Master’s Thesis
submitted in partial fulfilment of the
requirements for the course “Applied Computer Science”

Imitation Learning for Mobile Robot Control

Michael Krawez

Institute of Computer Science

Bachelor’s and Master’s Theses
of the Center for Computational Sciences
at the Georg-August-Universität Göttingen

Georg-August-Universität Göttingen
Institute of Computer Science

Goldschmidtstraße 7
37077 Göttingen
Germany

☎ +49 (551) 39-17200
✉ +49 (551) 39-14403
✉ office@informatik.uni-goettingen.de
🌐 www.informatik.uni-goettingen.de

First Supervisor: Prof. Dr. Florentin Wörgötter
Second Supervisor: Dr. Minija Tamosiunaite
I hereby declare that I have written this thesis independently without any help from others and without the use of documents or aids other than those stated. I have mentioned all used sources and cited them correctly according to established academic citation rules.

Abstract

Imitation Learning or Learning from Demonstration (LfD) is a recent approach for teaching robots new actions by presenting the desired behavior to the robot. In this work, a LfD-approach is proposed, which was designed to learn sequential manipulation and navigation tasks with mobile robots. Hereby the demonstrated action is decomposed in a sequence of simple movement primitives and conditions for transitions between those primitives. The transition conditions are derived from the input features of the robot, whereby two different condition types are used. The first bases on the feature value ranges throughout all demonstrations, and is obtained with low computational complexity. The second is acquired from regression analysis and has a high complexity. A heuristic for complexity reduction for the second condition type is proposed, which reuses information inferred by the first type conditions. The approach was tested on two tasks in a simulated environment. The first was a manipulation task, in which the learner had to push a box into a target area, the second an indoor navigation task, where the learner moved between different rooms. The simulation results showed that the learner was capable of reproducing those tasks, whereby the complexity for learning stayed low in steps where the heuristic could be applied. However, it was high in steps in which this was not the case.
List of Symbols

\( \mathcal{F} \) Set of input features. Each quantity, denoted by \( \mathcal{F} \) with a subscript or superscript, is a subset of \( \mathcal{F} \).

\( \mathcal{F}_{rel}^{i} \) Features, which are relevant for the \( i \)th transition.

\( \mathcal{F}_{con} \) Union of all \( \mathcal{F}_{rel}^{i} \), represents the feature context of the strategy.

\( \mathcal{F}_{used} \) Set of features, estimating \( \mathcal{F}_{con} \).

\( \mathcal{F}_{expl} \) In a regression, the set of explanatory features.

\( \mathbf{f} \in [0, 1]^{|\mathcal{F}|} \) Vector containing the current values of features in \( \mathcal{F} \).

\( \mathbf{m} \in [0, 1]^{2} \) Vector containing current wheel speed values.

\( \phi \) An unspecific feature in \( \mathcal{F} \)

\( \phi^{i} \) In a regression, the dependent feature.

\( \phi_{p}^{i} \) Primary triggering feature in the \( i \)th transition.

\( \phi(\mathbf{v}) \) The value of \( \phi \) in the feature vector \( \mathbf{v} \).

\( \phi(S) \) For a set \( S \) of feature vectors, the set of values of \( \phi \) in each of those vectors.

\( \mathcal{P} \) Set of primitives in the strategy at hand. Each \( P_{i} \in \mathcal{P} \) is a vector, describing a movement primitive executed in step \( i \).

\( \mathcal{T} \) Set of transitions in the considered strategy. Each \( T_{i} \in \mathcal{T} \) describes the conditions for the transition from \( P_{i} \) to \( P_{i+1} \).

\( t_{r}^{i} \in [0, 1]^{|\mathcal{F}|} \) The value of \( \mathbf{f} \), recorded during the \( i \)th transition in the \( r \)th demonstration run.

\( R \) Number of recorded demonstration runs.

\( T_{i}^{smpl} \) Set of all recorded \( t_{r}^{i} \) during the \( i \)th transition, throughout all demonstrations.

\( d(x, y) \) Distance between the values \( x \) and \( y \), which belong to the same feature.

\( d_{\phi}(\mathbf{v}_{1}, \mathbf{v}_{2}) \) Distance between the values of the feature \( \phi \) in \( \mathbf{v}_{1} \) and \( \mathbf{v}_{2} \).

\( I_{\phi}^{i} \) Minimal interval, containing all values of \( \phi \) in \( T_{i}^{smpl} \).
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$CI^i_\phi$</td>
<td>Interval, containing most values of $\phi$ in $T_{i}^{sampl}$, similar to a confidence interval.</td>
</tr>
<tr>
<td>$\Delta^i_\phi$</td>
<td>Mean change of $\phi$ during the $i$th step throughout all demonstration runs.</td>
</tr>
<tr>
<td>$\rho^i_\phi$</td>
<td>Precision coefficient of $\phi$ in step $i$.</td>
</tr>
<tr>
<td>$u$</td>
<td>A vector, which for each feature contains its estimated likelihood to be a trigger.</td>
</tr>
<tr>
<td>$w_{\mathcal{F}_{reg}}$</td>
<td>The weight which is assigned to a feature subset $\mathcal{F}_{reg}$, based on the $u$-values of the features in that subset.</td>
</tr>
</tbody>
</table>
## Contents

1 Introduction .......................... 1

2 Method Outline .......................... 3
   2.1 Motivational Example .......................... 3
   2.2 Application Domain .......................... 4
   2.3 Learning Approach .......................... 8

3 Implementation Details 13
   3.1 Preliminaries and Notation .......................... 13
      3.1.1 Linear and Circular Features .......................... 13
      3.1.2 Intervals and Ranges .......................... 14
   3.2 Data Sequencing .......................... 15
   3.3 Feature Selection .......................... 16
      3.3.1 Changing Criterion .......................... 16
      3.3.2 Range and Regression Criteria .......................... 17
      3.3.3 Computation of the Regression Model .......................... 20
   3.4 Task Reproduction .......................... 23
      3.4.1 Regression Triggers .......................... 23
      3.4.2 Triggering Conditions .......................... 25
   3.5 Main Algorithm .......................... 28
      3.5.1 Identification of the Error-Causing Step .......................... 28
      3.5.2 Used Features and Subset Generation .......................... 29

4 Experiments and Evaluation 31
   4.1 Manipulation Task .......................... 31
   4.2 Navigation Task .......................... 40

5 Discussion 45
1 Introduction

Learning from Demonstration (LfD), also known as Imitation Learning or Programming by Demonstration, is a recent approach to robot learning, which enjoys growing popularity. The core idea of LfD is to teach a robot (the learner) new skills by simply showing the robot the necessary actions, so that a human teacher, also one unfamiliar with robotics or programming, could easily extend or change the robot’s behavior. In the last two decades many LfD-based techniques were proposed, which differ in several aspects, such as: the way the skill is presented, e.g., by capturing the movements of the teacher with a camera [Kuniyoshi et al., 1994] or by teleoperating the robot [Saunders et al., 2006]; the method a control policy is acquired from the demonstration; or the level at which the learning occurs, reaching from motor level to complex behavior. A comprehensive review and a possible categorization of existing LfD approaches is given in [Argall et al., 2009].

Almost all those approaches have to tackle two major issues, typical for LfD: “How to imitate?” and “What to imitate?” [Billard et al., 2004], [Nehaniv and Dautenhahn, 2000]). The first, also known as the correspondence problem, refers to the question of how to reproduce the presented behavior. This task is non-trivial, since the physical abilities of the robot usually differ from those of the human teacher, and a one-to-one mapping of the teacher’s movements to robot states is not possible. However, this problem can be avoided, when the task to learn is executed passively by the robot itself, while being operated by the teacher, either through moving the robot’s body (e.g., Muelling et al. [2010]) or teleoperation. Thereby the robot records its internal states, and uses them to compute the inverse kinematics, required to reproduce the presented movements.

The second, “what to imitate?”, is the problem of identifying the possibly minimal subset of input features, necessary to describe or repeat the task. Redundant or unrelated features not only increase costs for computation and storage, but more importantly, make the generalization of the learned behavior more complicated or even not feasible in practice. A straightforward work-around is to provide the robot with only those features, which are needed for the task at hand. However, this strategy works only for small tasks, for more complicated behavior, consisting of multiple subtasks, a different solution is required. Several frameworks were proposed, which enable the teacher to hint the robot at important aspects of the presented action, for instance by marking an object of interest with a laser pointer [Veeraraghavan and Veloso, 2008], etc. But still, such approaches are domain-specific, since not all features can be accentuated in the same way. In addition, the robot’s internal world representation, i.e., the set of available features, must be known to the teacher.

The LfD-approach presented in this work tries to solve the problem of “what to imitate” automatically in the domain of learning goal directed behavior with a mobile (non-humanoid) robot. In particular, the method is optimized for learning tasks which can be decomposed into a sequence of simple steps, where each step is the execution of a fixed movement pattern (movement primitive). The condition for a transition between two steps is represented by a relation on the space of input
features, however, where only a small feature subset is responsible for triggering the transition. The main goal of the method is to find such subsets.

In general, the problem of finding an optimal feature subset for a given task is exponential in the number of features, although for practical tasks, many solutions for reducing the complexity were proposed. In this thesis, the main idea of complexity reduction was inspired by a cognitive mechanism known as attentional priming, or more precisely a special case of it: A feature is more likely to draw the focus of attention, if it was considered as important in the past [Kristjánsson and Campana, 2010]. Transferred to the problem of learning a sequence of steps, this means that features which were recognized as transition triggers in previous steps, are considered as being more likely to be important for transitions between other steps of the sequence. The idea is not entirely new in the field of LfD, and was already used in, e.g., [Pardowitz et al., 2007].

The above consideration is especially useful in the domain of navigation or manipulation tasks, where the transition conditions can be loosely separated into two classes: The first contains situations where the transition is triggered when one or more features simply reach a certain value or range (e.g., a fixed position), in the second class the triggering value depends on the values of several other features, i.e., the trigger is governed by a multiple regression. For a given step, the complexity of finding all triggers from the first category is linear in the number of features, but for the second category polynomial in the number of features, with the number of explanatory features as exponent. So, the simplified design of the method is the following: First, all transitions are analyzed for triggers from the computationally “cheap” first category, and then for those from the “expensive” second category, whereby primarily feature subsets are tested which contain triggering features from the first category. The conjecture is that the number of those triggers is small compared to the number of all features, and that triggers from the second category (regressions) employ at least some triggering features from the first category. If this is the case, then the triggers from the second category would be found much faster compared to the naive approach of testing all feature subsets.

The structure of the thesis is the following: In the next section, an overview of the method and its main ideas and concepts are presented, along with a motivational example of a simple manipulation task. Section 3 provides more details about the different parts of the approach. In section 4, the algorithm is evaluated by teaching a differential wheeled robot two tasks in a simulated environment. The first is a variation of the motivational example and the second a navigation task. The main parts of the approach are examined using the results of those experiments. The strengths and flaws of the method are summarized and discussed in section 5, where also possible future extensions and improvements are considered.
2 Method Outline

2.1 Motivational Example

The following scenario is used throughout this work to explain different steps of the learning method. A differential wheeled robot shall learn how to push an object (a rectangular box) into a target area on a plain. The robot is able to move forward, backward, turn left and right in place, and also turn left and right while moving forward or backward, so that there are eight movement patterns (primitives) in total. The motor signals are specified by the vector \( m \in [0, 1]^2 \), whereby 
\[(1, 1) \text{ encodes moving forward, } (0.5, 0.5) \text{ not moving and } (0, 0) \text{ moving backward, and all other patterns are derived similarly.}\]

The sensory input consists of two speed sensors, one for each wheel, and the set \( F \) of global features. Here, speed sensor values are identical to the corresponding motor signals, i.e., when some specific \( m \) is applied to the motors, the same \( m \) is returned by the speed sensors. Therefore, by \( m \) we refer to both, the motor signals and the speed sensor values.

The global features \( F \) describe the relationship between the single parts of the set-up (i.e., between the robot, the box and the target). For instance, \( F \) could be provided by a camera, positioned above the set-up (e.g., as in [Kuniyoshi et al., 1994]), and a preprocessor extracting the features which is, however, not further objected in this work. In this example, \( F \) contains (see also figure 1): The polar coordinates \( \gamma_{br} \) and \( \text{dist}_{br} \) of the robot regarding the box as pole; The angle \( \alpha_{br} \) between the orientation vectors of the robot and the box; Polar coordinates \( \gamma_{tb} \) and \( \text{dist}_{tb} \) of the box with the target as pole, and the angle \( \alpha_{tb} \) between their orientations. Assume that the orientations of the robot, box and target are uniquely defined and can be observed by the teacher and the visual preprocessor. The current values of the features in \( F \) (normalized to the unit interval) are referenced by the vector \( f \in [0, 1]^n \), \( n = |F| \). For a feature \( \phi \in F \), \( \phi(f) \) is the value of \( \phi \) in \( f \), and for a set of feature vectors \( S = \{t_1, t_2, \ldots, t_n\}, t_i \in [0, 1]^n \)

\[
\phi(S) = \{\phi(t_i) \mid t_i \in S\}.
\] (1)

For training, an expert teacher demonstrates the pushing task by teleoperating the robot. In each demonstration run the initial positions of the robot and the box, as well as their orientations, are randomly drawn from a uniform distribution (hereby the distances stay in some finite range). However, the teacher always follows the same strategy to solve the task, which is shown in detail in figure 2. We notice that each step description consists of a primitive movement and a condition for the transition to the next step. So, the strategy is expressed by a sequence of primitives and transition conditions. The idea of sequencing or segmenting the demonstrated strategy (in one form or another) is often used in LfD-methods, e.g., in [Niekum et al., 2012] and [Konidaris et al., 2011]. The notion of pre- or postconditions for the execution of a particular sequence part is also
Figure 1: Input features. The solid black arrows show the orientations of the box, the robot and the target area, the thin dashed arrows are projections of those orientations. The thick dashed lines represent the particular distances between the target, the box and the robot.

quite common in this context, see e.g. [Nicolescu and Mataric, 2001].

Here, the transition conditions are of two kinds: In the first case, a transition is triggered when one or more features reach a fixed value or value range, for instance in step 2, where the point C1 is defined in the polar coordinates of the box, i.e., by a fixed value of $\gamma_{br}$ and $\text{dist}_{br}$. Such conditions are referred to as range conditions (or range criterion). Here an observation is used, which is often exploited in the field of LfD to reduce dimensionality: considering several demonstrations of the same task, the relevant aspects of the task will be present more often than the irrelevant ones.

In the second case, a transition is triggered when one feature reaches a value which, however, is not fixed but depends on the values of other features, as in step 1. There, the desired starring direction of the robot, $\alpha_{br}$, depends on the relative position of the robot to C1, i.e., on $\gamma_{br}$ and $\text{dist}_{br}$. Since such conditions can be described in terms of a regression, we call them regression conditions (or regression criterion). In the next section, these observations are presented more formally.

### 2.2 Application Domain

The goal of the method presented here is to learn strategies from demonstration, which have the following properties. First, the strategy to learn is representable as a sequence of steps, where in each step $i$ the robot (learner) executes a fixed movement primitive $P_i$. Here, a movement primitive is a vector representing the motor command to the robot, where each dimension stands for a particular motor. If $s$ is the number of steps in a given strategy, then $\mathcal{P} = (P_1, P_2, \ldots, P_s)$ is the
Step 1: turn, until facing the point C1

Step 2: move forward, until point C1 is reached

Step 3: turn, until facing the box

Step 4: move forward, until box is reached

Step 5: turn in place by ca. 45°

Step 6: move forward, until the corner of the bumper is close to the corner of the box

Step 7: turn and move forward simultaneously, turning the box, until its orientation is pointing at the target

Step 8: turn in place, until the robot’s orientation is the same as that of the box

Step 9: move forward and push the box into the target area

Figure 2: Strategy for pushing a box into a target area. The top part shows the subtask of reaching the box, the bottom part the subtasks of turning the box into the right direction and pushing it to the target.
sequence of primitives executed during the presentation of that strategy. Note that \( \mathcal{P} \) remains the same for all initial conditions for which the particular strategy should be applicable.

Second, each transition condition \( T_i \) between primitives \( P_i \) and \( P_{i+1} \) can be derived from the input features of the robot at the time point of the transition: Let \( \mathcal{F} \) be the set of input features available to the learner, where the value of each feature \( \phi \in \mathcal{F} \) lies in \([0, 1]\), and let \( f \in [0, 1]^n \) with \( n = |\mathcal{F}| \) be the vector containing the current values of all features in \( \mathcal{F} \). Then, in the most general case, each \( T_i \) is a relation on the feature space of the learner, i.e., \( T_i \subseteq [0, 1]^n \), and a transition between \( P_i \) and \( P_{i+1} \) occurs when \( f \in T_i \). The sequence of all such transitions for a given strategy is \( \mathcal{T} = (T_0, T_1, ..., T_s) \), where \( T_0 \) represents all initial conditions, for which this strategy is applicable.

The sequence with verbal descriptions of the \( P_i \) and \( T_i \) for the motivational example is shown in figure 3, where \( P_0 \) is considered as the start and end state, where the robot does not move.

![Figure 3](image)

Figure 3: Strategy sequence for the pushing task. For each state \( P_i \) the executed movement primitive is given, for each transition condition \( T_i \) its verbal description is provided.

So, the aim of the approach is to learn a strategy \( \mathcal{S} = (\mathcal{P}, T) \) from demonstration, i.e., from the values of the motor sensors \( m \) and feature vector \( f \), available during the presentation of the strategy. For \( \mathcal{P} \) this is rather trivial, since for each step \( i \) the value of \( P_i \) is the value of \( m \), and a transition between two steps is indicated by a prolonged change in \( m \). To learn the transition conditions during the \( r \)th presentation and the \( i \)th transition, the value of \( f \) is assigned to the vector \( t^i_r \). After the strategy was presented \( R \) times, the task is to reconstruct \( T_i \) from the set \( T^{\text{smpl}}_i = \{t^1_i, t^2_i, ..., t^R_i\} \) of recorded feature values.

For a general relation, this is not possible with a limited number of training samples. Thus, we restrict the learning method to the two forms of transition conditions mentioned in the previous section. In range conditions the \( T_i \) has the following form:

\[
T_i = [a^i_1, b^i_1] \times [a^i_2, b^i_2] \times \cdots \times [a^i_n, b^i_n], \quad n = |\mathcal{F}|
\]  

(2)

with \( 0 \leq a^i_j \leq b^i_j \leq 1 \). The transition is triggered, iff \( f \in T_i \), i.e., each feature \( \phi_j \) falls within
the corresponding interval \([a_i^j, b_i^j]\). Note, however, that not each feature necessarily triggers the transition, because several values may already be in the correct range since the previous transition (if, for example, for some feature \([a_i^j, b_i^j] = [0, 1]\), this will always be the case).

The second kind of transition condition, which is objected by the learning method, is a multiple regression on a single feature. Hereby we assume that the transition is triggered when the value of a feature \(\phi^t\) (triggering or dependent feature) reaches a value \(v^t\), which depends on the values \(v^e_1, v^e_2, ..., v^e_m\) of \(m\) explanatory features \(F_{\text{expl}} = \{\phi^e_1, \phi^e_2, ..., \phi^e_m\}\), with

\[
v^t = F (v^e_1, v^e_2, ..., v^e_m) + \epsilon, \tag{3}
\]

where \(F\) is some function with

\[
F: [0, 1]^m \rightarrow [0, 1] \tag{4}
\]

and \(\epsilon\) is an error term. There is no direct restriction on the form of \(F\), since local regression analysis is used to detect and approximate the function. However, the number of training samples to do this reliably heavily depends on the shape of \(F\). The better \(F\) can be approximated by a linear function, the less training samples are needed. The error term is a random variable, which follows an unknown but fixed symmetric distribution with zero mean, and mainly results from two effects: First, although if the expert teacher tried to follow \(F\) strictly, small deviations would still arise due to execution inaccuracies. Second, the teacher may not follow \(F\) precisely, if this is not crucial for the presented strategy. For instance, in the motivational example the exact distance of the point C1 to the box is not as important as the polar angle, since the crucial property of C1 is that it lies “behind” the box. So, in step 2, the teacher might choose a C1 with a different distance in each demonstration. Nevertheless, we assume that \(F\) describes the expected value of \(v^t\).

We also expect that in each \(T_i\) there is at most one triggering feature governed by a regression. This means that if the corresponding \(\phi^t, F_{\text{expl}}\) and \(F\) are identified, then the strategy can be reproduced correctly. But this does not imply that \(T_i\) may not contain any other regressions, since this would mean that all other features are uncorrelated, which is rather unrealistic. However, we expect that the correlation between \(\phi^t\) and \(F_{\text{expl}}\) is stronger than in other feature subsets.

Further, it is required that the robot operates in a static environment, in the sense that all changes in the input feature values during the strategy execution (except for sensor noise) are caused by the actions of the robot itself. Thus, the learning method could be applied in indoor navigation or manipulation tasks. Also, those application areas typically provide another condition, exploited by the learning approach. As will be shown in the next section, the computational complexity of identifying a triggering regression in a single \(T_i\) benefits from a large total number of range triggers throughout all transition conditions. As can be seen in the motivational example, this is often given in navigation or manipulation problems.
2.3 Learning Approach

As implied in the previous section, the main difficulty of learning a strategy is to learn the $T_i$ from the set of recorded samples $T_i^{i\!mpl} = \{t_i^1, ..., t_i^R\}$, and in particular the subset of features which are actually relevant for the transition triggering. Informally, those are features which were consciously used by the teacher to determine the correct time point of the transition. To define these features more formally, we first introduce some notation. If $\mathcal{F}$ is the set of features, $n = |\mathcal{F}|$ and $v \in [0, 1]^n$ a feature vector, then for a feature $\phi \in \mathcal{F}$, $\phi(v)$ denotes the entry in $v$, corresponding to the value of $\phi$. Further, let $\mathcal{F}_{\text{sub}} \subseteq \mathcal{F}$ and $m = |\mathcal{F}_{\text{sub}}|$ is a feature vector in $\mathcal{F}_{\text{sub}}$ with $\phi(\mathcal{F}_{\text{sub}}(v)) = \phi(v)$ for all $\phi \in \mathcal{F}_{\text{sub}}$, i.e., $\mathcal{F}_{\text{sub}}(v)$ is a reduction of $v$ to the feature subset $\mathcal{F}_{\text{sub}}$. Finally, the reduction of a relation $T \subseteq [0, 1]^n$ from $\mathcal{F}$ to $\mathcal{F}_{\text{sub}}$ is

$$\mathcal{F}_{\text{sub}}(T) = \{v \in [0, 1]^m \mid \exists t \in T : \mathcal{F}_{\text{sub}}(t) = v\}. \quad (5)$$

Now, the subset $\mathcal{F}_{\text{rel}}^i \subseteq \mathcal{F}$ of features relevant for the $i$th transition is the minimal subset, such that after a correct transition in the previous step (i.e., $f$ was in $T_{i-1}$ at that point) and the subsequent execution of $P_i$, the transition condition can be reduced to the features in $\mathcal{F}_{\text{rel}}^i$, i.e.,

$$f \in T_i \iff \mathcal{F}_{\text{rel}}^i(f) \in \mathcal{F}_{\text{rel}}^i(T_i). \quad (6)$$

As the strategy to learn should be designed in such a way, that it could be performed by a human teacher, it seems justified to assume that the maximal number of relevant features in each $T_i$ is rather small, since the capability of humans to keep in mind and operate on several items or variables simultaneously is limited to just a few. Let $m$ be the maximal number of features we expect to be relevant in any transition condition, i.e., $|\mathcal{F}_{\text{rel}}^i| \leq m$ for all $i = 1, ..., s$. Next, we consider the computational costs of finding the relevant features of a single $T_i$ with the range and regression criteria.

The basic idea of the range criterion is that more relevant features have smaller ranges in $T_i$ than the less relevant (for details, see section 3.3.2). To make the necessary computations for each feature, only the recorded values of that particular feature are needed, so that the total computational costs of the range criterion are linear in the number of features in $\mathcal{F}$.

For the regression criterion, recall that we are looking for a dependent feature $\phi^t$, which is predicted by a set of explanatory features $\mathcal{F}_{\text{expl}}$, and we assume that $\phi^t \in \mathcal{F}_{\text{expl}}^i$ if the quality of prediction is the best among all other combinations of target and explanatory features. However, the prediction quality can only be tested if $\phi^t$ and all $\mathcal{F}_{\text{expl}}$ are considered simultaneously, so that the naive method would imply to check all feature subsets. Since from the definition of $\mathcal{F}_{\text{expl}}^i$ follows that for the desired regression $\phi^t \in \mathcal{F}_{\text{expl}}^i$ and $\mathcal{F}_{\text{expl}} \subset \mathcal{F}_{\text{expl}}^i$ must hold, the search can be limited to subsets with a maximal size of $m$. First, we consider the worst-case complexity of the naive method. Let $m$ be fixed and $|\mathcal{F}| = n$, and since there are at least one dependent and one explanatory feature, in
worts case \( \sum_{i=2}^{m} \binom{n}{i} \) subsets need to be tested. This quantity can be bounded as follows (compare, e.g., [Worsch, 1994], inequality (3.1)):

\[
\binom{n}{m} \leq \sum_{i=2}^{m} \binom{n}{i} \leq (m-1) \binom{n}{m},
\]

and due to \( \frac{n^m}{m!} \leq (n/m)^m \leq (m-1) \frac{n^m}{m!} \)

\[
\frac{n^m}{m!} \leq \sum_{i=2}^{m} \binom{n}{i} \leq (m-1) \frac{n^m}{m!},
\]

from which it is easy to see that the complexity is in \( O(n^m) \). This can become infeasible already for small \( m \), if \( n \) is large.

The idea for the heuristic, used in this method to reduce the above complexity, comes from a cognitive effect called attentional priming. In one of its forms, it is the tendency of the attentional mechanism to pick up features, which are associated with some past reward. For the problem of finding relevant features, this means that we consider a feature \( \phi \) to be more likely in \( F_{rel}^i \), if it already was identified as an element of some other \( F_{rel}^j \). The intuition is that for a large number of input features, only a small subset is used for the presented strategy, which can be seen as the context of the corresponding task:

\[
F_{con} = \bigcup_{i=1}^{s} F_{rel}^i,
\]

with \( |F_{con}| \ll |F| \). Further, we hope that features in that context are reused throughout multiple steps in the presented strategy, i.e.,

\[
|F_{con}| < \sum_{i=1}^{s} |F_{rel}^i|.
\]

Let \( F_{used} \subseteq F_{con} \) be the set of context features known so far, and assume that we can decide whether a given feature subset is in \( F_{rel}^i \) (however, possibly only when the subset contains all explanatory features). Then the order of testing feature combinations would be the following. First all combinations with \( 2, \ldots, m \) features from \( F_{used} \) are considered. If no proper subset was found, then again all \( 2, \ldots, m \)-subsets are examined, but now with exactly one feature from \( F \setminus F_{used} \), then this is repeated with exactly two features from \( F \setminus F_{used} \), and so forth, until the correct combination is found. Now we look at the worst-case complexity for this approach. Let the regression which we try to find use the maximal allowed number of features \( m \), and let \( k \) of them be in \( F_{used} \), with \( u = |F_{used}| \) and \( m \leq u \). Then up to

\[
\sum_{i=0}^{m-k} \sum_{j=2}^{m} \binom{u}{j-i} \binom{n-u}{i}
\]
combinations will be tested, where \( \binom{m}{y} = 0 \) for \( y < 0 \). Reusing (8), we determine the upper bound for this expression:

\[
\sum_{i=0}^{m-k} \sum_{j=2}^{m} \binom{u}{j-i} \binom{n-u}{i} \sum_{j=2}^{m} \binom{u}{j-i} \leq \sum_{i=0}^{m-k} \binom{n-u}{i} (m-1) \frac{u^n}{m!} \leq (m - k + 1) \frac{(n - u)^{m-k}}{(m-k)! (m-1) u^n}{m!} \leq \frac{(m + 1)m}{m! m!} (n - u)^{m-k} u^m,
\]

from what the complexity of \( O(u^m(n-u)^{m-k}) \) follows. Compared to the complexity of \( O(n^m) \) of the naive approach, for large \( n \) and \( u \ll n \) this is already a big advantage for \( k = 1 \). In the case of \( k = m \) even \( O(u^m) \) can be achieved. Note, however, that the above is a heuristic, which is not guaranteed to work, so that also \( k = 0 \) is possible, whereby the approach would fall back onto the complexity of the naive method. Notice also that in the actual implementation a slightly different order for subset testing is used (see section 3.5.2), which nevertheless leads to basically the same complexity.

From the above considerations, the rough form of the learning approach is the following: First, the teacher demonstrates the strategy several times with different initial conditions, whereby the learner records the sensory input, i.e., \( f \) and \( m \). Then, the recorded data is analyzed in offline modus, where the set of primitives \( P = \{P_1, \ldots, P_s\} \), and for each \( i = 0, \ldots, s \) the set \( T_i^{\text{sample}} = \{t_i^1, \ldots, t_i^R\} \) is constructed. Next, a preliminary set \( \mathcal{F}_{\text{used}} \) of context features is created, using the range criterion on the \( T_i^{\text{sample}} \), which has a low computational complexity. Finally, the former idea is used to find the remaining relevant features with the regression criterion, employing \( \mathcal{F}_{\text{rel}} \).

The last step needs to be considered in more detail, since it was not specified how the correct feature subset for regression is actually recognized. We assume that this subset has a better goodness of prediction (which we call \( \rho_{\text{reg}} \)) than other subsets, however, it is in general impossible to find the subset with the highest \( \rho_{\text{reg}} \) without actually testing all \( m \)-subsets of \( F \). Therefore, another criterion for verifying the chosen \( \mathcal{F}_{\text{rel}}^i \) is applied, which follows from its definition: If it was chosen correctly, then the learner is able to perform a proper transition to the next step using \( \mathcal{F}_{\text{rel}}^i \) only, provided that all previous steps were also executed correctly. So, if the learner can perform the demonstrated strategy error-free, then all \( \mathcal{F}_{\text{rel}}^i \) it has chosen must be correct.

This leads to the following scheme: After constructing \( \mathcal{F}_{\text{used}} \) with the range criterion, the learner tries to solve several problem instances with the demonstrated strategy autonomously. In each step \( i \), it constructs a hypothetical set \( \mathcal{F}_{\text{rel}}^i \) with the range and regression criteria. For the regression
criterion, the learner iteratively generates feature subsets, with subsets which we consider as more likely appearing first (see section 3.5.2 for the precise method). For each subset, its $\rho_{reg}$-value is computed (see section 3.3.3), which is compared with a threshold $thr_i$. If this threshold is exceeded, the subset is considered as a hypothetical triggering regression, and the search is aborted.

Then, the subset is added to $F_{rel}^i$, based on which the learner performs the next transition (see section 3.4). If no transition is triggered within some time frame, we assume that in one of the previous steps an $F_{rel}^j$ was chosen incorrectly. In this case the learner tries to identify the step $j$, in which the failure might have been caused (details for this procedure are provided in section 3.5.1).

Since the choice of features with the range criterion is much less error-prone than with the regression criterion, we assume that the failure resulted from a wrong chosen regression, so the learner increases $thr_j$ above the $\rho$-value of the last regression, selected in that step. Initially, all $thr_i$ are initialized with zero, and are only increased if in the corresponding step $i$ a wrong $F_{rel}^i$ is supposed. So, if in some $T_i$ a triggering regression exists with $k = m$, it will be found after a small number of subsets in $F_{used}$ were tested, and thereafter the threshold will remain constant, since no failure would be produced in this step. By contrast, in the worst case, $k = 0$, the threshold will be consistently increased due to repeated failure in this step, so that ultimately all $m$-subsets of $F$ can be tested.

The approach is summarized in algorithm 1. In the next section, the details of this method are discussed.
Algorithm 1: Main Algorithm

record $f$ and $m$ during $R$ demonstration runs;
compute $\mathcal{P} = \{P_1, ..., P_s\}$;
compute $T_i^{\text{smpl}} = \{t_i^1, ..., t_i^R\}$ for $i = 0, ..., s$;
compute $\mathcal{F}_{\text{used}}$ from $T_0^{\text{smpl}}, ..., T_s^{\text{smpl}}$ by range criterion;
for all $\text{thr}_i$: $\text{thr}_i \leftarrow -1$;
repeat
    generate new problem;
    $i \leftarrow 1$;
    repeat
        $\mathcal{F}_i^{\text{rel}} \leftarrow$ range criterion ($T_i^{\text{smpl}}$);
        if $\text{thr}_i \geq 0$ then
            repeat
                subset $\leftarrow$ get next subset ($\mathcal{F}_{\text{used}}$);
                $\rho_{\text{reg}} \leftarrow$ regression criterion (subset);
                until $\rho_{\text{reg}} > \text{thr}_i$;
                $\mathcal{F}_i^{\text{rel}} \leftarrow \mathcal{F}_i^{\text{rel}} \cup$ subset;
            $\text{tr}_{\text{condition}} \leftarrow$ compute transition condition ($\mathcal{F}_i^{\text{rel}}$);
            execute $P_i$ until $\text{tr}_{\text{condition}}$ or timeout;
            correct $\leftarrow$ not timeout;
            $i \leftarrow i + 1$;
        until $i \geq s$ or timeout;
    if not correct then
        $i_{\text{err}} \leftarrow$ get error-causing step;
        increase $\text{thr}_{i_{\text{err}}}$;
        update $\mathcal{F}_{\text{used}}$;
    until correct;
3 Implementation Details

In this section, several steps of Algorithm 1 are explained in detail. First, some general aspects of the implementation are considered and some notation is introduced in section 3.1. In section 3.2, the procedure of reading-in the training data during strategy demonstration is shown. Then, we consider the steps necessary to obtain $F_{\text{rel}}$ in section 3.3. In particular, the concepts of range and regression conditions are defined more formally, and also their computation is described. In section 3.4, we show how the learner reproduces the learned strategy using $F_{\text{rel}}$. Finally, in section 3.5, details about the detection of the error-causing step and the computation of $F_{\text{used}}$ are presented.

3.1 Preliminaries and Notation

3.1.1 Linear and Circular Features

One general aspect not mentioned in the outline section is that a distinction is made between linear features (e.g., distances) and circular features (e.g., angles and orientations), whereby the learner knows whether a future is linear or circular. This distinction is needed, because several quantities, used in this work, have different definitions for linear and circular data. However, on the higher level of the approach, as it was presented in section 2, all features are handled equally, and in following sections the distinction is omitted for sake of readability. Here, the most important definitions are presented.

Linear features are defined on the interval $[0, 1]$, circular features are defined modulo 1 (i.e., for any angle $\alpha$, $\alpha + 1 \equiv \alpha$), so that circular values appear only in $[0, 1)$. It is assumed, that all operations involving circular values, are implicitly computed modulo 1.

First, we consider the distance function $d$, which returns the distance between two values of the same feature. For linear features, $d(x, y) = |x - y|$, so that the range of $d$ is also $[0, 1]$. For circular features, $d(x, y)$ is the shortest path between $x$ and $y$, when moving on a unit circle, which means that the range is $[0, \frac{1}{2}]$. In cases, where distances between linear and circular data need to be compared directly, the circular distance is therefore multiplied by two. Formally, $d$ is defined as:

$$d(x, y) = \begin{cases} 
|x - y| & \text{linear feature} \\
-(|x - y| - 1) & \text{circular feature} \\
|x - y| & \text{if } |x - y| > 0.5 \\
|x - y| & \text{if } |x - y| \leq 0.5
\end{cases}$$

(12a) (12b)
By $d_\phi(t_1, t_2)$, we refer to the distance between the values of feature $\phi$ in $t_1$ and $t_2$, i.e.:

$$d_\phi(t_1, t_2) = d(\phi(t_1), \phi(t_2)).$$

Next, we define the mean and variance for a set $S = \{a_1, ..., a_n\}$ with $a_i \in [0, 1]$, belonging to the same feature. For a linear feature, we take the usual definitions, for a circular feature, definitions from [Zar, 1999] are used. Let

$$X_S = \frac{\sum_{a_i \in S} \sin(2\pi a_i)}{n},$$

$$Y_S = \frac{\sum_{a_i \in S} \cos(2\pi a_i)}{n},$$

$$Z_S = \frac{\text{atan2}(X_S, Y_S)}{2\pi},$$

$$sr = \sqrt{X_S^2 + Y_S^2},$$

then the mean of the set $S$ is

$$\bar{S} = \begin{cases} 
\frac{1}{n} \sum_{a_i \in S} a_i & \text{linear feature} \\
Z_S & \text{circular feature} \\
Z_S + 1 & \begin{cases} Z_S \geq 0 \\
Z_S < 0. \end{cases} \end{cases}$$

The variance of $S$ is defined as

$$\sigma^2_S = \begin{cases} 
\frac{1}{n-1} \sum_{a_i \in S} (a_i - \bar{S})^2 & \text{linear feature} \\
1 - sr & \text{circular feature.} \end{cases}$$

Note that the range of $\sigma^2_S$ lies in $[0, 1]$ for circular and in $[0, \frac{1}{2}]$ for linear features.

3.1.2 Intervals and Ranges

As outlined in section 2, the ranges, in which the sample values are spread for different features, will play an important role in the learning approach. We use two concepts to describe such ranges. The first, for a given set $S = \{a_1, ..., a_n\}$, is the smallest interval $I = [a, b]$, which contains all values in $S$. For linear data, $a$ is just the smallest and $b$ the largest value in $S$; for circular data, $a$ and $b$ are two values in $S$ with the smallest path length from $a$ to $b$ when moving clockwise, such that
all other values in $S$ lie on that path (the length of this path is also the size of the interval). By $I_{\phi}$, we denote such interval for the set $S = \{ \phi(t^i_1), \ldots, \phi(t^i_{R_i}) \}$, i.e., the set of sample values of feature $\phi$ after transition $i$. If the transition number is clear from the context, we just write $I_{\phi}$.

However, in several situations, the above definition is not well suited, since outliers in the sample data would easily bias the size and the center of the interval. In such cases, it is therefore more convenient to define the range in terms of the spread (e.g., standard deviation or variance) of the values in $S$. The idea is to capture the interval $CI$, containing most of the values in $S$, which is close to the notion of a confidence or tolerance interval. Here, however, the interval is derived from the variance of $S$. We expect that all features follow a uniform, a normal, or some other unimodal distribution (regarding a single step), but for computation of $CI$, all values are approximated by a uniform distribution. Note, that this approach is not suited for very small sample sizes and for multimodal distributions, where a different method should be chosen.

For a linear feature, uniformly distributed in the interval $[a, b]$, the population variance is $\frac{1}{12} (b - a)^2$. Therefore, for a sample set $S$ of linear data, the interval $CI$ is

$$CI = \left[ \bar{S} - \frac{1}{2} \sqrt{12 \sigma^2_S}, \bar{S} + \frac{1}{2} \sqrt{12 \sigma^2_S} \right].$$

The variance of the set $S$ of $n + 1$ angles, which are uniformly distributed on the circle segment $[\alpha, \alpha + n\beta]$ (i.e., $S = \{ \alpha + i\beta | i = 0, ..., n \}$), can be computed by

$$\sigma^2_S = 1 - \frac{\sin \left( \frac{(n + 1)\beta}{2} \right)}{(n + 1) \sin \left( \frac{\beta}{2} \right)}.$$

For a given set $S$ of circular data with $n + 1$ elements and the corresponding $\sigma^2_S$, the interval is

$$CI = \left[ \bar{S} - \frac{n\beta}{2}, \bar{S} + \frac{n\beta}{2} \right],$$

where $\beta$ can be calculated numerically using (21).

### 3.2 Data Sequencing

The first step of the learning approach is to sequence the sensor data, recorded from demonstrations, into movement primitive samples $p^r_i \in [0, 1]^2$ and transition condition samples $t^r_i \in [0, 1]^n$, $n = |F|$. For the $r$th demonstration run (whereby $R$ denotes the total number of demonstrations), $p^r_i$ is the $i$th primitive and $t^r_i$ the $i$th transition occurred in this run. The set of all recorded $t^r_i$ in step $i$ throughout all runs is denoted by $T^{empl}_i$. The sequencing can be performed either online or offline, after all demonstrations were recorded. Hereby the value of $m$ is observed, and a strong change is considered as a transition to a different movement primitive. For the $i$th transition in run $r$,
the value of $m$ before the transition is assigned to $p'_r$, and the value of $f$ in the moment of the transition is assigned to $t'_r$. The value of $f$ at the beginning of run $r$ (i.e., the initial condition) is assigned to $t'_0$, its last value in a run is considered as a transition to the goal state and is assigned to $t'_r$, where $s$ is the number of steps in the presented strategy. After $R$ demonstrations, the data is analyzed for runs in which the sequence of the $p'_r$ differs from other runs (e.g., the sequence has a different length or some $p'_r$ represents a different primitive). Such runs are filtered out and are not used for any further computation. The remaining $p'_r$ are averaged for each step $i$ to obtain $P_i$. The sequencing procedure is shown in Algorithm 2.

### Algorithm 2: Sequencing

```plaintext
r ← 0;
s ← 0;
while demonstration do
    r ← r + 1;
i ← 0;
t'_r ← f ;
while run do
    i ← i + 1;
    while $\dot{m} < \epsilon$ and run do
        m_old ← m ;
        p'_r ← m_old ;
        t'_r ← f ;
s ← i;
    R ← r ;
    filter out malicious runs, adjust indices and $R$ ;
for i in 1, ..., s do
    $P_i ← \frac{1}{R} \sum_{r=1}^{R} p'_r$ ;
```

#### 3.3 Feature Selection

##### 3.3.1 Changing Criterion

Before considering how triggering features are selected with the range and regression criteria, we first look at a necessary condition for all such triggering features. The teacher’s purpose of executing a primitive $P_i$ in step $i$ can be thought as changing one or more feature values, until they reach a new state given by $T_i$. So, each triggering feature must exhibit a considerable change between $T_{i-1}$ and $T_i$, and only such features, which we denote by $F_{ch}$, are considered by other criteria. Note that $F_{ch}$ will probably also contain other features, not regarded by the teacher as triggers.

A straightforward method of obtaining $F_{ch}$ is for each feature $\phi$ to observe its value range during
the demonstration of the $i$th primitive execution, and to compute the mean of such ranges over all demonstrated runs. For the $i$th step we call the length of the obtained mean range $\Delta_{\phi,i}$ and $\phi$ is put to $\mathcal{F}_{ch}$ in this step if $\Delta_{\phi,i}$ is greater than some threshold $\epsilon_{ch}$. If $\epsilon_{ch}$ is constant, however, problems may occur when the robot has to operate on different scales throughout the presented task. For instance, in the motivational example, changes of the trigger features in steps 4 and 5 are much subtle than in the previous steps. With $\epsilon_{ch}$ set too high, $\mathcal{F}_{ch}$ could remain empty in steps with only small changes. On the other hand, if it is set too low, $\mathcal{F}_{ch}$ might also contain features with marginal changes not intended by the teacher, resulting from inaccurate execution, noise, or other effects. Therefore an adaptive method is proposed, which is summarized in Algorithm 3.

The idea is to assign an individual scale $scale_{\phi}$ to each feature, which can change between consecutive steps. For the $i$th step, all $\Delta_{\phi,i}$ are computed as before, but then normalized by the corresponding $scale_{\phi}$. A local threshold is created by multiplying $\epsilon_{ch}$ with the highest of those values, and the remaining values are filtered by that threshold. For all $\phi$ in $\mathcal{F}_{ch}$ also the $scale_{\phi}$ are updated, setting them to the (not normalized) $\Delta_{\phi,i}$. This method neglects small changes, as long as features with large changes exist. If not, it will adapt the scales of the features with weaker changes. For the first step the $scale_{\phi}$ can be initialized with $I_{\phi}^0$.

**Algorithm 3**: Changing criterion

```plaintext
in $i$th step do:  $\mathcal{F}_{ch} \leftarrow \emptyset$

for $\phi$ in $\mathcal{F}$ do
    compute $\Delta_{\phi,i}$
    $\epsilon_{local} \leftarrow \epsilon_{ch} \cdot \max_{\phi \in \mathcal{F}} (\Delta_{\phi,i})$

for $\phi$ in $\mathcal{F}$ do
    if $\Delta_{\phi,i} > \epsilon_{local}$ then
        $\mathcal{F}_{ch} \leftarrow \mathcal{F}_{ch} \cup \{\phi\}$
        $scale_{\phi} \leftarrow \Delta_{\phi,i}$
```

3.3.2 Range and Regression Criteria

The range an regression criteria were already informally described in previous sections, here they are considered in more detail. Hereby a link between the two is established, which is then used to define both criteria formally.

First, we recall the range criterion: It describes situations, in which the transition is triggered when some features approach a specific value or value range, which is sufficiently different from the range in the previous step. If for some feature $\phi$ the range before step $i$ (i.e., directly after the transition $i-1$) is $CI_{i-1} = [a_{i-1}, b_{i-1}]$, and $CI_i = [a_i, b_i]$ after the execution of $P_{i+1}$, then there are two major scales on which $CI_{i-1}$ and $CI_i$ can be compared. The first is the size of the two.
If $|CI| < |CI_{i-1}|$, we consider $\phi$ to be likely a trigger, but if, however, $|CI| > |CI_{i-1}|$ holds, it would depend whether $CI_{i-1} \subset CI_i$. If this is the case, then $\phi$ cannot be a trigger, since the triggering condition would already have been fulfilled before the primitive execution (or at least, there must be a second triggering feature). This leads to the second comparison scale, namely the distance between both ranges, which we define here as the mean distance between the bounds of $CI_{i-1}$ and $CI_i$, i.e.,

$$
\delta = \frac{d(a_{i-1}, a_i) + d(b_{i-1}, b_i)}{2}.
$$

The likelihood of $\phi$ being a triggering feature by the means of the range criterion (not in statistical sense) is proportional to

$$
\frac{|CI_{i+1}|}{\delta},
$$

whereby $\phi$ is more likely a trigger if this term approaches zero, and more unlikely the bigger it gets.

Before defining this quantity in a more convenient way, we consider the regression criterion, which applies to situations where the triggering value $v^t$ of $\phi^t$ depends on the values $v_1, ..., v_m$ of $\{\phi_1, ..., \phi_m\} = F_{expl}$. The dependency is given by some function $F$ with

$$
v^t = F(v_1, ..., v_m) + \epsilon,
$$

where $\epsilon$ is a random error with zero mean. Note that $v_1, ..., v_m$ are taken from $T_{i-1}$ to predict $v^t$ in $T_i$, meaning that the triggering value must be predictable before the execution of $P_i$.

After $F$ has been approximated for a given $\phi^t$ and $F_{expl}$, the question is what likelihood of being a trigger should be assigned to that regression. We assume that the true triggering regression $F$ is a better predictor of $\phi^t$ than any other $\tilde{F}$ on some features $\tilde{\phi}^t$ and $\tilde{F}_{expl}$, in the sense that $\epsilon$ has a smaller spread than $\tilde{\epsilon}$. The spread of $\epsilon$ can be seen as the precision with which the teacher uses $F$ to trigger the transition, which indicates how important the corresponding trigger feature is for that transition. So, the spread can not only be used to detect the relevant feature subset for the regression trigger, but also to compare the importance of triggers given by the range and regression criteria. As a measure for the spread of $\epsilon$ we use the expected value of the absolute residuals, which for a set of $k$ samples $v_j, v_j^1, ..., v_j^m$, $j = 1, ..., k$, is approximated by

$$
\sigma_{reg} = \frac{1}{k} \sum_{j=1}^{k} |v_j^t - \hat{v}_j^t|,
$$

where

$$
\hat{v}_j^t = F(v_1^j, ..., v_m^j).
$$

However, if $\sigma_{reg}$ was used as the only reference for comparison of several regressions, it could be corrupted by different scales of the corresponding dependent features $\phi^t$ and $\tilde{\phi}^t$. For example, if
a $\phi^i$ moves only in the same small range in $T_{i-1}$ and $T_i$, it would be in general easier to obtain a small $\sigma_{\text{reg}}$ for $\phi^i$, than for some $\tilde{\phi}^i$ which is distributed over a large range in $T_{i-1}$ and $T_i$. Therefore, we normalize $\sigma_{\text{reg}}$ by the mean change of $\phi^i$ in step $i$, namely by $\Delta_{\phi^i}$, as defined in section 3.3.1. On the one hand, this solves the above problem, since in small ranges also the mean change will be small. On the other hand, one can also interpret the resulting term
\[
\frac{\sigma_{\text{reg}}}{\Delta_{\phi^i}}
\] as the effort the teacher invests to reach a certain value of $\phi^i$ precisely, which is an indicator of $\phi^i$ being a trigger.

Now, we can set this into relation with the range criterion, and derive a measure of likelihood of being a triggering applicable to both criteria. Another way to look at the range criterion is as a regression with no explanatory features, where $v^i$ is fixed and corresponds roughly to the center of $CI_i$. The size of $CI_i$ then depends on the spread of $\epsilon_i$, so that we replace $|CI_i|$ with $\sigma_{\text{rng}}$, a quantity similar to $\sigma_{\text{reg}}$. For a feature $\phi$ and step $i$, we replace $v^i$ by the mean of the sample set $S = \{\phi(t_1^i), ..., \phi(t_R^i)\}$, and use all samples in $S$ to attain
\[
\sigma_{\text{rng}} = \frac{1}{R} \sum_{j=1}^{R} \left| t_j^i - \bar{S} \right|,
\] which is the standard deviation of $S$. Next, we replace the denominator in (24) (a term reflecting the distance between the two ranges) by $\Delta_{\phi^i}$, i.e., the mean distance between each sample value of $\phi$ in $T_{i-1}^{\text{smpl}}$ and the corresponding value in $T_i^{\text{smpl}}$, and obtain
\[
\frac{\sigma_{\text{rng}}}{\Delta_{\phi^i}}.
\] The terms in (28) and (30) are close to zero, if the according feature is likely a trigger, and possibly become very large if not. To make comparison between features more natural, we introduce the quantity $\rho_{\phi^i}$, which we call the precision coefficient, given by
\[
\rho_{\phi^i} = \exp \left( -c \frac{\sigma}{\Delta_{\phi^i}} \right),
\] where $\sigma$ is either $\sigma_{\text{reg}}$ or $\sigma_{\text{rng}}$, depending on which criterion was used, and $c > 0$ is some constant. We omit the sub- and superscript, if they are clear from the context. For likely triggers, $\rho$ approaches 1, and for unlikely 0, whereby extreme values of the argument term are quenched in both directions. Increasing the constant $c$ can be used to increase the strength of suppressing unlikely features.

The computation of $\rho$ for the range criterion is straightforward, as the definition of $\sigma_{\text{rng}}$ can be directly employed. For the regression criterion this is more complicated, and will be addressed
in the next section. Note, however, that the learning approach is not restricted to use the method presented there, but rather any other regression analysis method can be applied, dependent on the expected form of \( F \), or some other aspect of the application area.

### 3.3.3 Computation of the Regression Model

To evaluate a feature \( \phi^t \) and a set of explanatory features \( F_{\text{expl}} \) with the regression criterion, one basically needs to compute \( \sigma_{\text{reg}} \), namely the mean distance between the value of \( \phi^t \), predicted by \( F \) for some values of \( F_{\text{expl}} \), and its true value in the same situation produced by the teacher. Here, a form of \( k \)-Nearest-Neighbor (kNN) regression is used, so \( F \) is by most part already given by the set of samples recorded from demonstration. Thus, the focus in this section lies on the method of constructing the sample subsets, which are used for evaluation.

The main idea behind kNN is that although \( F \) might not be linear, it behaves approximately linear in some small interval (if, of course, \( F \) is continuous in that interval). Therefore, for a query vector \( v_q \) in the domain of \( F \), \( F(v_q) \) can be approximated by a linear combination of \( F(v_1), \ldots, F(v_k) \), i.e.,

\[
F(v_q) \approx \sum_{j=1}^{k} w_j F(v_j),
\]

where \( v_1, \ldots, v_k \) are vectors near to \( v_q \) (in best case, the nearest \( k \) neighbors). To estimate \( \sigma_{\text{reg}} \) in step \( i \), we need to form some set \( S \subseteq \{ t^i_1, \ldots, t^i_R \} \), where we estimate each target value \( v^i_j = \phi(t^i_j) \), \( t^i_j \in S \) (i.e., the value of the triggering feature after the execution of \( P^i \)) by the means of the neighbors of the corresponding \( v^i_r = F_{\text{expl}}(t^i_{r-1}) \) (i.e., by values of the explanatory features before the execution of \( P^i \)). Now, for each explanatory feature \( \phi \) we construct the set \( S_{\text{local}}^\phi \) in the following way. We fix the values of all other explanatory features \( \tilde{\phi} \in F_{\text{expl}} \) to their current value \( \tilde{\phi}(f) \), and select only such \( t^i_{r-1} \), where \( \tilde{\phi}(t^i_{r-1}) \) is close enough to the fixed value:

\[
S_{\text{local}}^\phi = \left\{ t^i_{r-1} \mid \forall \tilde{\phi} \in F_{\text{expl}}, \tilde{\phi} \neq \phi : \text{local}(\tilde{\phi}) \right\},
\]

\[
\text{local}(\tilde{\phi}) = d_{\tilde{\phi}}(t^i_{r-1}, f) < \epsilon_{\text{local}} | T_{i-1} |^\phi
\]

where \( 0 < \epsilon_{\text{local}} \leq 1 \) and \( | T_{i-1} |^\phi \) is the size of the range containing all values of \( \tilde{\phi} \) in \( T_{i-1}^\text{expl} \). Next, the samples in \( S_{\text{local}}^\phi \) are reduced to the explanatory features, and along with the corresponding values of the target feature in \( T_{i-1}^\text{expl} \) sorted according to \( \phi \), such that the sequence \( S_{\text{sorted}}^\phi \) is obtained:

\[
S_{\text{sorted}}^\phi = \left\{ (v_j, v^i_j) \right\} \mid j = 1, \ldots, | S_{\text{local}}^\phi |\]
such that

\[ \exists t_{i-1}^r \in S_{\text{local}}^\phi : \mathbf{v}_j = \mathcal{F}_{\text{expl}}(t_{i-1}^r) \land v^t_j = \phi(t^r_i) \]

and

\[ \forall \mathbf{v}_j, \mathbf{v}_{j+1} \in S_{\text{sorted}}^\phi : \phi(\mathbf{v}_j) \leq \phi(\mathbf{v}_{j+1}). \]

The purpose of \( S_{\text{sorted}}^\phi \) is to minimize the influence of explanatory features other than \( \phi \) on \( \phi^t \), such that the distance between samples in \( S_{\text{sorted}}^\phi \) mainly depends on \( \phi \), and we therefore can assume that for some \( \mathbf{v}_j \) the samples \( \mathbf{v}_{j-1} \) and \( \mathbf{v}_{j+1} \) are reasonably close to it (see also figure 4 for the effect of \( S_{\text{sorted}}^\phi \) applied to the regression in step 1 of the motivational example).

Thus, if \( n_\phi = |S_{\text{sorted}}^\phi| \), the sum of residuals for that \( \phi \in \mathcal{F}_{\text{expl}} \) is

\[ SR_\phi = \sum_{j=2}^{n_\phi-1} |v^t_j - (w_j v^t_{j-1} + (1 - w_j) v^t_{j+1})|, \quad (36) \]

and computation of \( S_{\text{sorted}}^\phi \) and the corresponding \( SR_\phi \) for all \( \phi \in \mathcal{F}_{\text{expl}} \) finally leads to

\[ \sigma_{\text{reg}} = \frac{\sum_{\phi \in \mathcal{F}_{\text{expl}}} SR_\phi}{\sum_{\phi \in \mathcal{F}_{\text{expl}}} (n_\phi - 2)}. \quad (37) \]

Now, what is left to do is to define \( w_j \) in (36), which can be done in several ways. The simplest possibility is to set \( w_j = 0.5 \), which is equivalent to the unweighted \( k\text{NN} \) method, i.e., all neighbors receive the same weight. In the weighted version, the weight of each neighbor is proportional to its distance to the query point, so for the above case of two neighbors this leads to

\[ w_j = \frac{d(\mathbf{v}_{j-1}, \mathbf{v}_j)}{d(\mathbf{v}_{j-1}, \mathbf{v}_{j+1})}, \quad (38) \]

with \( d \) being a distance function, which can be defined, e.g., as the euclidean distance. However, the above approaches do not take into account how strong \( \phi^t \) is actually correlated with each of the explanatory features, i.e., how much change in \( \phi^t \) is caused by change of some \( \phi \in \mathcal{F}_{\text{expl}} \). To do so, one can try to fit the data with a linear model, and then use the coefficients to weight each explanatory feature in distance computation, as it is done in section 3.4.1 when computing the transition value during task reproduction. This method requires, however, more costs for testing one combination of \( \phi^t \) and \( \mathcal{F}_{\text{expl}} \), which can become noticeable if many, or in worst case, all such combinations must be tested. Also, this variation has rather small improvement on just identifying the triggering regression, i.e., on estimating \( \rho_{\phi^t} \).
Figure 4: Selection of samples for estimating $\sigma_{reg}$. The top part shows samples in $T_{smpl}^1$ of the motivational example, illustrated by arrows in the polar coordinates of the box (the outer circle is the unit circle). The position of an arrow is determined by $\gamma_{br}$ and $dist_{br}$ (the explanatory features), its direction by $\alpha_{br}$ (the trigger feature). The dashed (arc) and the dotted (circle segment) areas symbolize $S_{\gamma_{br}}^{sorted}$ and $S_{dist_{br}}^{sorted}$, respectively. The plots in the middle show the mapping of $\alpha_{br}$ against $\gamma_{br}$ (left) and $dist_{br}$ (right), using all samples from above. The bottom plots show the same, but using the samples in $S_{\gamma_{br}}^{sorted}$ on the left and those in $S_{dist_{br}}^{sorted}$ on the right.
### 3.4 Task Reproduction

In this section, we consider how the learner decides when to trigger a transition during task reproduction, after the triggering features are known. The first part focuses on the computation of the triggering value for a regression trigger, whereby a custom \( k \)NN method is used. The second part describes the actual triggering policy, considering all trigger features.

#### 3.4.1 Regression Triggers

Similar to the detection of the possible triggering regression, a form of \( k \)NN is used to actually compute the triggering value during strategy reproduction. However, a different method is applied, since the accuracy of the triggering value is more important during execution, and higher costs are acceptable, because the computation is carried out only for a single regression and one query.

As mentioned in section 3.3.3, the impact of each explanatory feature \( \phi \in F_{\text{expl}} \) on the dependent feature \( \phi_t \) might be important for distance computation, and therefore also for the selection of suitable neighbors. Let \( v_q \) be the query, \( v \) and \( \tilde{v} \) possible neighbors of \( v_q \), \( \phi \) and \( \tilde{\phi} \) two explanatory features, and let \( v \) and \( \tilde{v} \) be equal in all features but in \( \phi \) and \( \tilde{\phi} \). Suppose that \( d_\phi(v_q, v) = 0 \), \( d_{\tilde{\phi}}(v_q, v) = \delta \), \( d_{\phi}(v_q, \tilde{v}) = \delta \) and \( d_{\tilde{\phi}}(v_q, \tilde{v}) = 0 \), so that both samples would have the same distance to the query if the features were unweighted. But if \( \phi \) had a greater influence on \( \phi_t \) than \( \tilde{\phi} \), it would be more appropriate to select \( v \) as neighbor. Therefore, for each explanatory feature \( \phi \) we introduce a weight \( \beta_\phi \), which reflects the impact of that feature on \( \phi_t \). If the regression was linear, then one just could fit the model

\[
\phi_t^{'} = \beta_{\phi} \phi_1 + ... + \beta_{\phi_m} \phi_m,
\]

\( \phi_1, ..., \phi_m \in F_{\text{expl}} \)

and the model to the samples by some standard technique. However, for non-linear cases this might be problematic, e.g., if increasing some \( \phi \) in one half of the samples caused a strong increase in \( \phi^{'} \), but in the other half a strong decrease, this would lead to \( \beta_\phi \approx 0 \), which would be misleading about the true effect of \( \phi \) on \( \phi^{'} \). Therefore, we informally define \( \beta_\phi \) as the mean absolute change in \( \phi^{'} \), caused by \( \phi \) which we estimate with the method below.

In step \( i \), let \( \phi^{'} \) and \( F_{\text{expl}} \) be the features of the triggering regression in that step, and for each \( \phi \in F_{\text{expl}} \) the set \( S_{\phi}^{\text{sorted}} \) be defined as in section 3.3.3. For some \( \phi \in F_{\text{expl}} \), let \( n_\phi = |S_{\phi}^{\text{sorted}}| \), and...
for each $j = 1, ..., n_\phi - 1$ and $\phi^e \in \mathcal{F}_{\text{expl}}$, let

$$
\delta^e_j = d^e(v_j, v_{j+1}),
\delta^t_j = d(t_{j}, t_{j+1}),
(v_j, v_{j+1}) \in S^\text{sorted}_\phi,
$$

i.e., each $\delta^e_j$ is the distance (change) in $\phi^e$ between two neighboring samples in $S^\text{local}_\phi$, and analogously $\delta^t_j$ for $\phi^t$. Now let

$$
\text{err}_j = \left| \delta^t_j - \sum_{\phi^e \in \mathcal{F}_{\text{expl}}} \delta^e_j \beta_{\phi^e} \right|,
$$

and

$$
SE_{\phi} = \sum_{j=1}^{n_\phi-1} \text{err}_j,
$$

where $\text{err}_j$ is the difference between the true change of $\phi^t$ and the change predicted by individual changes of the explanatory variables, for given coefficients $\beta_{\phi^e_1}, ..., \beta_{\phi^e_m}$ (where $m$ is the number of explanatory features). The sum of all $\text{err}_j$ for one $S^\text{sorted}_\phi$ is denoted by $SE_{\phi}$. Now, we want to select such coefficients, which minimize the sum of error terms for all $S^\text{sorted}_\phi$, i.e.,

$$
\beta_{\phi^e_1}, ..., \beta_{\phi^e_m} = \arg \min_{\beta_{\phi^e_1}, ..., \beta_{\phi^e_m}} \left( \sum_{\phi^e \in \mathcal{F}_{\text{expl}}} \sum_{\phi^e \in \mathcal{F}_{\text{expl}}} SE_{\phi} \right),
$$

This problem can be solved efficiently as a linear program, by, e.g., some form of the simplex algorithm.

After estimating the weights for the explanatory features, we can compute the value of $\phi^t$ for a query $v_q$ with kNN. Recall, that thereby we try to approximate that value by $w_1 F(v_1) + ... + w_k F(v_k)$, where $v_1, ..., v_k$ are samples (neighbors) close to $v_q$, and $w_1, ..., w_k \geq 0$ are weights with the total sum of 1. This is justified by the assumption that $F$ behaves almost linear, provided that the neighbors are close enough to the query. So, the weights are chosen in such manner, that the linear combination of the neighbors approximates the query, i.e., $v_q \approx w_1 v_1 + ... + w_k v_k$. For $w_j \geq 0$, the approximation can only be exact, if the query lies within the convex hull of the neighbors. Thus, for $m$ explanatory features, at least $k = m + 1$ neighbors are needed to enclose the query with an $m$-simplex, i.e., for one feature the query must lie on the line between two neighbors, for two features it must be within the triangle spanned by three neighbors, and so forth. So, we want to select $m + 1$ samples as neighbors, which possibly enclose the query point, and are as close to the query as possible. To achieve this, the following heuristic is employed.

First, all samples are sorted according to their distance to the query, using the weighted distance
function:

\[ d_\beta(v_1, v_2) = \sum_{\phi \in F_{\text{expl}}} \beta_\phi d_\phi(v_1, v_2). \]  

(42)

We denote the sorted samples by \( v_1, ..., v_{R\ell} \), with \( v_1 \) being the nearest to the query and \( v_{R\ell} \) the farthest. Next, the set \( N \) of neighbors is constructed iteratively, whereby after the \( l \)th iteration \( N \) has \( l \) elements, and in the first iteration we set \( N = \{ v_1 \} \). Now consider the \( l \)th iteration with \( l > 1 \), where in order to find the next neighbor \( N \) is combined with one of the remaining samples (\( v_j \) with smaller \( j \) are chosen first). Then, the obtained set is examined whether it can improve (by optimizing \( w_1, ..., w_l \)) the approximation of the query point, i.e., if \( \text{dist}_{\text{app}}(N \cup \{ v_j \}) > \text{dist}_{\text{app}}(N) \) with

\[ \text{dist}_{\text{app}}(N) = \min_{w_1, ..., w_l} d_\beta(v_q, v_{\text{app}}), \]  

(43)

\[ v_{\text{app}} = \sum_{i=1}^{l} w_i v_{N_i}, \]  

(44)

\[ l = |N|, \ v_{N_i} \in N. \]

If this is not the case, the sample is removed from \( N \) and the method proceeds with the next one. If an improvement was achieved with all weights being greater zero, the sample is kept in \( N \), and the iteration step is completed. However, if there is an improvement with one or more weights becoming zero, the method tries to find another improving sample with non-zero weights. If this is not possible, the improving sample with smallest distance to the query is added to \( N \). The method is summarized in [algorithm 4](#algorithm4). Note, that for the computation of \( \text{dist}_{\text{app}}(N) \) a linear solver can be used.

After \( N \) is constructed, the weights \( w_1, ..., w_k \) are the same as used to obtain \( \text{dist}_{\text{app}}(N) \), and the triggering value for \( v_q \) is computed using the \( \phi^i \)-values corresponding to the samples in \( N \), namely

\[ v_{q}^i = \sum_{i=1}^{k} w_i v_{N_i}^i, \ v_{N_i}^i \in N. \]  

(45)

### 3.4.2 Triggering Conditions

For step \( i \), after the set of relevant features \( F_i^{rel} \) and the value \( v^i \) for the triggering regression (if one was needed in that step) has been computed, the question arises how to choose the condition for transition to step \( i + 1 \). For features \( \phi \in F_i^{rel} \), selected by the range criterion, from its definition follows that the transition should occur when all \( \phi(f) \) are in the corresponding ranges. In practice, however, this can be problematic. Due to inaccuracies in previous steps, the state of the robot during strategy reproduction might deviate from the state expected from the training samples.
Algorithm 4: Nearest-Neighbor Selection

sort all samples according to distance to \(v_q\);
\[ N \leftarrow \{v_1\}; \]
\[ \text{dist}_{\min} \leftarrow d_{\beta}(v_q, v_1); \]
\[ l \leftarrow 1; \]
\[ \text{for } l < k \text{ do} \]
\[ l \leftarrow l + 1; \]
\[ j \leftarrow 1; \]
\[ \text{found} \leftarrow \text{false}; \]
\[ j_{\min} \leftarrow R; \]
\[ \text{while } j \leq R \text{ and not } \text{found do} \]
\[ \text{if } v_j \not\in N \text{ then} \]
\[ N \leftarrow N \cup \{v_j\}; \]
\[ \text{if } \text{dist}_{\text{app}}(N) < \text{dist}_{\min} \text{ then} \]
\[ \text{if } \forall i = 1, ..., l : w_i > 0 \text{ then} \]
\[ \text{found} \leftarrow \text{true}; \]
\[ \text{dist}_{\min} \leftarrow \text{dist}_{\text{app}}(N); \]
\[ \text{else} \]
\[ \text{if } j < j_{\min} \text{ then} \]
\[ j_{\min} \leftarrow j; \]
\[ N \leftarrow N \setminus \{v_j\}; \]
\[ \text{else} \]
\[ N \leftarrow N \setminus \{v_j\}; \]
\[ j \leftarrow j + 1; \]
\[ \text{if not } \text{found then} \]
\[ N \leftarrow N \cup \{v_{j_{\min}}\}; \]
\[ \text{dist}_{\min} \leftarrow \text{dist}_{\text{app}}(N); \]
This often means that one or more ranges cannot be reached at all by executing \( P_i \), or they are not reached simultaneously.

Nevertheless, the strategy can still be carried on correctly if the missed ranges are rather unimportant, i.e., the \( \rho \)-values of the according features are low. As example, consider again step 2 of the strategy from the motivational section. There, the transition condition is that the robot is close to \( C_1 \), but the polar angle \( \gamma_{br} \) is more important than the polar distance \( \text{dist}_{br} \), since the strategy execution can still succeed if the transition occurs above or below \( C_1 \), but not when it occurs left or right of it.

To tackle this issue, we develop a transition policy which focuses mostly on the triggering feature with the highest \( \rho \)-value, which we call the primary trigger, and integrates the remaining triggering features proportionally to their \( \rho \)-values. First, the set of triggering features \( F_{tr} = F_{rel} \setminus F_{expl} \) is computed (if no regression is employed in that step, \( F_{expl} \) is empty), and the primary trigger

\[
\phi_p = \arg \max_{\phi \in F_{tr}} \rho_{\phi}
\]  

(46)

is chosen. Then, for each \( \phi \in F_{tr} \), we construct a transition interval \( CI_{\phi} \) as shown in section 3.1.2, where we use \( \sigma_{\phi} \) as standard deviation, and \( v^t \) as the interval center for the regression trigger \( \phi_t \), if one exists.

The necessary condition for a transition is that the primary trigger value lies within its transition intervals, i.e.

\[
d_{\phi_p} (f, t_c) < \frac{|CI_{\phi_p}|}{2},
\]  

(47)

where \( t_c \) is a vector which, for each triggering feature, contains the center of the corresponding transition interval. If \( \phi_p \) once enters \( CI_{\phi_p} \), the transition must occur as long as \( \phi_p \) stays there, and thus the last time point for a transition is when \( \phi_p \) leaves \( CI_{\phi_p} \). Before that, the point of transition is determined by \( \text{dist}_{tr} \), which is the cumulative distance of all triggers \( \phi \) to the according \( CI_{\phi} \):

\[
\text{dist}_{tr} = \sum_{\phi \in F_{tr}} \frac{\text{dist}_{tr}^\phi}{\text{scale}_{\phi}} \rho_{\phi},
\]  

(48)

with

\[
\text{dist}_{tr}^\phi = \begin{cases} 
  d_{\phi_p} (f, t_c), & \phi = \phi_p \\
  \left( d_{\phi} (f, t_c) - \frac{|CI_{\phi}|}{2} \right)^+, & \text{else}
\end{cases}
\]

So, \( \text{dist}_{tr} \) is the sum of individual trigger distances, weighted by their \( \rho \)-value. Hereby, the distance is measured against the center of the triggering interval for the primary trigger, and against the border of the interval for all other triggers (which is zero if they are inside the corresponding interval). That means that if all \( \phi \) are in the according \( CI_{\phi} \), \( \text{dist}_{tr} \) will decrease as long as \( \phi_p \)
approaches the center of $CI_{\phi_p}$, and becomes zero when it is reached. Now, the transition is triggered on three occasions: first, $dist_{tr}$ drops beneath some threshold value close to zero; second, $dist_{tr}$ starts to increase instead of decreasing; and finally, $\phi_p$ leaves $CI_{\phi_p}$.

3.5 Main Algorithm

In this section, several details of algorithm 1 are presented, which were not mentioned in section 2.3 or were explained only briefly. Recall that algorithm 1 aims to find a feature subset for a triggering regression in each step, such that features used earlier are employed first, and verifying this selection by executing the learned strategy on a generated problem. Hereby, in each step $i$, feature subsets are constructed according to some order induced by $F_{used}$, and their $\rho$-values are computed as in section 3.3.3, until one $\rho$-value is higher than the current threshold $thr_i$. The threshold is increased, if the strategy execution fails due to this step. Therefore, we first need to identify when the strategy was performed correctly, which is only the case when the goal state was reached. If no transitions were triggered within some time frame, we conclude that an execution error has occurred. So, after an error was detected, the incorrect regression which led to this error could lie in each step performed in this run, and a method is needed to identify this step.

The issues, considered in this section, can be divided into two groups: First, the identification of the step which has caused the error; and second, the generation of feature subsets from $F_{used}$, and issues related to representation and update of $F_{used}$.

3.5.1 Identification of the Error-Causing Step

As mentioned earlier, we expect errors to happen mainly due to wrongly chosen regressions, since triggers based on the range criterion can be identified with rather high certainty. Further, we assume that the correct triggering regression in a given step will have (usually by far) the highest $\rho$-value among all other possible regressions in that step. Thus, the $\rho$-value of $\phi^t$ in the error-causing step $i_{err}$ is expected to be small. On the other hand, for an error to occur all other triggers (and especially the primary trigger) must have sufficiently low $\rho$-values, otherwise they are likely to outweigh the malicious effect of the wrong regression trigger. Hereby, the same argumentation is used which justified the notion of a primary trigger. Therefore one indicator for the error-causing step is that its primary trigger has a small $\rho$-value, compared to other steps.

A second criterion is that the error-causing step $i_{err}$ is more likely to be close to the step, where the error was detected (which we denote by $i_{det}$). Thus, we weight all steps $j \leq i_{det}$, on one hand, with the $\rho$-values of their primary triggers (i.e., $\rho^j_{\phi_p}$), on the other hand, with their distance to $i_{det}$.
More precisely, for each step \( j \) we define the coefficient \( c_{\text{err}}^j \) as

\[
c_{\text{err}}^j = \rho_j^p + \bar{\rho} \frac{i_{\text{det}} - j}{s},
\]

where \( s \) is the total number of steps in the strategy. The error-causing step is then the one with the smallest \( c_{\text{err}}^j \).

After finding \( i_{\text{err}} \), the threshold \( \text{thr}_{i_{\text{err}}} \) should be increased, such that in the next run a regression with a higher \( \rho \)-value can be found. However, an error can also occur due to, e.g., inaccurate computation of the triggering value by kNN (which heavily depends on the number of samples, their distribution, etc.), although \( \phi^j \) and \( F_{\text{expl}} \) were selected correctly. In this case, increasing the threshold would lead to an exhaustive subset search in the next run. To prevent this, we introduce an error counter \( \text{count}_{i_{\text{err}}} \) for each step \( i \), which is decreased in each run for \( i_{\text{err}} \) and increased for all other \( j \leq i_{\text{det}} \). All \( \text{count}_{i_{\text{err}}} \) are initialized with some small number, e.g., 3, and the threshold \( \text{thr}_i \) is only increased if \( \text{count}_{i_{\text{err}}} \) reaches 0, so that a step must be selected as \( i_{\text{err}} \) systematically before the threshold is increased. After increasing \( \text{thr}_i \), the corresponding \( \text{count}_{i_{\text{err}}} \) is again set to some small number.

To prevent search for regression in steps where no such is needed, the procedure of regression detection is only executed for steps, whose error counter has reached 0 at least once. In algorithm 1 this is symbolized by initializing all \( \text{thr}_i \) with \(-1\), which prevents the regression search as long as the corresponding step does not cause errors. Another mechanism, employed to speed up the simulation, is to reuse a feature subset without a search procedure in each run, as long as the corresponding \( \text{count}_{i_{\text{err}}} \) stays above 0. Further, a method was used to boost the threshold increase in situations, where the feature selection heuristic does not fully apply. Thereby the increase is higher, when the difference between the highest and second-highest \( \rho \)-values achieved so far in this step is small. This significantly lowers the number of runs after which the correct feature subset is found in such cases. But it also tends to “overshoot” the corresponding \( \rho \)-value, e.g., by setting the threshold to one, so that the feature selection algorithm is forced to test all possible feature combinations.

### 3.5.2 Used Features and Subset Generation

Initially, \( F_{\text{used}} \) was defined as the set containing the features which were previously used to trigger a transition. Basically, it is the union of all successfully chosen \( F_{i_{\text{rel}}} \) throughout all executed steps \( i \) in all autonomous training runs. However, some features are more important for a transition than others (higher \( \rho \)-value), and some features might serve as trigger in several steps. To incorporate that information, it is more convenient to assign a continuous value to each feature, which repre-
sents the (assumed) likelihood that the particular feature is a trigger in an arbitrary step. Therefore, we introduce the vector \( u \) with the length \( |F| \), where for each \( \phi \in F \) this likelihood is represented by \( \phi(u) \in [0, 1] \). Another way to see \( F_{used} \) is as a fuzzy set, where the degree of membership for each feature is given by \( u \) (we also refer to \( \phi(u) \) as the \( u \)-value of \( \phi \)).

To keep \( u \) updated, an \( s \times n \) matrix \( U \) is used, where \( s \) is the total number of steps and \( n = |F| \). There, the \( i \)th row contains the \( \rho \)-values of all \( \phi \in F \) which were used during the last successful execution of step \( i \) (i.e., when this step was not considered as error-causing in the same run). Hereby, for \( \phi \in F_{rel} \) the \( \rho_{\phi} \) are computed as usual (if a regression was used, the explanatory features obtain the same \( \rho \)-value as the dependent feature), for all other features the \( \rho \)-value is set to 0. Before the first autonomous training run, \( U \) is initialized with \( F_{rel} \) for each step \( i \), computed only by the range criterion. After the \( n_a \)th autonomous training run, \( u \) is updated with the average of the rows of \( U \), by computing

\[
   u_{n_a+1} = \frac{n_c u_{n_a} + u_{mean}}{n_c + 1}
\]

with

\[
   u_{mean} = \frac{1}{s} \sum_{i=1}^{s} (row_i U)^T,
\]

whereby \( u_1 \) (i.e., \( u \) before the first training run) is just set to \( u_{mean} \) as in (53), and \( n_c \in \mathbb{N} \) is a constant which makes sure that recent runs are considered. In runs with \( n_a < n_c \), \( n_c \) is replaced by \( n_a \).

Now, the remaining question is how the order, in which feature subsets are tested with the regression criterion, can be defined in terms of \( u \). One possibility is to assign a weight \( w_{F_{reg}} \) to each such subset \( F_{reg} = \{ \phi^f \} \cup F_{expl} \), where \( w_{F_{reg}} \) is the product of the \( u \)-values of each feature in \( F_{reg} \):

\[
   w_{F_{reg}} = \prod_{\phi \in F_{reg}} \phi(u).
\]

If, before computation, some small value \( \mu \) is assigned to each zero-entry in \( u \), then \( w_{F_{reg}} \) will also reflect the number of features in \( F_{reg} \), with smaller values being associated with more features (note, however, that \( \mu \) must be smaller than the smallest non-zero entry in \( u \)). So, the subsets are ordered according to \( w_{F_{reg}} \), and such subsets with higher value are considered first, i.e., regressions using the more likely features and also using fewer explanatory features.
4 Experiments and Evaluation

In this section, a number of experiments and their results are presented, which are used to show the applicability of the learning approach to manipulation and navigation problems, to evaluate the effectiveness of the main aspects of the method, but also to identify several problematic issues of the approach. In particular, three main questions are examined: First, how well is the learner able to solve the task with the learned strategy? Second, to what degree does this ability depend on the correct feature subset selection for regression, and how reliable is the feature selection method? Finally, how efficient is the proposed heuristic for complexity reduction in feature selection? Also, it is explored how these issues depend on the number of used training samples.

There were two tasks for which the experiments were performed. The first was an extension of the motivational example, the second a navigation task, and all experiments were carried out in a simulated environment. Since basically the same set of tests was done for both tasks, the methodology of testing and result presentation is described in detail only for the first task. For both tasks, the same parameters were used (besides the set of input features), in particular the error counts $count_{err}$ were initialized with 3, and the maximal number of explanatory features in regressions was set to 2.

4.1 Manipulation Task

Set-Up

The experimental set-up was almost the same as described in section 2.1, consisting of a differential-wheeled robot, a target area and a rectangular (object) box, but in addition two other (distractor) boxes of same size as the first were added. For each of the boxes, the same six input features were defined (i.e., 18 features in total), namely the polar angle and distance of the robot in the coordinate system of the box, the orientation of the robot towards the box, the polar coordinates of the box in the coordinate system of the target, and the orientation of the box towards the target (see also figure 1).

Execution

The task was to learn to push the object box into the target, using the strategy shown in section 2.1. For strategy demonstration a computer teacher was employed, i.e., a program which performed the pushing task with the given strategy by controlling the robot. A computer teacher was chosen to speed up the demonstration process, and also to make the experiments more reproducible. No noise was explicitly added to the teacher’s performance, but it was programmed in such manner (each transition was triggered slightly before the time point, specified by the strategy), that the
initial position of the robot had influence on the further strategy execution. This resulted in enough deviation from the “ideal” trajectory to simulate human performance to some degree. In fact, in about 30% of the runs the computer teacher could not complete the task due to too great divergence from the strategy. However, only successful runs were used as training samples.

Before each demonstration run, a new problem instance was generated. Thereby, the initial positions of the boxes and the robot, and also their orientations, were drawn independently from a uniform distribution (with distances being in some fixed range), however, with several restrictions. All boxes and the robot had mutually to keep a minimal distance to each other, and additionally the boxes were placed with some larger minimal distance to the target (precisely, it was 0.3 of the maximal distance). Also, the distractor boxes had not to interfere with the strategy execution (e.g., if a distractor box was in the way of the robot, when it would try to reach the object box), in such cases a new position for the boxes was generated.

All experiments were segmented in so-called training attempts. At the beginning of each attempt, the learner was set back to an initial state, and a set of training samples was chosen which was fixed for the rest of the attempt. Then the learner performed a series of autonomous training runs, in which he tried to reconstruct the strategy from the training samples. For each training run a new problem instance was generated, to which the learner applied the strategy he obtained from the training samples and all previous runs in that attempt. The run was considered as successful, if the learner reached the goal state within some time frame, and as unsuccessful otherwise.

The data, examined in the next section, was obtained in the following manner. The computer teacher generated 20 training sample sets, each consisting of 50 task demonstrations on distinct problem instances. For each such set, the learner performed three training attempts with 100 autonomous runs, whereby a subset of 15 samples was used in the first attempt, 30 samples in the second and 50 in the third. So, in total, 60 training attempts were executed with 100 autonomous runs each. Also, another 10 sample sets with 50 samples were created and used for training attempts, in which the regression parameters were chosen manually (see next section). As before, three attempts with different subsets were carried out for each sample set.

In order to compare the learner’s performance on computer-generated and manually obtained training data, two sample sets with 50 training samples were recorded from the demonstration of a human teacher. Hereby, the robot was controlled by four key commands, which corresponded to moving forward, backward, turning left and right. Since the strategy at hand requires high precision and timing in several steps (especially when turning the box), it is rather natural for a human teacher not to execute a strategy step in one go, but by pushing the same key several times in order to fine-tune the robot’s position. However, the learning algorithm would assume a transition if it detected a pause in the recorded data. To prevent this, the simulation was paused whenever no control key was pressed, so that data was only recorded when the teacher actually moved the robot. For both sample sets, 10 training attempts with 15, 30 and 50 samples were
performed, so that again there were 60 attempts in total.

Results

Performance First, we consider the overall performance of the learner on solving the task in the above experiments with computer-generated data. In the top three panels of figure 5, the (goal) success rate of the learner in 20 training attempts (separated by the size of the used sample subset) is plotted against the number of autonomous runs $r$, performed after the beginning of an attempt. For each $r = 1, ..., 100$, the corresponding plot point indicates in how many attempts the $r$th training run was successful, e.g., a success rate of 0.8 for $r = 55$ means that in 16 of the 20 attempts the 55th run was successful. The left, middle and right panels present the data for the attempts which used 15, 30 and 50 training samples, respectively. For all sample subset sizes, the development of the success rate through an attempt is the same: About 20 first runs were by most part unsuccessful, thereafter the success rate starts to grow rapidly until it reaches a plateau, around which it fluctuates for the remainder of the attempt. For 15 samples, it has an approximate value of 0.5, for 30 and 50 samples it is 0.7.

This behavior is explained by the fact that the learner first needs to find the “correct” feature subsets for regression at the beginning of each attempt, which we will discuss later. For now, consider the top part of figure 6, where the success rates for attempts are shown, in which all regression-relevant parameters were fixed, and the “correct” feature subsets were set by hand. It can be seen that the mean success rate increases between attempts with 15 and 30 samples, and then increases only marginally for attempts using 50 samples. So we can say that after the learner finds the proper regression feature subsets, its average performance only depends on the number of training samples.

The reason for this is that the quality of the $k$NN-method used here to compute the triggering values (and of $k$NN-methods in general) strongly relies on the number of underlying training samples. Since the robot does not correct its course during the execution of a moving primitive, it is crucial that the triggering values for a transition are computed as precise as possible. Small inaccuracies in the triggering value might therefore result in large deviations from the strategy, if the following primitive is executed long enough, e.g., in situations where a turning primitive is followed by a move-forward-primitive. And because the problem instances for training and presentation are generated on a random basis, even for larger sample sets it is possible that the robot hits an area, only sparsely covered by training samples, which probably leads to imprecision. That explains why no 100% goal success rate was reached even with correct features.

Finally, we consider how the performance of the learner differs on data recorded from demonstration of a computer and a human teacher. Therefore, we only compare the corresponding goal success rates, which for the human-generated data are shown in the bottom panels of figure 6. For each of the three sample set sizes, the course of the rate throughout an attempt is similar to the
Figure 5: Experimental results for the manipulation task. At the top, the rate of a successful task execution is shown, dependent on the number of runs already performed. Additionally the plots are separated by the number of training samples, which were used in those attempts, namely 15, 30 and 50. Similarly, the bottom plots present the rate of correct regression feature selection for the first (blue) and the seventh (purple) steps. The data was gathered from the same experiments as in the plots at the top.
Figure 6: The top panels show the goal success rate with fixed regression features, which were set by hand before the training. The bottom plots present the goal success rate in attempts with human-generated training data.
computer-generated data in figure 5, but in general with a slightly higher success rate. The reason for this is that in this particular set-up, the computer teacher was not programmed to perform the strategy as exact as possible but to allow small inaccuracies, whereas the human teacher tried to solve the task as exact as possible.

Feature selection

Now, we examine the selection of triggering features (and subset selection for regression in particular) and how it influenced the goal success rate. Since demonstration runs \(r\), in which the resulting primitive sequence \(p'_r\) varied greatly from other demonstrations, were filtered out in the preprocessing, in each training attempt the learned primitive sequence was basically identical to the “ideal” movement sequence (see figure 2) used by the teacher. Therefore the success rate depended (besides execution inaccuracies) only upon the correct selection of the triggering features.

First we consider features determined by the range criterion, which served as triggers in all steps except 1 and 7. Recall that the range criterion bases only on the spread of the feature values in the training samples (which do not change during autonomous runs), so that all range-based triggers were the same for all autonomous runs in one training attempt. Thus, it is sufficient to consider only one run in each attempt to examine those triggers (of course, only if all steps were performed in this run). However, in order to see the final strategy obtained by the learner in each attempt, we look at runs where also the regression triggers in steps 1 and 7 were chosen correctly. For all of the 60 training attempts, one such run was selected, and the \(\rho\)-values in each of the nine steps were gathered. The mean and spread of that data is shown in figure 7, whereby only features related to the object box and one of the distractor boxes are presented (the values for the second distractor were nearly the same). As described in section 3.4.2, the feature with the highest \(\rho\)-value was taken as the primary trigger in the corresponding step. In all considered runs, in steps 1-5 those were accordingly \(\alpha_{br}, \gamma_{br}, \alpha_{br}, dist_{br}\) and \(\alpha_{br}\). In step 6, in 50 runs \(\alpha_{br}\) was the primary trigger and in 10 runs \(\gamma_{br}\), whereby both can be seen as proper there. In the remaining three steps, the triggers were \(\gamma_{tb}, \alpha_{br}\) and \(dist_{tb}\), where in two runs \(\alpha_{br}\) was falsely taken as primary trigger in step 7. Nevertheless, the feature selection method with the range criterion demonstrated very good results (the above choice of primary triggers reflects the strategy as described in figure 3) with almost no variation. So, the success of strategy reproduction depended mainly on the right choice of regression features in steps 1 and 7.

In step 1, the triggering feature was \(\alpha_{br}\) (the orientation of the robot regarding the box), and the explanatory features were \(\gamma_{br}\) and \(dist_{br}\) (the robot’s position in the local coordinates of the box). Since the individual influence of each of the explanatory features on \(\alpha_{br}\) varied with the particular position of the robot (e.g., when the robot was far from the box, \(\alpha_{br}\) was mainly determined by \(\gamma_{br}\)), we consider the subsets \(\{\alpha_{br}, \gamma_{br}, dist_{br}\}\), \(\{\alpha_{br}, \gamma_{br}\}\) and \(\{\alpha_{br}, dist_{br}\}\) as correct in step 1. In step 7, the trigger was \(\alpha_{tb}\) (the orientation of the box with respect to the target), which was determined by \(\gamma_{tb}\) the polar angle of the box in the target’s coordinates. Thus, in step 7 we call only the subset \(\{\alpha_{tb}, \gamma_{tb}\}\) correct.
Figure 7: Mean $\rho$-values in each step of the learned manipulation strategy, obtained from 60 training attempts. For each step and for each feature the $\rho$-values are represented by a boxplot, where the black dot is the mean and the blue box is enclosing the second and third quartiles of the data (due to small variance, in most cases it is just a line). The blue dots are representing outliers. The data was taken from runs, in which the regression triggers were selected correctly. Only $\rho$-values for features concerning the object box (the left six) and one distractor box (the right six) are shown.
The three bottom panels in figure 5 show the (selection) success rate of choosing the correct feature subsets in steps 1 and 7. Besides objecting the proper feature selection (and not reaching the goal) as success in one run, this rate was computed identically to the goal success rate in the top panels (if in some run the execution was aborted before step 7, the same feature subset was taken for the statistics as in the previous run). For step 1, we see that for all three sample set sizes the success rate increases steeply in the first few runs of an attempt, until it reaches a value of almost 1. For step 7, the rate of success stays at 0 for ca. the first 20 runs, and then grows until reaching a stable level. However, this level is close to 1 only for attempts using 30 and 50 training samples, for 15 samples it is around 0.75. This is a result of small (and statistically less significant) sample sets used for regression analysis, where odds rise for wrong feature subsets to achieve better results than the correct feature subset. We notice that the growth onset and stabilization of the success rate in step 7 coincide with the growth onset and stabilization of the goal success rate. This demonstrates that the goal can be reached on high rate only when the correct regression features are selected (it can still be reached by chance at a low rate, though).

**Complexity**

Next we look at the effectiveness of the heuristic for complexity reduction of feature selection. In step 1, all relevant features $F_{reg}$ were already used as triggers in other steps (see also figure 7), so that their values in $u$ and the resulting weight $w_{F_{reg}}$ (as defined in [54]) were higher than of all other features. Therefore already the first feature subset, considered by the algorithm, was the correct one, what explains the high selection success directly after the beginning of an training attempt. By contrast, the correct features in step 7 had both an almost zero value in $u$, so that this feature subset was examined relatively late by the algorithm, after several other subsets were dismissed due to constant execution failures.

Now we consider the number of tested feature combinations in both steps and how this number scaled with the amount of input features. Hereby we are interested only in runs, in which the proper feature subset was found for the first time in a training attempt, since after that the same subset was usually used until the end of the attempt without further testing. As previously mentioned, in step 1 the first tested combination was correct in all considered runs. In step 7, on average 1200 combinations had to be tested in one run, until the correct one was found (with almost no differences among the underlying sample set sizes).

To see how those values depend on the number of input features, the number of distractor boxes was gradually increased to six, and 10 training attempts with 30 samples were performed in each case. For each box again six features were used, so that together with the object box up to 42 features were involved. The quantity of tested feature subsets in the run, in which the correct subset was first detected, is shown in figure 8 for each attempt from the above experiment. We see that in step 1, this number did not change and stayed at almost 0. In step 7, however, a strong increase can be observed, whereby the values seem to follow two “branches” with different progressions. The lower branch results from cases where the threshold stayed below the $\rho$-value of the correct subset, as intended. Since that subset had two features in step 7, from the considerations in section 2.3...
follows that the progression has to be quadratic, and this was also confirmed by the data. The upper branch represents situations in which the threshold overshot the maximal $\rho$-value, so that no subset (not even the correct one) could reach it. This forced the algorithm to test all feature subsets with at most three features, which led to a cubic progression in the number of features. The threshold overshooting is a result of the used threshold boosting method, and is examined in more detail in the discussion section.

During the above experiment, also the CPU-time needed for feature subset selection was measured. The results show that the average time to test one feature subset did not depend on the total number of features, so that the total needed time was proportional to the number of tested feature combinations. On a quadcore CPU with 3.3GHz, for 42 features it took up to 50 seconds to test ca. 20,000 combinations. Notice, however, that such delays only occurred at the beginning of step 7, and only before the proper feature subset was found. Besides that, the strategy execution did not require heavy computation, so that the simulation speed could be increased at over 50× of real time.
4.2 Navigation Task

Set-Up

For the second experiment, an indoor navigation task was chosen. The environment consisted of several rectangular rooms, placed in a row and connected by a straight corridor (see also figure 9), whereby all rooms had the same depth, but their width varied. Each room had only one door to the corridor, which had the same fixed size but possibly a different location in distinct rooms. The task of the robot was to move from a start room into some different goal room.

The following set of eight features was used for this task. First, the learner received the local cartesian coordinates of the robot, regarding the room it was currently located in. The values of $x$ and $y$ were normalized by the width and the depth of the room, accordingly, so that their values reached from 0 to 1, with $x = 1$ corresponding to the right-most and $y = 1$ to the top-most position. Note that unlike in the manipulation task example, where each box received an own set of features, here only one $x$ and one $y$ feature existed for all rooms, whereby they represented the coordinate system of the current room of the robot. When the robot was at the corridor, $x$ and $y$ were both set to 1.

Next, three features were defined to handle doors, namely the polar coordinates $\gamma_{dr}$ and $dist_{dr}$ of the robot considering the center of the door as basis, and the angle between the orientations of the robot and the door $\alpha_{dr}$ (similar to $\alpha_{br}$ in the manipulation task). The distance, at which $dist_{dr}$ reached its maximal value of 1, was a multiple of the door width (see figure 9, where the range of $dist_{dr}$ is represented by a gray-shaded area). Equivalently to $x$ and $y$, the features $\gamma_{dr}$, $dist_{dr}$ and $\alpha_{dr}$ were assigned to the door of the room, which was closest to the robot (also when the robot was at the corridor).

Finally, a marker point was used, which was placed in front of the goal room. The marker was associated with the features $\gamma_{mr}$, $dist_{mr}$ and $\alpha_{mr}$, defined similarly to the door features, except for $dist_{mr}$, which was a categorical feature. It approached 0 when the distance between the robot and the marker was smaller than half the width of the goal room (i.e., when the robot was in front of or inside the goal room, see the gray area around the marker in figure 9), and 1 otherwise. Notice that the orientation vectors of the door and the marker, used for $\alpha_{dr}$ and $\alpha_{mr}$, were the same, so that $\alpha_{dr}$ and $\alpha_{mr}$ had always the same value.

Execution

The solution strategy for the task was the following: Turn towards a point in front of the door and go to that point. Then, turn towards the door and move through it, until the middle of the corridor is reached (i.e., the robot is at the same height as the marker). Turn in the direction of the goal
Figure 9: Set-up of the navigation task. The figure shows an exemplary problem instance with three rooms, with the leftmost being the start and the rightmost the goal room. The point “door sensor” marks the basis for $\gamma_{dr}$ and $dist_{dr}$, whereby the gray area around it symbolizes the value of $dist_{dr}$ (the darker, the closer to 0). Equivalently, “marker” shows the reference point for $\gamma_{mr}$ and $dist_{mr}$, where $dist_{mr}$ is 0 in the gray area and 1 anywhere else. The coordinate axes in the left room represent the features $x$ and $y$, and the dashed arrows the solution trajectory for this problem instance. In steps 1, 3, 5 and 7 the robot turns, and in steps 2, 4, 6 and 8 it moves forward.
room and move forward, until approaching its door, turn again and enter the room. The strategy trajectory is also shown in [figure 9].

Before each demonstration by the teacher or autonomous run of the learner, a new problem instance was generated. Hereby, three rooms were created, with the width for each of them randomly drawn between some fixed minimal and maximal values. The door locations for each room were also chosen randomly, although the door width, as well as the room and floor depth were always the same for all rooms and problem instances. Then, a start room was randomly selected, where the robot was again randomly positioned. At last, a goal room was chosen from the remaining two, and the marker was placed accordingly. All random values and parameters were drawn from a uniform distribution. Note that in this set-up the number of rooms does not matter, since the features are assigned dynamically to the nearest room, and a larger number of rooms would only increase the distance the robot has to cover in each run. However, three is the minimal number of rooms, for which the learner actually needs to make a choice in which direction to turn at the corridor.

As for the manipulation task, 20 training sample sets with 50 samples were created, and for each of them a training attempt with 15, 30 and 50 samples was performed. Also, for another 20 sets of 50 training samples the same was done with manually chosen regression features. However, this time no training data from a human teacher was used.

**Results**

**Performance**  Again, we first look at the goal success rate of the learner, presented in the top panels of [figure 10]. It exhibits a similar behavior as in the manipulation task, with a sharp increase at the beginning and the following hovering around a stable level, which is circa 0.6, 0.7 and 0.75 for attempts using 15, 30 and 50 samples, respectively. However, the increase sets on much earlier compared to the manipulation task, and the stable levels are higher. Consider now the success rates of the attempts with hand-picked regression features, which are shown in the bottom panels of [figure 10]. There, the mean success rate lies around 0.7 for 15, 0.75 for 30, and 0.8 for 50 samples, so that it is only slightly higher than in the case when the features are actually learned.

**Feature selection**  The better performance in the navigation task compared to the manipulation task results, on one hand, from the smaller set of input features and from the fact that it is generally less prone to execution inaccuracies, whereas reaching and turning the object in the manipulation task required high precision. On the other hand, it follows from a different regression type, used in the navigation task. There were two steps, in which regression analysis was applied. In step 1, the robot had to turn towards a point in front of the door, which is similar to step 1 in the pushing task. Here, however, the correct triggering feature could have been either $\alpha_{dr}$ or $\alpha_{mr}$, with the set of explanatory features being $\{\gamma_{dr}\}$, $\{\gamma_{dr}, dist_{dr}\}$ or $\{\gamma_{tb}, y\}$. Due to high $\rho$-values of the two possible dependent features and of $\gamma_{dr}$, the feature selection heuristic fully applied here. This can be seen
in the middle panels of figure 10, where the selection success rate for step 1 approaches a value close to 1 at the beginning of each training attempt.

In the fifth step, the robot had to turn in the corridor in the direction of the marker, so that again $\alpha_{dr}$ and $\alpha_{mr}$ both were proper triggers, but only $\gamma_{mr}$ (the polar angle of the robot regarding the marker) was the correct predictor. Since in this situation the learner had only the choice to go left or right along the corridor, the regression can be seen as a binary classification problem. This had a major influence on the learning behavior. Consider the selection success rate for step 5 in figure 10, which starts to grow after circa the first 15 runs, and asymptotically reaches a stable value. This value, anyhow, is considerably lower than for step 1, namely 0.65, 0.6 and 0.7 for attempts with accordingly 15, 30 and 50 training samples. Partially, this is explained by the low value of $\gamma_{mr}$ in $u$, which means that other features were tested as predictors before $\gamma_{mr}$. But for instance in step 7 of the manipulation task, the dependent and explanatory features had low $u$-values as well, and still a higher selection success rate was reached.

Recall that in order to test feature subsets with smaller weights $w_{F_{reg}}$ in step $i$, the corresponding threshold $thi$, has to be increased sufficiently, which happens after the error counter $count_i^{err}$ reaches zero. Here, however, each of the correct dependent features, together with an arbitrary set of explanatory features, will lead to moving left or right down the corridor. The choice between left and right is random, unless the set of explanatory features contains $\gamma_{mr}$, so that even with wrong predictors the learner has a chance of 50% to take the correct direction and to reach the goal state. So, in approximately half of the runs $count_i^{err}$ would be incremented and in the other half decremented, which is basically an one-dimensional random walk. In this case the probability that $count_i^{err}$ reaches zero and a wrong feature subset is discarded approaches one, but only with an increasing number of runs, and this is reflected by the slow increase of the selection success rate. On the other hand, the learner was able to reach the goal in about half of the runs with wrong features, which explains the high goal success rates.

**Complexity**

In the end, we consider the number of tested subsets in the first run (within one attempt), in which a correct feature subset was selected. For the first step, this number lied between 1 and 5 (with a mean of 1.8) for attempts using 15 training samples; for attempts with 30 samples it was again in the range $[1,5]$ with 1.4 as mean; and for 50 samples, it lied in $[1,3]$ with the mean of 1.2. In step 5, those numbers were higher due to the low $u$-value of $\gamma_{mr}$. The ranges and the corresponding mean values were $[10, 19]$ and 14.9, $[10, 56]$ and 12.2, and $[10, 19]$ and 15.2, for attempts using 15, 30 and 50 training samples, respectively. There were only two potential triggers in this step, and since the maximal number of explanatory features was set to 2, 56 feature combinations were possible. Those were all tested only in a single run, where the corresponding threshold overshot the maximal $\rho$-value. Although the heuristic was not as effective in step 5 as in step 1, it nevertheless achieved a reduction of computational costs.
Figure 10: Experimental results for the navigation task. The top panels show the rate of a successful task execution, separated by the number of training samples. The plots in the middle present the selection success rate of the proper regression feature subsets for the first (blue) and the fifth (purple) steps. The bottom panels present the goal success rate for training attempts, in which in steps 1 and 5 the correct regression features were set manually.
5 Discussion

In this section, the main results of the thesis are discussed, several problems of the learning approach with possible improvements are considered, and also some directions for future work are presented.

The goals of this work were, on one hand, to develop an LfD-method which could be applied to sequential navigation and manipulation tasks for a mobile robot. On the other hand, the chosen method design required a reduction of the feature input space to features relevant for a particular task step, in other words feature subset selection was needed. For this purpose, a method basing on two different mechanisms was developed, namely on the range of a feature value in distinct demonstrations, and on regression analysis. In order to reduce the computational complexity of the regression analysis, a heuristic was used. It proposes to examine such feature subsets first, which were considered as relevant in other steps of the demonstrated task due to their value range. The integration of this heuristic had a major influence on the shape of the learning approach, and can be considered as its main characteristic.

The experimental results in the previous section show that the proposed method is able to learn smaller navigation and manipulation tasks from demonstration. Also, the reduction of complexity for feature subset selection with the discussed heuristic can work in principle. But the experiments also show that, like all heuristics, it cannot be assumed to work in every situation. Nevertheless, the complexity reduction approach might be more useful in combination with other methods, and especially when exploiting more domain knowledge, or knowledge about connections between the input features. For instance, consider again step 7 from the manipulation task, where the heuristic did not apply: If the learner had had the information, that all trigger features in other steps are related to the object box, it would have been obvious to test $\alpha_{tb}$ and $\gamma_{tb}$, before beginning to test features related to the two distractor boxes. So, one direction of future work might be to apply the heuristic to an object-oriented framework, where features can be grouped by objects which they describe.

One conceptual flaw of the method, at least in the presented form, is that its ability to filter out unrelated features by the range criterion bases on the assumption that those features have higher variance (throughout different training runs) than related features. Although this is not uncommon in practice, other situations are likely as well. Consider the following example: The robot has to operate in a room with several objects inside, whose positions are fixed, e.g., desks, cupboards, etc. Assume that the input features are similar as in the manipulation example, i.e., for each object, a polar coordinate system exists, and assume also that global coordinates for the room exist as well (as in the navigation task). If in some step the robot needed to move to a fixed point, say, the center of the room, the corresponding transition conditions could be described in terms of the global coordinates, but equally eligible, also by the coordinate system of any object in the room.
This would be less problematic for the range criterion, since all redundant features could just be taken as triggers, but it could undermine the principle of the heuristic for the regression criterion, because $F_{used}$ might become very large. Therefore a method is needed to detect such redundancies, possibly by using some standard techniques.

Another weak point of the method is the detection of the error-causing step in an unsuccessful run, which mainly bases upon the fact that the $\rho$-value of the primary trigger in such a step is smaller than the corresponding $\rho$-values in the steps with correctly selected features. Although this approach has worked for the presented examples, it is again rather a heuristic than a rule, and might be unreliable in the general case. Related to this is also the problem of choosing the right increase rate for the threshold $thr_i$ in an error-causing step. As could be seen in the experiments for the manipulation task, it can happen that the threshold is set higher than the maximal $\rho$-value, which could be reached by any subset in that step. This leads to the worst case (from the computational view point), where the algorithm tests all feasible feature subsets and chooses the one with the highest $\rho$-value. This issue could be solved by increasing the threshold only marginally over the $\rho$-value of the last subset used in this step. However, this results in very slow threshold growth in situations as in step 7 of the manipulation task, so that the correct feature subset is found only after a very high number of runs (when tested with the manipulation task, the right subset in step 7 was often not found after 100 runs). Therefore, a more sophisticated method is needed, which sufficiently boosts the threshold increase without overshooting the maximal $\rho$-value.

Further, there are several other flaws, which, however, concern only implementational aspects of the method and could be fixed more easily. For one, the procedure of testing the goodness of fit of a regression (i.e., a feature subset), proposed in section 3.3.3, can be improved in several ways. For each feature subset, it constructs training sample subsets which are local to the current position of the robot. This is computationally expensive, and could be enhanced by preprocessing of the samples and the use of more sophisticated data structures, e.g., $k$-d trees. For another, the $k$NN method used in task reproduction (see section 3.4.1) could also be replaced by a more efficient and effective approach.

Besides that, there are several other possibilities to extend and improve the learning approach in future. Most important, the support for branches, alternatives and repetitions in the demonstrated task could be implemented, i.e., the task would be represented by a general graph instead of a sequence. The approach could be applied to such tasks without much change, provided that the graph structure is known, and that the data from each demonstration run could be mapped to a path through the graph. This, however, is a non-trivial task, and probably could be addressed by methods from the theory of Hidden Markov Models (HMMs), see e.g., [Niekum et al., 2012] or [Yang et al., 1994].

Moreover, the method may be extended by implementing hierarchical learning. This means
that simpler tasks and behavior, learned in the past, could be reused as a part of more complex tasks, as in, e.g., [Saunders et al., 2006]. This might also be combined with a mechanism for generalization, since at the moment each learned task is bound to the features which were used during demonstration. For instance, imagine that there are two types of objects described by similar features, which, however, are regarded as distinct by the robot. If the task was, e.g., to avoid such objects, but the demonstration was performed only on one object type, it would be useful to recognize this behavior when it is demonstrated for the second type of objects. In such case, avoidance would not be relearned, but the known strategy for the first type would be used instead.

In conclusion, the presented LfD-method can be applied to learn some less complex tasks, but its main purpose is to demonstrate the feature selection approach and the complexity reduction heuristic in particular. However, the approach and the heuristic might be of greater practical relevance after some of the above extensions are implemented.
References


