to note is that this is not connected in any way with the stable vector field used in the latent dynamics, it is used solely to model the diffeomorphism

### 3.3.2 Stable Latent Space Dynamics as Movement Primitives

Consider the simple dynamic system:

$$\dot{\mathbf{z}} = g(\mathbf{z}) = \begin{cases} -\mathrm{LogMap}_{\mathbf{z}_H}(\mathbf{z}) & \text{if } \mathbf{z} \in U_{\mathcal{N}} \\ \mathbf{0} & \text{if } \mathbf{z} \in \mathcal{N} \setminus U_{\mathcal{N}} \end{cases} \tag{3.15}$$

By selecting the points in $\mathcal{M} \setminus U_{\mathcal{M}}$, sources can be favorably placed on the manifold, since the flow constraint $\alpha(\mathbf{x})$ makes $\Phi$ the identity in these points. The point $\mathbf{z}_H$ becomes the sink. Additionally, workspace limits can be imposed by putting them in $\mathcal{M} \setminus U_{\mathcal{M}}$. For rotational dimensions, we assign $\pm\pi \in \mathcal{M} \setminus U_{\mathcal{M}}$, making these orientations sources. Dynamic systems can be augmented with parameters, that are dependent on $\mathbf{z}$ and are predicted by a neural network during execution[23]. It is important, however, to ensure the integration of those predictors in a way that retains the stability of the dynamic system.

In Lie groups, vector fields are always defined with respect to an origin. In LieFlows, the origin of the tangent space is always shifted to $\mathbf{0}^n$, in the task space it is set to the target transformation of the skill. The $\mathrm{LogMap}$ inside $\Phi$ normalizes $\mathbf{x}$ so that $\hat{\mathbf{x}}_H = \mathbf{0}$. This normalization is reversed in the pullback (see Figure 3.5).

This thesis uses the notation $\mathbf{x}$ for coordinates in task space. The notation $\mathbf{z}$ is chosen for coordinates in latent space. Coordinates in the first cover of the tangent are denoted with a hat $\hat{\mathbf{x}}, \hat{\mathbf{z}}$. Coordinates in the tangent space representation encoding velocities from the dynamic system are denoted with dots $\dot{\mathbf{x}}, \dot{\mathbf{z}}$.

### 3.3.3 Defining $f_\theta$ on $\mathrm{SE}(3)$

Here we show an exemplary definition of $\alpha(\mathbf{x})$ that can be used to model $f_\theta$ on $\mathrm{SE}(3)$. Other solutions are possible, we discuss them in chapter 4. Constraining the flow depends on the type of information to represent. We split the scaling function into a positional and a rotational one.

$$\alpha(\mathbf{x}) = \alpha_{\mathrm{pos}}(\mathbf{x}^{(0,3)})\alpha_{\mathrm{ori}}(\mathbf{x}^{(3,6)}) \tag{3.16}$$

Positional dimensions X, Y, and Z may be constrained to a certain workspace interval. To do this, $\alpha(\mathbf{x})$ is chosen such that it becomes zero at the workspace limits and differentiable in between.

$$\alpha_{\mathrm{pos}}(\mathbf{x}^{(n)}) = 0 \qquad \text{if } \mathbf{x}^{(n)} \in \mathbf{x}^{(n)}_{\mathrm{lim}} \qquad \text{for all} \quad n \in \{x, y, z\}$$

Orientations are represented in the sets $\hat{U}_{\mathcal{M}} = \hat{U}_{\mathcal{N}} = \{\mathbf{x} \in \mathbb{R}^3, \|\mathbf{x}\| < \pi\}$. The scaling function must guarantee $\alpha(\mathrm{x}) = 0$ if $\|x\| = \pi$, which can be done multiple ways. It could be defined like this:

$$\alpha_{\mathrm{rot}}(\mathbf{x}) = \max\left(0, 1 - \left(\frac{\mathbf{x}}{\pi}\right)^2\right) \tag{3.17}$$

This enforces $f_\theta$ to be the identity map at rotations with amount $\pi$. The function $f_\theta$ must be constructed according to the set $\mathcal{M} \setminus U_{\mathcal{M}}$. Roots of $f_\theta$ denote elements in $\mathcal{M} \setminus U_{\mathcal{M}}$.

Combined with the dynamic system from subsection 3.3.2 it is ensured, that the boundaries of the intervals are projected back to the task space as a source. Note, that the mapping $\Phi$ can move the stable point of the dynamic system since the position of this sink in the vector field is a result of the training process. The vector field in task space is asymptotically stable, however. If we chose $\mathbf{x}_H \in \mathcal{M} \setminus U_{\mathcal{M}}$ the sink would be projected to $\mathbf{x}_H$ in the task space. The scaling function would have to be constructed accordingly.

### 3.3.4 Movement Segmentation

Representing motion with the LieFlow method imposes certain constraints on the types of movement sequences that can be represented. This is dependent on the characteristics of the latent dynamics and the structure of $\Phi$ especially with respect to the flow constraint and the Lie group representing the robot state. Since the vector field induced to the task space is a functional relation, each unique point in the state space can map to exactly one tangent vector. This rules out movements that traverse the same point in space more than one time with different velocity vectors. To represent movements like this with LieFlows, they have to be segmented to separate phases by time into multiple skills. Each skill is then represented by its own vector field. Additional logic has to be deployed keeping track of the state space and the progress of the task execution to switch to different states at appropriate times. Incorporating neural networks that enable the learning of speed parameters to drive the movement primitives, can reduce the morphing complexity that the diffeomorphic map needs to represent. The neural ODE does not need to contract the latent space as much to represent different speeds.

## 3.4 Training a Control Policy using the LieFlows Method

The training dataset consists of state action pairs. Since LieFlows is a form of behavioral cloning, we train $\psi(\mathbf{x})$ such that the loss between the actual velocities from the demonstration and the movement predicted by the vector field in the same point is minimized. We use $n$ state action pairs taken from demonstration movements as ground truth.

$$\mathcal{D} = \begin{bmatrix} \mathbf{x} \\ \dot{\mathbf{x}} \end{bmatrix}^n \tag{3.18}$$

We search a mapping $\Phi$ minimizing the loss between the ground truth action and the stable vector field prediction.

$$\begin{aligned} &\min_{\Phi} \|\dot{\mathbf{x}} - \mathrm{d}\Phi_x^*(g(\Phi(x)))\|_2^2 \\ &\text{s.t.} \quad \Phi \qquad \text{is a diffeomorphism} \end{aligned} \tag{3.19}$$

The diffeomorphic property of $\Phi$ needs to be ensured by the construction of the scaling function.

**Algorithm 1** Behavioral Cloning with LieFlows

**function** TRAIN($\mathrm{svf}, \theta_0, E, \mathcal{D}$) ▷ svf is the model to train
▷ with $\mathrm{svf}(\mathbf{x}_k, \theta_i) = \mathrm{d}\Phi_x^*(g(\Phi(x)))$
▷ $\theta_0$ are the initial parameters
▷ the number of epochs $E$
▷ demonstrations $\mathcal{D}$
    **for** $i = 0$ to $E$ **do**
        $\mathbf{x}_b, \dot{\mathbf{x}}_b \sim \mathcal{D}$ ▷ sample from data set
        $\mathrm{loss}(\theta_i) = \frac{1}{B} \sum_{k=0}^{B} \|\dot{\mathbf{x}}_k - \mathrm{svf}(\mathbf{x}_k, \theta_i)\|_2^2$
        $\theta_{i+1} = \theta_i + \alpha \nabla_\theta \mathrm{loss}(\theta_i)$
    **end for**

    return $\theta^*$ ▷ parameters of best model
**end function**

The training process is described by Algorithm 1.

## 3.5 Robotic Environment

The Telemax robot was modified by the SIM group at TU Darmstadt to enable control using a ROS interface. Robot OS (ROS) can be described as a middleware specialized for robotic tasks. It is composed of a set of software tools installed on top of the Ubuntu Linux distribution. The ROS environment enables the integration of sensors such as cameras with depth sensors, LIDAR, joint state information, radiation, and chemical sensing with control logic that can actuate the robot using the same platform.

Since the robot is an old model that does not support trajectory control from the factory yet, a simple P controller has been developed at SIM to control the robot using the native joystick-based control interface exposed by the robot for communicating with the control tablet. During normal operation, this interface is used to send control commands from the handheld control unit to the robot's internal joint controller. This retrofitted controller utilizes this interface to expose a trajectory control interface to ROS which is exposed via topics. Trajectories are transmitted by sending JointTrajectory messages. The robotic environment has been integrated into the Webots simulator, this way the learned policies can be tested in simulation in a realistic environment.

The scenario inside the Webots simulation is modeled after a physical practice room present at Kerntechnische Hilfsdienst GmbH near Karlsruhe. This place is used as a training ground for robot teleoperators and formed the base scenario to develop and test robotic AI assistance systems in the scope of the KIARA project. The perceptions of the robot are combined and reconstructed into a 3D scene that visualizes the spatial positioning of the robot in its environment. This is done with a software module inside of ROS called RViz. This 3D Visualizer uses different ROS sensor topics to get, assemble, and render information about the scene. Integrated into this visualization environment are certain UI features to control the robot. There is a Waypoint tool to plan and execute movement trajectories for the robot base, as well as different camera views, snapshot features, and options to activate different sensor visualizations. RViz depicts the world from the view of the robot, while Webots shows the 3D simulation environment that represents the "true state" of the environment. The map consists of stairs, valves, tables, and doors that all can be operated in simulation.

### 3.5.1 Control Scheme

The stable vector field method is used to calculate velocity commands given a current robot state in task space. This results in state-action pairs that form the basis of the control policy. The velocity vector predicted by the SVF is integrated with the control interval $\Delta t$, set as the waypoint at the next timestep $t+1$, and sent to the robot's trajectory controller. Since the Telemax robot has a stiff, noncompliant controller that is unable to profit from reactivity, we formulate a simple control policy. This control scheme generates a trajectory by generating via points of the frequency $\frac{1}{\Delta t}\mathrm{Hz}$.

$$\begin{aligned} \mathbf{x}_{des}^{(t+1)} &= \mathbf{x}_{des}^{(t)} + \Delta t \dot{\mathbf{x}}_{des}^{(t)} \\ \dot{\mathbf{x}}_{des}^{(t+1)} &= \mathrm{svf}(\mathbf{x}_{des}^{(t+1)}) \end{aligned} \tag{3.20}$$

A reactive control policy is based on the measured robot state instead.

$$\begin{aligned} \mathbf{x}_{des}^{(t+1)} &= \mathbf{x}^{(t)} + \Delta t \dot{\mathbf{x}}^{(t)} \\ \dot{\mathbf{x}}_{des}^{(t+1)} &= \mathrm{svf}(\mathbf{x}_{des}^{(t+1)}) \end{aligned} \tag{3.21}$$

If trained policies were to be deployed onto a different, compliant robot, this policy could be used.

Since the trajectory controller on the robot is joint-based, we do operational space control[11] by differential kinematics to calculate and control the target state in joint space.

## 3.6 Problem Statement

We want to extend the LieFlows method to enable precise positioning and cyclic motion. To do this, we need to examine how to change the different parts of the method, we need to find good flow constraints $\alpha(x)$ for these movement tasks and find latent dynamics representations $g(z)$ that can reproduce these new movement types. We want to control the gripper of the robot, so we need to adapt the mapping $\Phi$ to enable the control of an additional dimension. Since additional dimensions can not be represented by the special Euclidean group $\mathrm{SE}(3)$, an extended representation is needed. The movement primitives are augmented with state-dependent velocity multipliers which are predicted by neural networks.

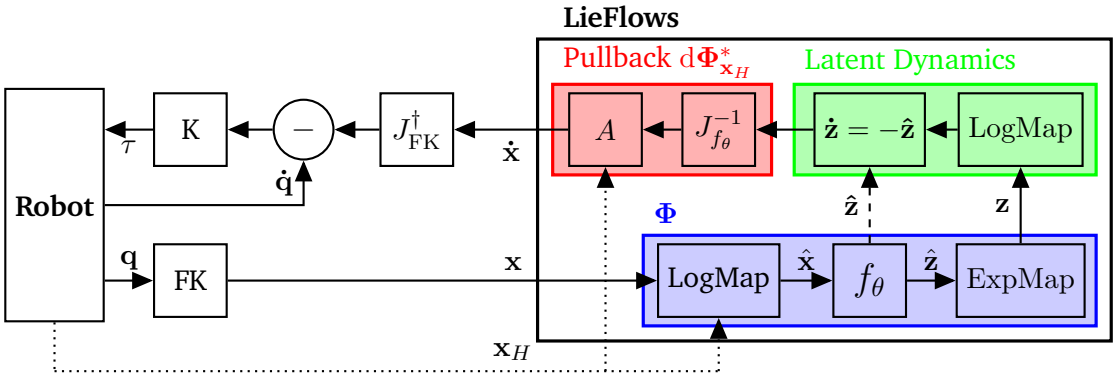


Figure 3.5: Reactive operational space control scheme using the LieFlows method

$$\begin{aligned} & \min_{\theta} \|\dot{\mathbf{x}} - \mathrm{svf}(\mathbf{x}, \theta)\|_2^2 \\ \text{with} \quad & \mathrm{svf}(\mathbf{x}, \theta) = \mathrm{d}\Phi^*_{\theta x} \circ g_\theta \circ \Phi_\theta(x) \\ \text{s.t.} \quad & \Phi \qquad \text{is a diffeomorphism} \end{aligned} \tag{3.22}$$

We consider two scenarios. First, we try to modify the learning architecture with respect to $g(\mathbf{x})$ and $\alpha(\mathbf{x})$ for precise positioning movements including one extra dimension controlled by the stable vector field. Secondly, we search a cyclic movement primitive $g_{\mathrm{cycle}}(\mathbf{x})$ and a flow constraint $\alpha(\mathbf{x})$ yielding stable execution of cyclic motion.

# 4 Methodology

We propose an extension to the learning architecture for two classes of tasks. These adaptations are designed to enable the precise execution of point-to-point movements with an additional degree of freedom and introduce circular movements to the LieFlows method. Currently, the flow constraint $\alpha(\mathbf{x})$ has been used to enforce sources at the boundaries for rotations at the antipodal point inside the $\Phi$ function.

Precise positioning requires asymptotical stability around the target pose $\mathbf{x}_H$. We tweak the flow constraint to enforce a sink at the target pose $\mathbf{x}_H$. We add a movement primitive that involves a slowing of the movement when the target is approached. Since we want to represent movements, where movement in the configuration space may only happen partially at times, pose rotation while staying at the same position for example, the movement primitive is augmented with a predictor of attraction speeds depending on the configuration of the robot in task space. To control one extra dimension, the representation of the task and latent space as well as the the tangent space vector field must be adapted for this extra degree of freedom. Representing the control velocities, as well as the first cover of the tangent space requires this dimensionality extension to integrate nicely with lie groups. Especially the diffeomorphic function $f_\theta$ needs to work with a new representation including the extra dimension.

To enable cyclic motions we define a limit cycle movement primitive in latent space with a corresponding flow constraint matching this class of tasks. To improve the expressiveness of the learning architecture, we also augment the limit cycle dynamics by including neural networks trained to predict attraction speed.

## 4.1 Precise Point to Point Movements

### 4.1.1 Extend $\mathrm{SE}(3)$ to represent additional Degrees of Freedom

We extend the $\mathrm{SE}(3)$ manifold, which already represents the position and orientation of the end effector by a single scalar value $r \in \mathbb{I}$ , $\mathbb{I} := \{x | j_{lower} \leq x \leq j_{upper}, x \in \mathbb{R}\}$ with $j_{lower}$ and $j_{upper}$ denoting the lower and upper joint limits for the gripper. In the case of our robot, this amounts to the range $[0, 0.2]$. This results in the following definition of our manifold:

$$\mathrm{SE}(3)\mathbb{R}^1 := \{\ [\mathbf{x}, r] : \quad \mathbf{x} \in \mathrm{SE}(3),\ r \in \mathbb{I}\ \} \tag{4.1}$$

The $\mathrm{LogMap}$ and $\mathrm{ExpMap}$ functions are modified to accommodate the additional dimension the SE3 manifold is extended by. This definition includes shifting the coordinate system to the target frame $\mathrm{x}_H$ according to the control scheme highlighted in Figure 3.5.

$$\mathrm{logmap}_{\mathrm{SE}(3)\mathbb{R}^1}(\mathbf{x}) = \left[\mathrm{logmap}_{\mathrm{SE}(3)}(\mathbf{x}^{(0,6)}, \mathbf{x}_H^{(0,6)}), \mathbf{x}_H^{(6)} - \mathbf{x}^{(6)}\right], \qquad \mathbf{x}, \mathbf{x}_H \in \mathrm{SE}(3)\mathbb{R}^1 \tag{4.2}$$

where

$\mathbf{x}$ is a state of the robot

$\mathbf{x}_H$ target transformation, origin in the first cover of the tangent space

Since $\mathrm{SE}(3)$ is a lie group, and $\mathbb{R}$ is also a Lie group, we can treat $\mathrm{SE}(3)\mathbb{R}^1$ like Lie group.

### 4.1.2 Constraining the Flow

We want the vector field to be stable at the target $\mathbf{x}_H$ so that executing it on the robot causes the end effector frame to converge to the configuration $\mathbf{x}_H$ with arbitrary precision. Therefore the latent dynamics need to be stable in $\mathbf{0}$ and $\Phi$ needs to be constructed so that $\mathbf{x}_H$ lies in the set mapped by the identity function. We constrain the flow separately for position and rotation.

$$\hat{\mathbf{z}} \ = f_\theta(\hat{\mathbf{x}}) = \hat{\mathbf{x}} + \int_0^1 \alpha_{\mathrm{poly}}(\hat{\mathbf{x}}) \psi_\theta(\hat{\mathbf{x}}) \tag{4.3}$$

with

$$
\begin{aligned}
\alpha_{\text{poly}}(\mathbf{x}) &= \begin{bmatrix} \alpha_{\text{pos}}(\mathbf{x}^{(0,3)}) \\ \alpha_{\text{ori}}(\mathbf{x}^{(3,6)}) \\ \alpha_{\text{pos}}(\mathbf{x}^{(6)}) \end{bmatrix}, \qquad \text{for } \mathbf{x} \in \mathbb{R}^7 \\
\alpha_{\text{pos}}(\mathbf{x}) &= \min\left(1, 1 + (\|\mathbf{x}\|_1 - 1)^{2\delta+1}\right) \\
\alpha_{\text{ori}}(\mathbf{x}i) &= \min\left(1, 1 - \left(\frac{2\|\mathbf{x}\|}{\pi} - 1\right)^{2\delta}\right) \\
\delta \in \mathbb{N} \setminus 0 \quad & \text{being a coefficient controlling steepness} \\
\psi_\theta(\hat{\mathbf{x}}) \quad & \text{the vector calculated by the neural network}
\end{aligned}
\tag{4.4}
$$

We call $\alpha_{\text{poly}}(\mathbf{x})$ the *polynomial flow constraint*, with $\delta$ being a constant controlling the steepness of the scaling function near the boundaries. It is inversely proportional to the area around the target where the flow is pinned to the identity map.

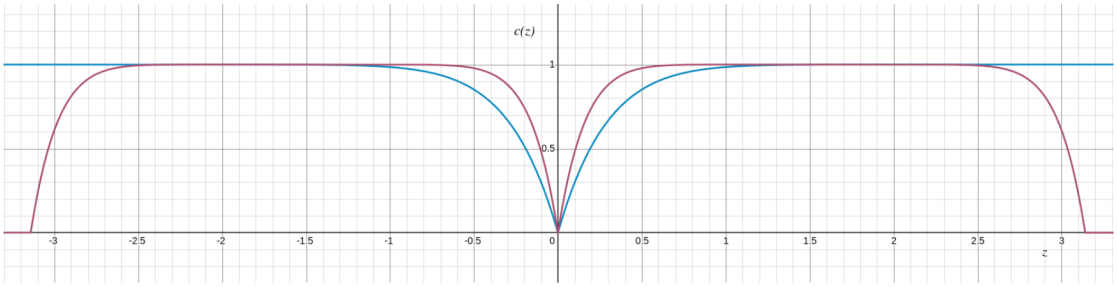


Figure 4.1: $\alpha_{\text{pos}}(\mathbf{x})$ in blue and $\alpha_{\text{ori}}(\mathbf{x})$ in red plotted for a single dimension with $\delta = 5$

This yields a complete model for $\Phi$ defining $f_\theta$ with neural ODEs (Equation 3.14)

$$
\hat{\mathbf{z}} = f_\theta(\hat{\mathbf{x}}) = \hat{\mathbf{x}} + \int_0^1 \begin{bmatrix} \alpha_{\text{pos}}(\mathbf{x}^{(0,3)})\psi_\theta(\hat{\mathbf{x}})^{(0,3)} \\ \alpha_{\text{ori}}(\mathbf{x}^{(3,6)})\psi_\theta(\hat{\mathbf{x}})^{(3,6)} \\ \alpha_{\text{pos}}(\mathbf{x}^{(6)})\psi_\theta(\hat{\mathbf{x}})^{(6)} \end{bmatrix} \tag{4.5}
$$

### 4.1.3 Parametrized Movement Primitive for Precise Positioning

We define a simple dynamic system that acts as a point attractor.

$$
\dot{\mathbf{z}} = g_{tanh}(\mathbf{z}) = \beta(\mathbf{z}) \tanh(\mathbf{z}), \qquad \mathbf{z}, \dot{\mathbf{z}} \in \mathbb{R}^n \tag{4.6}
$$

with

$$\beta(\mathbf{z}) = \exp\left(\mathrm{nn}(\mathbf{z})\right) + \varepsilon \tag{4.7}$$

The attraction speed is dependent on the distance, we choose the $tanh(x)$ function to limit the maximum speed of the attraction and to slow the movement down near the target. Using tanh is practical since it is differentiable and strictly monotonically increasing. The additional term $\beta(x)$ is added as a mechanism by which the training process is able to fine-tune the attraction speeds for different components of the spatial manifold controlled by the dynamical system. This term consists of a neural network prediction that's exponentiated to obtain a positive, strictly monotonically increasing mapping of the network prediction that retains stable attraction. We bias this prediction by $\varepsilon > 0$ such that the movement speed described by this attractor is never zero[6]. This guarantees movement of a minimal speed described by this constant. This construction allows learning the attraction parameters by a black box since the prediction can not make the dynamics unstable. We set $n$ to the dimensionality of the tangent space of the manifold, for precise positioning on $\mathrm{SE}(3)\mathbb{R}^1$, $n = 7$.

Rotations of the amount of $\pm\pi$ are projected to sources by $\alpha_{\mathrm{ori}}$ while positional components are projected to the manifold as a sink at the target pose $\mathrm{x}_H$. Since the origin $\mathbf{x}_H$ is shifted to $\mathbf{0}$ in the first cover of the tangent space, the latent dynamics $g_{\tanh}$ is stable in $\mathbf{0}$, and the scaling functions $\alpha_{\mathrm{pos}}$ and $\alpha_{\mathrm{ori}}$ produce the identity mapping at $\mathbf{0}$.

## 4.2 Circular Movements

As $f_\theta$ is defined on the $\mathrm{SE}(3)$ manifold the scaling function $\alpha(\mathrm{x})$ needs to satisfy criteria similar to the ones discussed in subsection 3.3.3. Cyclic movements have no stable point, they are defined with respect to a cycle (i.e. a set of states that is taken repeatedly). Stable dynamic systems that converge to such cycles are called *stable limit cycles*.

### 4.2.1 Constraining the Flow

We use the scaling functions laid out in Equation 4.4 to construct a polynomial flow constraint.

$$\alpha_{\mathrm{poly}}(\mathbf{x}) = \begin{bmatrix} \alpha_{\mathrm{pos}}(\mathbf{x}^{(0,3)}) \\ \alpha_{\mathrm{ori}}(\mathbf{x}^{(3,6)}) \end{bmatrix}, \qquad \text{for } \mathbf{x} \in \mathbb{R}^6 \tag{4.8}$$

[6] Allowing zero would create unwanted equilibrium points

This constraint pins the center point inside the limit cycle and makes it the identity transformation. Training the model will not cause a static offset in the flow model $\Phi$. The center of the flow at $\mathbf{x}_H$ is guaranteed to become a source.

### 4.2.2 Dynamic System

We model a stable limit cycle of radius one in two dimensions with a differential equation. Since we want to represent non-cyclic (point-to-point movements) in the remaining dimensions we extend the missing dimensions of the dynamics with tanh attractors. This way the dynamics are defined on 6DOF.

$$
\begin{aligned}
g_{\text{cycle}}(\mathbf{z}) &= \begin{cases} e^{\beta} \cdot \tanh(d-1)\cos(\phi) - d \cdot e^{\omega} \cdot \sin(\phi) \\ e^{\beta} \cdot \tanh(d-1)\sin(\phi) - d \cdot e^{\omega} \cdot \cos(\phi) \end{cases} \\
\dot{\mathbf{z}} = g_{\text{lcd}}(\mathbf{z}) &= \begin{cases} g_{\text{cycle}}(\mathbf{z}^{(0,2)}) \\ e^{\gamma} \cdot \tanh(\mathbf{z}^{(2,6)}) \end{cases}
\end{aligned}
\tag{4.9}
$$

with

$d = \|\mathbf{z}^{(0,2)}\|,$ The distance to the origin

$\phi = \text{atan2}(\mathbf{z}^{(1)}, \mathbf{z}^{(0)})$

$\begin{bmatrix} \beta \\ \omega \\ \gamma \end{bmatrix} = \text{nn}_{\theta}(\mathbf{z}), \qquad \beta \in \mathbb{R},\ \omega \in \mathbb{R},\ \gamma \in \mathbb{R}^4$

$\beta$ the speed the limit cycle is approached

$\omega$ the rotation speed of the cycle

$\gamma$ the attraction speed for the remaining dimensions[7]

Together with the scaling functions, we impose a source at zero for the two dimensions laying on the limit cycle and the boundaries ($\pm\pi$) of the rotation in the latent space. The remaining dimensions are defined using tanh attractors, this way they have sinks at zero.

[7] one is positional ($z$ dimension), the three remaining are the rotational dimensions from the latent space mapping of the $\text{SE}(3)$ Lie group

During training, the cyclic motion can be mapped to any space between the boundaries, the shape can vary arbitrarily.

Since the target of the motion $\mathbf{x}_H$ is shifted to $\mathbf{0}$ in the first cover of the tangent space and $f_\theta$ is the identity at $\mathbf{0}$ due to the construction of the scaling functions $\alpha_{\text{pos}}$ and $\alpha_{\text{ori}}$, the target position is projected as a source to the task space manifold in the X and Y dimensions, with the stable limit cycle projected to the surrounding space. The Z dimension of the task space manifold has a sink at $\mathbf{x}_H^{(z)}$, and the rotational degrees of freedom also have sinks at $\mathbf{x}_H$ due to the scaling functions mapping the identity at $\mathbf{0}$. This makes $\mathbf{x}_H$ a saddle point with respect to position.

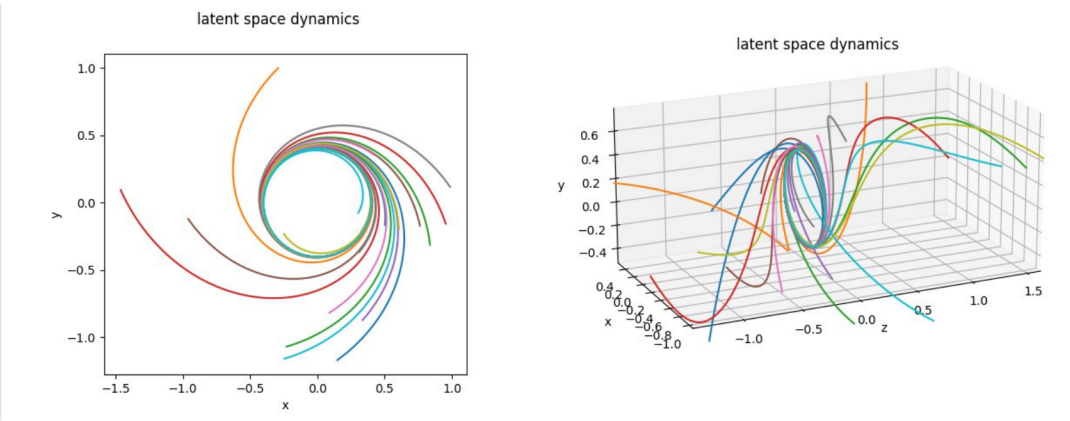


Figure 4.2: Limit cycle in the latent space (left). The limit cycle extended to three dimensions by adding a tanh attractor to form the Z axis (right).

# 5 Experiments

We apply the new architectures to two new tasks. The door opening task will be modeled by the architecture introduced for precise positioning movements. The task is partitioned into two skills with identical configurations and is described in section 5.2.

The architecture modeling cyclic movements is applied to train the swiping task. It is trained as a single skill using a limit cycle as movement primitive to drive the motion and described in detail in section 5.3.

We develop a user interface plugin that integrates into a control environment developed by Team Hector of the SIM group at TU Darmstadt[13]. The plugin integrates into the ROS program RViz. A detailed description of the plugin can be found in section 5.4.

## 5.1 Data Preparation

The demonstration data is a list of $m$ example trajectories of length $n$, each defined by transformations at time $t$.

$$\mathcal{D}_{\text{task}} = [\mathbf{d}]_m, \quad \mathbf{d} = [\mathbf{x}]_n, \quad \mathbf{d}^{(t)} = \mathbf{x}_t = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^\intercal & 1 \end{bmatrix} \tag{5.1}$$

These trajectories are then converted to the first cover of the tangent space and filtered using a Savitzky Golay filter with window size 25 and a polynomial degree of three, resulting in smooth trajectories.

$$\mathcal{D}_{\text{filt}} = [\text{savgol}(\mathbf{d})] \tag{5.2}$$

Since the demonstration trajectories are given without velocity information we obtain the derivative by filtering the trajectories position data using the same filter. This gives a smooth estimate of the velocity information. If the demonstration trajectories include

velocity information, filtering may be skipped, depending on the amount of noise in the data.

## 5.2 Door Opening

Opening a door works best by grabbing its handle. A robot arm needs a gripper to do this, with a control policy including a representation for the gripper. The $\mathrm{SE}(3)$ Lie group has been extended in the point-to-point architecture for this reason.

We assume the door handle position to be known. Also, since the training data is taken from a single door being opened, the radius of the handle with respect to the door hinge is assumed to be identical to the demonstrations and constant. The policy is trained with a constant handle rotation angle needed to unlock the door in mind. We assume the real-world door to have the same dimensions as the door used for generating the demonstrations. The skills learn the grabbing, twisting, and pulling motions.

### 5.2.1 Segmentation of the Movement

The door opening movement can be generally dissected into five specific sections.

1. Opening the gripper
2. Move the end effector to the door handle
3. Close the gripper to grab the handle
4. Turn the handle in an arc to unlock the door
5. Pull open the door

Since very precise positioning near the door handle is needed, we segment the movement into two separate skills at the time when the gripper hovers near the door handle. The first skill will (theoretically[8]) reach arbitrary precision with respect to the target frame around the door handle. This is due to the stability guarantees granted by the construction of the stable vector field. In the second skill the grabbing, twisting, and pulling will happen. This

[8] Precision on a real system will be limited due to the sim-to-real gap, caused by errors in the low-level trajectory control system

way the stability guarantees of the stable vector field method help position the gripper at the door handle precisely.

The target $\mathbf{x}_H$ of the first skill is the handle position to grab, and the target frame for the second skill is the position of the twisted handle at the opened door.

### 5.2.2 Training

The neural ODE is modeled using a fully connected neural network topology with two hidden layers using 512 features, seven input dimensions, and seven output dimensions and relu activations. The dynamics attraction parameters are predicted with fully connected neural networks as well, using one hidden layer with 100 features per layer and relu activations. The network has an input dimensionality of seven per definition of the dynamics function. We learn $\alpha(\mathbf{z})$ and $\omega(\mathbf{z})$, the attraction and rotation speed of the limit cycle, $\beta(\mathbf{z})$ is a vector predicting the attraction speed for the remaining degrees of freedom. The networks are trained simultaneously using the Adamax optimizer with the learning rate $1 \times 10^{-6}$ with a weight decay constant of $1 \times 10^{-5}$.

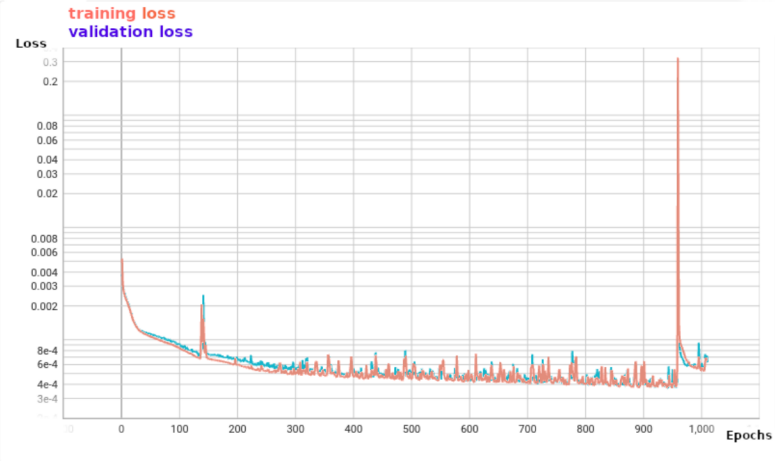


Figure 5.1: Training loss for the door opening skill. Note the noisiness and the spikes in the loss.

From the loss, it is apparent that the training is quite noisy. We tuned the hyperparameters in different ways, without a conclusive solution to the learning problems. The spikes

hint at gradient problems, tweaking $L_2$ parameter regularization did not help, however. Further examination is needed to locate the underlying issue.

### 5.2.3 Deployment in Simulation

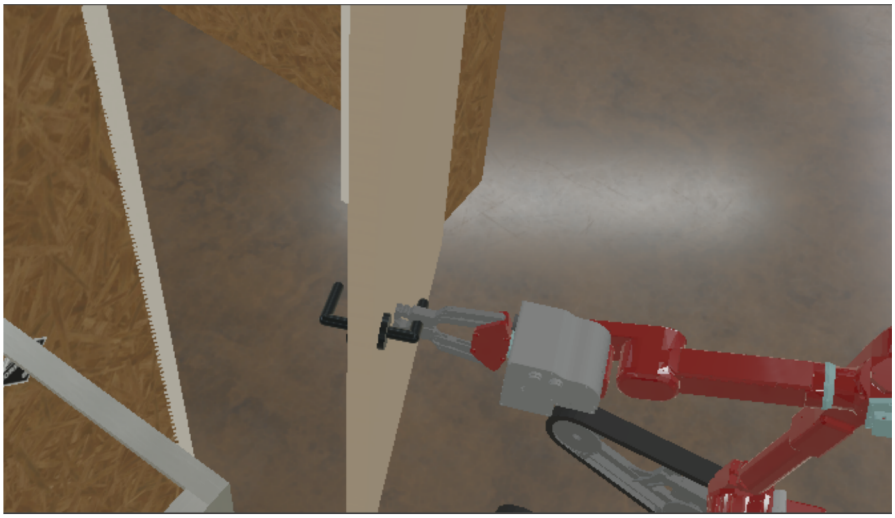

Figure 5.2: Door opening task, executed in simulation. Note that the robot does not match the aperture of the gripper to the handle exactly.

The model is able to express the basic movement of the task. The two phases, opening the gripper while going to the handle and grabbing, twisting, and pulling work and are able to open the door in simulation. The second skill shows problematic behavior while grabbing the handle, however. While the end effector position should stay stationary while closing the gripper, it moves slowly. Different training runs result in different speeds and directions of this movement, so clearly the method needs some adaption to represent movements showing very little to no motion in one domain while moving in another.

Running the policy in simulation shows this behavior, on the Webots simulator the gripper slides into the door handle while in PyBullet the door first moves in the wrong direction.

The demonstration data for the second task is very localized and spans a small area of the state space, since the operational space while grabbing the handle is constrained by the door geometry. The demonstrations lie on a thin band in the task space. Decreasing the amount of $\varepsilon$ did not result in a significant difference in performance.

## 5.3 Swiping

Repeated positions in operational space only occur on points lying on the limit cycle. Since we want to train cyclic motion this is the desired behavior. Thus there is no need to segment the swiping task into separate skills. The dynamic system defining the limit cycle is augmented with parameters that are trained during gradient descent. The swiping demonstrations consist of different trajectories where a circular swiping in an oval shape is performed. The demonstrations approach the circle from different directions.

### 5.3.1 Flow Constraint

The coefficient $\delta$ needs to be big enough to reduce the size where the "pinning" takes place to not interfere with the movement that should be represented by our learning process. This can be hard to spot and may prevent learning the skill altogether. We came to this realization and added additional plotting to visualize the morphing of the space by plotting the determinant of the jacobian $\det(J)$ of the morphing function as a heatmap.

Trajectories resulting from this error in the parametrization are displayed in Figure 6.1 in the appendix.

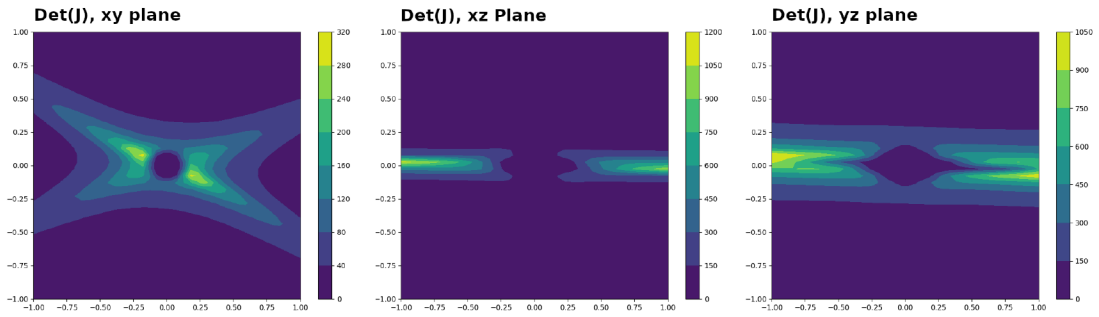


Figure 5.3: Heatmap of $\mathrm{Det}(\mathbf{J})$ for policy trained with coefficient $\delta$ being too low

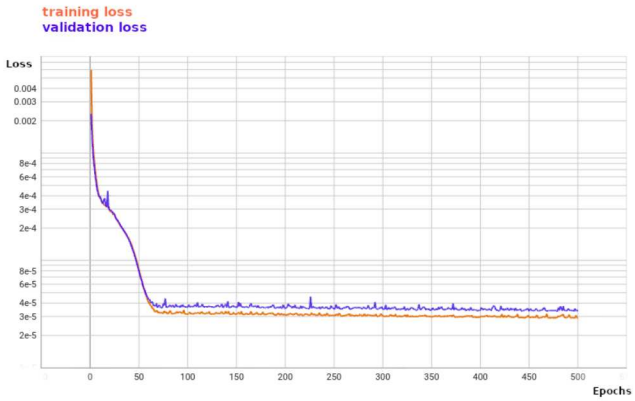


Figure 5.4: Training and validation loss of the swiping skill

### 5.3.2 Pre-Rotation of the Dynamics

We observed, that learning the rotation of the limit cycle converges to the best solution if we pre-rotate the limit cycle to the correct orientation first. We analyze the demonstration rotations manually to determine their orientation and construct a rotation matrix based on the analysis. This sets a good initialization for the learning process, and results in the convergence to a better solution.

### 5.3.3 Training and Deployment

For the function $\psi(\hat{\mathbf{x}}$ morphing the space, we choose a fully connected neural network topology with two hidden layers using 512 features, six input dimensions, and six output dimensions and ReLU activations. We use 50 demonstrations and use $\frac{2}{3}$ for training and $\frac{1}{3}$ for validation.

We predict the dynamics attraction parameters using fully connected neural networks using two hidden layers with ReLU activations as well. We use a network with 100 features per layer with an input dimensionality of six to predict $\alpha(\mathbf{x})$ and $\beta(\mathbf{x})$ with an output dimensionality of two. We learn $\omega(\mathbf{x})$, the rotation speed of the limit cycle using a neural network with the same topology, but using 200 features per layer, using input dimensionality of six and output dimensionality of one.

Figure 5.5: Swiping skill executed on the real robot

The networks are trained simultaneously using the Adamax optimizer with the learning rate $1 \times 10^{-4}$ with a weight decay constant of $1 \times 10^{-5}$.

The controller of the Telemax robot we used to deploy the skill is a workaround since the robot is not upgraded to the new control interface provided by the manufacturer. Our experiments on the robot show this clearly. While achieving very good tracking in simulation, executing the policy on the robot reveals noticeable errors in the spatial positioning. Tracking an exact height in the rotation plane poses difficulty. When observing the robot it is apparent, that the velocity is subject to large variations, resulting in oscillation-like movement. The tracking error is highly dependent on the kinematic configuration of the robot. Nevertheless, the basic movement is executed and the circular motion is performed properly.

We trained the skill with different sets of hyperparameters and different amounts of epochs. We have yet to see overfitting with this training configuration, the flow constraint and the limit cycle appear to be very strong inductive biases preventing this.

## 5.4 RViz Plugin for Simple User Interaction in the Hector UI

We developed a plugin for Rviz to provide a simple user interface for operators of the robot. At the mouse location, an arrow is created inside the scene that denotes a 3D coordinate where the swipe skill should be executed. We call this location the target frame and publish it using the tf2 system of ROS. The orientation is set upright since we assume the execution of the swipe skill on top of surfaces parallel to the ground right now. The skill is rotated around the Z-axis so that the oval is pointing away from the robot. Since the skill is defined in task space we could execute it in any orientation. Due to the noisiness of the 3D surface reconstruction however, executing the skill with a 3D orientation based on the surface normal is not robust enough right now due to excessive

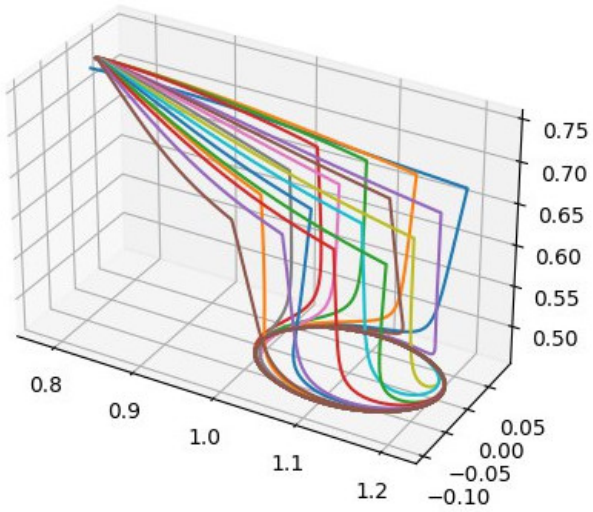


Figure 5.6: Swiping skill in simulation. The hover points are approached with tanh dynamics and are not part of the swiping skill.

noise in the reconstructed surface normal, depending on the location and makeup of a surface in the real environment. Extending the application of the launcher to this surface selection scheme would require further preprocessing tricks, for example, heavy smoothing of the sensor data or (user-defined) quantization of specific degrees of freedom in the rotation manifold. The plugin is written in C++ and built using the catkin build system that is part of ROS.

Figure 5.7 illustrates the layout of the RViz UI with our tools integrated, the gray arrow being a preview of the target position $\mathbf{t}_{raytrace}$.

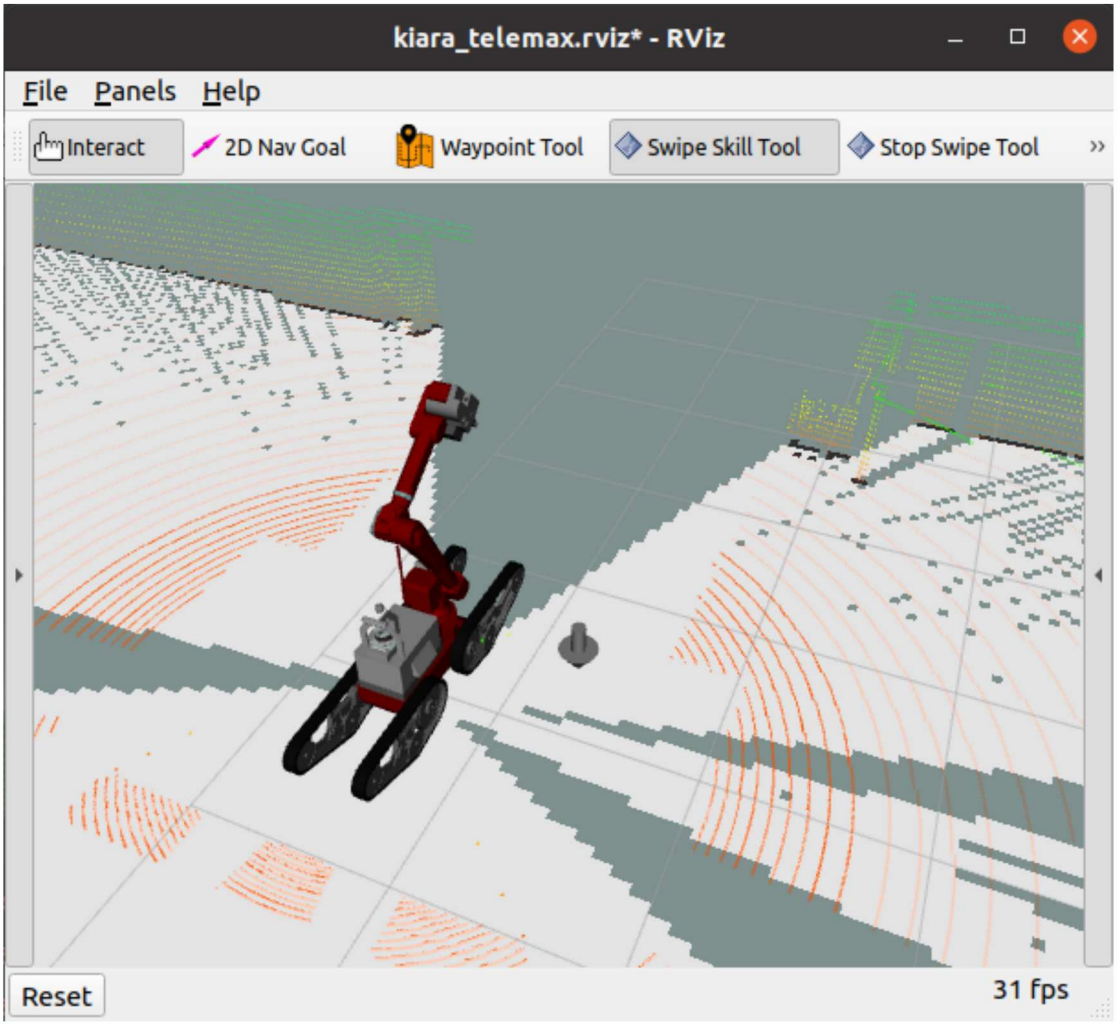


Figure 5.7: The user interface for triggering the swipe skill in RViz. On the top right, the target selector tool can be chosen, the button to its right aborts the action if it currently runs. The target selector is activated, with the target preview displayed with a gray arrow. RViz shows a reconstruction of the environment with a 2D map and a point cloud based on a LIDAR sensor. This scene reproduction is handled by the HECTOR user interface.

### 5.4.1 Target Publication with TF2

The plugin publishes the target of the swiping motion as a static transformation using the TF2 library[10]. The transformation is published at

`/tf2_static/swipe_target`

as an *TransformStamped* Message. The swipe target $\mathbf{x}_{target}$ is picked by raytracing the scene to get the picked point from the RViz 3D environment model. The rotational orientation is based on the robot base and turned such that the $z$ axis points upward while the $x$ axis points directly away from the robot[9].

$$\mathbf{x}_{target} = \begin{bmatrix} \mathbf{R}_{robot}\mathbf{R}_{xyz}(0,0,\phi) & \mathbf{t}_{raytrace} \\ \mathbf{0}^\intercal & 1 \end{bmatrix} \tag{5.3}$$

with

$\phi = \text{atan2}(y_{raytrace} - y_{robot}, x_{raytrace} - x_{robot})$

$\mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$

$R_{xyz}(\psi, \theta, \phi)$ the rotation matrix from cardan angles with the order xyz

This way the oval shape of the swiping motion is correctly aligned to the robot base.

### 5.4.2 Interfacing with the Control Policy

The control policy provides a *TriggerService* that can be accessed to start the skill.

`/start_swipe_skill`

[9] the coordinate system is right-handed

The service returns success if it is available and $\mathbf{x}_{target}$ is set to the target topic. This is visualized by a static green arrow in the scene reconstruction marking the target position for the swiping motion (visualized in Figure 5.8). If the swipe target is too far from the robot base or the robot is busy executing the last task, the request fails. This is displayed by a red selection arrow.

The task can also be aborted by a second trigger service at

```
/stop_swipe_skill
```

This topic is called by activating a second plugin, aborting the skill. When the execution of a skill is stopped, the robot returns to its initial position.

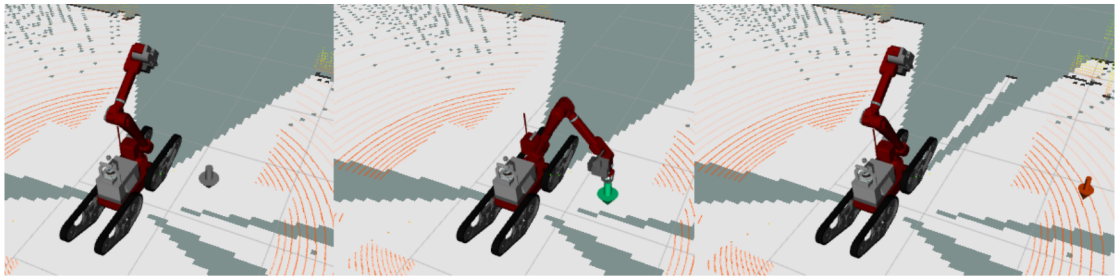

Figure 5.8: Selection preview in gray, accepted selection with movement execution in green and rejected selection in red, from left to right

## 5.5 User Feedback

We demonstrated our results to real-world users who deploy rescue robotics in real scenarios and asked them for feedback.

These remarks are based on informal discussions. One interesting observation was, that different domains sometimes require different approaches. Bomb defusing specialists for example are not concerned about the preservation of the environment. In radiation containment scenarios, however, the operators are very concerned about further contamination and non-destructive navigation and manipulation. Since teleoperation is very stressful to humans, any help in the form of automation systems is appreciated. It was also


Figure 5.9: Demo station to test UI interaction for the swiping skill

deemed important to keep user interfaces simple and only expose the necessary details to the operator. The simple point-and-click interface for probe-taking by swiping was well received.

A notable wish expressed by the teleoperators present at the KIARA presentation is the ability to add features by the end user. This fits nicely with our demonstration-based machine learning learning method. Motion capture systems have become cheap and capable [5][8][15], making it possible for end users to create and supply their own demonstration movements for training. Applying the method still requires domain knowledge, however. Movement segmentation needs to be done manually by a domain expert who understands the learning architecture, to choose the correct segmentation, flow constraints, and movement primitives.

# 6 Conclusions

We extend the work of the ImitationFlow and Stable Vector Fields on Lie groups (LieFlows) method [23][22] and employ it for two new tasks. We showed that the swiping task can be learned in a sufficiently precise manner. The door-opening task needs further improvement, however. Precisely positioning the end effector works fine at the target position, yet the end effector does not stay stationary when closing the gripper away from it. Interviews with real-world operators of disaster response robots confirmed the usefulness of our approach to (semi-) autonomous assistance functions.

This work can be extended in multiple ways: Task space trajectory tracking is currently done using very simple operational space control. Depending on the initial position of the robot it is possible, that the vector field pushes the robot towards a joint limit. Adding nullspace control in the case of deployment on a 7DOF robot or adding control mechanisms in joint space to avoid collisions and joint limits would help make policy execution robust with respect to the starting position so that the configuration of the robot during skill execution is independent on its initialization. The version of the Telemax robot deployed at the SIM lab is lacking a precise controller due to a missing software and hardware upgrade, therefore we could not evaluate the policies trained for positioning without considerable risk.

Further examination is needed to learn why the door-opening task does not converge to a better model during training. Experimenting with different flow constraints is an obvious starting point for this. Generalization of the skill with respect to different door geometries is not incorporated yet, since handle position with respect to the door hinge is assumed to be consistent between demonstration movements and the deployment scenario. This could be done by introducing an additional parameter into the learning process encoding the distance between the handle and the hinge. Also, the door-opening task is currently defined with the assumption of a static base that doesn't move during the execution of the skill. Since real doors often use retaining springs that keep them closed when not used simply pulling the door open does not work for every door. Discussions with people

working as teleoperators made it apparent, that the task of opening a door can be very involved, including the operation of the flippers to keep the door in place while the robot arm is repositioned and a complete repositioning of the base. In practice, it's useful to keep doors open that have been opened. There are specialized door wedges designed by these task forces to lock the doors once opened. Combining these aspects to create a fully autonomous door-opening routine can be approached by extending the manifold used for pose representation. By adding the robot base as a SE(2) coordinate to the state LieFlows can be applied to train a more comprehensive skill.

Augmenting the operational space controller with additional safety layers like collision avoidance logic, will be useful for this extension of the door-opening skill as well. Since the robot will be able to move in a larger space, the risk of self and environment collision increases. Therefore a more careful control scheme will be needed at some point. As we have seen in subsection 3.3.4, some tasks have to be segmented into multiple skills. Analyzing a set of demonstrations and pre-processing them accordingly by time-dependent segmentation would generalize this method to arbitrary demonstration movements. This removes the need for a domain expert to set up the training configuration and annotate the skill boundaries manually.

# Bibliography


[1] Pieter Abbeel and Andrew Y. Ng. “Apprenticeship learning via inverse reinforcement learning”. In: *Proceedings of the Twenty-First International Conference on Machine Learning*. ICML ’04. Banff, Alberta, Canada: Association for Computing Machinery, 2004, p. 1. ISBN: 1581138385. DOI: 10.1145/1015330.1015430. URL: https://doi.org/10.1145/1015330.1015430.

[2] Jeffrey Abouaf. “Trial by fire: teleoperated robot targets Chernobyl”. In: *IEEE Computer Graphics and Applications* 18.4 (1998), pp. 10–14.

[3] Michael Bain and Claude Sammut. “A Framework for Behavioural Cloning”. In: 15 (Mar. 2000).

[4] S. Calinon. “A Tutorial on Task-Parameterized Movement Learning and Retrieval”. In: *Intelligent Service Robotics* 9.1 (2016), pp. 1–29. ISSN: 1861-2776. DOI: 10.1007/s11370-015-0187-9.

[5] Anargyros Chatzitofis et al. “DeMoCap: Low-cost marker-based motion capture”. In: *International Journal of Computer Vision* 129.12 (2021), pp. 3338–3366.

[6] Boston Dynamics. *Door Manipulation with the Spot Arm*. https://support.bostondynamics.com/s/article/Door-manipulation-with-the-Spot-Arm. [Online; accessed 8-July-2024]. 2023.

[7] Luca Falorsi et al. “Reparameterizing distributions on lie groups”. In: *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR. 2019, pp. 3244–3253.

[8] Matthew Field et al. “Motion capture in robotics review”. In: *2009 IEEE international conference on control and automation*. IEEE. 2009, pp. 1697–1702.

[9] Tamar Flash and Binyamin Hochner. “Motor primitives in vertebrates and invertebrates”. In: *Current Opinion in Neurobiology* 15.6 (2005). Motor sytems / Neurobiology of behaviour, pp. 660–666. ISSN: 0959-4388. DOI: https://doi.org/10.1016/j.conb.2005.10.011. URL: https://www.sciencedirect.com/science/article/pii/S0959438805001601.

[10] Tully Foote. “tf: The transform library”. In: *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*. Open-Source Software workshop. Apr. 2013, pp. 1–6. DOI: 10.1109/TePRA.2013.6556373.

[11] Oussama Khatib. “A unified approach for motion and force control of robot manipulators: The operational space formulation”. In: *IEEE Journal on Robotics and Automation* 3.1 (1987), pp. 43–53.

[12] Jens Kober and Jan Peters. “Learning motor primitives for robotics”. In: *2009 IEEE International Conference on Robotics and Automation*. 2009, pp. 2112–2118. DOI: 10.1109/ROBOT.2009.5152577.

[13] Stefan Kohlbrecher et al. “Hector Open Source Modules for Autonomous Mapping and Navigation with Rescue Robots”. In: *RoboCup 2013: Robot World Cup XVII*. Ed. by Sven Behnke et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 624–631. ISBN: 978-3-662-44468-9.

[14] J. M. Lee. *Introduction to Smooth Manifolds*. Germany: Springer, 2006.

[15] Thomas B Moeslund, Adrian Hilton, and Volker Krüger. “A survey of advances in vision-based human motion capture and analysis”. In: *Computer vision and image understanding* 104.2-3 (2006), pp. 90–126.

[16] R.M. Murray et al. *A Mathematical Introduction to Robotic Manipulation*. Taylor & Francis, 1994. ISBN: 9780849379819. URL: https://books.google.de/books?id=D_PqGKRo7oIC.

[17] Stephane Ross, Geoffrey Gordon, and Drew Bagnell. “A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning”. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, Nov. 2011, pp. 627–635. URL: https://proceedings.mlr.press/v15/ross11a.html.

[18] Stefan Schaal. “Dynamic movement primitives-a framework for motor control in humans and humanoid robotics”. In: *Adaptive motion of animals and machines*. Springer, 2006, pp. 261–280.

[19] Stefan Schaal. “Learning from Demonstration”. In: *Advances in Neural Information Processing Systems*. Ed. by M.C. Mozer, M. Jordan, and T. Petsche. Vol. 9. MIT Press, 1996. URL: https://proceedings.neurips.cc/paper_files/paper/1996/file/68d13cf26c4b4f4f932e3eff990093ba-Paper.pdf.

[20] Fritz Schwarz. *Algorithmic Lie Theory for Solving Ordinary Differential Equations*. Sankt Augustin, Germany: Chapman & Hall/CRC, 2007.

[21] Joan Sola, Jeremie Deray, and Dinesh Atchuthan. “A micro Lie theory for state estimation in robotics”. In: *arXiv preprint arXiv:1812.01537* (2018).

[22] Julen Urain, Davide Tateo, and Jan Peters. “Learning Stable Vector Fields on Lie Groups”. In: *IEEE Robotics and Automation Letters* 7.4 (2022), pp. 12569–12576. DOI: 10.1109/LRA.2022.3219019.

[23] Julen Urain et al. “ImitationFlow: Learning Deep Stable Stochastic Dynamic Systems by Normalizing Flows”. In: Oct. 2020, pp. 5231–5237. DOI: 10.1109/IROS45743.2020.9341035.

[24] J.S. Yuan. “Closed-loop manipulator control using quaternion feedback”. In: *IEEE Journal on Robotics and Automation* 4.4 (1988), pp. 434–440. DOI: 10.1109/56.809.

[25] Martijn J. A. Zeestraten et al. “An Approach for Imitation Learning on Riemannian Manifolds”. In: *IEEE Robotics and Automation Letters* 2.3 (2017), pp. 1240–1247. DOI: 10.1109/LRA.2017.2657001.

# Appendix

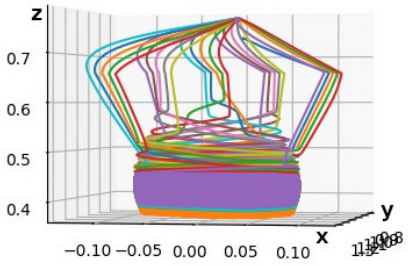


Figure 6.1: Trajectories demonstrating erratic behavior exhibited by skill trained with $\delta$ being too low. This creates a sphere that is unaffected by the diffeomorphic mapping $\Phi$

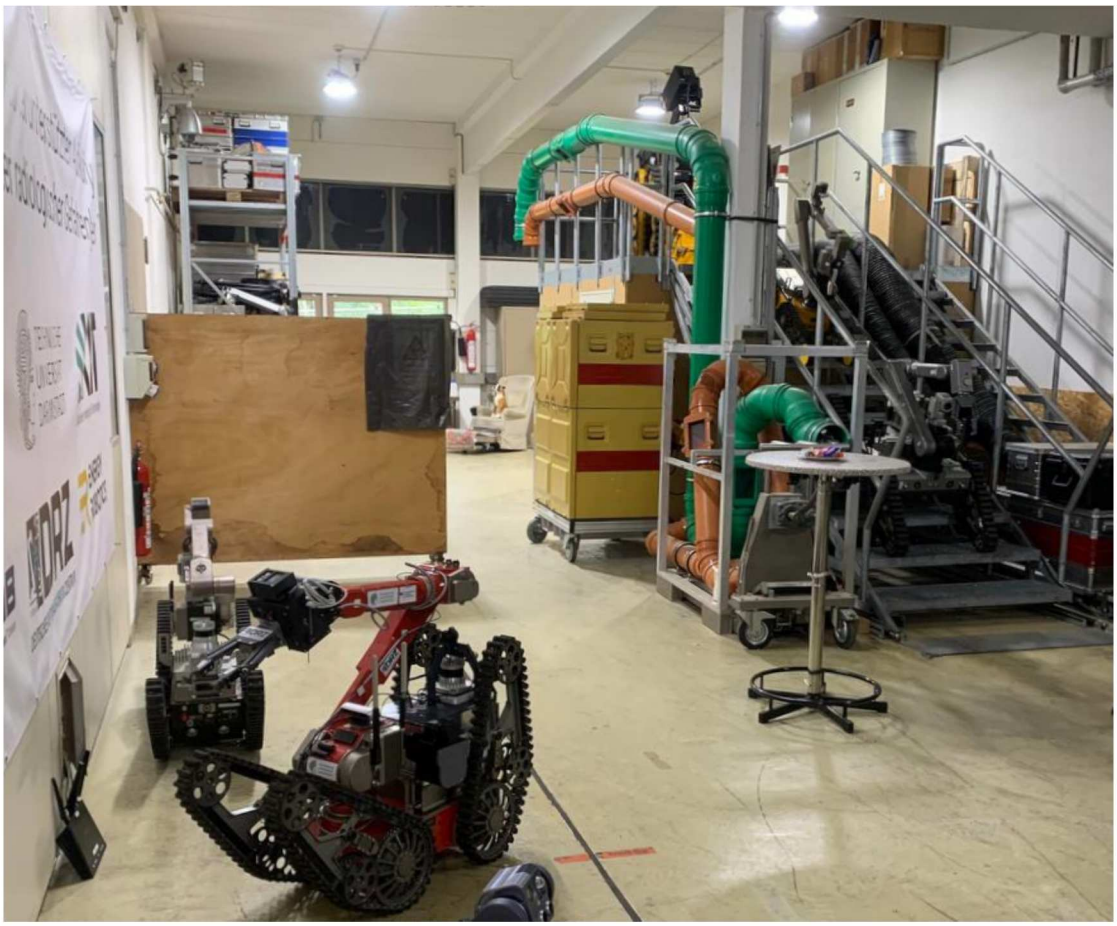

Figure 6.2: Test scenario at KHG. This is virtually reproduced in Webots.