Modelling and learning of building blocks for shared control templates.

handed in
MASTER'S THESIS

B.Sc. Sebastian Bader

Human-centered Assistive Robotics
Technical University of Munich

Supervisor:	M.Sc. Gabriel Quere (DLR), Dr. Freek Stulp (DLR), Univ.-Prof. Dr. Donghui Lee
Start:	17.05.2021
Intermediate Report:	19.10.2021
Delivery:	30.11.2021

With my signature below, I assert that the work in this thesis has been composed by myself independently and no source materials or aids other than those mentioned in the thesis have been used.

Place, Date	Signature

TECHNISCHE UNIVERSITÄT MÜNCHEN
Human-centered Assistive Robotics
UNIV.-PROF. DR.-ING. DONGHEUI LEE

May 21, 2021

M A S T E R ' S T H E S I S
for
Sebastian Bader
Student ID 03675381, Degree EI

Modeling and learning of building blocks for shared control templates.

Problem description:

The goal of this thesis is to improve the automation of the design of Shared Control Templates skills [1]. Those skills are represented as a Finite State Machine where each state can define an Input Mapping (IM, mapping of low-dimensional user inputs to main directions of motions of the robot relative to some feature frame) and Active Constraints (AC, synonym to Virtual Fixtures, typically geometric constraints on the robot end-effector). States are connected via transitions that are followed on certain conditions.
For this purpose, SCT building blocks like feature frames, IM and transitions will be extracted from multiple demonstrations. The thesis will apply an iterative approach, first experimenting with the identification of a single IM on low-dimensional generated data with a reduced set of models, then progressively increase the complexity of the data (towards 6 DoF robot demonstrated data) and the complexity of the models (using multiple IMs generating a segmentation of the data). Learning of the transitions and identifying new features frames will be investigated.

Tasks:

- Literature review
- Identify single feature frames
- Identify single input mappings with feature frames
- Identify different feature frames and input mappings
- Learn the transitions of the states between SCT skills

Bibliography:

[1] Gabriel Quere, Annette Hagengruber, Maged Iskandar, Samuel Bustamante, Daniel Leidner, Freek Stulp, Joern Vogel, and Jörn Vogel. Shared control templates for assistive robotics, 2020.

Supervisor:	M.Sc. Gabriel Quere (DLR), Dr. Freek Stulp (DLR), Univ.-Prof. Dr. Lee
Start:	17.05.2021
Intermediate Report:	19.10.2021
Delivery:	30.11.2021

(Prof. Dr. Lee)
Univ.-Professor

Abstract

In the field of assistive robotic robots aim to help users with impairments to execute daily living activities in everyday household environments. Controlling the high degrees of freedom of those robots directly, with a joystick, is often tedious and can cause a high cognitive workload. Using Shared control methods, like Shared Control Templates [GQ20], provide task-specific guidance and constraints to facilitate control. However, designing SCTs requires robotic expertise and was previously hand-crafted. To make such designs easier and faster, we extend the SCT approach to facilitate the generation of SCT templates on the basis of demonstrations. We show the capabilities of this approach in a set of daily living tasks such as opening a drawer and pouring with the wheelchair-mounted robot EDAN [VHI+20].

Contents

Chapter 1

Introduction

In the field of assistive robotics, robots nowadays are capable of doing a lot of impressive tasks such as helping people with disabilities to restore manipulation capabilities by enabling them to perform daily living tasks. To perform those tasks in human environments robots need to be capable of sensing, planning and acting accordingly. However, pre-programming a robot for all potential uses cases and especially unpredictable events is impossible. Even only extending a task by adding a new skill requires more time, effort and resources. Therefore, in the field of Human-Robot Interaction researchers look for potential solutions. In this field, several architectures are designed for humans to interact with (partially) autonomous robots. On the one hand, they try to overcome the problems of fully autonomous robots that require an accurate model of the world they interact with. This model might be heavily constrained by the validity of its environment modelling and might not allow adaption to new environments [MFD+17]. Furthermore, it has been shown that users prefer to be in control rather than have a fully autonomous robot[KHKCG+12]. On the other hand, they improve on manual control (eg. by a Joystick) of robots as it can be slow, challenging and might require many control mode switches depending on the task [HHS16].
These design methods are referred to as Shared Control (SC) or Shared Autonomy (SA). This is a symbiosis between human skills and robot skills and combines the high-level cognitive understanding of humans and the rapid computation of robots [She92, CVW90]. The difference between SC and SA is the degree of autonomy the system can comprise. In SA robots are capable of adapting the autonomy based on understanding the environment and human actions and intentions. The dynamics of an autonomous system can be described as follows:

$$\begin{aligned} \dot{x}(t) &= f(x(t), u(t)) \\ u(t) &= h_\theta\left(u_h(t), u_a(t); \theta(t)\right) \end{aligned} \tag{1.1}$$

where x is the state of the robot, u is the user input and h is an arbitrary function that combines/modulates autonomous control and user input. h_θ models the autonomy level of the system and θ is the robot's understanding of the human or environment

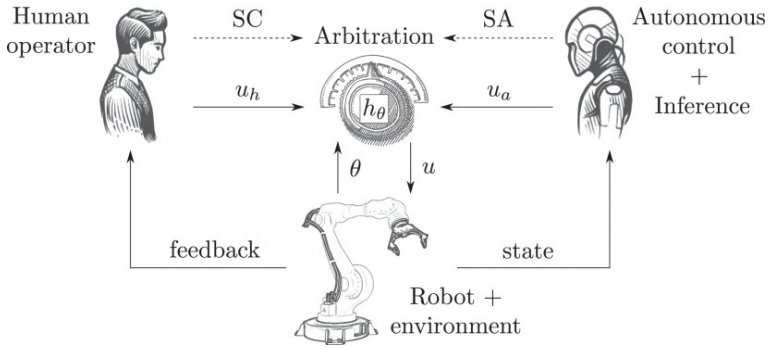


Figure 1.1: Autonomy model of robot systems. Taken from [SCN+21].

(see Figure 1.1). In the case of SC, the arbitration function reduces to $h(u_h(t), u_a(t)$ as it only depends on human/autonomous control inputs and no other external variables [SCN+21].
A popular field where SC and SA are used is assistive robotics, where, for example, robots help people with disabilities to conduct activities of daily living. One implementation is the research project EDAN, an Electromyography (EMG) Daily AssistaNt [VHI+20]. This is an electric powered wheelchair with a mounted robot arm that can be controlled via a surface EMG-based interface or a joystick. Manually controlling everything might be difficult as it leads to a high cognitive load of the user. For this purpose, SC and SA can help the user by guiding a specific task. For example, the robot helps users trying to pour a bottle of water into a glass or open a shelf. The guidance comprises different approaches that extract and use information from the human operator, which is in charge of trading control. Usually, these laborious design procedures require robotic expertise. To make such designs easier and faster, this master's thesis is interested in automating the process of setting up new skills based on Shared Control Templates. With the overall goal of reducing the amount of work required by programmers to create a desired shared control skill, a system for learning skills from demonstrations is desired. Improving on previous work [GQ20] and [QHI+20], this thesis aims to create an automated pipeline to learn these skills from demonstrations and explore to what degree automation is possible. The contribution of this thesis is three-fold:

1. Improve the formalism to automatically generate SCT,

2. Design two algorithms to model states, Input Mappings and Active Constraints,

3. Test both algorithms on a set of experiments on the real robot.

This thesis is structured as follows: Chapter 2 introduces some related work in this field. Linear Input Mappings models and vector fields models are introduced in

Figure 1.2: A demonstration of Edan. Taken from [VHI+20].

Chapter 3. Chapter 4 describes the conducted experiments and Chapter 5 discusses them. Chapter 7 summarizes the master's thesis and outlines future work.

Chapter 2

Related Work

Several complementary approaches have been developed in Shared Control where some approaches predict and leverage user information, while others attempt to constrain the available workspace.

2.1 Predict and Leverage human intention

First, one can realize shared control with policy blending [DS13]. Dragan et al. study everyday manipulation tasks where the robot tries to predict and augment the user input. Two policies, one from the robot for the most likely goal and one from the user through a device (ie. GUI, joystick) are blended. The amount of assistance is determined by the prediction confidence of the goal.
In [JSB15] Javdani et al. formulate the SC problem as a Partially Observable Markov Decision Process (POMDP) with uncertainty over the user's goal. Rather than splitting the prediction and then assisting for a single goal, this approach assists for an entire distribution of goals. They use hindsight optimization in order to find the best assistance command and provide assistance even if the confidence is low.
A first model-free approach is made by Oh et al. in [OKTM19], where they define a differentiable strategy. It attempts to integrate motion generation, user intention inference, and arbitration to train a shared autonomy system using reinforcement learning. In their user experiments, the approach had problems identifying the correct intention and resulted in incorrect predictions.
A more sophisticated approach was conducted by Reddy et al. in order to overcome unknown dynamics, user policies and a particular goal representation [RLD18]. It tries to learn an end-to-end mapping from observations of the environment and user inputs to agent actions values in form of a Q-function. The supervision is solely based on task rewards which consist of known terms for every state and terminal rewards by succeeding or failing the task.
Mehr et al. present a method for inferring end-effector constraints online from operator input and a confidence-based method that assists the user in maintaining those constraints by learning a mapping to the task space [MHD16].

2.2 Restricting the environment

Second, some approaches try to exploit the environment by constraining the available workspace. This control method is usually referred to as active constraints or virtual fixtures [BDB14]. They can take various forms and can be applied to any dimension of the task space, usually the end-effector position, orientation, or full pose. Those constraints range from simple lines, planes, axial rotations or fixed joints [SZG18] to generalized cylinders [AC18].
A more implicit restriction is made with Latent Actions or Input Mappings (IM). The goal is to find a mapping from high-dimensional robot actions to low-dimensional controls. With so-called learned latent actions, Losey et al. provide a low-dimensional embedding of high-dimensional robot behaviour, where they change the meaning of human inputs depending on the confidence in the goal [LSM+20, LJL+21].
In [ACHT14], Alexandrova et al. attempt to learn multi-step manipulation tasks from a single demonstration. To do this, they learn so-called "landmarks" that represent target frames (target position and orientation in Cartesian space) with respect to relevant objects. An interactive graphical user interface allows the user to modify the target frames and landmarks a posteriori.
Another aspect within the environment is to capture task-specific information. Learning from demonstrations (LfD) therefore has become a very popular approach. Robots acquire new skills by learning to imitate an expert or, likewise, non-experts to teach robots manipulation [ACVB09]. These methods have helped teach manipulators a variety of skills, such as robotic surgery [SMO+21] or assistance with daily activities [MNM18]. The recorded trajectories are usually clustered based on distance metrics, density [BTTT18], or predominantly statistically using, for example, Gaussian mixture models (GMM) and Gaussian mixture regression (GMR) [CGB07, Cal16] or GMR in Riemannian Manifolds [ZHS+17]. In order to generalise it is important to learn the representation in terms of relevant frames of reference, i.e. objects being manipulated [Mas81, LBH12]. In approaches like [AM16] the frames of reference are often given. Therefore, there is a necessity to extract potential candidate feature frames directly from the recorded data. A first attempt to cluster trajectories with vector fields was done by Ferreira et al. By using Vector Field k-Means they try to identify streams of trajectories as vector fields and capture velocity and magnitude insights [FKSS13]. Their approach makes use of a grid-based approximation. Baro et al. use Gaussian random fields in [BM17] and [BM18] to avoid this discretization. Making use of GMMs or vector fields can also fall into the first category where they act as a policy to blend commands.

2.3 Multiple models

Third, some approaches attempt to combine multiple models simultaneously, e.g. by constraining the end-effector with multiple models and using different Input Mappings.

One example is C-Learn (Constraints learning), which uses a multi-model representation to build arbitrary capabilities [PDS17]. They attempt to learn hard geometric constraints from a single demonstration using machine learning and motion planning techniques. The resulting finite state machine consists of a sequence of keyframes and a set of geometric constraints per keyframe. A generalization of this approach is [CBO18], where they also learn nonlinear constraints defined not only by the state space but by an arbitrary constraint space.
Another approach on which this work builds is the Shared Control Template (SCT) framework, introduced by Quere et al. in [GQ20]. The framework provides templates to define task-specific capabilities that provide guidance during task execution. The system leverages knowledge about the task and objects in the world and guides the user by mapping input commands to task-relevant robot motions. A follow up approach is [QHI+20], where Quere et al. focus on learning Active Constraints.

Chapter 3

Background

This chapter briefly summarises the main concepts of SCTs and describes its pipeline, explains relevant aspects of EDAN and describe the process of learning SCTs from demonstrations.

3.1 Example

Before starting with the SCT formalization, the pouring task will be explained to grasp the main concepts. Within this task, users try to pour a bottle into a mug and the movements consist of different phases. In the first phase, the user moves the bottle closer to the target. In the second phase, the user starts to slowly rotate towards the target and in the last state, the pouring takes place. In all these phases the bottle needs to fulfil geometric constraints like to not tilt too far, which can be expressed in different coordinate frames. In addition, to simplify the users'

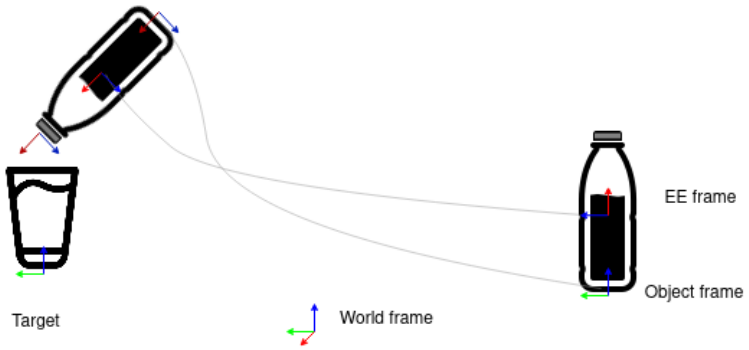


Figure 3.1: Conceptual image of the pouring task, where the user pours a bottle into a target. It shows the world frame as well as the object and end-effector frame throughout the task.

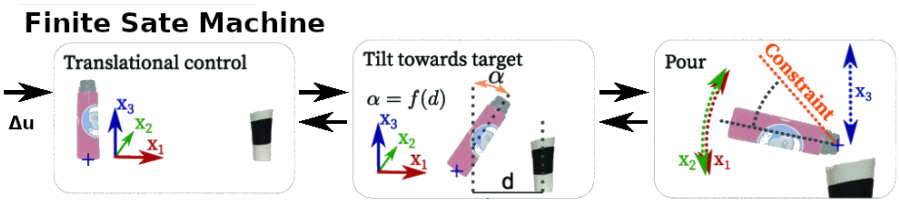


Figure 3.2: Schematic overview of SCT modelling the pouring tasks. User commands Δu are processed by a Finite State Machine with different states, Input Mappings and Active Constraints. Adapted from [GQ20].

execution, the input commands can be mapped in different motion directions.

3.2 SCT pipeline

The pouring skill from above can then be described more formally within the SCT pipeline (see Figure 3.2).

Input

At the start of the pipeline is the user input $\Delta u \in \mathcal{R}^3$, which comes either from an sEMG interface (see [VHI+20]) or a joystick. The space $\mathcal{R}^3$ allows translational and rotational motions and the commands will be further processed within a state machine.

States

A state is a structure optionally composed of robot parameters (end-effector configuration, control parameters, etc), Input Mappings (IMs), and Active Constraints (ACs).

Formalism

An SCT models different states and transitions, which resembles a finite state machine. The transitions depend on metrics such as estimated force thresholds, distances, or timeouts. For example in the pouring skill, the transitions depend on the distance between mug and bottle. There is a defined start state, ie. when a skill starts and only one state can be active. It also defines a distinct end state that finishes the execution of a skill once its reached.

Input Mappings

IM models constrain the velocity commands on the end-effector and maps user velocity commands $\dot{\mathbf{u}}(t)$ to the end-effector velocity $\dot{\mathbf{g}}(t)$ at each time step. They are

depending on the end-effector pose and result in an end-effector increment $\mathbf{\Delta g(t)}$ (i.e. small end-effector motion) called displacement. Those displacements are continuously applied to the end-effector pose $\mathbf{g}_\epsilon$ and create a trajectory. In the first state of the pouring task (translational control), the user input is mapped to the x_1, x_2, x_3 translation of the grasped thermos. In the second state (Tilting towards the target), the IMs maps two control signals to translation and one control signal to rotation. In the last state where the pouring happens, the IMs map only one signal to translation and two signals to rotation.

Active Constraints

AC models are used to restrict and guide the user along with the task. An AC is a geometric limit that affects the robots end-effector pose and restricts its space of permissible poses in the special Euclidean group SE(3). They can comprise scalars, functions or manifolds like cones or cylinders and are independent of the starting position. In our pouring task, ACs are used to keep the bottle upright at a specific height in the first state. In the second state and third steps, ACs restrict the end-effector pose to not tilt too far by setting a maximum tilt angle to prevent spillage. Both of these elements assist the user while executing a task and both define a manifold of allowable end-effector poses at a given state $G(s)$, where $G \subset SE(3)$ and s is a state in the FSM.

Output

The output of the template are homogeneous transformation matrices representing end-effector poses $\mathbf{H}_t$. Those poses are given to a task space interpolator like a Cartesian impedance controller to apply them to the robot.

Summary

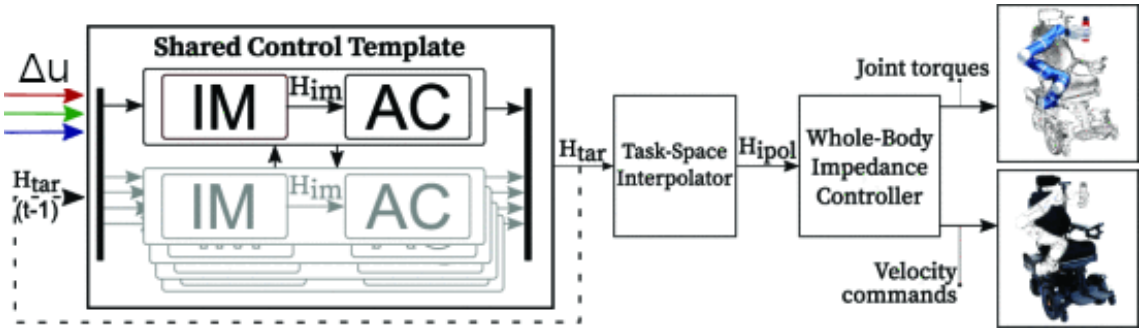


Figure 3.3: Overview of the system architecture. User commands are processed by Input Mappings and Active Constraints to obtain a new end-effector pose. Taken from [GQ20].

To sum up, SCTs provide task-aware mappings and constraints for each state of a skill to help users achieve a task. The state machine triggers transitions between them depending on the progress. An SCT is described in a YAML file to easily adapt and modify without changing the underlying framework (see [GQ20] for examples). Figure 3.3 shows the overall system architecture combining all different aspects within a SCT for the research platform EDAN. In addition to the task-space interpolator, EDAN also includes a whole-body controller that coordinates the motion of the wheelchair (2DoF) together with a modified DLR Light-Weight Robot III (8DoF) and five-fingered DLR-HIT hand mounted [VHI+20] to follow the target pose H_{tar} [IQH+19].

3.3 System integration

SCTs and this master's thesis are integrated into the EDAN system. Therefore, this section outlines important aspects such as the world and object representation and the user interface.

World Representation and Object Database

Following [LBH12], SCTs use an object-centric action representation. Skills are defined for object classes and their inheritance in a database. These object classes can be inherited, e.g. a thermos can derive from a virtual class bottle, which can derive from a container. A centralized world representation then handles object instances and describes the robot belief of the current world state. EDANs perception systems localises available instances and estimates their pose [VHI+20].

User Interface

EDAN also uses a high-level graphical user interface on a mounted display which provides various information (see Figure 3.4). For SCTs, a continuous 3D velocity interface is used with a space mouse or sEMG-based interfaces as input.

3.3.1 Learning SCTs from demonstrations

In previous work, LfD was used to learn new skills. In this pipeline, several demonstrations were gathered to help capture variability within the task. They contain end-effector trajectories and forces applied to them. Kinesthetic teaching or teleoperation is used to obtain the kinematic part. Force sensors or joint torque sensors are used to measure external forces. These recordings are then with respect to the target object, subsampled a given distance and preprocessed with dynamical-time-warping (DTW) [BC94] to align the timestamps. Afterwards, they were usually segmented based on curvature or force contact points with different ACs fitted. So

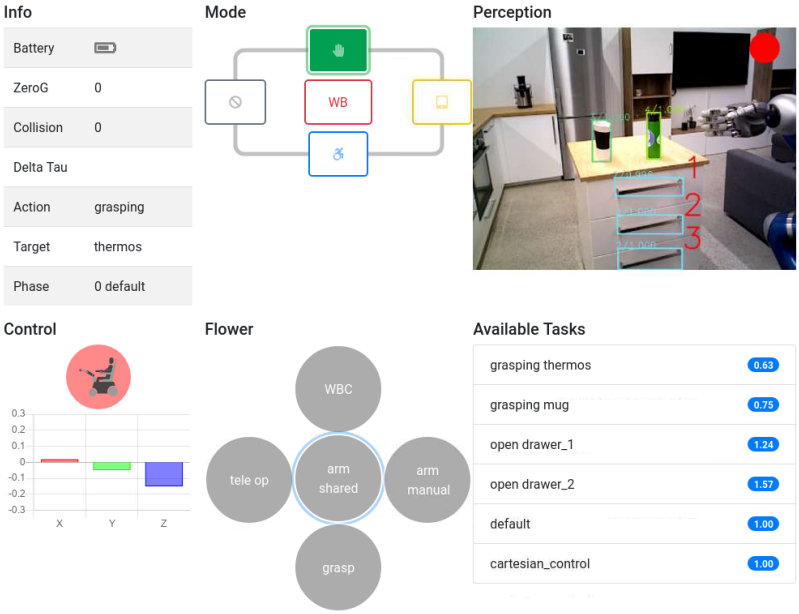


Figure 3.4: The user interface provides general information (Info), the controlled device (Mode), perception module feedback (Perception), the measured control signal (Control), different default tasks (Flower) and a list of available tasks from the world state representation (Available Tasks). Taken from [GQ20].

far, neither learning IMs nor transitions have been considered in the pipeline, which will be explored in the following.

Chapter 4

Methodology

A challenge within SCTs is to hand-craft IMs, ACs and states. In this thesis, we develop two different approaches to facilitate the generation of SCTs. The first method is Linear Input Mappings that tries to identify the main direction of motion and learn transitions based on them. The second method uses vector fields to generate IM and AC with a smooth transition based on their superimposition. All of the poses and transforms used are elements of SE(3), which means their representation (pose, matrix, ...) is interchangeable.

4.1 Linear Input Mappings

The most elementary IM is a 1-to-1 mapping, where one user input DoF controls one motion DoF. This can be achieved with a matrix with at most one element per line and column. For EDAN, where the user input dimension is $\mathcal{R}^3$, it gives a $\mathbf{M} \in \mathbb{R}^{6\times3}$ matrix to control 6-DoF motion. This linear input model is applied as IM within a skill (see Figure 3.3) and computes with the user input $\mathbf{\Delta u}$ a displacement $\mathbf{\Delta T}$ at every time step. This displacement is then applied on the current end-effector pose to get a new pose.

$$\mathbf{H}_{IM} = \mathbf{M} \cdot \mathbf{\Delta u} \in \mathbb{R}^{6\times1} \tag{4.1}$$

An example of such a Linear IM is displayed in Figure 4.1. It maps the user input in a translation in x and a rotation in y. For EDAN, the first three columns can be seen as a mapping to the translation vectors and the last three columns as a mapping to the rotation vectors.

The proposed idea for this algorithm is to define a set of possible Linear IMs $\mathbf{M} \in \mathcal{M}$, that control 1-3 motion DoF and can be used by SCTs as IMs. This set can then be used to identify the main direction of motion within trajectories.

$$\mathbf{M}_{txry} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figure 4.1: Example of a Linear IM with a translation in x (tx) and a yaw-rotation (ry).

4.1.1 Identification of Linear IMs

To identify Linear IMs, this approach looks at adjacent poses from the subsampled recorded trajectories. Those discrete poses $\mathbf{g}_\epsilon$ of the end-effector consist of three Cartesian coordinates (x,y,z) and orientation as quaternion (q_0, q_1, q_2, q_3) expressed with regards to the target. The displacement between the current pose $\mathbf{g}_{\epsilon,t}$ to their next pose $\mathbf{g}_{\epsilon,t+1}$ is used in order to identify the main direction of motion within a time step t. This displacement is given with the transformation $\mathbf{\Delta T}_t$ with

$$\mathbf{g}_{\epsilon,t+1} = \mathbf{g}_{\epsilon,t} * \mathbf{\Delta T}_t \tag{4.2}$$

where $\mathbf{\Delta T}_t$ consists of a rotation matrix $\mathbf{R}_t$ and translation $\mathbf{t}_t$:

$$\mathbf{R}_t = \mathbf{R}_{g_\epsilon,t+1} \cdot \mathbf{R}_{g_\epsilon,t}^T \tag{4.3}$$

$$\mathbf{t}_t = \mathbf{t}_{g_\epsilon,t+1} - \mathbf{R}_t \cdot \mathbf{t}_{g_\epsilon,t} \tag{4.4}$$

The rotation difference is transformed back to Euler angles and together with the translation difference, it forms an Euler pose. Even though the Euler pose has some drawbacks like non-unique solutions and the gimbal lock [SN03], it is yet a practical way to define small displacement.
Afterwards, the approach checks by using any command $\mathbf{\Delta u}$, which particular Linear IM $\mathbf{M} \in \mathcal{M}$ best approximates the displacement $\mathbf{\Delta T}_t$, and solves following linear equation:

$$\mathbf{M\Delta u} = \mathbf{\Delta T}_t \tag{4.5}$$

The system has more variables m than equations n, which means it is an overdetermined: $m > n$. This means it has full rank and a solution for

$$\mathbf{\Delta u} = \mathbf{M}^+ \mathbf{\Delta T}_t \tag{4.6}$$

exists with the Moore-Penrose inverse $\mathbf{M}^+$. The Moore-Penrose inverse is defined as

$$\mathbf{M}^+ = (\mathbf{M}^T\mathbf{M})^{-1}\mathbf{M}^T \tag{4.7}$$

The residual is given by

$$\mathbf{r}_t = \mathbf{M\Delta u} - \mathbf{\Delta T}_t \tag{4.8}$$

To show how well $\mathbf{M}$ approximates the displacement, the transform norm is used. This norm for a given pose $\mathbf{g}$ is

$$||\mathbf{g}||_{TF} = \|\mathbf{g}_{trans}\|_2 + \alpha \cdot \mathbf{g}_{rot} \tag{4.9}$$

where $\mathbf{g}_{trans}$ is the translation part and $\mathbf{g}_{rot}$ is the angle-axis of the rotation. The angle-axis is defined as

$$\mathbf{g}_{rot} = \arccos \frac{Trace(\mathbf{R}_g) - 1}{2} \tag{4.10}$$

with the rotation matrix $\mathbf{R}_g$ and the trace of the matrix $Trace(\cdot)$. The parameter α tunes the trade-off between the Euclidean distance of the translation and the angle difference in the axis-angle representation.
The cost of one Linear IM and one displacement is

$$Cost(\mathbf{M}, \mathbf{\Delta T}_t) = ||\mathbf{r}_t||_{TF} \tag{4.11}$$

The best displacement approximation for $\mathbf{\Delta T}_t$ for of all Linear IMs in $\mathcal{M}$ is given by the minimum cost.

$$\tilde{\mathbf{M}}_t = \min_{\mathbf{M} \in \mathcal{M}} \; Cost(\mathbf{M}, \mathbf{\Delta T}_t) \tag{4.12}$$

4.1.2 Using feature frames

The main direction of motion might differ for feature frames. A feature frame is a frame of interest such as the end-effector pose, the tip frame of a grasped object

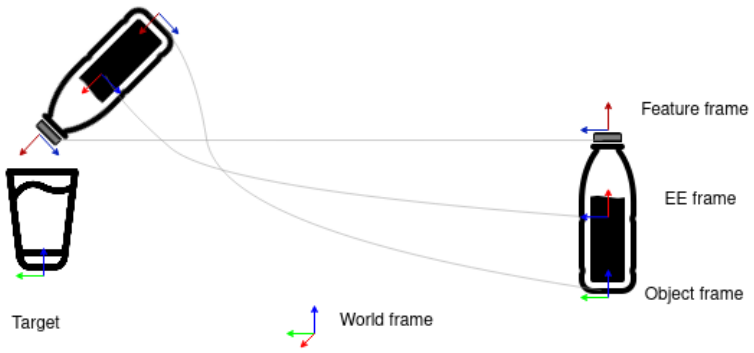
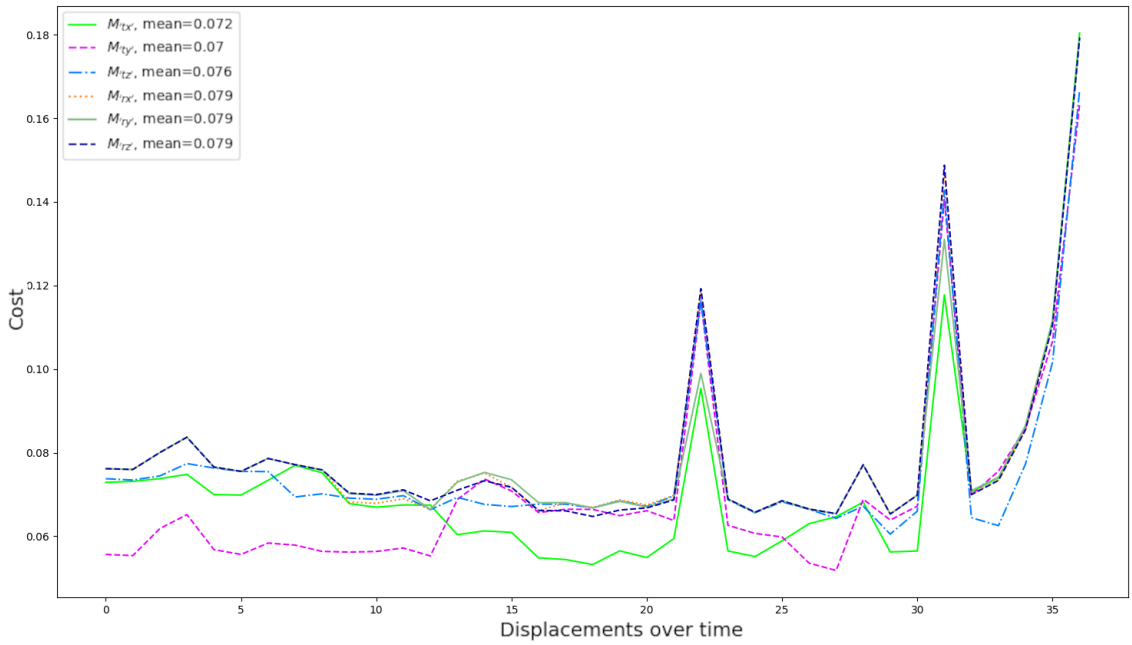


Figure 4.2: Conceptual illustration of the pouring skill now with an additional feature frame like the tip of the bottle. Using the tip frame as a reference for the motion simplifies the necessary motion to accomplish the task to only rotate without any translation.

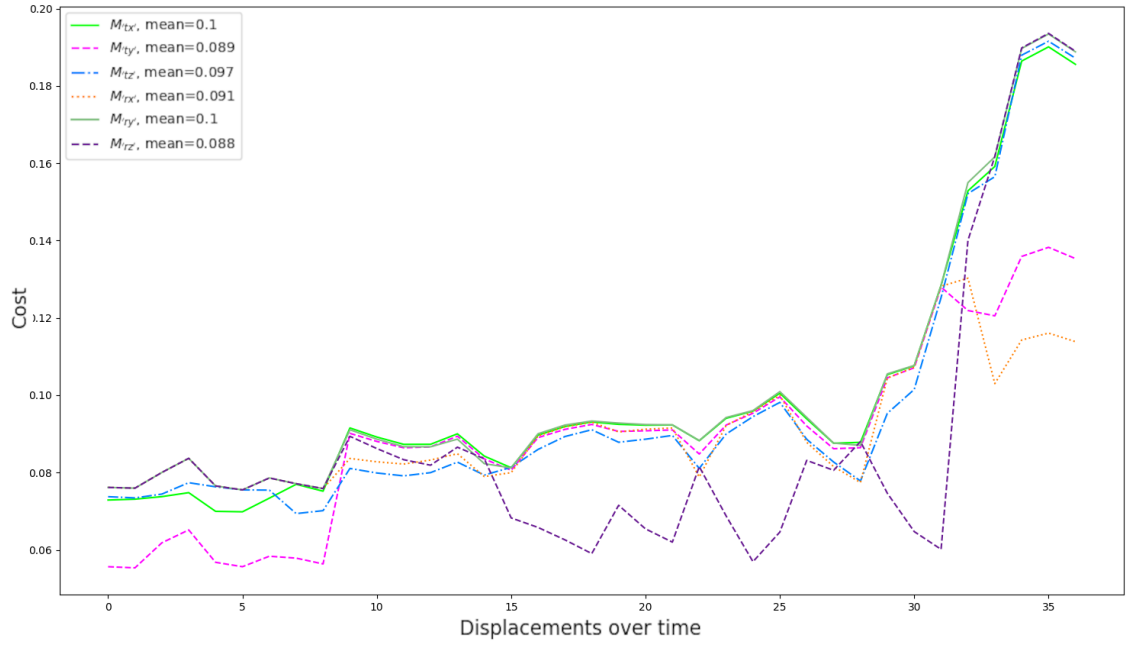
or the grasped frame of a target. The demonstrations can be observed from those different viewpoints [Cal16]. By looking at the demonstration data from different feature frames, the motion can be simplified (see Figure 4.2). In the pouring skill example, the main direction of motion simplifies to only a rotation in the pour state for the tip of the bottle. Both concepts together are used to analyze the recorded trajectories.

Linear IMs with 1-DoF

The introduced concept will be illustrated with the following figures. Figure 4.3 shows one trajectory from two different viewpoints for the pouring task. On the x-axis are the displacements (4.2) over time. On the y-axis is the cost (4.11) of the different 1-DoF Linear IMs, which include translation in either x,y,z or rotation of roll (rx), pitch (ry), yaw (rz). Figure 4.3(a) shows the trajectory by looking at the end-effector. We can see that the trajectory consists of x-translation up to the 14 displacements. From 14 to 25, it consists of y-translation, then a small x-translation and a slight yaw-rotation at the end. Figure 4.3(b) shows the same trajectory by looking at the tip of the bottle. Here, the main movement consists of a translation in y up to the 10 displacements. Afterwards, the yaw-rotation is the main motion.



(a) The cost of 1-DoF Linear IMs by looking at the end-effector. It is mainly generated by x- and y-translation.



(b) The cost of 1-DoF Linear IMs by looking at the tip of the bottle. While the first part consists of an x-translation, the main movement in the second part is the yaw rotation.

Figure 4.3: The identification of the main direction of motion using 1-DoF Linear IMs on two different feature frames for the same trajectory.

4.1.3 Segmentation of trajectories

By using different paths to execute the same task we obtain multiple trajectories and multiple displacements with different costs associated. Therefore, we would like to find a labelling of Linear IMs which represent the trajectories in a piecewise smooth and consistent manner. Meaning the same Linear IM is used in whole regions of the trajectories. This labelling can result, depending on the task, in different segments. For example in the pouring skill, we would obtain 3-DoF translation Linear IM in the first state and 3-DoF rotation and translation Linear IM in the second state. The smoothing comes from exploiting neighbourhoods since adjacent displacements should be more likely to be generated by the same Linear IM than by different ones. Consistency means that it is desirable to have fewer Linear IMs changing within a state [Sch12]. This problem is formulated as an energy term where the energy for a label f is given by [BVZ01]:

$$E(f) = E_{data}(f) + E_{smooth}(f) \tag{4.13}$$

$E_{data}(f)$ indicates how well the labels represent the data. For example how many models are used or if the motion can be described with more or less DoF. To account for this, the cost in (4.11) is extended to include the number of controlled DoF of the Linear IM:

$$Cost(\mathbf{M}, \mathbf{\Delta T}_t) = ||\mathbf{r}||_{TF} + \beta \cdot n_{DoF}(\mathbf{M}) \tag{4.14}$$

With the parameter β, it favours simpler Linear IMs over more complex Linear IMs. $E_{data}(f)$ is then the sum of the costs of the Input Mappings representing the displacement $\mathbf{\Delta T}$ over all time steps.

$$E_{data}(f) = \sum_{t=1}^{T} Cost(\mathbf{M}, \mathbf{\Delta T}_t) \tag{4.15}$$

$E_{smooth}(f)$ shows how smooth the labels represent the data. It assumes some specific neighbourhood system $\mathcal{N}$ for the displacements. For example, displacements (p, q) that are within a specific distance can be considered as neighbours.

$$E_{smooth}(f) = \lambda \sum_{p,q \epsilon \mathcal{N}} w_{pq} \cdot \delta(M_p \neq M_q) \tag{4.16}$$

λ is a parameter to tune the influence of the smoothness term, where on the one hand, labels should not change too often. On the other hand, Linear IMs must represent a skill in a meaningful manner. Therefore, the smoothing parameter should not be too high, otherwise, important information will be lost due to over-generalisation and the displacements will be labelled with only one Linear IM. Contrary, if the smoothing parameter is too low, too many IMs will be considered and the representation will not be smooth enough. The weighting factor w_{pq} is based on the tf-norm for the spatial distance from (4.9) between neighbours. This distance is then used in $w_{pq} =$

$exp(-||p-q||_{TF})$. The metric $\delta(M_p \neq M_q)$ is called Potts model, where $\delta(\cdot)$ is 1 if it fulfills the conditions and 0 otherwise. It is 1 if the IM of displacement p is different to the IM for displacement q and 0 otherwise.

4.1.4 Energy Minimization via Graph Cuts

By minimizing (4.13), a piecewise smooth and consistent labelling will be found. This can be done either locally or globally. According to [Sch12], graph cuts show the best performance as a global optimization. A popular algorithm based on this is the $\alpha - \beta$ *swap* algorithm [BVZ01]. It allows swapping the labels between an arbitrary set of α-labeled displacement and β-labeled displacement. At each iteration, the algorithm changes the $\alpha - \beta$ combination until all possible combinations have been tried and the algorithm converges. To do this, a graph is constructed for a current labelling f with different weights at the edges and nodes. An optimal cut is found that solves the maximum flow problem and minimizes the total energy in equation (4.13). For more details see Section A.3 in the Appendix.

4.1.5 Separation via Support Vector Machines

After identifying the different Linear IMs and finding a smooth and consistent label, the resulting data points x_i with their associated labels $y_i \in f$ were used to train a support vector machine (SVM) classifier [CST00] with the Python SciKit Learn package. The classifier separates the data points by finding an optimal hyperplane when minimizing the following equation:

$$\min_{\mathbf{w},\mathbf{b},\zeta} \frac{1}{2}\mathbf{w}^{\mathbf{T}}\mathbf{w} + C\sum_{i=1}^{n}\zeta_i \tag{4.17}$$

with subject to

$$y_i\left(\mathbf{w}^T\phi\left(x_i\right) + \mathbf{b}\right) \geq 1 - \zeta_i \tag{4.18}$$

$$\zeta_i \geq 0, i = 1, \ldots, n \tag{4.19}$$

The goal is to find hyperparameters $\mathbf{w} \in \mathcal{R}^3$ and $\mathbf{b} \in \mathcal{R}^3$ so that the prediction $sign(\mathbf{w}^T\phi(\mathbf{x}) + \mathbf{b})$ is correct for most samples. The parameter ζ allows a certain distance for these samples to be away from the correct margin boundary, in the case of some samples violating the hyperplane. With the penalty term C, the strength of the violation can be regularized. There is also the possibility to use polynomial or radial-basis kernels, but as we want a linear switch (i.e. simple and intuitive transition) between the different states of the skill, a linear kernel was sufficient. Figure 4.4 shows multiple trajectories for the pouring skill segmented and classified. As expected, the algorithm distinguishes with a hyperplane between two states with different Linear IMs. Algorithm 1 illustrates the whole pipeline for this approach.

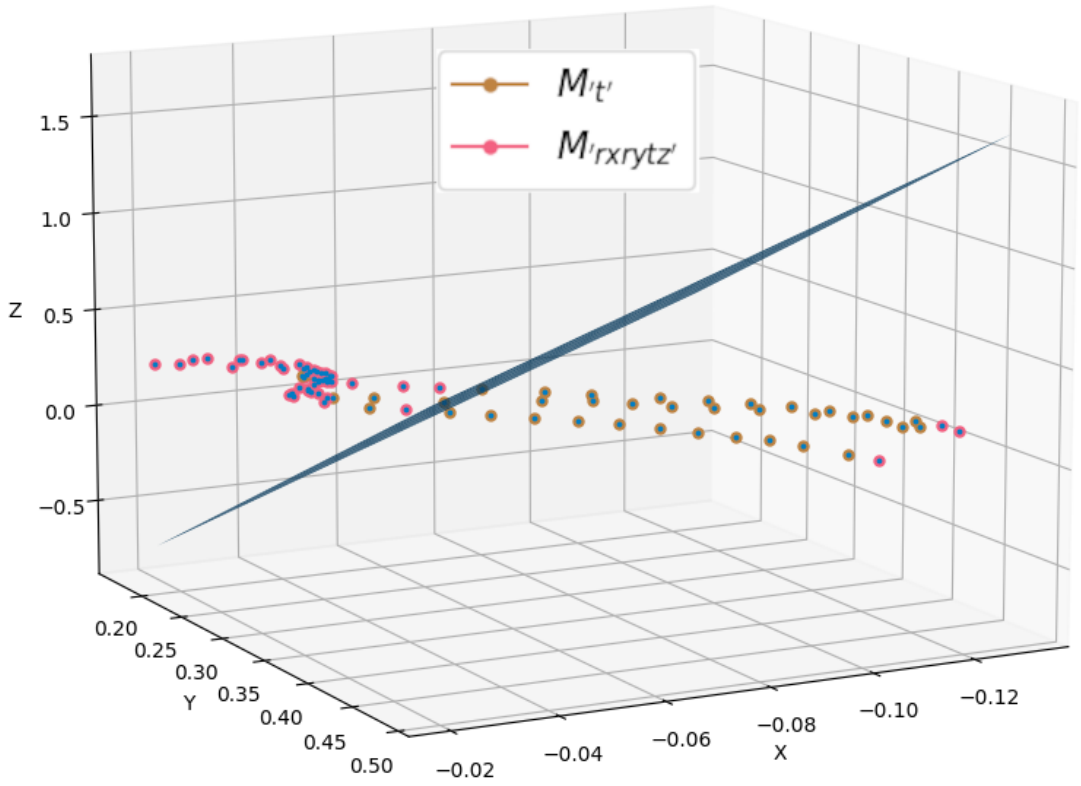


Figure 4.4: Identifying different Linear IMs on multiple trajectories with hyperplane separating different states.

4.1.6 Building a skill

After various Input Mappings have been identified and separated with a hyperplane, the outcome is a fast and simple function used by the skill. For this purpose, all necessary parameters are described in a YAML file (see Listing 4.1). This file describes three main parts. First, the transitions for the finite state machine. They depend on the EE position wrt. the target. The SVM predicts the labelling and returns the current state (see Algorithm 3 in the Appendix). Second, the individual states where the Linear IMs are applied. The current active state applies the user input on the Linear IM to generate $\mathbf{H}_{IM}$ (see Algorithm 2 in the Appendix). Third, possible Active Constraints that can be used to apply geometric constraints on the intermediate pose $\mathbf{H}_{IM}$. For all three parts, it refers to a serialized model automatically generated from above.

Listing 4.1: Linear IM skill

```
fsm:
 model:
  path: "path/to/model"
  display_rviz: [True]|[False]
  frame: [Ref. frame]

states:
  state_0:
    input_mapping:
      - model:
          path: "/path/to/model"
          frame: [Ref. frame]
  ...

active_constraints:
   - ref: [Ref. frame]
     model:
       path: "path/to/model"
       scaling: [Int]
```

4.2 Vector fields

Another promising approach to automatically generate SCTs is by looking at vector fields. The successful separation via hyperplanes depends hugely on the skill and the main direction of motion. Vector Fields, on the other hand, do not need to segment trajectories and allow us to exploit further knowledge within the trajectories. The idea is to model the translational displacements of the prerecorded trajectories as vector fields. Their usage is twofold: either as Input Mappings or to identify candidate feature frames (see Figure 4.5). In the first case, the vector fields guide the end-effector along its way either by scaling the users input command or optional by by blending the commands. Being able to identify candidate feature frames allows us to manipulate unknown object in the case perception failed. Vector fields are a tool to provide us with this prior. For example, if the bottle rotates around the tip when it is poured, a spherical vector field is created. This tip frame can then be identified because it is located in the centre of this sphere.

4.2.1 Definitions

A vector field is a function $F : R^3 \rightarrow R^3$ that assigns each point $\mathbf{p} \in R^3$ a vector $\mathbf{v} \in R^3$. It it usually written as

$$F(\mathbf{p}) = (f_1(\mathbf{p}), f_2(\mathbf{p}), f_3(\mathbf{p})) \tag{4.20}$$

with f_1, f_2, f_3 as function $f : R^3 \rightarrow R$. To simplify the calculations, this approach looks only at the translation of the end-effector pose $\mathbf{p}_\epsilon \subset \mathbf{g}_\epsilon$. We did not consider

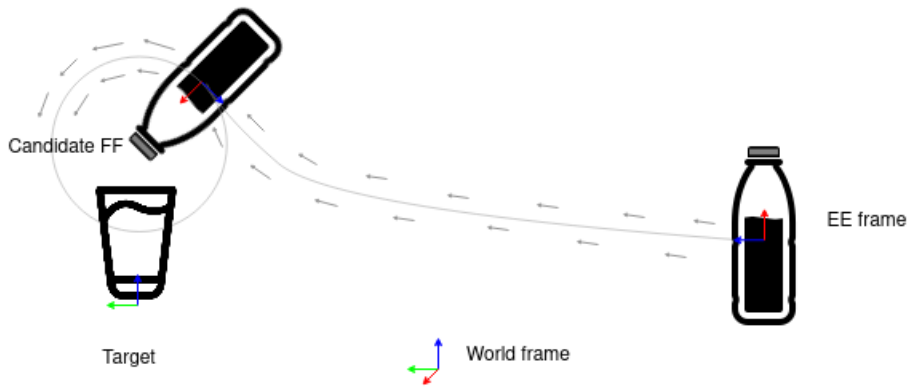


Figure 4.5: Conceptual illustration of the vector field approach. The movement can either be described with vector fields and they are used as IMs. Or a candidate feature frame can be extracted as the center of the spherical vector field.

the orientation because of their high DoF, their difficulty to visualize and the preference to control translation over-rotation in most tasks. To obtain the vectors $\mathbf{v}_\epsilon$ associated with points $\mathbf{p}_\epsilon$, this approach computes the translation difference between the current point $\mathbf{p}_{\epsilon,t}$ to its successor $\mathbf{p}_{\epsilon,t+1}$ in Euclidean space.

$$\mathbf{v}_{\epsilon,t} = \frac{\mathbf{p}_{\epsilon,t+1} - \mathbf{p}_{\epsilon,t}}{\|\mathbf{p}_{\epsilon,t+1} - \mathbf{p}_{\epsilon,t}\|} \tag{4.21}$$

4.2.2 Basic forms of vector fields

The recognition of basic shapes is a major topic in point cloud segmentation. Following the approach of Schnabel et al. [SWK07] basic shapes like spheres, cylinders or planes just need a few parameters and only a minimum set of points and vectors to estimate pose and shape. For this proof of concept, we selected a few basic fields in a set $\mathcal{F}$.

Translation field A translation vector field is defined by a magnitude and its direction and is described only by one point and one vector. The vectors associated with this field have the same direction (see Figure 4.7(a)).

Cylindrical field To describe a cylindrical vector field, a center and a rotation axis (see Figure 4.7(c)) is needed. The center and axis are described by two points with corresponding vectors. They define a plane, where the intersection of the planes is a candidate for the rotation axis.

Spherical field A spherical vector field is only defined by a center (see Figure 4.7(e)). Similar to the cylindrical vector field it is described by two points with corresponding vectors. They again define a plane, where the midpoint of the intersection of the planes is a candidate for the center.

4.2.3 Fitting basic vector fields

In order to robustly find basic vector fields we use a two-step approach. First, random sample consensus (RANSAC) [FB81] is used to find fields in the presence of many outliers and a high degree of noise. Second, an expectation-maximisation (EM) algorithm is used to achieve a smoother fit in a probabilistic manner [ZHS+17]. Algorithm 4 is in Section A.2 of the Appendix shows the generation pipeline for this approach.

Random sample consensus

RANSAC randomly selects a minimum set of points and vectors to describe the models and then checks for inliers (points that fit the model) and outliers (points that do not fit the model). Depending on the field, different vectors are considered as

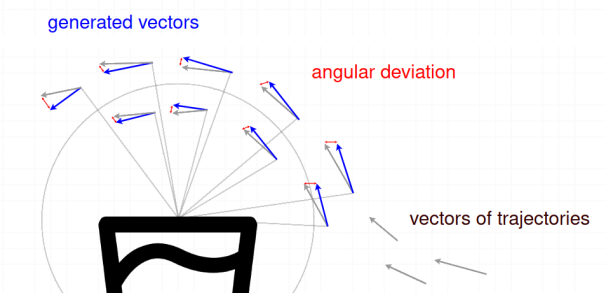


Figure 4.6: Conceptual illustration of calculating the deviation angles for a spherical vector field. The blue vectors are generated of the vector field at the different points. The grey vectors are the vectors from the data. The red lines indicate the angular deviation between both vectors. Maximizing the set of inliers (i.e. minimizing the sum of angular deviation) gives the desired field.

inliers or outliers. The distinction is based on the angular difference. To calculate the angular difference, the algorithm uses the vectors from (4.21) and compares them to the generated vectors from each field in $\mathcal{F}$. In other words, each field of $\mathcal{F}$ generates one vector at point $\mathbf{p}_{\epsilon,t}$ and compares it to vector $\mathbf{v}_{\epsilon,t}$ associated with it. If the angle deviation is smaller than a predefined angle α, the vector is considered as inlier otherwise as outlier. For more details about the calculation of these angles check Section A.1 in the Appendix.

Number of iterations The number of iterations can be determined such that there is at least one outlier-free subset with a certain probability p. Given the number of points S needed to compute a model and e, as the relative proportion of outliers in the data, the probability that no outlier is selected after N iterations is given by:

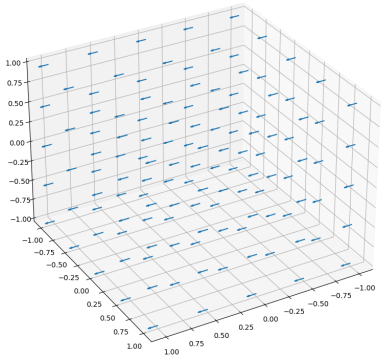
$$p = 1 - (1 - (1 - e)^S)^N \tag{4.22}$$

In order to determine how many samples are needed that at least one sample is free of outliers, the number of iterations follows:

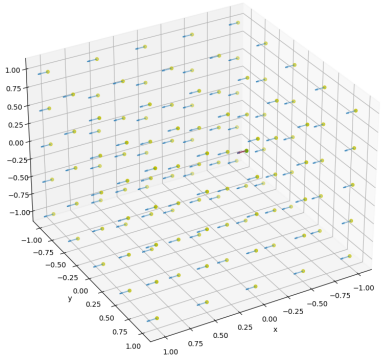
$$N = \frac{log(1 - p)}{log(1 - (1 - e)^S)} \tag{4.23}$$

This number is independent of the total number of measured values and depends only on the proportion of outliers, the number of parameters of the model function and the given probability of drawing at least one outlier-free subset.
The algorithm was first tested on synthetic data (see figure 4.7) and then on real data (see figure 4.8). A set of possible vector field models $\mathcal{F} \in$ [Translation, Spherical, Cylindrical] is defined and each model was fitted to the data. After each iteration, the fitted model with the highest inlier set was stored. The inlier points were

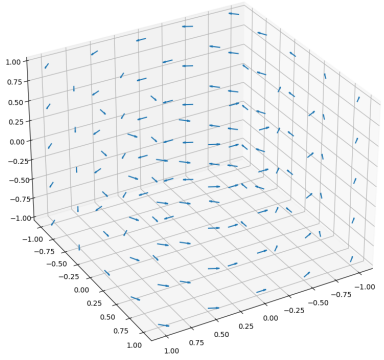
excluded from the data. The defined set of models was again fitted to the reduced data. After n_R iterations, n_R fields are obtained to describe the data. In the pouring skill, the algorithm identifies after two iterations two vector fields. The first one is a translation field for translation control. The second one is a spherical field for the pouring (see Figure 4.8(a)). In the open a drawer skill, the algorithm fits two translational fields after two iterations (see Figure 4.8(b)). To get a meaningful representation of vector fields, it was important to set the hyperparameters correctly. Besides the number of iterations, which depends on the skill, the angles to distinguish between outliers and inliers are important. If this angle is too large, the vector field would contain too many points, if it is too small, too few points. See Algorithm 5 in Section A.2 of the Appendix for the pseudo code.



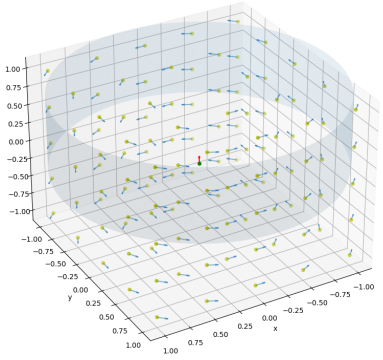
(a) Example of a translation vector field.



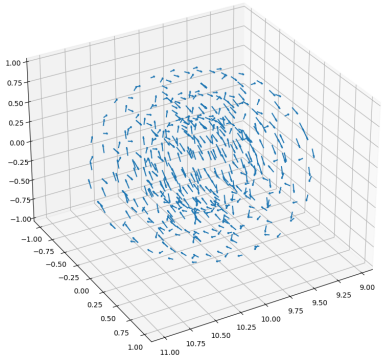
(b) Fitted translation field.



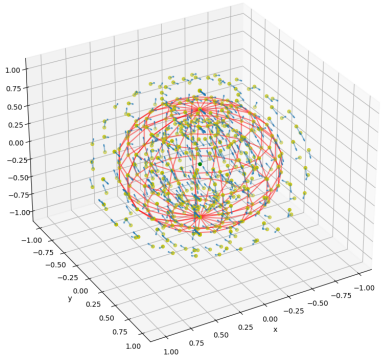
(c) Example of a cylindrical vector field.



(d) Fitted cylindrical field.

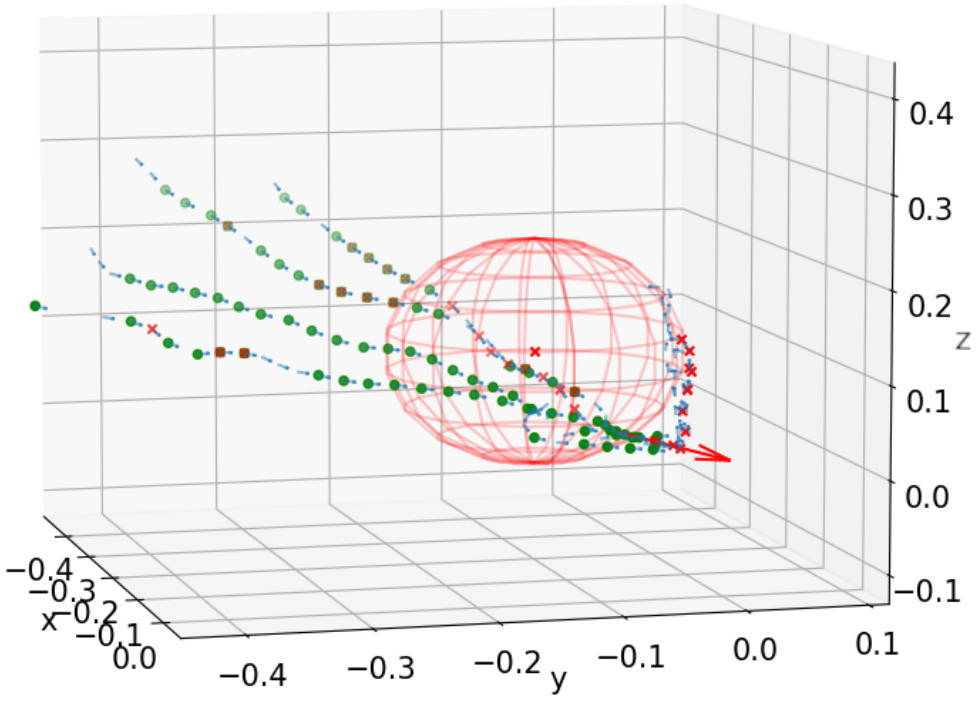


(e) Example of a spherical vector field.

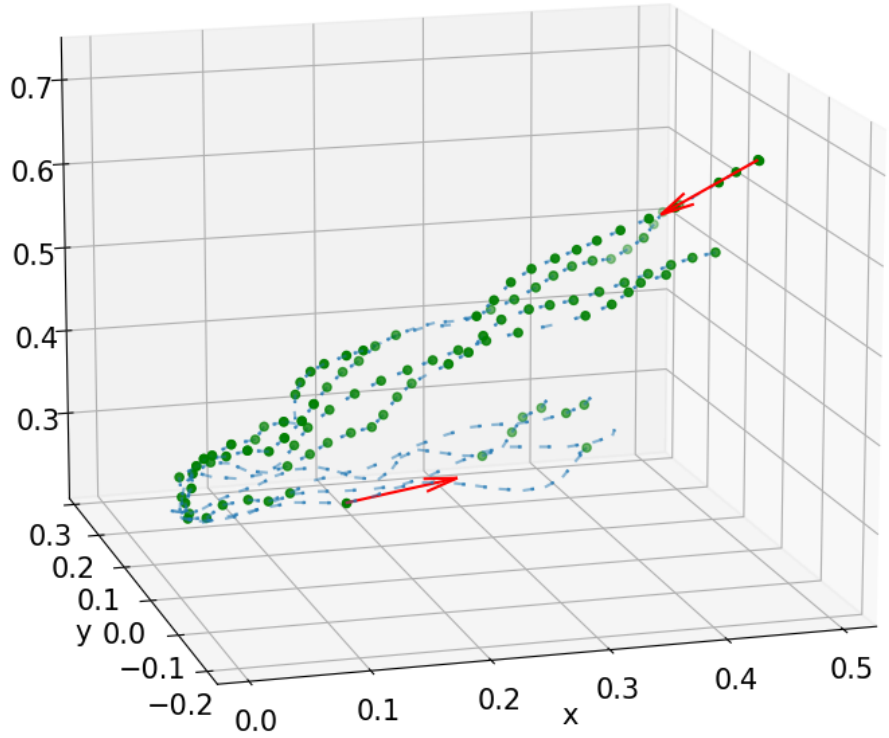


(f) Fitted spherical field.

Figure 4.7: Simple forms of vector fields on the left. Fitted vector fields on the right with RANSAC. The yellow points are considered as inliers. The synthetic data is generated in a way that there are no outliers.



(a) Fitting a spherical vector field and translation vector field for pouring skill on real data.



(b) Fitting two translational fields for open drawer task skill on real data.

Figure 4.8: Basic vector fields fitted on real data for the pouring and open drawer task.

4.2.4 Expectation Maximisation with Gaussians Mixture Models

Due to the nature of RANSAC algorithms to distinguish between outliers and inliers, an EM algorithm is implemented as a second step, to get a smoother fit. This ensures that all points of the trajectories are considered and associated with probabilities [ZHS+17]. The idea is that the vectors are generated from a Gaussian distribution with a zero mean and the spread of the vectors captured with σ. When there are multiple vector fields, there are multiple Gaussian distributions called a Gaussian mixture model (GMM). The general form of a probability density distribution is:

$$\mathcal{N}(x|\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \tag{4.24}$$

The GMM aims to describe the angles $\theta_1, ..., \theta_n$ as:

$$P(\theta_j) = \sum_{i=1}^{K} \pi_i \, \mathcal{N}(\theta_j|\mu_i,\sigma_i) \tag{4.25}$$

where K is the number of Gaussians and π_i is the mixing proportion with following constraint:

$$\sum_{i=1}^{K} \pi_i = 1 \tag{4.26}$$

Expectation Maximisation algorithm

The EM algorithm is an iterative algorithm to soft cluster data based on the probabilistic of GMMs. It consists of two steps: first, the Expectation step (E-Step), which assigns each data point a probability. Second, the Maximization step (M-Step), which updates the parameters of each cluster based on the assigned points.

Expectation Steps To assign probabilities to data points, this approach takes the deviation angle of the data. The probability r_{im} that the angle deviation θ_i belongs to model m is given by:

$$r_{im} = \frac{\pi_m \, \mathcal{N}(\theta_i \mid 0, \sigma_m)}{\sum_{k=1}^{K} \pi_k \, \mathcal{N}(\theta_i \mid 0, \sigma_k)} \tag{4.27}$$

where σ_m captures the spread of the angles of the vectors assigned to the model m.

Maximisation Steps After all, probabilities are assigned to each vector, the model parameters are updated. First, the total weight w_m, which is the sum of the probabilities of the proportion of points assigned to model m,

$$w_m = \sum_i r_{im} \tag{4.28}$$

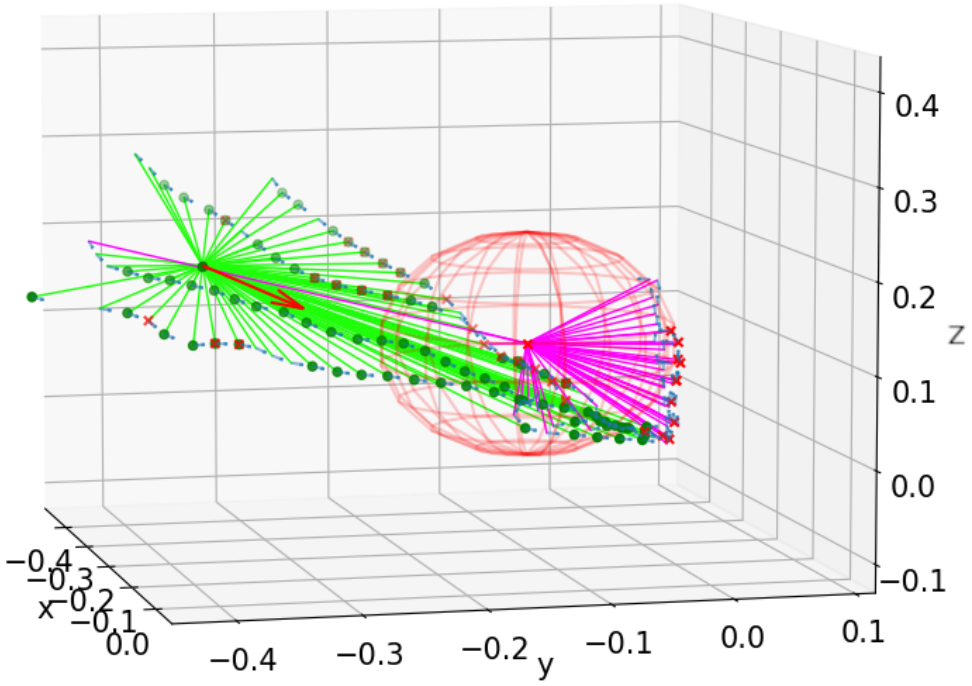
then the mixing proportion

$$\pi_m = \frac{w_m}{\sum_m w_m} \tag{4.29}$$

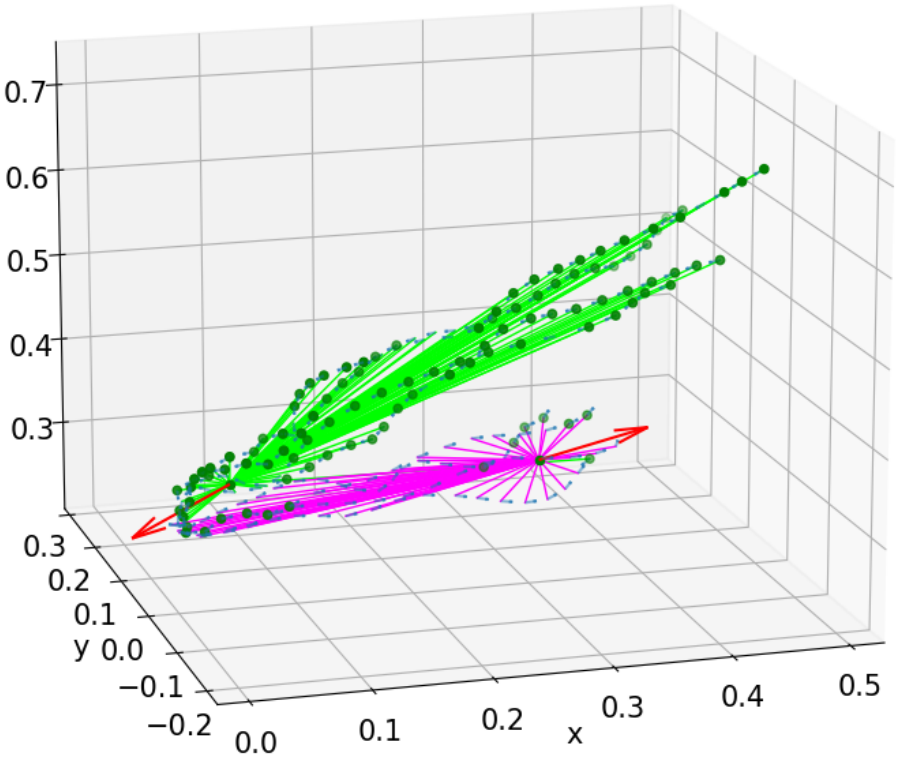
and the variance

$$\sigma_m = \frac{1}{w_m} \sum_i \theta_i \tag{4.30}$$

The center of the vector fields is then updated according to the assigned points. For a translation field, the new center is the mean overall associated points. For a cylindrical field, the axis is given by the intersection between all planes from the points and vectors. Similarly, for the spherical field, the center is given by the intersection of all the planes, and then the intersection of the intersection lines. Furthermore, it is possible to check for Spatio-temporal coherence. For example, points that are too far away or have a less frequent timestamp are removed. If there are too many models, the models with too few points are pruned. See Algorithm 6 in Section A.2 of the Appendix for the pseudo code.



(a) The initialised vector fields for the pouring task from Figure 4.8(a) were generalised with the EM algorithm. The colours indicate which points are associated to which field.



(b) The initialised translation vector fields for the open drawer task from Figure 4.8(b) were generalised with the EM algorithm. The colours indicate which points are associated to which field.

Figure 4.9: Generalised basic vector fields from 4.8.

4.2.5 Vector fields as IM

Compared to Linear IMs, realizing vector fields as IM has two advantages. First, the transition between the different states (fields) are smooth and based on probabilities. Second, applying Input Mappings can be done in two optional ways: either apply scaling to the user commands vector or as policy blending.
To apply a scaling to the user commands, the scalar product s between the command vector $\mathbf{\Delta u}$ and the generated vector(s) $\mathbf{v}_F$ at $\mathbf{p}_{\epsilon,t}$ is calculated. The generated vector is either from the most probable vector field (4.31), or from multiple vector fields (4.32), added together and scaled by the probability $P(\mathbf{v}_F|F)$ that this vector is associated with the vector field $F \subset \mathcal{F}$.

$$s = \mathbf{\Delta u} \cdot \mathbf{v}_F \tag{4.31}$$

$$s = \sum_{F}^{\mathcal{F}} \mathbf{\Delta u} \cdot \mathbf{v}_F \cdot P(\mathbf{v}_F|F) \tag{4.32}$$

The scalar product follows the shape of a cosine between $[0, \frac{\pi}{2}]$. If the command vector is aligned with the vector field vector v_F, then there is a scaling of 1. If the command vector is not aligned, then the scaling is reduced. If the command vector is orthogonal to v_F then the scaling is 0. In order to tune this scaling with a parameter, the scalar product was normalized with tunable sigmoid function, where the parameter k_s adjusts the curve.

$$s_s = \frac{s - k_s s}{k - 2k_s|s| + 1} \tag{4.33}$$

To still maintain motion and avoid 0 input, the scalar product is limited to $[min\_v, 1]$ with the clip function. The parameter min_v sets a minimum velocity and is defined in the skill as shown in Listing 4.2. Applying the scaled user command, the outcome is more guidance: the end-user is always in control rather than blended by a policy. The intermediate pose from Figure 3.3 becomes the following:

$$\mathbf{H}_{IM} = clip(\mathbf{\Delta u} \cdot s_s, min\_v) \tag{4.34}$$

A different possibility is using vector fields for policy blending. Therefore, the current end-effector position is used to determine the nearest neighbour points with vectors within a distance ϵ. This is done with a KD-tree using the poses relative to the target with the Python SciKit-Learn package. Each neighbour influences the current end-effector position by adding their vectors together scaled by the probability of the most probable field.

$$\mathbf{H}_{IM} = \mathbf{\Delta u} + \sum_{F}^{\mathcal{F}} \mathbf{v}_F \cdot P(\mathbf{v}_F|F) \tag{4.35}$$

When the end-effector is coming closer to the target pose, the vector field influence becomes stronger as there are more vectors which cause the end-effector to automatically go closer to the target.
After applying IMs, geometric constraints can be applied. Similar to the blending option, the nearest neighbours are located and the average orientation of the neighbours is used, scaled by the distance to them. The parameters for the maximum number of neighbours *max_n* and maximum distance *max_d* reduce the number of neighbours. First, it checks within a *max_d* for neighbours and if there are more than *max_n*, they constrain the number to *max_n*. See Algorithm 7 in Section A.2 of the Appendix for the pseudo code.

4.2.6 Building a skill

Listing 4.2: VF IM skill

```
states:
 default:
  input_mapping:
    - model:
        path: "path/to/model"
        ref_frame: [Ref. frame]
        min_v: [Float]
        max_dt: [Float]
        k_s: [Float]
        display_rviz: [True]|[False]
        max_n: [Int]
```

4.2.7 Identification of candidate feature frames

If perception fails and no additional feature frames are available other than the end-effector frame, vector fields can be used as a tool to obtain additional frames of unknown objects to have the possibility to further manipulate them. In the pouring skill, the bottle is poured out around the tip of the bottle and forms a sphere (see Figure 4.5), which means that the center of the sphere is a candidate feature frame for the tip of the bottle. Figure 4.10 shows the identified vector fields in the pouring technique. Only the translational part of the centre is given and we assume that it has the same orientation as the world. Figure 4.11 shows the frame of the end effector and the identified candidate feature frame projected onto a vertical plane. As expected, the candidate feature frame initially follows the end-effector, but then, as the end effector follows arc circle, it stays in place.

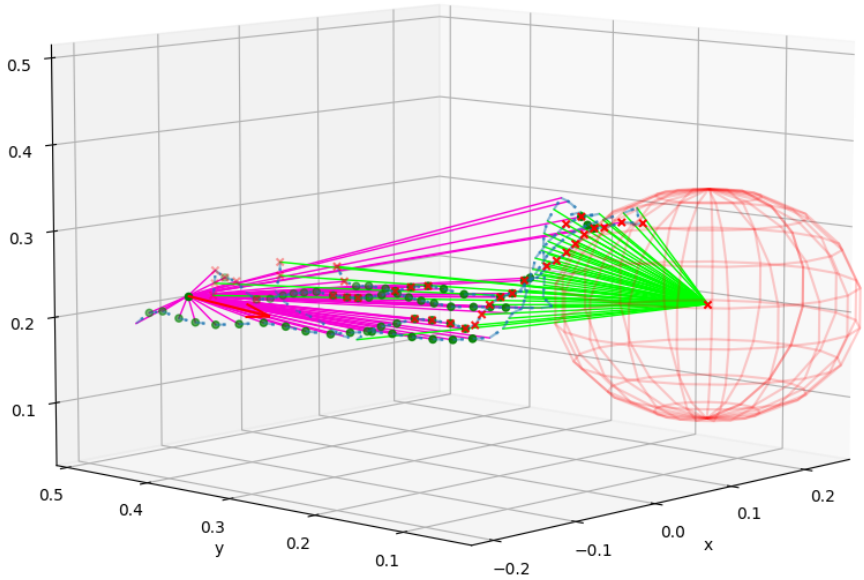


Figure 4.10: Vector fields identified on the end-effector trajectories wrt. to the world. A candidate feature frame is the center of the sphere at (0.087, 0.068, 0.265). The orientation is assumed to be the same as the world.

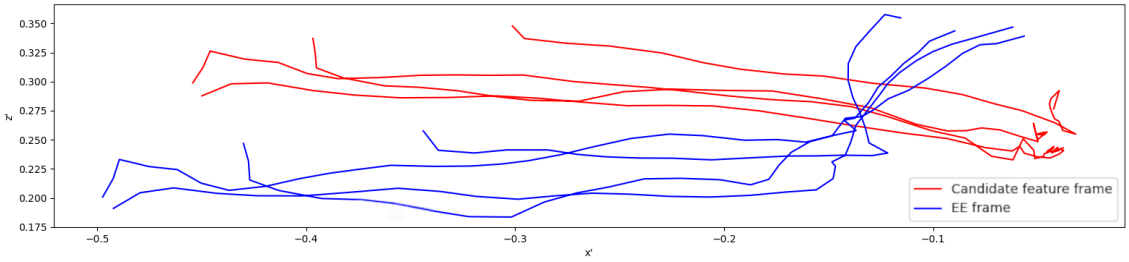


Figure 4.11: End-effector frame and candidate feature frame from sphere center projected on a vertical plane.

Chapter 5

Experimental Results

This chapter verifies the current approach based on conducted experiments on EDAN. The experiments aim to show a proof of concept of our approach on two tasks. Furthermore, it shows that the reproduced trajectories are similar to the ones used for training.

5.1 Metrics

The similarity between trajectories can be measured with plenty of different metrics. One of the most important and commonly used similarity measures is dynamic time warping [TBS+21]. In general, DTW measures an optimal match between two given sequences. A characteristic is that the cost of path represents how well two trajectories are aligned in that path. In a 2D case, the cost is the sum of the squared distances between all pairs of aligned points. Using the tf-norm from (4.9) enables us to compare trajectories in 6D. A low value indicates that the trajectories are similar.

5.2 Pouring task

The first task in our experiments was the pouring task. Therefore, new trajectories with kinesthetic teaching were recorded, segmented and fitted to generate the different aspects of the skill. During this process, different hyperparameters have to be set.

Linear IM The first hyperparameter for the identification of Linear IMs is α in (4.11) with a reasonable value between $[0.3, 0.5]$. These values slightly favour rotations, because empirically the algorithm detected translation easier. This might come from the fact that rotation movements are more affected by noise than translation movements. The second hyperparameter is β in (4.14) with a small value around $[0.01]$. It tunes the controlled number of DoF and enabled 3-DoF, which

Figure 5.1: The pouring task demonstrated on the real system.

was a sufficient for the pouring task (more in the discussion). The third parameter within the segmentation process is λ in (4.16) to tune smoothing and consistency. A value between $[0, 0.007]$ showed a good trade-off. The number of neighbours is determined with the maximum neighbour distance witch was around $4cm$. In general, it was good to consider up to 10 neighbours. The last parameter within the separation is the penalty term C in (4.17) and was set to 1000. The commands during execution were applied onto the end-effector frame in the first state and onto the tip of the bottle frame in the second state. No active constraints were used.

Vector fields For the vector fields, the deviation angle of the spherical field was 15°, for the translation field 25° and for the cylindrical 20°. The number of iterations for the RANSAC algorithm was two and for the EM algorithm was five with the coherence filter enabled. The parameters for execution in Listing 4.2 were min_vel equal to 0.1, the max_dis equal to 0.15, the $sigmoid\_k$ equal to 0.5 and the $max\_neighbours$ equal to 5.
The generated models were added as skills to the thermos mug in the object representation. The red mug was used as a target. Figure 5.1 shows images of the task with the following parameters:

- Phases: grasp the thermos bottle, lift it, pour ball into the mug.
- Initialization: hand, mug and bottle at a fixed position.
- Success: no spillage and ball poured into the mug.

Table 5.1 shows the results and combines the success rate as well as the completion time for both approaches. The completion time is calculated in the following to enable a fair comparison. The start of the task is when the manipulated bottle is closer than 35cm to the target. The end of the task is when the pouring is done. It can be seen, that the vector field approach was more robust and faster. A reason for that is that vector fields in this experiment provided Active Constraints. They helped the user in the end by constraining the orientation towards the target. With linear IMs, the user had to control more DoF and especially the control of orientation in the last state was not as intuitive as expected.

Table 5.1: Comparison of conducted experiment with the pouring task and two approaches. It shows the success rate and execution times (s), $\mu \pm \sigma$ over five trials.

Approach	Success	Failed	Execution time [s]
Linear IMs	4	1	14.25 ± 7.28
Vector Fields	5	0	6.00 ± 1.24

Another measure besides the time and success rate is the similarity to the trajectories used for training. Table 5.2 shows the metric from section 5.1.

Table 5.2: Mean and variance of the similarity to training trajectories over five trials for each approach.

Approach	DTW
Linear IMs	7.13 ± 0.68
Vector Fields	4.28 ± 0.52

Concerning the vector field approach, we can gather interesting information, presented in Figure 5.2 below. The first image shows the movement of the end-effector wrt. the target projected on a vertical plane. The second image shows the influence of the different vector fields as mean and variance for the five trials over the duration of the skill. We can see that in the first state the translational field mainly influences the end-effector. When the end-effector comes closer to the target, the influence of the spherical vector field increases. In the last state, the spherical vector field took over. The third image shows the mean and average of the scalar product between the generated vector from the vector field and the user command vector together with the mean and average of sigmoid-normalized scalar product for five trials. We can see that the user had difficulties rotating the bottle in the pouring state because the scaling is low.
The projected plane for the Linear IMs is shown in Figure 5.3.

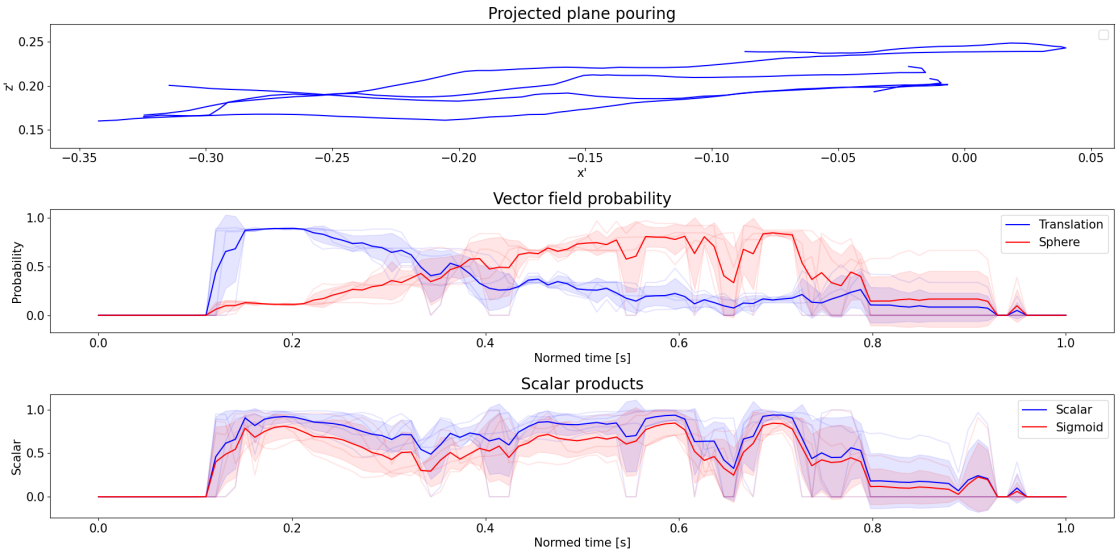


Figure 5.2: The trajectories for the vector fields are projected on a vertical plane with the influence of the vector fields and scaling factors.

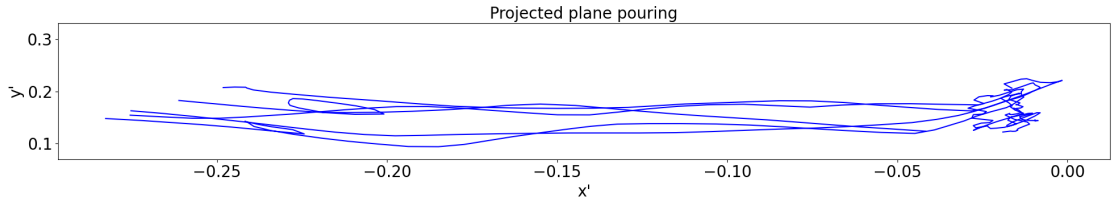


Figure 5.3: The trajectories for the Linear IM approach are projected on a vertical plane.

5.3 Opening a drawer task

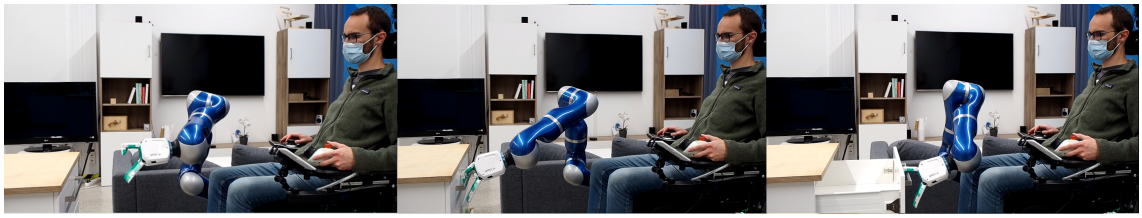


Figure 5.4: The open a drawer task demonstrated on the real system.

In a second experiment opening a drawer was tested with the vector field approach. Therefore, new trajectories were recorded by kinesthetic teaching, segmented, fitted and the skill was to the object representation. The Linear IMs were not considered here because the movement consists only of translations hence resulting in a single state with little interest compared to manual control. Figure 5.4 shows images for each state of the skill with these parameters:

- Phases: move towards the target, apply force, pull open drawer.
- Initialization: Hand at a fixed position with default orientation.
- Success: Drawer opened.

Table 5.3 shows the results with success rate and completion time. To compute the completion time, the start and end of the task are when the end-effector is within 20cm of the target.

Table 5.3: Success rate and execution times (s), $\mu \pm \sigma$ for drawer task over five trials.

Approach	Success	Failed	Execution time [s]
Vector Fields	5	0	14.73 ± 4.21

Concerning the vector field approach, we can gather a similar Figure 5.5 as in Section 5.2. The first image shows the movement of the end-effector wrt. the drawer handle projected on a vertical plane. The second image shows the mean and variance of the influence of two translational fields for five trials. We can see, that the influence changes when the fingers touched the handle and the arm tries to open the drawer. The third image shows the mean and variance of both scaling products for five trials. We can see that the user had more difficulties when grasping the handle because the orientation of the hand to be precise.
The similarity metric from section 5.1 wrt. to the training trajectories is shown in table 5.4.

Table 5.4: Mean and variance of the similarity to training trajectories over five trials

Approach	DTW
Vector Fields	4.04 ± 0.25

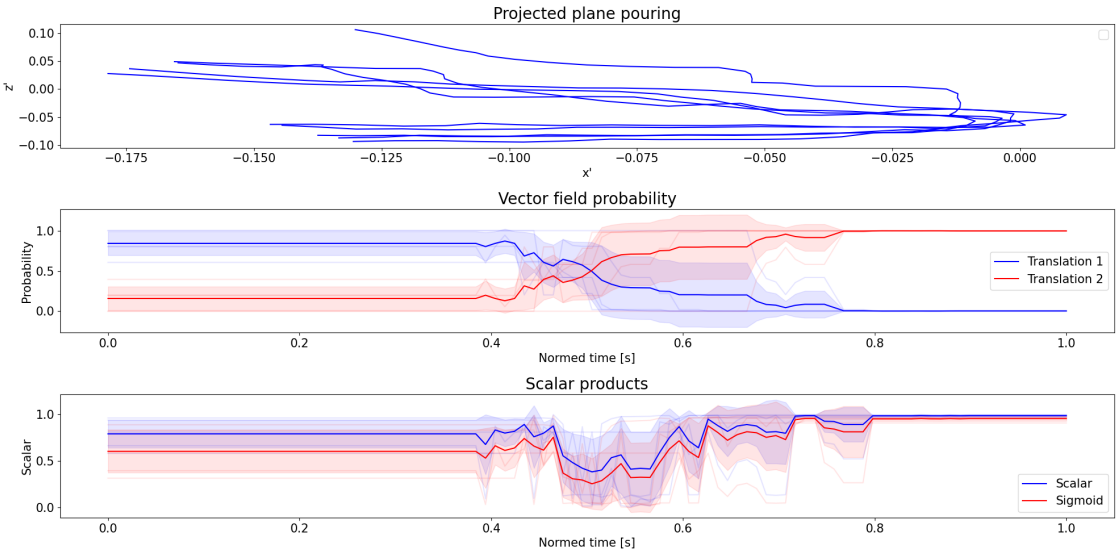


Figure 5.5: The trajectories for the vector fields are projected on a vertical plane with the influence of the vector fields and the scaling factors.

Chapter 6

Discussion

Overall, on our proof on concept experiments both implemented approaches succeeded on the EDAN systems. Nevertheless, there were quite some differences in the required setup. This chapter highlights the difference in the design procedure as well as points out flaws and strengths when using them.

6.1 Design procedure

Linear IM allows model-based segmentation based on the main direction of motion without taking curvature, distances or timeouts into account. In addition, states, transitions and IMs are generated automatically. A certain disadvantage is the quality of these transitions. For example, the hyperplane is sometimes too far from the target, too flat or too simple. It allows clean transitions, but for more complex tasks, a more complex decision boundary may be desired. The adjustment of the hyperplane is directly related to the segmentation result, which is another main shortcoming of this approach. The result is not as robust to new data or feature frames as expected. This means that the hyperparameters may have to be readjusted to achieve the desired result. There is a small gap between smooth and consistent trajectories, which is strongly related to the quality of the recording and the sampling distance. If those parameters are defined incorrectly, it can happen that there is only one Linear IM for all trajectories. Compared to the vector fields, the Linear IMs were more difficult to set up. They required more time for training and fine-tuning and resulted in poorer task performance. However, the higher time expenditure also resulted from the greater degree of freedom that the linear IMs allow in performing the task, for example, controlling the timing and intensity of the rotation. A positive point is that any DoF can be controlled.
The advantages of vector fields are the robust fitting with RANSAC. Even in the presence of high noise, RANSAC allows models to be fitted with only a few parameters. But depending on the model, the fitting result can sometimes be ambiguous. For example, the spherical or cylindrical fields showed weird centers estimation. Within some fitting iterations, the center jumped with a high distance. Vector

fields are easier to create because there is no need to segment the trajectories. The influence comes either as a weighted sum of vector fields or individually with the highest probable vector field. A disadvantage is the limited number of models and the reduced dimensional space by not making use of the orientation.

6.2 Usability

When using Linear IMs, the decision boundary may be too simple to capture the underlying constraints. We lack the guarantees to be certain to cross the transitions. Non-linear kernels or non-linear IMs could help, but are not guaranteed. In addition, the current label may not allow the user to transition to another state. Therefore, the end-effector could be stuck, without the possibility to transition further. An example is when the SCT designer wants to use only 1 DoF to control rotation. The user then moves towards the decision boundary and once the IMs change, he cannot transition back. So depending on the task, it is important to know how many DoFs to control. Another problem with controlling rotations was that they were not intuitive. The users do not really see around which axis they are allowed to rotate. This makes controlling the arm not as easy as expected. The intuitiveness in this approach came from applying the Linear IM rotation onto the hand and the translation wrt. world. Vector fields, on the other hand, worked quite well. The user could always feel the influence of the vector fields.

Chapter 7

Conclusion

In this thesis, we have proposed two novel approaches which expand the Shared Control Templates to facilitate the learning of skills from demonstrations. By modelling skills with Linear IMs and vector fields, we have shown that states, transitions, Input Mappings and Active Constraints can be generated and successfully tested on a real system. Linear IMs identify the main direction of motion, but has fickle segmentation, are not intuitive and do not provide guarantees for the the state machine. Vector fields, in contrast, allow smooth transitions and guidance. We have further highlighted the advantages, disadvantages and differences between those two. We also demonstrated the potential use of vector fields to identify candidate feature frames in a proof of concept experiment. For further research, it would be worth investigating non-linear IMs, where the mapping model is not a matrix but a function that maps the user command $\varphi : \mathbb{R}^n \to \mathbb{R}^n$. Another way is to look at non-linear decision boundaries to enable more complex transitions. For the vector fields, more complex models or a general model could be considered without having to find individual models. This general one could smooth all displacements and the generated vector would be a smoothed version of its neighbours, which means, there is no need to find individual models. Another option is to exploit different properties of the vector field such as the gradient, divergence or curl when describing a vector field. A popular approach within the field of shape estimation is the Hough transform [ANC13], which is a method for detecting parameterized objects such as spheres, cylinders and cones. Compared to RANSAC, they have a faster runtime and can detect multiple instances of the shape at once.

Appendix A

Appendix

A.1 Calculating the angular difference

To calculate the angular difference, the cosine-similarity and sinus-similarity are used. The sinus-similarity is used because it was simpler to construct for the spherical and cylindrical field, but in the end the principle is the same.

Translation field The deviation is calculated according to the cosine similarity relationship of two vectors **a** and **b**:

$$cos(\theta) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\|\|\mathbf{b}\|} \tag{A.1}$$

Cylindrical field To calculate the deviation angle, we need to construct the desired vector. Therefore we take the normal vector from the associated point on the cylinder axis. The normal vector $\mathbf{v}_\perp$ of the point **p** to the rotation axis **a** and axis center **c** is given by:

$$\mathbf{v}_\perp = (\mathbf{p} - \mathbf{c}) \times \mathbf{a} \tag{A.2}$$

This normal vector then is used, to calculate the deviation. The deviation angle is defined as:

$$sin(\theta) = \frac{\mathbf{v}_\perp \times \mathbf{v}_p}{\|\mathbf{v}_\perp\| \cdot \|\mathbf{v}_p\|} \tag{A.3}$$

Spherical field Similarly to the cylindrical field, we need to construct our desired vector. Now we take the vector from the associated point to the center of the sphere. The vector of a point to the centre is given by:

$$\mathbf{v}_{pc} = (\mathbf{p} - \mathbf{c}) \tag{A.4}$$

The deviation angle is calculate via:

$$sin(\theta) = \frac{\mathbf{v}_{pc} \times \mathbf{v}_p}{\|\mathbf{v}_{pc}\| \cdot \|\mathbf{v}_p\|} \tag{A.5}$$

A.2 Algorithms

A.2.1 Linear Input Mappings

Algorithm 1 Generation Pipeline

Input: Feature frame poses $g_{\mathcal{F}}$, Set of Linear Input Mappings $\mathcal{M}$,
Hyperparameters: $\alpha, \beta, \lambda, max\_n, C$
Output: Linear Input Mapping FSM and IM models

$T^{\mathcal{F}} \leftarrow pose2tf(g_{\mathcal{F}})$
// Compute displacements
$T^{D} \leftarrow displacements(T^{\mathcal{F}})$
for each Linear Input Mapping M **do**
    // Assign costs for all displacements
    $c_M \leftarrow cost(M, T^{D}, \alpha)$
end for
// Segment
$S \leftarrow graph\_cut(c_M, \beta, \lambda, max\_n)$
// Separate
$S' \leftarrow SVM.fit(S, C)$
// Generate transition model and IM model
$L_{FSM}, L_{IM} \leftarrow generate\_IM(S')$
return L_{FSM}, L_{IM}

Algorithm 2 Input Mapping

Input: User input Δu, end-effector pose $g_{\mathcal{E}}(t-1)$, Linear Input Mappings $\mathcal{M}$
Output: Displaced target end-effector pose $g_{\mathcal{E}im}(t+1)$

$T^{\mathcal{E}im} \leftarrow pose2tf(g_{\mathcal{E}}(t-1))$
for each *input_mapping* in $\mathcal{M}$ **do**
    // Compute reference frame
    $T^{f_i} \leftarrow input\_mapping$.reference_frame$(T^{\mathcal{E}im})$
    // Compute transform from $T^{\mathcal{E}im}$ *to* T^{f_i}
    ${}_{f_i}T^{\mathcal{E}im} \leftarrow {T^{f_i}}^{-1} * T^{\mathcal{E}im}$
    // Compute displacement
    $T^{D} \leftarrow euler2tf(M_i * \Delta u)$
    // Apply displacement to reference frame:
    // rotation relative, translation global
    $T^{f} \leftarrow (T^{D}_{trans} * T^{f_i}) * T^{D}_{rot}$
    // Update target EE pose from reference frame
    $T^{\mathcal{E}im} \leftarrow T^{f} * {}_{f_i}T^{\mathcal{E}im}$
end for
$g_{\mathcal{E}im}(t) \leftarrow tf2pose(T^{\mathcal{E}im})$
return $g_{\mathcal{E}im}(t)$

Algorithm 3 Finite state machine

Input: End-effector pose $g_{\mathcal{E}}(t-1)$, Linear Input Mapping SVM
Output: New state

return $SVM.predict(g_{\mathcal{E}}(t-1))$

A.2.2 Vector fields

Algorithm 4 Generation Pipeline

Input: Feature frame poses $g_{\mathcal{F}}$, Set of possible vector field models $\mathcal{F}$, RANSAC iterations n_R, EM iterations r_E

Output: Vector field IM model

1: *// Calculate points and vectors*
2: $p \leftarrow g_{\mathcal{E},trans}$
3: $v \leftarrow calc\_vec(p)$
4: *// Apply RASNAC*
5: $F_R \leftarrow RANSAC(F, p, v, n_R)$
6: *// Apply EM*
7: $F_E \leftarrow EM(F_R, v, r_E)$
8: *// Generate and IM model*
9: $F_{IM} \leftarrow generate\_IM(F_E)$
10: **return** F_{IM}

Algorithm 5 RANSAC

Input: Points p, Vectors v, Set of possible vector field models $\mathcal{F}$, Number of Iterations n_R
Output: Initialized vector field models F_R

for n_R iterations **do**
  for each model **in** $\mathcal{F}$ **do**
    // Apply RANSAC
    // Sample random points
    $s \leftarrow random\_samples(p, v)$
    // Estimate model parameters
    $model.parameters \leftarrow model.estimate\_parameters(s)$
    // Estimate inliers based on angle difference
    $model.estimate\_inliers(v)$
  end for
  // Save model with highest inliers and exclude them
  $best\_model \leftarrow best(F)$
  $p, v \leftarrow delete(p, v; best\_model.inliers)$
  $F_{R,n_R} \leftarrow best\_model$
end for
return F_R

Algorithm 6 EM with GMM

Input: Vectors v, Initialized models F_R from RANSAC, Number of Iterations n_E
Output: Smoothed vector field models F_E

for n_E iterations **do**
  // Expectation step
  for each model **in** F_E **do**
    // Calculate angle deviation prob
    $r_{model} \leftarrow model.angle\_diff\_prob(v)$
  end for
  // Maximisation step
  for each model **in** F_E **do**
    // Get vector associated to model
    $v_{model} \leftarrow associate(v, r_{model})$
    // Filter for spatial-temporal coherence
    $v_{model} \leftarrow filter(v_{model})$
    // Update centers
    $model.parameters \leftarrow model.update\_parameters(v_{model})$
  end for
end for
return F_E

Algorithm 7 Vector Field Input Mapping

Input: User input Δu, end-effector pose $g_{\mathcal{E}}(t-1)$,
Skill parameters: max_n, max_d, min_v, k_s
Output: Displaced target end-effector pose $g_{\mathcal{E}im}(t+1)$

// Get neighbours close to end-effector pose
$n \leftarrow neighbours(g_{\mathcal{E}}(t-1), max\_n, max\_d)$

// Orientation
$g_{\mathcal{E},rot}(t+1) \leftarrow weighted\_average\_orientation(n.orientations, n.distances)$

// Translation
for each $vector\_field$ in n.fields **do**
    // Generate vector field vector
    $v_F \leftarrow vector\_field.generate\_vector(g_{\mathcal{E},trans}(t-1))$
    // Compute scalar and scale it
    $s_s \leftarrow sigmoid(\Delta u \cdot v_F, k\_s)$
    // Apply command
    $g_{\mathcal{E},trans}(t+1) \leftarrow g_{\mathcal{E},trans}(t) + clip(\Delta u \cdot s_s, min\_v)$
end for
return $g_{\mathcal{E}im}(t+1)$

A.3 Alpha-Beta Swap algorithm

Algorithm 8 $\alpha - \beta$ swap. Taken from [BVZ01].

1: *// Start with an arbitrary labeling f*
2: *// Set success := 0*
3: **for each** pair of labels $\alpha, \beta \in L$ **do**
4: *// Find $\hat{f}$ = argmin E(f') among f' within one $\alpha - \beta$ swap*
5: *// If $E(\hat{(}f)) < E(f)$, set $f := \hat{(}f)$ and success := 1*
6: **end for**
7: *// If success = 1 goto 2*
8: **return** f

The $\alpha - \beta$ swap algorithm is a popular algorithm used in computer vision. The version explained here is adapted to our particular problem. The algorithm generates a labelling f by exchanging an arbitrary set of α-labelled displacements with another arbitrary set of β labelled displacements. After creating a labelling, no other swap move decreases the energy term. In order to do this, the assumption is that there is a semi-metric interaction potential V. It means that the space of labels L for any pair of α, β in L satisfies two properties:

1. $V(\alpha, \beta) = V(\beta, \alpha) > 0$ and
2. $V(\alpha, \beta) = 0$ if $\alpha = \beta$.

Finding the optimal swap move

Given an input labelling f and a pair of α, β labels, the algorithm tries to find a labelling $\hat{f}$ that minimizes the Energy over all labels within one $\alpha - \beta$ swap of f. Therefore it constructs a graph $G_{\alpha,\beta} =< \mathcal{V}_{\alpha,\beta}, \mathcal{E}_{\alpha,\beta} >$ with vertices and edges (see Figure A.1 for the 1D structure of the graph). The set of vertices includes two terminals α and *beta* and the displacements p in the set of P_α (i.e. set of displacements labelled with α) and P_β (i.e. set of displacements labelled with β). The links of the graph are the terminal links and neighbours links. The terminal links connect the displacements to the terminals α and β with the edges t_p^α and t_p^β. The neighbours links connect each neighbour displacement $p, q_\alpha\beta$ which are neighbours $p, q \in (N)$. The weights of each edge is displayed in Table A.1.
The label assignment is then the following: when the cut C seperates p from the terminal α then p is assigned label α if the displacement p is in $P_{\alpha,\beta}$. Similarly, when C seperates p from the terminal β, then p is assigned label β. Otherwise, if p is not in $P_(\alpha, \beta)$ it keeps the inital labelling f_p (see Figure A.2).

Table A.1: Weights of the edges. Taken from [BVZ01].

edge	weight	for
t_p^α	$D_p(\alpha) + \sum_{q \in \mathcal{N}p} V\{p,q\}\,(\alpha, f_q)$	$p \in \mathcal{P}_{\alpha\beta}$
t_p^β	$D_p(\beta) + \sum_{q \in \mathcal{N}p} Vq\{,q\}\,(\beta, f_q)$	$p \in \mathcal{P}_{\alpha\beta}$
$e_{\{p,q\}}$	$V_{\{p,q\}}(\alpha, \beta)$	$\{p,q\} \in \mathcal{N}$ $p, q \in \mathcal{P}_{\alpha\beta}$

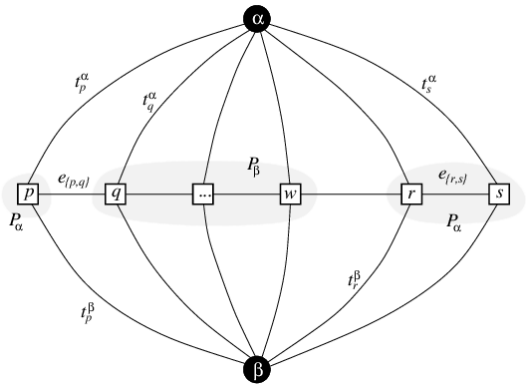


Figure A.1: Example of a graph for 1D with different terminal links and edge links. Taken from [BVZ01].

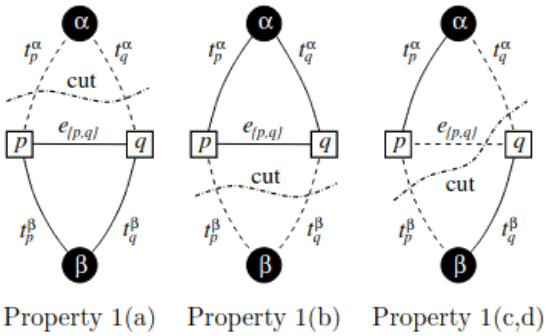


Figure A.2: Illustration of a cut. Taken from [BVZ01].

List of Figures

Acronyms and Notations

HRC Human-Robot Collaboration

HRI Human-Robot Interaction

HRT Human-Robot Team

SCT Shared Control Template

SC Shared Control

SA Shared Autonomy

IM Input Mapping

AC Active Constraints

GMM Gaussian Mixture Models

EM Expectation maximisation

EE End effector

DTW Dynamical-time-warping

SE Special Euclidian Group

RANSAC Random sample consensus

Bibliography

[AC18] Seyed Reza Ahmadzadeh and S. Chernova. Trajectory-based skill learning using generalized cylinders. *Frontiers in Robotics and AI*, 5, 2018.

[ACHT14] Sonya Alexandrova, Maya Cakmak, Kaijen Hsiao, and Leila Takayama. Robot programming by demonstration with interactive action visualizations. In *Robotics: Science and Systems (R:SS)*, pages 48–56, 2014. doi:10.15607/RSS.2014.X.048.

[ACVB09] B. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics Auton. Syst.*, 57:469–483, 2009.

[AM16] Tohid Alizadeh and Milad S. Malekzadeh. Identifying the relevant frames of reference in programming by demonstration using task-parameterized gaussian mixture regression. *2016 IEEE/SICE International Symposium on System Integration (SII)*, pages 453–458, 2016.

[ANC13] Anas Abuzaina, Mark S. Nixon, and John N. Carter. Sphere detection in kinect point clouds via the 3d hough transform. In *CAIP*, 2013.

[BC94] D. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *KDD Workshop*, 1994.

[BDB14] Stuart A. Bowyer, B. Davies, and F. Baena. Active constraints/virtual fixtures: A survey. *IEEE Transactions on Robotics*, 30:138–157, 2014.

[BM17] M. Barão and J. Marques. Gaussian random vector fields in trajectory modelling. 2017.

[BM18] M. Barão and J. Marques. Clustering of gaussian random vector fields in multiple trajectory modelling. *2018 13th APCA International Conference on Control and Soft Computing (CONTROLO)*, pages 352–357, 2018.

[BTTT18] Jiang Bian, Dayong Tian, Yuanyan Tang, and D. Tao. A survey on trajectory clustering analysis. *ArXiv*, abs/1802.06971, 2018.

[BVZ01] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, 2001. doi:10.1109/34.969114.

[Cal16] Sylvain Calinon. A tutorial on task-parameterized movement learning and retrieval. *Intelligent Service Robotics*, 9:1–29, 2016.

[CBO18] Glen Chou, Dmitry Berenson, and Necmiye Ozay. Learning constraints from demonstrations. *ArXiv*, abs/1812.07084, 2018.

[CGB07] Sylvain Calinon, Florent Guenter, and Aude Billard. On learning, representing, and generalizing a task in a humanoid robot. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37:286–298, 2007.

[CST00] N. Cristianini and J. Shawe-Taylor. An introduction to support vector machines and other kernel-based learning methods. 2000.

[CVW90] L. Conway, R.A. Volz, and M.W. Walker. Teleautonomous systems: projecting and coordinating intelligent action at a distance. *IEEE Transactions on Robotics and Automation*, 6(2):146–158, 1990. doi:10.1109/70.54731.

[DS13] A. Dragan and S. Srinivasa. A policy-blending formalism for shared control. *The International Journal of Robotics Research*, 32:790 – 805, 2013.

[FB81] M. Fischler and R. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24:381–395, 1981.

[FKSS13] Nivan Ferreira, J. Klosowski, C. Scheidegger, and Cláudio T. Silva. Vector field k-means: Clustering trajectories by fitting multiple vector fields. *Computer Graphics Forum*, 32, 2013.

[GQ20] Maged Iskandar Samuel Bustamante Daniel Leidner Freek Stulp Joern Vogel Gabriel Quere, Annette Hagengruber. Shared control templates for assistive robotics. 2020. doi:https://ras.papercept.net/proceedings/ICRA20/0530.pdf.

[HHS16] Laura V. Herlant, Rachel M. Holladay, and Siddhartha S. Srinivasa. Assistive teleoperation of robot arms via automatic time-optimal

mode switching. In *The Eleventh ACM/IEEE International Conference on Human Robot Interaction*, HRI '16, page 35â42. IEEE Press, 2016.

[IQH+19] Maged Iskandar, Gabriel Quere, Annette Hagengruber, Alexander Dietrich, and Jörn Vogel. Employing whole-body control in assistive robotics. *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5643–5650, 2019.

[JSB15] Shervin Javdani, S. Srinivasa, and J. Bagnell. Shared autonomy via hindsight optimization. *Robotics science and systems : online proceedings*, 2015, 2015.

[KHKCG+12] Dae-Jin Kim, Rebekah Hazlett-Knudsen, Heather Culver-Godfrey, Greta Rucks, Tara Cunningham, David Portee, John Bricout, Zhao Wang, and Aman Behal. How autonomy impacts performance and satisfaction: Results from a study with spinal cord injured subjects using an assistive robot. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 42(1):2–14, 2012. doi:10.1109/TSMCA.2011.2159589.

[LBH12] Daniel Leidner, Christoph W. Borst, and Gerd Hirzinger. Things are made for what they are: Solving manipulation tasks by using functional object classes. *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, pages 429–435, 2012.

[LJL+21] Dylan P. Losey, Hong Jun Jeon, Mengxi Li, Krishna Parasuram Srinivasan, Ajay Mandlekar, Animesh Garg, Jeannette Bohg, and Dorsa Sadigh. Learning latent actions to control assistive robots. *Autonomous Robots*, pages 1 – 33, 2021.

[LSM+20] Dylan P. Losey, K. Srinivasan, Ajay Mandlekar, Animesh Garg, and D. Sadigh. Controlling assistive robots with learned latent actions. *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 378–384, 2020.

[Mas81] Matthew T Mason. Compliance and force control for computer controlled manipulators. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(6):418–432, 1981.

[MFD+17] Pat Marion, Maurice Fallon, Robin Deits, Andréas Valenzuela, Claudia Pérez D'Arpino, Greg Izatt, Lucas Manuelli, Matt Antone, Hongkai Dai, Twan Koolen, John Carter, Scott Kuindersma, and Russ Tedrake. Director: A user interface designed for robot operation with shared autonomy. *Journal of Field Robotics*, 34(2):262–280, 2017. URL: https:

//onlinelibrary.wiley.com/doi/abs/10.1002/rob.21681, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.21681, doi:https://doi.org/10.1002/rob.21681.

[MHD16] Negar Mehr, R. Horowitz, and A. Dragan. Inferring and assisting with constraints in shared autonomy. *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 6689–6696, 2016.

[MNM18] Christina Moro, Goldie Nejat, and Alex Mihailidis. Learning and personalizing socially assistive robot behaviors to aid with activities of daily living. *ACM Transactions on Human-Robot Interaction (THRI)*, 7:1 – 25, 2018.

[OKTM19] Yoojin Oh, Han-Gyoo Kim, M. Toussaint, and Jim Mainprice. A differentiable policy for shared autonomy. 2019.

[PDS17] Claudia Pérez-D'Arpino and J. Shah. C-learn: Learning geometric constraints from demonstrations for multi-step manipulation in shared autonomy. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4058–4065, 2017.

[QHI+20] Gabriel Quere, Annette Hagengruber, Maged Iskandar, Samuel Gomez, Daniel Leidner, Freek Stulp, and JÃ¶rn Vogel. Shared control templates for assistive robotics. 05 2020. doi:10.1109/ICRA40945.2020.9197041.

[RLD18] S. Reddy, Sergey Levine, and A. Dragan. Shared autonomy via deep reinforcement learning. *ArXiv*, abs/1802.01744, 2018.

[Sch12] K. Schindler. An overview and comparison of smooth labeling methods for land-cover classification. *IEEE Transactions on Geoscience and Remote Sensing*, 50:4534–4545, 2012.

[SCN+21] Mario Selvaggio, Marco Cognetti, Stefanos Nikolaidis, Serena Ivaldi, and Bruno Siciliano. Autonomy in physical human-robot interaction: A brief survey. *IEEE Robotics and Automation Letters*, 6:7989–7996, 2021.

[She92] Thomas B Sheridan. *Telerobotics, automation, and human supervisory control*. 1992.

[SMO+21] Hang Su, Andrea Mariani, Salih Ertug Ovur, Arianna Menciassi, Giancarlo Ferrigno, and Elena De Momi. Toward teaching by demonstration for robot-assisted minimally invasive surgery. *IEEE Transactions on Automation Science and Engineering*, 18:484–494, 2021.

[SN03] W. Schindler and C. Natale. Interaction control of robot manipulators: Six degrees-of-freedom tasks. 2003.

[SWK07] R. Schnabel, R. Wahl, and R. Klein. Efficient ransac for point-cloud shape detection. *Computer Graphics Forum*, 26, 2007.

[SZG18] Guru Subramani, Michael R. Zinn, and Michael Gleicher. Inferring geometric constraints in human demonstrations. *ArXiv*, abs/1810.00140, 2018.

[TBS+21] Yaguang Tao, Alan Both, Rodrigo I. Silveira, Kevin Buchin, Stef Sijben, Ross Purves, Patrick Laube, Dongliang Peng, Kevin Toohey, and Matt Duckham. A comparative analysis of trajectory similarity measures. *GIScience & Remote Sensing*, 58:643 – 669, 2021.

[VHI+20] J. Vogel, Annette Hagengruber, M. Iskandar, Gabriel Quere, Ulrike Leipscher, Samuel Bustamante, Alexander Dietrich, Hannes Höppner, Daniel Leidner, and A. Albu-Schäffer. Edan: An emg-controlled daily assistant to help people with physical disabilities. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4183–4190, 2020.

[ZHS+17] Martijn J. A. Zeestraten, Ioannis Havoutis, João Silvério, Sylvain Calinon, and Darwin Gordon Caldwell. An approach for imitation learning on riemannian manifolds. *IEEE Robotics and Automation Letters*, 2:1240–1247, 2017.

License