Learning and Interactive Design of Shared Control Templates

Gabriel Quere, Samuel Bustamante, Annette Hagengruber, Joern Vogel, Franz Steinmetz and Freek Stulp

Learning and Interactive Design of Shared Control Templates

Gabriel Quere, Samuel Bustamante, Annette Hagengruber, Jörn Vogel, Franz Steinmetz and Freek Stulp

***Abstract*— Controlling a robotic arm to achieve manipulation tasks is challenging for humans. Especially if only low-dimensional input signals can be provided, as is often the case for users with motor impairments. Using shared control to provide task-specific guidance and constraints facilitates control – for instance with the Shared Control Templates (SCT) framework – and enables even complex activities of daily living to be performed successfully. However, designing SCTs is a laborious task requiring robotic expertise. To make such design easier and faster, we propose a method for semi-automatically designing SCTs on the basis of demonstrations. Furthermore, we propose two similarity metrics, and demonstrate how these can be used to transfer knowledge from one SCT to another. We demonstrate that the SCTs so acquired can be successfully used in shared control for everyday tasks such as opening a drawer or a cupboard on our assistive robot EDAN.**

I. Introduction

In shared control, human input commands are combined with semi-autonomous control systems to achieve a common goal. Examples include the control of assistive devices, such as our wheelchair-based robotic assistant EDAN [1]. EDAN is an electrical wheelchair with a robotic arm and hand, and its aim is to enable users with impairments to perform activities of daily living independently. Controlling the robotic arm and the wheelchair is done with a continuous 3D signal from either a joystick or signals extracted from the muscles with surface electromyography (sEMG). Shared control alleviates the many difficulties that arise from the direct control of a high degree-of-freedom (DoF) system, such as cumbersome mode switches between position, orientation, gripper and wheelchair control [2].

We have previously proposed Shared Control Templates (SCT) [3], which are shared control skills particularly suited for tasks that consist of different phases. For instance, a bottle should be kept upright during transport towards a glass. However closer to the glass, it should be tilted with its opening above the glass. SCTs achieve this by defining a skill as a Finite State Machine (FSM), with one state for each phase of a task. Within each state, the low-dimensional user input commands are mapped to task-relevant end-effector motions, e.g. during transport inputs map to translations, but during pouring these same inputs map to tilting motions of the bottle above the glass [3]. Furthermore, constraints on the end-effector pose ensure that task requirements are not violated, e.g. not tilting too much or not hitting the table during pouring.

All authors are with the Institute of Robotics and Mechatronics, German Aerospace Center (DLR), Wessling, Germany. Email: Gabriel.Quere@dlr.de. This work was supported by the Bavarian Ministry of Economic Affairs, Regional Development and Energy(StMWi) by means of the project SMiLE2gether (LABAY102).

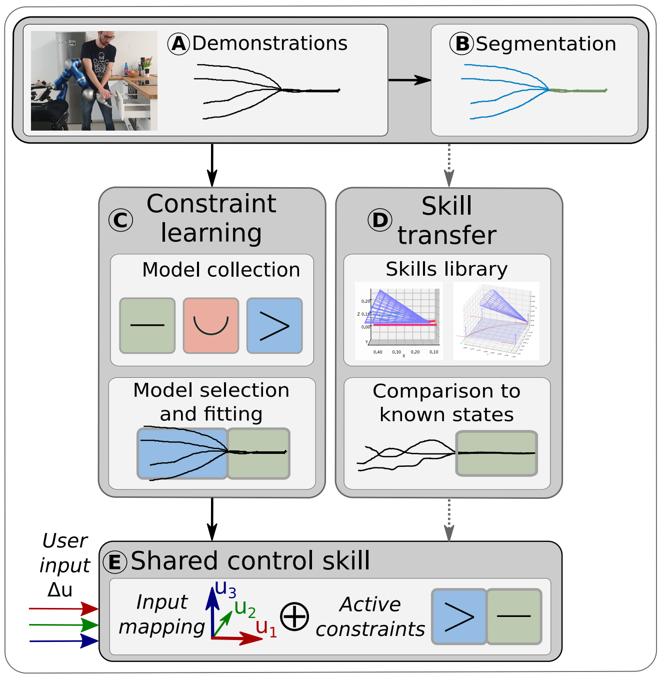


Fig. 1: SCT design method. **A.** Gathering of demonstrations via kinesthetic teaching or direct teleoperation. **B.** Data segmentation according to contacts or trajectory curvature. **C.** Models are selected and fitted to represent the constraints of each phase of the task. **D.** Optionally, information from previously learned SCTs can be used if phases are similar. **E.** An SCT is represented as a Finite State Machine with state specific Input Mapping and Active Constraints.

To allow such goal-directed support, SCTs – which are detailed in Section III – assume that the goal is known or inferred. However, the user is always in control, as there is no robot motion without user input, and they can at any time stop the task or change to another task.

A shortcoming of SCTs is that they require an expert programmer to manually code the command mapping and constraints for each task. To overcome this, our aim is to automatically learn SCTs from demonstrations, to make the design of new shared control skills faster and more intuitive. To do so, the first contribution of this paper – described in Section IV – is a method for semi-automatically learning Shared Control Templates from demonstrations, as illustrated in Fig. 1. We start with a set of end-effector trajectories which are segmented based on trajectories curvature or contact forces. Constraints can be learned from those segments. Our algorithm helps to determine which models best represent the constraints underlying the different phases of the task.

Additionally, demonstrations for a new SCT can be compared to a library of existing SCTs, to identify states matching the demonstrations. This allows the transfer of components from existing SCTs to a new SCT. Our second contribution – described in Section IV-E – is to define two comparison metrics to identify known states matching new data, to enable this transfer. This process can serve to reduce the number of demonstrations required to build an SCT, or reduce the number of parameters which need to be hand-designed.

To validate those contributions, we verify that the SCTs that have been extracted from the demonstrations support users during complex tasks of daily living, given only 3D input. This evaluation is done on the EDAN robot, and described in Section V.

II. Related work

In shared control of a robot, a user and an assistive system jointly command the robot state variables, such as the end-effector position or velocity. To this end, various complementary approaches have been developed.

First, one may realize shared control as a policy blending between the user-created teleoperation trajectory and the robot commands, resulting from a motion planner or a control policy, as formalized by Dragan et al. [4]. This has been used to guide a teleoperated robot with an invasive brain-machine interface with constraints in the form of capture envelopes [5]. Mehr et al. [6] additionally infer end-effector constraints online while the user is doing a task.

A second facet is to constrain the available workspace of the robot, as is also done in some of the above works [5], [6]. This control method usually called virtual fixtures or active contraints was surveyed by Bowyer et al. [7]. Constraints can take various forms; in particular they can be applied to any dimension of the task space, usually either the end-effector position, its orientation or the full pose. Simple constraints like lines, planes, axial rotations or fixed joints can be used [8]. Restrictions in the 3D Euclidean manifold when using multiple demonstrations can also be done with generalized cylinders [9]. Another example is from Zeestraten et al. [10] who use statistical trajectory representations to constrain the end-effector orientation depending on its position in teleoperation scenarios.

Third, one can restrict the workspace by modeling the mapping of the user input. For example, Losey et al. [11] learn an input mapping conditioned on the end-effector pose, while Herlant et al. [2] present a human-inspired metric to select the end-effector DoF controlled by the user at any instant. Both methods implicitly constrain the available workspace.

Our approach makes use of some of those constraint models in a multi-model representation. This can be achieved by constraining the end-effector with multiple models at once, for example on different DoFs, or by having a skill representation in multiple sections, corresponding to different phases of the task. To do so, SCTs are represented with a Finite State Machine. A FSM is a finite set of states connected by transitions, with one state active at a time. Its use is quite common [12], [13]. Some works learn the transitions [14], while others learn both the segmentation of the data in different states and the model fitting in parallel [15]. In this work, we segment considering either trajectory curvature or contact points, and focus on the constraints representation, using the FSM model of our previous work [3].

One of the contributions closest to ours is C-learn [16], where a similar multi-model representation is used to build a library of skills, which can then be reused. The key difference is that we use it for shared control instead of supervised autonomy. As such, we use a FSM with flexible transitions instead of a succession of keypoints. We also use different constraints models and metrics for skill transfer due to our focus on shared control.

III. Shared Control Template (SCT) Formalism

As illustrated in Fig. 2, an SCT takes low-dimensional user commands as input and output desired end-effector poses $g_{\mathcal{E}}$, applying various Input Mappings (IM) and Active Constraints (AC) during the individual phases of a task. Those phases are modeled with a FSM, where transitions between states depend on predicates with metrics such as estimated force exceeding thresholds, distances between feature frames or timeouts. On EDAN, those values are computed from a world representation built by a perception system, see [1] for details. When the defined transition predicate of a state becomes true during the execution a task, the state changes automatically.

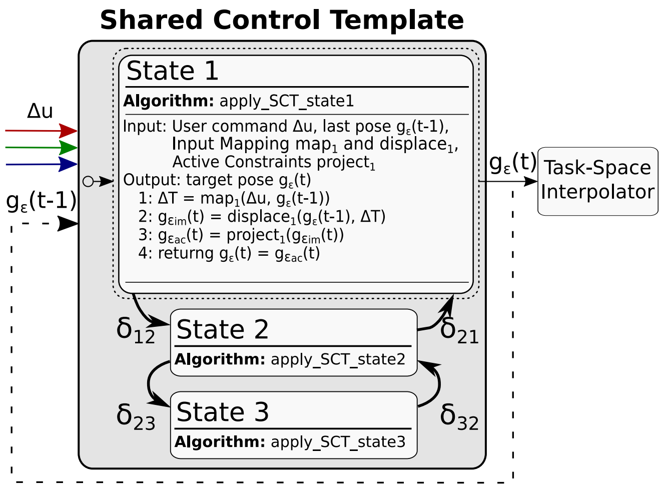

Shared Control Template
Δu
$g_\varepsilon(t-1)$
State 1
Algorithm: apply_SCT_state1
Input: User command Δu, last pose $g_\varepsilon(t-1)$, Input Mapping map_1 and $displace_1$, Active Constraints $project_1$
Output: target pose $g_\varepsilon(t)$
1: $\Delta T = map_1(\Delta u, g_\varepsilon(t-1))$
2: $g_{\varepsilon im}(t) = displace_1(g_\varepsilon(t-1), \Delta T)$
3: $g_{\varepsilon ac}(t) = project_1(g_{\varepsilon im}(t))$
4: returng $g_\varepsilon(t) = g_{\varepsilon ac}(t)$
$g_\varepsilon(t)$
Task-Space Interpolator
δ_{12}
δ_{21}
State 2
Algorithm: apply_SCT_state2
δ_{23}
δ_{32}
State 3
Algorithm: apply_SCT_state3

Fig. 2: Overview of an SCT, which is modeled as a Finite State Machine. In each SCT state an Input Mapping and Active Constraints can act. Transitions δ_{ij} need to be defined to switch between states. The output $g_{\mathcal{E}}(t)$ of the active SCT state is given to the task space interpolator, in order to apply it on the robot. To limit error accumulation, $g_{\mathcal{E}}$ is reset to the output of the interpolator each time user commands are null.

Within the active state q, at every time-step iteration, the IM are first applied. An IM first computes a displacement ΔT :

$$\begin{aligned} \text{map}_q \colon R^n, SE(3) &\to SE(3) \\ \Delta u(t), g_{\mathcal{E}}(t-1) &\mapsto \Delta T \end{aligned} \tag{1}$$

which is then applied on the end-effector pose $g_{\mathcal{E}}$:

$$\begin{aligned} \text{displace}_q \colon & SE(3), SE(3) \to SE(3) \\ & g_{\mathcal{E}}(t-1), \Delta T \mapsto g_{\mathcal{E}\text{im}}(t) \end{aligned} \quad (2)$$

with $\Delta u(t)$ the user command for a time step ($\Delta u \in [-1, 1]^K$, K the dimension of the user input, $g_{\mathcal{E}\text{im}}(t) = \Delta T * g_{\mathcal{E}}(t-1)$, and finally $g_{\mathcal{E}}(t-1)$ and $g_{\mathcal{E}}(t)$, the target pose of the end-effector at time-steps $t-1$ and t, respectively. Our default IM is to map the 3-DoF input to x, y, z translations of the end-effector. However when pouring liquid for instance, an IM could map to a rotation around the tip of the object grasped by the robot end-effector.

After applying the Input Mapping, which results in the end-effector pose $g_{\mathcal{E}\text{im}}(t)$, geometric constraints can be applied. An Active Constraint from a model m applies a projection function project_m of the form

$$\begin{aligned} \text{project}_m \colon & SE(3) \to SE(3) \\ & g_{\mathcal{E}\text{im}}(t) \mapsto g_{\mathcal{E}\text{ac}}(t) \end{aligned} \quad (3)$$

where $g_{\mathcal{E}\text{ac}}$ is the constrained end-effector pose. An AC may also apply to specific DoFs only. For example, to open a door, the end-effector is constrained on the arc-circle path traced by the door handle. Even with a 3 DoF Cartesian IM, $g_{\mathcal{E}}$ will be restricted to the 1 DoF arc-circle. In opposition to *force constraints* for haptic teleoperation which provide more forces the more they are violated, ACs can not be overcome by the user.

We define project_q as the sequential computation (if multiple models are defined) of project_m in state q. Finally, we define step_q as the composition of the map_q, displace_q and project_q functions, i. e. the action of an SCT on the end-effector target pose $g_{\mathcal{E}}$ at one time-step.

To sum up, ACs operate on poses and explicitly constrain the workspace of the end-effector, while an IM does so implicitly and operates on velocities. Providing appropriate input mappings and active constraints during different phases of the task facilitates the achievement of the task. We refer to [3] for additional details on SCTs.

A. *Constraint representation*

Our aim is to semi-automatically build SCTs from demonstrations. Whereas the previous section focused on explaining SCTs top-down, in this section, we explain the individual building blocks – the constraints – from which they are built bottom-up.

There is a vast literature on constraints representations [7], from which we use three different classes of models: 3D surfaces, 3D volumes, and a third one related to force control. Some examples are shown in Fig. 3. For example, pulling open a drawer is a heavily constrained motion, which can be represented with a prismatic constraint model. On the other hand, approaching the drawer handle can be achieved when constrained by a curved funnel, providing the user with a large workspace to move the end-effector, in case they want to switch to another task for example.

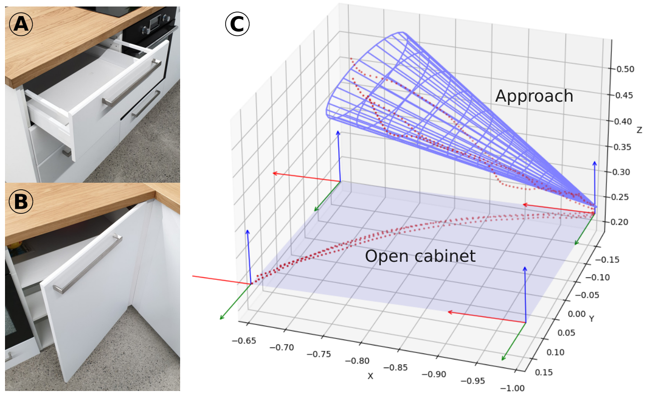


Fig. 3: Examples of constraints. **A.** A prismatic motion is needed to open the drawer. **B.** An axial rotation is needed to open the cabinet. **C.** Two phases from the data recorded by opening the cabinet door, fitted by a cone and a plane, respectively.

1) Parameterized surfaces: Explicit constraints such as lines or axial rotations are needed in constrained object manipulation, e. g. while pulling on a drawer or opening a door. They deliver consistent behavior from the user point of view, and can be attached to a semantic meaning, like a drawer being open or closed. The constraints we use in this work, with their respective DoFs, split along translation and rotation, are plane (2+3), planar (2+1), line (1+3), prismatic (1+0), arc circle (3+0) and axial rotation (0+1).

2) Parameterized volumes: A parametric representation of explicit volumes like cones, curved funnels or cylinders can provide practical and easily identifiable constraints. They are useful to guide the user motion while leaving some freedom for fine control, e. g. towards the beginning or the end of the task, or to limit the end-effector to safe regions. Implemented volumes currently only concern the position (i.e. with constrained DoFs (3,0)), but future work could consider orientation, e. g. Task Space Regions [17].

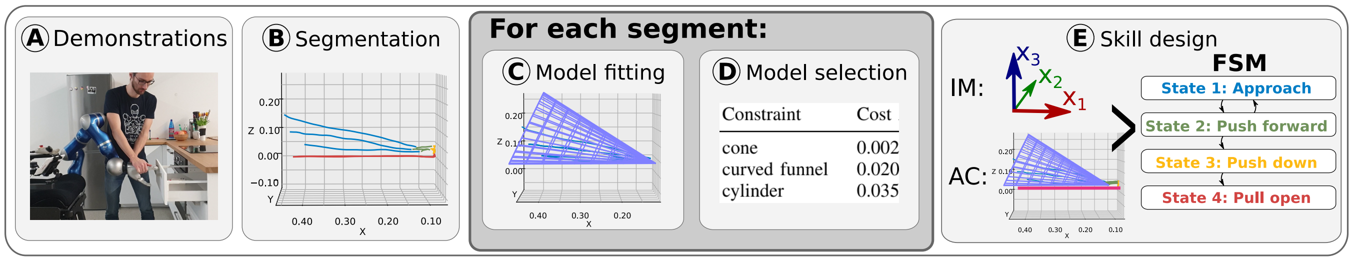
3) Force constraints: Depending on the task and the end-effector used, adding a force component to the control of the end-effector can be efficient, for example to keep the robot hand pressed on the drawer handle. This hybrid position/ force control has already been explored in various works; here, within a specific state of an SCT, we simply apply a constant force in a specific direction. This is controlled with a Cartesian impedance controller [18] via a virtual end-effector target. Force constraints can easily be computed from the data by taking the mean direction and value of the force from the input data.

IV. Interactive Design Procedure

The pipeline for learning a new SCT is depicted in Fig. 4 and presented in this section.

A. *Data acquisition through robot demonstrations*

To build an SCT, multiple demonstrations are recorded to help capture variability within the task. Usually, up to ten demonstrations are sufficient. The recorded data is composed of end-effector trajectories and forces applied on the end-effector during a task. The kinematic part can be acquired



Constraint	Cost
cone	0.002
curved funnel	0.020
cylinder	0.035

Fig. 4: Pipeline for interactive SCT design. **A.** Demonstration data is acquired. **B.** Recorded data is segmented. **C.** Multiple constraints models are fitted for each segment. **D.** The constraints are heuristically ordered based on a cost for the most adapted ones to be selected by the SCT designer. **E.** Finally, the Input Mappings and the Active Constraints of each state are specified by the SCT designer and assembled as a FSM.

from demonstrations in teleoperation or by kinesthetic teaching. Force recordings can be obtained with either an end-effector force sensor or with joint torque sensors during teleoperation. The target object is used as a reference point, to have an object-centric SCT representation.

B. *Segmentation*

The recorded data is then segmented to obtain a set of segments S, each containing portions of each trajectory. The goal of the segmentation process is to assemble data from the trajectories subjected to the same constraint. This forms the different states of an SCT and allows to model the AC.

Trajectories are first pre-processed with Dynamical Time Warping [19] to adjust timestamps. Then, segmentation candidate points are created on each demonstration, using either sharp turns in the curvature (following [20]) or the contact forces on rising and failing edges, under a minimal segment length condition. Finally, each segmentation point is only kept if there is also a segmentation point on each of the other trajectories at a time step in close vicinity.

As an example, the segmentation results in four segments when considering the demonstration of the SCT *open drawer* are shown in Fig. 4.B. The first segmentation point arises when the end-effector arrives at a pre-grasp pose, the second when it gets in contact with the drawer and the third when it starts to pull the drawer open. The contact force and its location can be detected and estimated using a momentum-based framework [21].

C. *Constraints fitting*

Once data is segmented, geometric constraints can be assigned to the segments (see Fig. 4.C & 4.D). Here, for every segment, all geometric models described in Section III-A.1 and III-A.2 are fitted on the trajectories and the force constraint (Section III-A.3) is calculated from the force profile. Parameters of parameterized surfaces are learned by fitting the geometric models to the kinematic information as described in [8]. To fit parameterized volumes on the input data, models are first initialized on a few random points, then the black box optimization algorithm CMA-ES [22] is used, optimizing the cost defined in (6). To reduce the probability of finding a bad local optimum, the best out of 20 runs is taken for each model.

A cost value is determined for each class of constraints models, which helps the skill designer in choosing the most appropriate constraints. The cost functions make use of a custom distance metric D between two poses g_A and g_B, defined as

$$D(g_A, g_B) = \|\text{position}_A - \text{position}_B\| + \alpha \cdot |\theta|, \tag{4}$$

calculating the sum of the Euclidean distance in meters and the weighted angle θ in radian between g_A and g_B in an axis-angle representation. The weighting term or trade-off factor α is set to 0.2.

The cost function for a parameterized surface model m_S is defined using (3) and (4) as

$$\text{cost}_{\text{surface}}(m_S) = \frac{1}{N}\sum_{n=1}^{N} D(g_{\mathcal{E}}(n), \text{project}_{m_S}(g_{\mathcal{E}}(n))) \tag{5}$$

with number of recorded poses N .

The cost function for a parameterized volume model m_V is also defined using (3) and (4) as

$$\begin{aligned}\text{cost}_{\text{volume}}(m_V) = &\frac{1}{N}\sum_{n=1}^{N} D(g_{\mathcal{E}}(n), \text{project}_{m_V}(g_{\mathcal{E}}(n))) \\ &+ \beta \cdot \text{volume}(m_V)\end{aligned} \tag{6}$$

with trade-off factor β empirically set to 0.4, leading to higher costs for bigger volumes.

Having calculated the costs for every geometric constraint, the models are ordered by their costs within each model class (surfaces and volumes). An example of this can be seen in Table I. Models of different classes are not compared with each other as they are calculated with different cost functions. Based on these costs, as well as a visualization for each model, a skill designer can choose the most appropriate geometric constraint.

D. *FSM building*

Once AC have been modeled for each state, the skill designer can finalize the skill representation. First, transitions between states have to be implemented to build the FSM. The IM of each state can then be specified if the default Cartesian control mapping is not appropriate, for example by scaling down motions in DoFs perpendicular to the direction of motion. Finally, various parameters like the end-effector fingers configuration or the safety thresholds (i.e. force limits the robot may not exceed) can be set. This methodological step is conceptualized in Fig. 4.E.

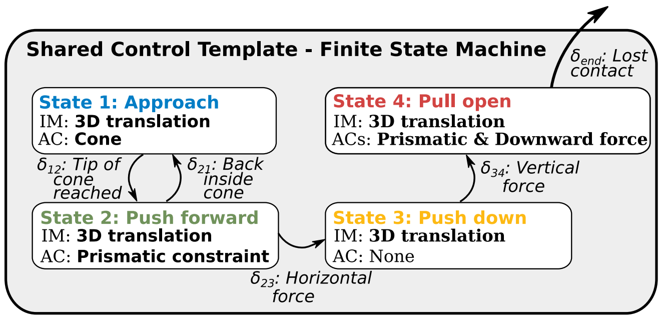


Fig. 5: Results of the pipeline from Fig. 4. We show an SCT - in the form of a FSM - semi-automatically built from demonstrated data. The SCT consists of four states with different Input Mappings (IM) and Active Constraints (AC). The transitions (δ) are specified by the skill designer.

For our *open drawer* example, the final result is shown in Fig. 5. Here, the skill designer has set the transitions as distance to the handle (between states 1 and 2) and force threshold (for the others). A default Cartesian control IM is selected for each state, leaving no possibility for the user to control the orientation, as it is not required for this task.

E. Transferring knowledge for new shared control skills

Our multi-model FSM representation can optionally be used to bootstrap the learning of new SCTs from an existing SCTs library, for example to reduce the number of required demonstrations or transfer hand-defined parameters. As a concrete example, once the SCT *open drawer* has been learned from demonstrations (using the pipeline from Section III), it can be used to help build a new SCT *open cabinet door*.

Although the last section of opening the cabinet door has different constraints, the constraints for the first three states, guiding the grasp of the handle, can be transferred from the *open drawer* SCT, as in our experimental kitchen setup the drawer and cabinet have the same type of handle.

Given a set S of segments from newly demonstrated and segmented data, each segment s is compared to every state q of a learned SCT of interest with two similarity metrics. The first metric, m_{AC}, only takes the ACs into account and considers the demonstrated data as a set of poses. A set of N_s poses consists of all poses of all segments of trajectories of s. m_{AC} is defined similarly to (5):

$$m_{AC}(s,q) = \frac{1}{N_s}\sum_{n=1}^{N_s} D(g_{\mathcal{E}}(n), \text{project}_q(g_{\mathcal{E}}(n))) \quad (7)$$

The function project_q projects the pose onto the workspace permissible by the ACs. This metric calculates the average distance between the demonstrated poses of the given segment to those same poses constrained by the ACs.

Similarly, we define a second metric m_{IM+AC} which considers the demonstrated data as a set of displacements between consecutive poses $g_{\mathcal{E}}(n)$ and $g_{\mathcal{E}}(n+1)$. It takes as input a known state q and its associated IM and ACs. For each demonstrated displacement, we evaluate if this displacement could have been commanded by the user with state q being active, and if not look for the most similar commanded displacement. Hence, this metric computes how close demonstrated data can be reproduced under the IM and ACs of state q. For example, using this metric on the data used to learn the *Approach* state while considering the *Approach* state itself as reference state will bring a null or almost null cost, while using data for tasks requiring completely different motions, positions or orientations will have a high cost.

Computing this cost means applying the optimal commands with $step_q$ to reproduce the demonstrated trajectories as best as possible under the constraints of state q. Those optimal commands are not known, and the function $step_q$ is non-invertible in the general case, i. e. there is no function that can determine the command responsible for a specific displacement from one recorded pose to the next. Therefore, the optimal command to get the closest constrained displacement is found using a Evolution Strategy (ES) algorithm on the command input. The ES algorithm uses the latest best command (initialized as 0 for the first displacement) to initialize its current solution and only keeps the best solution at each iteration, see Algorithm 1.

Algorithm 1 Evolution Strategy: Computes the optimal command to find the closest displacement from $g_{\mathcal{E}}(n)$ to $g_{\mathcal{E}}(n+1)$, constrained by the IM and ACs of state q.

```
Input: $g_{\mathcal{E}}(n), g_{\mathcal{E}}(n+1), \Delta u, \text{step}_q$
Output: $\text{closest\_constrained\_pose}_n$
 1: while not_converged do
 2:     solutions = $[\text{sol}_1, \text{sol}_2, ...]$,  $\text{sol}_i \sim \mathcal{N}(\Delta u, \sigma)$
 3:     for sol in solutions do
 4:         $\text{cost}_{\text{sol}} = D(\text{step}_q(\text{sol}, g_{\mathcal{E}}(n)), g_{\mathcal{E}}(n+1))$
 5:     end for
 6:     $\Delta u = \text{argmin(cost)}$
 7:     if successful_mutations_ratio $< 0.2$ then
 8:         $\sigma = \sigma * (1 - 0.6)$
 9:     else
10:         $\sigma = \sigma * (1 + 0.6)$
11:     end if
12: end while
13: $\text{closest\_constrained\_pose}_n = \text{step}_q(\Delta u, g_{\mathcal{E}}(n))$
```

This simple black-box algorithm is efficient because of the low DoF of the input, all the input dimensions having the same order of magnitude, and the low complexity of $step_q$. We use those $\text{closest\_constrained\_pose}_n$ to compute $m_{IM+AC}(s,q)$, similarly to Eq. 7.

Those two metrics provide for each segment s of the demonstrated trajectories a similarity measure to the states of learned SCTs. An example for this is shown in Table II. Those metrics guide the skill designer in the decision to associate a known state q to a newly demonstrated segment. If that is the case, components such as the IM and SCT parameters such as the end-effector finger configuration of q are transferred to a newly created state based on s. The ACs can be transferred as well or alternatively trained on the new data using the same model.

To sum up, those two metrics are defined independently from the models used for the skill representation. They favor the reuse of states from learned SCTs of interest.

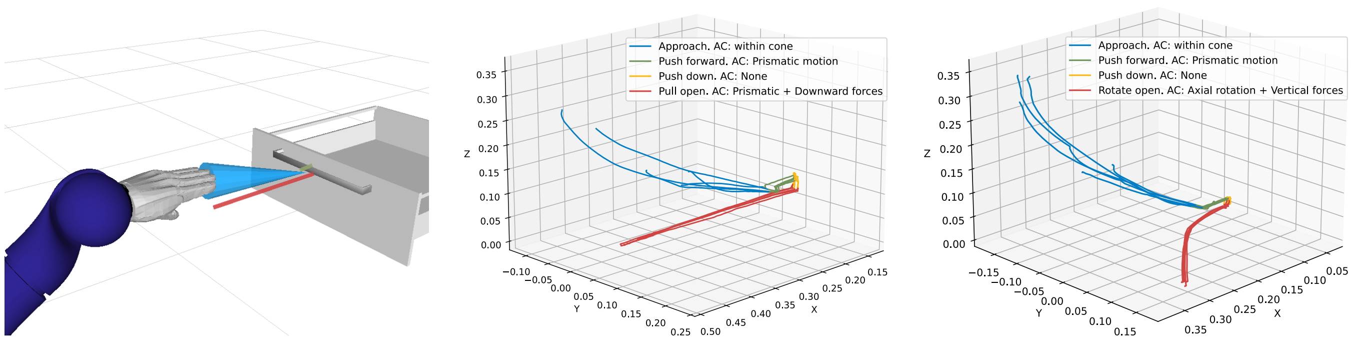


Fig. 6: Learned SCTs. We show a visualization of the learned *open drawer* SCT (left), and the measured trajectories of the real robot of five executions of opening a drawer (center) and a cabinet door (right) with a 3 DoF joystick. The colors illustrate the four different states of each SCT. The main difference between the two SCTs is the fourth state, where an axial rotation constraint is used for *open cabinet door* instead of the prismatic motion of *open drawer*.

Constraint class	Constraint	Cost Approach	Cost Pull
Parameterized Surfaces	plane	0.022	0.001
	line	0.051	0.004
	planar	0.035	0.004
	arc circle	0.051	0.008
	prismatic	0.060	**0.013**
	axial rotation	0.063	0.031
Parameterized Volumes	cone	**0.002**	0.0006
	curved funnel	0.020	0.009
	cylinder	0.035	0.012
Force	N	4	23
	Direction	[-0.5,-0.7,-0.3]	[0.2,0.3,0.9]

TABLE I: SCT *open drawer* models fitting results for segment 1. Due to the lack of an end-effector force sensor, segmentation results come from kinesthetic demonstrations, seen in Fig. 4.B, while force values arise from an extra teleoperated demonstration. The selected models are highlighted in bold.

V. Evaluation

The aim of this evaluation is to demonstrate that the extracted SCTs can be used for successful task completion. For more details on the general functionality of the Shared Control Templates concept, the EDAN user-interface and related user-studies, we refer to [3], [23], [24], [25].

We build SCTs for two tasks: the first one is learned uniquely from the demonstrations, then serves to populate an empty SCT library. The second showcases an example of SCT transfer.

A. Learning to open a drawer from demonstrations

With an empty SCT library, we first learn the SCT *open drawer* from three kinesthetic demonstrations (plus an extra one in teleoperation to compute a force estimate). The trajectories are automatically segmented according to the curvature method, resulting in four segments for each trajectory, as seen in Fig. 4.B. A variety of models are learned on those segments, providing the designer with the results seen in Table I.

The designer then selects different models: a cone for the first state to guide the user to a pre-grasp pose while leaving a large freedom of motion, prismatic motions to get in contact with the drawer and when pulling it, and a force applied on the handle when pulling it so that the end-effector doesn't slip. Finally, transitions are specified and the default IM is set. The final skill representation is shown in Fig. 5 and Fig. 6.Left.

B. Learning to open a cabinet door from demonstrations and from the SCT open drawer

The second task of interest is to open a cabinet door. One demonstrated trajectory is recorded with kinesthetic teaching and automatically segmented. Since the two tasks are similar, the resulting segments match the ones used to learn *open drawer*. Then for each segment of the demonstrated trajectory and each state of *open drawer*, m_{AC} and m_{IM+AC} are computed, with results shown in Table II.

Segment \ State	Approach	Push forward	Push down	Pull
1	AC: **0.022** IM+AC: **0.013**	AC: 0.061 IM+AC: 0.057	AC: 0 IM+AC: 0.001	AC: 0.08 IM+AC: 0.078
2	AC: 0.039 IM+AC: 0.030	AC: **0.005** IM+AC: **0.005**	AC: 0 IM+AC: 0.002	AC: 0.030 IM+AC: 0.031
3	AC: 0.054 IM+AC: 0.044	AC: 0.013 IM+AC: 0.017	AC: 0 IM+AC: 0.001	AC: 0.016 IM+AC: 0.013
4	AC: 0.153 IM+AC: 0.1	AC: 0.082 IM+AC: 0.084	AC: 0 IM+AC: 0.003	AC: 0.062 IM+AC: 0.064

TABLE II: Comparison of a newly recorded *open cabinet* trajectory to the SCT skill *open drawer*. In bold the relevant results for components transfer. Costs from state *Push down* are negligible as there are no AC. m_{AC} and m_{IM+AC} are similar because the default IM is used for each state in this example, but this could change greatly depending on the chosen IM.

Guided by those results and his prior knowledge, the skill designer builds the first three states of *open cabinet* as copies of the SCT *open drawer*, which correspond to grasping an identical handle (see Fig. 3.B), with their associated transitions. We can note in particular, highlighted in bold in Table II, the similarities between segment 1 and state *Approach*, as well as segment 2 and state *Push forward*. The fourth segment does not match any of the existing states and therefore is subject to a new set of constraints, which is modeled by a combination of *axial rotation* to constrain the position in the horizontal plane and a downward force constraint for stability of the contact with the handle.

C. Successful task completion with learned SCTs

To demonstrate that the SCTs that have been extracted from the demonstrations enable users to execute tasks even with low-dimensional command signals, we demonstrate (as able-bodied authors of this paper) five successful executions of opening the drawer and the cabinet door while teleoperating EDAN. Command inputs are generated with a 3

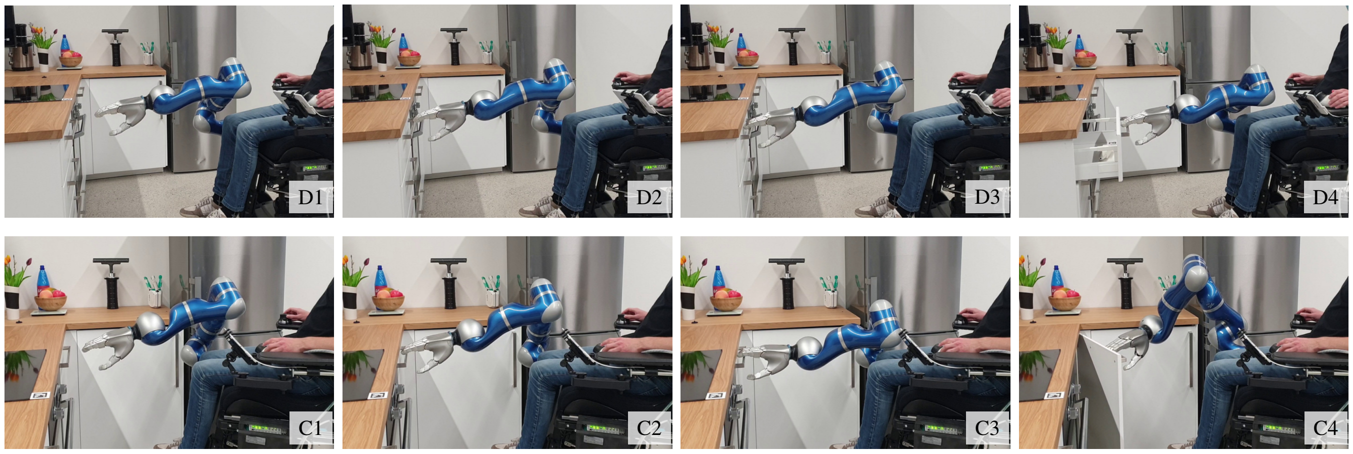


Fig. 7: Photo series of the different states of *open drawer* in the upper row (D1-D4) and *open cabinet door* in the lower row (C1-C4). The robot is controlled with a 3D joystick by the user sitting in the robotic wheelchair. **D1 & C1** *Approach*: The robotic manipulator is restricted within a cone to guide the user. **D2 & C2** *Push forward*: After reaching the pre-grasp pose, the user gets to the drawer through a prismatic motion. **D3 & C3** *Push down*: Transition to the next state is upon contact with the handle. **D4** *Pull open*: A prismatic constraint and downward forces lead the robot to robustly pull open the drawer. **C4** *Rotate open*: An axial rotation contraint and downward forces are in effect to open the cabinet door.

DoF joystick. During execution of the *open drawer* task, albeit only the end-effector is controlled by the user, the wheelchair adjusts its position automatically to keep end-effector manipulability by means of a whole-body impedance control, presented in [26].

The constraints of *open drawer* and the trajectories resulting from the SCT skill's executions are shown in Fig. 6 and the attached supplementary video. The trajectories illustrate that some constraints (e.g. the approach cone) provide more freedom of movement (e.g. when starting to approach the drawer handle) whereas other movements are more heavily constrained (e.g. the linear movement required to open the drawer).

VI. Discussion

The presented method speeds up the design of SCTs compared to [3], as it supports the skill designer in the most time-consuming aspect of the SCT creation process, namely the design of constraints. It does so both by auto-fitting and evaluating multiple models, as well as by transferring components from known SCTs.

The current number of different model classes may not be that high, but the underlying methods proof the viability and success of the approach. Expanding the range of model classes will widen the scope of our skills, without burdening the skill designer with additional work, but only add more freedom of choice. Models of interest are for example Task Space Regions [17], GMMs [10], Generalized cylinders [9] and input mappings conditioned on the end-effector pose [11]. Nevertheless, the current model classes already give each SCT state more semantic meaning. For example, it models that drawers are to be opened with a prismatic motion and doors with a rotational motion.

The calculated costs for the different models may raise the question, why – based on these costs – the best model cannot be chosen automatically. This can be explained using the trajectory segments for pulling open a drawer. Here, a plane has the lowest cost, but arguably the phase is best constrained by a prismatic motion. This is were the knowledge of the skill designer comes into play. Additional refinement of the cost metric, e.g. higher costs for higher dimensions, might be a point for further automation.

Future work will also consider learning from demonstrations the transitions between states, to further facilitate the SCT design.

VII. Conclusion

In this paper, we have proposed a method to semi-automatically learn Shared Control Templates (SCT) with a multi-model representation from demonstrations. We also presented an approach that allows transferring knowledge from existing SCTs to new ones. The evaluation on a real-world scenario proofed the effectiveness of our methods. The reduced time required for creating new SCTs allows to provide users with impairments more SCTs that are individually fitted to the requirements of the user and the environment, enabling more activities of daily living.

Acknowledgments

The authors thank Maged Iskandar, Peter Lehner, Maximilian Mühlbauer and Hanna Riesch for the support in writing this paper.

References

[1] J. Vogel, A. Hagengruber, M. Iskandar, G. Quere, U. Leipscher, S. Bustamante, A. Dietrich, H. Höppner, D. Leidner, and A. Albu-Schäffer, "EDAN-an EMG-controlled daily assistant to help people with physical disabilities," in *Int. Conf. Intelligent Robots and Systems (IROS)*, 2020.

[2] L. V. Herlant, R. Holladay, and S. Srinivasa, "Assistive teleoperation of robot arms via automatic time-optimal mode switching," in *The Eleventh ACM/IEEE International Conference on Human Robot Interaction*. IEEE Press, 2016, pp. 35–42.

[3] G. Quere, A. Hagengruber, M. Iskandar, S. Bustamante, D. Leidner, F. Stulp, and J. Vogel, "Shared control templates for assistive robotics," in *Int. Conf. Robotics and Automation (ICRA)*. IEEE, 2020, pp. 1956–1962.

[4] A. D. Dragan and S. Srinivasa, "A policy-blending formalism for shared control," *The International Journal of Robotics Research*, vol. 32, no. 7, pp. 790–805, 2013.

[5] K. Muelling, A. Venkatraman, J.-S. Valois, J. E. Downey, J. Weiss, S. Javdani, M. Hebert, A. B. Schwartz, J. L. Collinger, and J. A. Bagnell, "Autonomy infused teleoperation with application to brain computer interface controlled manipulation," *Autonomous Robots*, vol. 41, no. 6, p. 1401–1422, 2017.

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

[7] S. A. Bowyer, B. L. Davies, and F. R. y Baena, "Active constraints/virtual fixtures: A survey," *Transactions on Robotics*, vol. 30, no. 1, pp. 138–157, 2013.

[8] G. Subramani, M. Zinn, and M. Gleicher, "Inferring geometric constraints in human demonstrations," *arXiv preprint arXiv:1810.00140*, 2018.

[9] S. R. Ahmadzadeh and S. Chernova, "Trajectory-based skill learning using generalized cylinders," *Frontiers in Robotics and AI*, vol. 5, p. 132, 2018.

[10] M. J. A. Zeestraten, I. Havoutis, and S. Calinon, "Programming by demonstration for shared control with an application in teleoperation," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1848–1855, 2018.

[11] D. Losey, K. Srinivasan, A. Mandlekar, A. Garg, and D. Sadigh, "Controlling assistive robots with learned latent actions," *arXiv preprint arXiv:1909.09674*, 2019.

[12] S. G. Brunner, F. Steinmetz, R. Belder, and A. Dömel, "Rafcon: A graphical tool for engineering complex, robotic tasks," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 3283–3290.

[13] C. Willibald, T. Eiband, and D. Lee, "Collaborative programming of conditional robot tasks," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 5402–5409.

[14] S. Manschitz, J. Kober, M. Gienger, and J. Peters, "Learning to sequence movement primitives from demonstrations," in *Int. Conf. Intelligent Robots and Systems*. IEEE, 2014, pp. 4414–4421.

[15] S. Niekum, S. Osentoski, G. Konidaris, S. Chitta, B. Marthi, and A. G. Barto, "Learning grounded finite-state representations from unstructured demonstrations," *The International Journal of Robotics Research*, vol. 34, no. 2, pp. 131–157, 2015.

[25] A. Hagengruber and J. Vogel, "Functional tasks performed by people with severe muscular atrophy using an semg controlled robotic ma-

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

[17] D. Berenson, S. Srinivasa, and J. Kuffner, "Task space regions: A framework for pose-constrained manipulation planning," *The International Journal of Robotics Research*, vol. 30, no. 12, pp. 1435–1460, 2011.

[18] A. Albu-Schäffer, C. Ott, and G. Hirzinger, "A Unified Passivity-based Control Framework for Position, Torque and Impedance Control of Flexible Joint Robots," *International Journal of Robotics Research*, vol. 27, no. 1, January 2007.

[19] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space," *Intelligent Data Analysis*, vol. 11, no. 5, pp. 561–580, 2007.

[20] S. Calinon, "Robot programming by demonstration: A probabilistic approach," 2009.

[21] M. Iskandar, O. Eiberger, A. Albu-Schäffer, A. De Luca, and A. Dietrich, "Collision detection, identification, and localization on the dlr sara robot with sensing redundancy," in *Proc. of the 2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021.

[22] N. Hansen, S. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es)," *Evolutionary computation*, vol. 11, no. 1, pp. 1–18, 2003.

[23] S. Bustamante, G. Quere, K. Hagmann, X. Wu, P. Schmaus, J. Vogel, F. Stulp, and D. Leidner, "Toward seamless transitions between shared control and supervised autonomy in robotic assistance," *IEEE Robotics and Automation Letters (preprint)*, 2021.

[24] J. Vogel and A. Hagengruber, "An sEMG-based Interface to give People with Severe Muscular Atrophy control over Assistive Devices," in *Int. Conf. Engineering in Medicine and Biology Society (EMBC)*. IEEE, 2018, pp. 2136–2141.

nipulator," in *Int. Conf. Engineering in Medicine and Biology Society (EMBC)*. IEEE, 2018, pp. 1713–1718.

[26] M. Iskandar, G. Quere, A. Hagengruber, A. Dietrich, and J. Vogel, "Employing whole-body control in assistive robotics," in *IEEE Int. Conf. Intelligent Robots and Systems*, 2019, pp. 5643–5650.