# Robot Motion and Task Learning with Error Recovery

by

Guoting Chang

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2013

© Guoting Chang 2013

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

## Abstract

The ability to learn is essential for robots to function and perform services within a dynamic human environment. Robot programming by demonstration facilitates learning through a human teacher without the need to develop new code for each task that the robot performs. In order for learning to be generalizable, the robot needs to be able to grasp the underlying structure of the task being learned. This requires appropriate knowledge abstraction and representation. The goal of this thesis is to develop a learning by imitation system that abstracts knowledge of human demonstrations of a task and represents the abstracted knowledge in a hierarchical framework. The learning by imitation system is capable of performing both action and object recognition based on video stream data at the lower level of the hierarchy, while the sequence of actions and object states observed is reconstructed at the higher level of the hierarchy in order to form a coherent representation of the task. Furthermore, error recovery capabilities are included in the learning by imitation system to improve robustness to unexpected situations during task execution.

The first part of the thesis focuses on motion learning to allow the robot to both recognize the actions for task representation at the higher level of the hierarchy and to perform the actions to imitate the task. In order to efficiently learn actions, the actions are segmented into meaningful atomic units called motion primitives. These motion primitives are then modeled using dynamic movement primitives (DMPs), a dynamical system model that can robustly generate motion trajectories to arbitrary goal positions while maintaining the overall shape of the demonstrated motion trajectory. The DMPs also contain weight parameters that are reflective of the shape of the motion trajectory. These weight parameters are clustered using affinity propagation (AP), an efficient exemplar clustering algorithm, in order to determine groups of similar motion primitives and thus, performing motion recognition. The approach of DMPs combined with APs was experimentally verified on two separate motion data sets for its ability to recognize and generate motion primitives.

The second part of the thesis outlines how the task representation is created and used for imitating observed tasks. This includes object and object state recognition using simple computer vision techniques as well as the automatic construction of a Petri net (PN) model to describe an observed task. Tasks are composed of a sequence of actions that have specific pre-conditions, i.e. object states required before the action can be performed, and post-conditions, i.e. object states that result from the action. The PNs inherently encode pre-conditions and post-conditions of a particular event, i.e. action, and can model tasks as a coherent sequence of actions and object states. In addition, PNs are very flexible

in modeling a variety of tasks including tasks that involve both sequential and parallel components. The automatic PN creation process has been tested on both a sequential two block stacking task and a three block stacking task involving both sequential and parallel components. The PN provides a meaningful representation of the observed tasks that can be used by a robot to imitate the tasks.

Lastly, error recovery capabilities are added to the learning by imitation system in order to allow the robot to readjust the sequence of actions needed during task execution. The error recovery component is able to deal with two types of errors: unexpected, but known situations and unexpected, unknown situations. In the case of unexpected, but known situations, the learning system is able to search through the PN to identify the known situation and the actions needed to complete the task. This ability is useful not only for error recovery from known situations, but also for human robot collaboration, where the human unexpectedly helps to complete part of the task. In the case of situations that are both unexpected and unknown, the robot will prompt the human demonstrator to teach how to recover from the error to a known state. By observing the error recovery procedure and automatically extending the PN with the error recovery information, the situation encountered becomes part of the known situations and the robot is able to autonomously recover from the error in the future. This error recovery approach was tested successfully on errors encountered during the three block stacking task.

**Acknowledgements**

I would like to thank my supervisor Professor Dana Kulić for her invaluable guidance and support throughout this research work. I am very thankful for her patience, advice and expertise, as well as for the countless paper edits and informative discussions during meetings.

I am also very grateful to Professor David Wang for the undergraduate research assistantship opportunity that eventually led to my Master's placement.

I would like to thank the Professors of the courses that I have taken for sharing their knowledge and for providing input to course projects related to my research work. In particular, Professor Ali Ghodsi who suggested the use of the affinity propagation (AP) algorithm that is now an integral part of this thesis.

Thank you also to everyone who has taken the time to review this work, including the readers of this thesis - Professor Fakhreddine Karray and Professor Jonathan Kofman - who have provided valuable feedback.

I would like to thank everyone in our lab group for sharing their insights during the many lab meeting discussions and for being all-round great people who have made this experience that much more enjoyable.

I would like to thank the Natural Science and Engineering Research Council of Canada, the Ontario Graduate Scholarship Program and the University of Waterloo for providing funding without which this research work would not have been possible.

Lastly, I would like to thank my parents for their unyielding support and love.

## Dedication

To my parents.

# Table of Contents

**List of Tables** **xi**

**List of Figures** **xii**

**List of Abbreviations** **xv**

**1 Introduction** **1**

1.1 Contributions . . . . . 3

1.2 Outline . . . . . 4

**2 Related Work** **6**

2.1 Motion Learning . . . . . 8

2.1.1 Motion Modeling . . . . . 9

2.1.2 Motion Recognition . . . . . 10

2.2 Task Learning . . . . . 11

2.2.1 Action Based . . . . . 11

2.2.2 Object Based . . . . . 12

2.2.3 Object and Action Based . . . . . 13

2.3 Error Recovery . . . . . 13

2.3.1 Plan Based . . . . . 14

2.3.2 Graph Based . . . . . 14

2.3.3 Social Learning Based . . . . . 15

2.4 Summary . . . . . 16

**3 Background** **17**
3.1 Dynamic Movement Primitives (DMPs) . . . 17
3.2 Affinity Propagation (AP) . . . 20
3.3 Self Organizing Maps (SOM) . . . 21
3.4 Petri Nets (PN) . . . 23
3.4.1 Petri Net Structures . . . 24
3.4.2 Petri Net Graphs . . . 26
3.4.3 Reachability . . . 27
3.5 Summary . . . 28

**4 Motion Learning** **29**
4.1 Motion Segmentation . . . 30
4.2 Motion Modeling . . . 31
4.3 Motion Recognition . . . 32
4.4 Experiments . . . 33
4.4.1 Motion Model Implementation . . . 34
4.4.2 Small Motion Data Set . . . 36
4.4.3 Large Motion Data Set . . . 41
4.4.4 Discussion . . . 49
4.5 Summary . . . 51

**5 Task Learning** **52**
5.1 Object and Object State Recognition . . . 53
5.2 Task Modeling . . . 55
5.2.1 Petri Net Training . . . 55
5.2.2 Petri Net Generation . . . 56
5.3 Experiments . . . 57
5.3.1 2 Block Task . . . 58
5.3.2 3 Block Task . . . 64
5.4 Summary . . . 71

**6 Error Recovery** **72**
6.1 Petri Net Extension . . . 73
6.2 Experiments . . . 75
6.2.1 Known Error State . . . 77
6.2.2 Unknown Error State . . . 79
6.3 Summary . . . 82

**7 Conclusions and Future Work** **83**
7.1 Future Work . . . 85
7.1.1 Motion Learning . . . 85
7.1.2 Task Learning . . . 86
7.1.3 Error Recovery . . . 87

**References** **88**

**Appendices** **97**

**A Dynamic Movement Primitives Implementation** **98**
A.1 Motion Trajectory Data, $y$ . . . 98
A.2 Temporal Scaling Factor, $\tau$ . . . 99
A.3 Time Constants, $\alpha_z, \beta_z$ . . . 99
A.4 Goal Position, $g$ . . . 100
A.5 Gating Term, $v$ . . . 102
A.6 Gaussian Basis Functions, $\psi_i$ . . . 103
A.6.1 Number of Basis Functions, $N$ . . . 103
A.6.2 Center and Spread, $c_i$, $h_i$ . . . 104

**B Object Feature Extraction Implementation** **107**

**C Petri Net Task Learning Implementation** **109**

C.1 Petri Net Training . . . . . 111

C.2 Petri Net Generation . . . . . 113

**D Petri Net Error Recovery Implementation** **116**

# List of Tables


3.1 Components of the SOM algorithm . . . . . 22
3.2 Example PN structure . . . . . 25
3.3 Example PN markings . . . . . 26

4.1 Small motion data set clustering confusion matrix . . . . . 39
4.2 Clustering algorithm computational complexities . . . . . 50

5.1 Pre-selected features extracted from video stream data . . . . . 54
5.2 SOM parameter values used in implementation . . . . . 54
5.3 2 block task exact imitation PN structure . . . . . 60
5.4 2 block task PN place meanings (L = left, R = right, C = center) . . . . . 61
5.5 2 block task multiple demonstration PN structure . . . . . 62
5.6 3 block task exact imitation PN structure . . . . . 66
5.7 3 block task place meanings (L = left, R = right, C = center) . . . . . 67
5.8 3 block task subset imitation results . . . . . 69

6.1 Valid markings of 3 block task PN (L = left, R = right, C = center) . . . . 77

# List of Figures


1.1 Overview of the proposed learning by imitation system . . . . . 2

3.1 Example PN graph . . . . . 26
3.2 Example PN graph with marking . . . . . 27
3.3 Example PN reachability tree (dotted line connects repeated nodes) . . . . 28

4.1 The motion recognition component in the proposed learning by imitation system . . . . . 30
4.2 Segmentation results for right to left motion on top of another object (RLt) 31
4.3 The four possible groups of motions (LR, RLt, RL and LRt) when two blocks are stacked . . . . . 37
4.4 The experimental setup for capturing videos using the NAO robot . . . . . 37
4.5 Input preference values and the resulting number of clusters for the small motion data set using DMP weights and AP (the circle marks the median of all similarities) . . . . . 38
4.6 Example motion segments in the LR cluster (asterisks mark the exemplar motion found by AP and K-medoids) . . . . . 41
4.7 NAO robot performing the left to right motions (LR and LRt) . . . . . 42
4.8 NAO robot performing the right to left motions (RL and RLt) . . . . . 42
4.9 Input preference values and the resulting number of clusters for the large motion data set using DMP weights and AP (the diamond marks the minimum of all similarities and the circle marks the median of all similarities) . 43

4.10 Clustering results of the large motion data set using DMPs of 5 weights and two preference values for the AP algorithm. The total number of clusters found is indicated at the beginning of each row (including the preference value used in parentheses). The types of motions contained in each cluster are indicated through their respective labels, where WA = walking, SU = sumo leg raising, CH = cheering, KI = kicking, PU = punching, SQ = squatting, TH = throwing, BO = bowing, DA = dancing. If one cluster contains more than one type of motion, the type of motion of the representative exemplar for the cluster is italicized. . . . . 44

4.11 Input preference values and the resulting number of clusters for the large motion data set using HMM negative KL distances and AP (the diamond marks the minimum of all similarities and the circle marks the median of all similarities) . . . . . 45

4.12 Clustering results of the large motion data set using HMMs with 5 states and two preference values for the AP algorithm. The total number of clusters found is indicated at the beginning of each row (including the preference value used in parentheses). The types of motions contained in each cluster are indicated through their respective labels, where WA = walking, SU = sumo leg raising, CH = cheering, KI = kicking, PU = punching, SQ = squatting, TH = throwing, BO = bowing, DA = dancing. If one cluster contains more than one type of motion, the type of motion of the representative exemplar for the cluster is italicized. . . . . 46

4.13 Clustering results of the large motion data set using DMPs of 5 weights and the HAC minimum-variance algorithm. The total number of clusters found is indicated by the number of branches intersecting a specific cluster distance value. The 8 and 9 cluster cases are marked with dotted lines. The types of motions contained in each cluster are indicated through their respective labels, where PU = punching, TH = throwing, DA = dancing, SQ = squatting, CH = cheering, BO = bowing, KI = kicking, WA = walking, SU = sumo leg raising. . . . . 47

4.14 Visualization of the representative kicking (KI) motion . . . . . 48

4.15 Visualization of the representative squatting (SQ) motion . . . . . 48

5.1 The object recognition and task learning components in the proposed learning by imitation system . . . . . 53

5.2 2 block stacking task . . . . . 58

5.3 NAO robot imitating the 2 block stacking task . . . . . 59
5.4 2 block task exact imitation . . . . . 60
5.5 2 block task multiple demonstration PN graph . . . . . 62
5.6 2 block task multiple demonstration PN reachability tree . . . . . 63
5.7 3 block stacking task . . . . . 64
5.8 NAO robot imitating the 3 block stacking task . . . . . 65
5.9 3 block task exact imitation PN graph . . . . . 66
5.10 3 block task exact imitation PN reachability tree . . . . . 68

6.1 The error recovery component in the proposed learning by imitation system 73
6.2 Example of failing PN transition . . . . . 74
6.3 New reachability trees . . . . . 78
6.4 Human demonstrator corrects green block misalignment in 3 block stacking task . . . . . 80
6.5 PN with learned error correction . . . . . 80
6.6 PN reachability tree with learned error correction . . . . . 81
6.7 NAO robot performing error correction action after stacking the green block 81

B.1 Example video frame of a 2 block stacking task and the process of extracting the blue block . . . . . 108

C.1 Overview of the task learning code . . . . . 110

# List of Abbreviations

| | |
|---|---|
| AMS | Automated Manufacturing Systems |
| ANN | Artificial Neural Network |
| AP | Affinity Propagation |
| DMP | Dynamic Movement Primitive |
| DoF | Degree of Freedom |
| DTW | Dynamic Time Warping |
| EM | Expectation Maximization |
| FFNN | Feed-Forward Neural Network |
| FHMM | Factorial Hidden Markov Model |
| GMM | Gaussian Mixture Model |
| GSOM | Growing Self-Organizing Map |
| HAC | Hierarchical Agglomerative Clustering |
| HMM | Hidden Markov Model |
| KL | Kullback-Leibler |
| MDS | Multi-Dimensional Scaling |
| PCA | Principal Component Analysis |
| PHMM | Parametric Hidden Markov Model |

| | |
|---|---|
| PN | Petri Net |
| RGB | Red Green Blue |
| RL | Reinforcement Learning |
| RNN | Recurrent Neural Network |
| SEC | Semantic Event Chain |
| SIFT | Scale Invariant Feature Transform |
| SOM | Self-Organizing Map |
| SVM | Support Vector Machine |
| ZVC | Zero Velocity Crossing |

# Chapter 1

# Introduction

One of the long term goals in robotics, particularly humanoid robotics, is for robots to work and function within a human environment. Due to the dynamic nature of human environments, the ability to learn new tasks and to extend what has been learned is crucial. *Robot programming by demonstration* allows robots to learn new tasks from a human demonstrator without new code having to be written each time. This is desirable as writing new program code and maintaining existing program code is costly and requires special programming knowledge [70], [33]. But this also means that the underlying robot system must be capable of a more generalized form of learning to support learning new tasks and extending tasks it has already learned. To achieve this form of generalization, knowledge has to be transferred from the human demonstrator to the robot and the robot must be able to abstract relevant information from this transferred knowledge and appropriately represent it.

There are two main approaches to transferring knowledge from the human demonstrator to the robot via robot programming by demonstration: *learning by imitation* and *social learning*. Learning by imitation involves either having the human demonstrator move the robot through the required motions of the task [21], [12] or the robot observes the human demonstrator via vision based approaches [59], [72]. Social learning on the other hand is more interactive and involves feedback from a human teacher [63], [75].

Even though social learning requires a more complex interface for human-robot communication than learning by imitation, the underlying method of abstracting and representing information can be the same. Different levels of abstractions can be used. These levels form a hierarchical structure, where the lower levels of the hierarchy abstract either motion or object states from human demonstration and represent those using motion trajectory

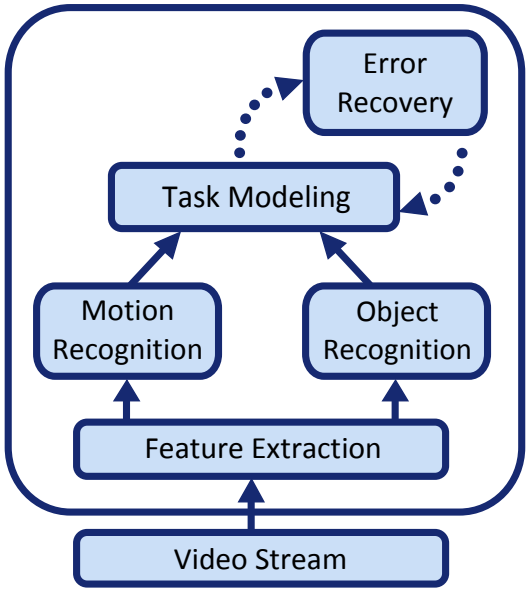



Figure 1.1: Overview of the proposed learning by imitation system

models [92], [53] or graphical models [89], [2], respectively. At a higher level of the hierarchy, the sequence of motions and object states can be combined into a task representation, usually based on a graphical model [59], [72].

This thesis proposes a learning by imitation system that relies on visual observations of human demonstrations. The proposed system uses a hierarchical setup where motions, objects and object states are abstracted from video stream data at the lower levels of the hierarchy and then further combined into a coherent task model at the higher levels of the hierarchy as shown in Figure 1.1. To improve the robustness of the learning by imitation system, error recovery capabilities during task execution have also been added.

In the proposed learning by imitation system, dynamic movement primitives (DMPs) [41], [42] are used for motion modeling and affinity propagation (AP) [31] clustering is used for motion recognition, while self-organizing maps (SOMs) [51], [52] are used for recognizing objects and their states in the environment. In order to construct a task model using the motions and objects recognized, it is noted that a goal-driven task consists of a sequence of pre-conditions needed for action, an action and post-conditions resulting from the action. As the pre- and post-conditions are the object states in the environment and the actions are the observed motions, a task model should reflect the sequence of object states and motions observed. This is achieved using Petri nets (PNs) [83], [82], [71].

Once PN task models have been created based on a single observation of a human demonstration, they can be used to generate the sequence of actions needed to perform the entire task as well as sequences for performing subsections of the task. Moreover, multiple demonstrations of the task can be summarized into a single PN model and the task can be performed with minor generalizations involving different combinations or configurations of objects. Furthermore, by checking current object states in the environment against the expected object states, the PN task model can be used to perform error detection and to request assistance from the human demonstrator to learn new error correction strategies. The demonstrated error correction strategy is incorporated into the task model, so that automated error recovery becomes possible when the error states are encountered again.

## 1.1 Contributions

1. **Motion Recognition via Affinity Propagation**
   Motion trajectories are segmented and modeled using DMPs. DMPs are chosen as they can be used to generate trajectories with variable initial and final positions during task imitation. A new approach for recognizing motions based on DMP weights and AP clustering is proposed. The weight parameters of the DMP are used as features for the AP clustering algorithm which will place similar motion trajectories into the same cluster. Each cluster represents the motion trajectories of a particular type of action within the task, thus, performing motion recognition. AP is exemplar based, meaning it will identify an exemplar motion trajectory to represent each cluster. This exemplar motion can be used to generate motions for the entire cluster during task imitation. The proposed approach is verified on two motion data sets.

2. **Task Learning via Petri Nets**
   A new approach for task modeling and learning based on PNs is proposed. The PN model can be expressed as a directed bipartite graph with vertices representing places and transitions. For task learning, the places in the PN are used to represent object states, while the transitions are used to represent actions. An approach for automatically generating the PN graph from observation of human task demonstration is proposed. The sequence of object states and motions observed in the video stream data of the human task demonstration will indicate the sequence of places and transitions of the PN. The place(s) that represent the object states before an action form the inputs to the transition and the place(s) representing the object states after an

action form the outputs of the transition, where the transition represents the action. The approach is verified experimentally on block stacking tasks.

3. **Error Recovery via Petri Net Extensions**
   A novel approach for error recovery during task performance is developed based on the PN task modeling framework. The automatically created PN model can be used for direct error recovery when a known, but unexpected state is encountered. The error recovery procedure consists of finding the place representing this unexpected state and performing the appropriate search for a sequence of actions to lead to the place representing the desired goal state. When an unknown and unexpected state is encountered, the PN can be extended via observation of the human demonstrator performing error correction. Once the PN has been extended with the error correction, the error state is no longer unknown and error recovery can be performed directly the next time this state is encountered. The proposed approach is tested on a block stacking task.

## 1.2 Outline

The remainder of this thesis is organized as follows:

Chapter 2 outlines existing work related to robot motion learning, task learning and error recovery. Existing learning systems usually do not combine motion learning, task learning and error recovery into a single framework.

Chapter 3 describes the various existing algorithms, i.e. DMPs, AP, SOMs and PNs, used in the proposed learning by imitation system. This includes the theoretical formulation of DMP motion models and how they are trained, the iterative computation of AP clustering and SOM training, as well as the relevant theoretical background and concepts of PN models.

Chapter 4 covers motion learning, i.e. motion segmentation, motion modeling via DMPs and motion recognition via AP clustering. Experimental results using two different motion data sets are given to illustrate the robustness of the proposed motion recognition approach. Comparisons to hidden Markov models (HMMs) for motion modeling as well as other common clustering approaches are performed.

Chapter 5 presents the proposed task learning approach, including object and object state recognition and the procedure for automatically creating PNs from the information extracted from observations of human demonstration. The PN creation process uses both

recognized motions and object states and a created PN model can be used to generate sequences of actions for task imitation. Experimental results for the PN creation process and the subsequent task imitation are provided for block stacking tasks.

Chapter 6 covers the error recovery approach which includes dynamically adjusting the sequence of actions for unexpected, but known error states and learning new error correction strategies from the human demonstrator for unexpected and unknown error states. Experimental results of error recovery are shown for a block stacking task.

Lastly, Chapter 7 summarizes the key contributions of the thesis and outlines directions for future work.

# Chapter 2

# Related Work

Robot programming by demonstration allows robots to learn tasks via human demonstration without having to add new program code to the robot system for each new task. Initially, robots used in assembly line tasks were programmed by recording trajectory keypoints via teleoperation [66], but there was a demand for more flexibility and generalization in learned tasks than just replaying fixed recordings of motions [33]. The addition of humanoid robots further motivates an increased ability to learn for robots, as humanoid robots are likely to be employed in dynamic human environments which requires learning a wide variety of new tasks and extensions to tasks.

One important modality for robot learning is through observation of a human demonstrator performing the task to be learned. As part of a generalized learning process, knowledge has to be transferred from the human demonstrator to the robot. The robot must then abstract the knowledge by extracting relevant information and reorganizing the extracted information into an appropriate representation of the task. In order to learn the abstraction and representation in a generic form, machine learning techniques are usually used to implement the learning system, e.g. [92], [53]. Machine learning is traditionally divided into three main paradigms [8], [16]: *supervised learning* (requiring data labels), *unsupervised learning* (no data labels required) and *reinforcement learning* (requiring a scalar reward signal).

Two main approaches exist for transferring knowledge in robot programming by demonstration: *learning by imitation* and *social learning*. There are also learning approaches that do not require a human demonstrator (at least for part of the learning process), as the robot autonomously explores its surroundings [78], [91] or improves what it has learned through self-practice [29], [7], [75]. The focus of this thesis will be on learning from obser-

vation, autonomous exploration and self-practice are not the main focus of this thesis and will not be discussed in further detail.

The learning by imitation approach can be further divided depending on how the robot records the human demonstration. One method involves the robot learning to perform a task by being moved through the sequence of required motions by a human demonstrator using a teach pendant or force/moment sensors [21] or kinesthetic training [12]. These methods avoid the correspondence issue [5] where a mapping between the different embodiments of humans and robots is required. However, it can be difficult to simultaneously provide accurate guidance for many degrees of freedom (DoF) when using teach pendants or kinesthetic training. Another method uses computer vision to record the demonstration, so that the human can demonstrate the task naturally without having to put the robot through the motions. The robot simply observes the demonstration and abstracts the relevant features for task representation [59], [72]. In this case the robot must resolve the correspondence problem, for example by imitating the Cartesian coordinates when performing goal based tasks and solving the inverse kinematics for its specific embodiment.

The social learning approach, on the other hand, is more interactive and involves feedback from a human teacher [28], [63], [75]. Due to the interactive nature of learning, social learning approaches often include the ability to adjust learned tasks [28], [63], [75]. In order for this form of learning to occur naturally, the robot would require the ability to interpret and convey social cues such as intention and confusion [63] during communication with the human teacher. Though the human teacher would be able to communicate more naturally with the robot in this case, capabilities beyond the ability to abstract and represent the task are required.

Both the learning by imitation and social learning approaches require an underlying system for abstracting and representing the task. The different levels of abstraction form a hierarchical structure, where the lower levels in the hierarchy contain more detailed information about the task, such as motion trajectory information and object and object state features, whereas the higher levels of the hierarchy may contain an overview of the sequence of motions and object states required for the task. This hierarchical structure is inspired by human based learning models, e.g. Deacon's symbolic human language model [26]. Deacon argues that the emergence of human language involves a hierarchy consisting of *icons*, *indices* and *symbols*. Icons, at the lowest level of the hierarchy, represent the associations we make between objects based on similarities in physical properties. On a higher level of the hierarchy, indices represent the associations we make between objects based on their correlations in space, time or otherwise. On the highest level, symbols form associations between indices and also between other symbols. When symbols are arranged in a particular order "language" emerges. Despite the usefulness of a hierarchical

arrangement, not all learning methods mention a hierarchical setup explicitly, though some works refer to a sub-symbolic (control or trajectory) level versus a symbolic level learning approach [33], [70], [35], [11] which correspond to the lower and higher levels of a hierarchy of abstraction, respectively.

Because of the highly dynamic nature of the human environment, the outcomes of actions may not always be as expected. Unexpected outcomes while performing tasks can be considered error conditions. For a robust learning system, the ability to handle error conditions via proper error recovery is very useful. Moreover, to be able to generalize handling error conditions, the ability to extend the learned task model to incorporate demonstrated error recovery procedures is necessary. It has been shown that the ability to recover from error conditions adds to the positive image of a service robot [60]. Learning by imitation does not always consider error recovery, but social learning may inherently involve the ability to extend what has been learned based on human feedback which can include error recovery.

The learning system presented in this thesis is based on learning by imitation with error recovery capabilities. Such a system requires both motion modeling and task modeling in order for the robot to be able to generate the motions necessary to imitate a demonstrated task. Therefore, motion modeling and recognition is discussed first in the following sections before task modeling and error recovery.

## 2.1 Motion Learning

In order to learn the motions associated with a task, the motion trajectory recorded during a task demonstration is usually divided into meaningful atomic units called *motion primitives* [86]. Learning these motion primitives allows longer motion trajectories to be reconstructed by sequencing a number of learned motion primitives together. In order to generate motion trajectories similar to the demonstrated trajectories, it is necessary to train motion models based on the observed motion trajectories. The ability to distinguish new motion primitives from known ones is necessary to prevent repeatedly relearning the same motion primitives. This requires the ability to recognize motions. The following subsections will present existing methods of motion modeling and recognition.

### 2.1.1 Motion Modeling

A number of motion modeling techniques have been applied for representing and learning motion primitives or longer motion segments, including artificial neural network (ANN) approaches such as recurrent neural networks (RNNs) [77], [44], stochastic approaches such as Gaussian mixture models (GMMs) [10], [12] or hidden Markov models (HMMs) [58], [56], [57], [93], [38], [54] and dynamical system approaches such as dynamic movement primitives (DMPs) [41], [42], [87], [86].

Ogata et al. [77] use an ANN approach that applies Jordan type RNNs consisting of a normal feed-forward neural network (FFNN) with added context units. The context units form extra input and output nodes to the normal FFNN to reflect the current state of the system corresponding to past inputs. The resulting RNN is able to operate in two modes once trained using back propagation: an open loop mode for generating the motion trajectory and a closed loop mode for rehearsing learned motion trajectories. The advantage of RNNs is that new motion sequences can be learned without damaging previously learned motions, which is a problem for normal FFNNs. The main disadvantage is that there is no way of specifying specific goal positions for the trajectory for a manipulation task and it is also not clear how motion recognition could be performed on trained RNN models. Furthermore, RNNs can be computationally expensive and have to be trained offline. Also, the accuracy of the generated trajectories may not be sufficient as an additional collision avoidance system had to be implemented to override the RNN output at times [77].

Calinon and Billard [10] propose a stochastic motion model which uses principal component analysis (PCA) to transform the recorded motion trajectories to a lower dimensional space before using a GMM consisting of a number of Gaussian distributions to model the shapes of the motion trajectory, as well as the variabilities observed across different demonstrations of the same trajectory. GMMs are normally trained offline via the expectation maximization (EM) algorithm, but [10] presents two methods for training GMMs incrementally. Furthermore, the average log-likelihood of the GMMs can be used for motion recognition. Another type of stochastic motion model is the HMM which consists of a set of observations and a set of hidden states along with the observation to state probabilities and the state transition probabilities [84]. HMMs are commonly used for motion modeling as they can be trained on multiple demonstrations of the same motion trajectory and can be used to generate motions similar to the observed demonstrations, though the standard HMM does not have a mechanism for parameterizing the generated trajectories based on properties such as the goal position, which is important for task manipulation. Thus, extensions such as parametric HMMs (PHMMs) and factorial HMM (FHMMs) become necessary at an additional computational cost. PHMMs explicitly add parameters

for specifying the goal positions of trajectories [93], while FHMMs train multiple HMMs [58].

DMPs are a dynamical system approach which model motion trajectories using a second order differential equation with an additional nonlinear term that consists of weighted Gaussian functions and is independent of the progression of time [41], [42], [87], [86]. The second order differential equation is guaranteed to converge to a parameterized goal value as the effects of the nonlinear term are forced to diminish towards the end of the motion trajectory. Thus, once trained on a demonstrated motion trajectory, DMPs can generate generalized motions between a specific start and end position. The separation of the nonlinear term from the progression of time allows the DMP to reject disturbances by halting the progression of the motion trajectory until the disturbance has been removed [41]. The correlations between weight parameters of the DMP can be used as a distance metric for motion recognition as they have been shown to reflect similarities between motions [41], [42]. These properties make the DMP an attractive choice for modeling motions for manipulation tasks. Furthermore, extensions for improving the similarity between demonstrated and generated trajectories [40], [80], as well as for providing explicit control over acceleration [73], velocity at the end of the trajectory [50] and obstacle avoidance capabilities [79], [40] have been added to the DMP formulation. Many of these extensions are possible due to the analytic formulation of the DMP as a second order differential equation with additional terms. One drawback of the DMP is the sensitivity of the weight parameters towards noise, which may require motion trajectories to be filtered to reduce the effects of noise.

### 2.1.2 Motion Recognition

Many algorithms for learning motions assume that segmented motion trajectories are readily available and assume that the identification of which motion exemplar belongs to which primitive is provided a priori [9]. Fewer algorithms consider automating the process of assigning exemplars to primitives or identifying motion primitives from continuous data.

The RNNs in [77], for example, were trained on the entire motion trajectories without identifying the motion primitives and no method of motion recognition is proposed. As previously mentioned, it is not clear which metrics should be used for determining the similarity between RNNs trained using different demonstrations of motion trajectories. The GMM in [10] uses the average log-likelihood as a similarity measure, but motion recognition is simply based on a tuned threshold of this log-likelihood value and the robustness of this recognition system is unclear.

Some algorithms, especially those using HMMs or their extensions as motion models, have used clustering methods for creating a further abstracted representation of motion. Of these, [90] models motion primitives with HMMs and uses the Kullback-Leibler (KL) distance and multi-dimensional scaling (MDS) to reduce the dimensionality of the data, effectively finding clusters of motions. On the other hand, [58] models motions as FHMMs and uses hierarchical agglomerative clustering (HAC) for grouping and storing motions during on-line observation. However, the resulting dendrogram tree structures are not stable and can be presentation order dependent [58]. Thus, it may be necessary to apply online tree correction [56], which adds additional computational expense. Furthermore, as the tree structure does not pick out a particular motion exemplar to represent each group of motion, additional computational expense is incurred in training FHMMs to represent a group of motions for motion generation. In addition, as multiple motions are grouped together to train the generative model, there is no guarantee that the resulting motion will remain kinematically consistent.

While the use of weight parameters to compare DMPs has been previously proposed [41], [42], there is not much work on motion recognition using DMPs. Meier et al. [68] propose an approach for simultaneous motion recognition and segmentation using DMPs [68]. However, the approach in [68] requires a pre-existing library of movement primitives to be created. Creating a comprehensive library of such movement primitives can be time consuming, while a small library can be insufficient for proper motion recognition.

## 2.2 Task Learning

Robot task learning can be divided based on the level of abstraction. At the lowest level, the actions represented by motion trajectories are modeled [57], [1], [92], [53]. At a higher level, object relationships are modeled [89], [2], [3] and at the highest level, both actions and objects are considered [59], [72]. Representative examples of each level of abstraction are presented in the following subsections.

### 2.2.1 Action Based

Action based approaches to task modeling tend to model both motion trajectories and the sequence of motion trajectories observed [57], [1], [92], [53]. Both [57] and [1] model motion trajectories and the sequence of motion trajectories using separate HMMs. Though task imitation using this setup is not verified.

Vicente et al. [92] use motions to analyze a task. In this approach, a combination of support vector machines (SVMs) and HMMs are used to model and identify actions either as simple primitives or as a composite of primitives. The composite of primitives model forms a directed graph that models the sequence of actions for performing a task. However, this work focuses on task recognition rather than task imitation.

Konidaris et al. [53] propose an approach where demonstrated trajectories are segmented into skill chains using reinforcement learning (RL) and change point detection. The skill chains represent the sequence of actions required to perform a task. In the case of multiple demonstrations, multiple skill chains may be created and then merged into skill trees. This task modeling approach does not work when multiple demonstrations form a more general graph than a tree.

### 2.2.2 Object Based

Object based approaches tend to use graphical models for representing tasks [89], [2], [3]. However, imitation of the task becomes difficult without motion models.

Summers-Stay et al. [89] use object relations to analyze a task, where events corresponding to objects touching other objects or the hand of the human demonstrator are extracted into a tree structure called an activity tree. This method of modeling the task is capable of separating two tasks that are interleaved, as the objects involved in each task will not touch and hence belong to separate activity trees. However, this approach does not incorporate storing and generating the actions so that they can be performed by a robot.

Aksoy et al. [2] use changes in the spatial relation of objects captured in videos to construct a transition matrix called the semantic event chain (SEC) to model the human demonstrated task. The SEC is a condensed sequence of spatial relations between objects, which encode whether objects are touching or overlapping. The challenge with using object spatial relations is how to exclude spurious cases where objects only seem to be touching due to the position of the camera. The original SEC model does not incorporate actions such that motions could be reproduced by the robot to actually perform the task. This work has been extended, however, in [3] to execute simple pushing actions for manipulation tasks. These pushing actions do not encode trajectory shapes of the motions via motion modeling. Instead the objects states are used to determine simple straight-line motions between the starting and ending position of an object.

### 2.2.3 Object and Action Based

Approaches that involve both objects and actions tend to be able to perform task imitation, though the models used to represent the overall task do not possess the modeling power of Petri nets (PNs) [59], [72].

An early example of learning by imitation involving both actions and objects is the work by Kuniyoshi et al. [59], where the robot learns from a human demonstrator via video recordings. Vision processing is applied to extract objects, actions and the overall goal of the task. The task is modeled only minimally using a stack of actions and environmental states. The goal extraction allows the robot to perform the task with some level of generalization. However, the extent of the generalization is limited to small changes, such as a slight difference in the initial placement of objects.

Nagai [72] provides a more recent example of task learning using both actions and objects. Their proposed learning system extracts actions and objects separately from video image frames. This abstracted information is then combined to model the overall task. A lot of emphasis is put on the vision techniques for selecting the area of the image that the attention should be focused on, while the task modeling is simply a sequential ordering of the actions and objects detected. This task modeling method does not allow for the learned task to be generalized to different situations.

PNs have been applied in the robotics domain to model human-robot interaction [20], multi-robot systems [22], [98] and tasks with qualitative performance evaluation [61], [69]. In all these applications the PNs were created manually and not automatically generated by learning from observation of human demonstration. Ziparo et al. [98] use a modularized approach to make dynamic adjustments to the PN structure based on new observations, i.e. a large PN is composed of known elementary PNs that are arranged in a sequence based on observations of the states of the environment. However, the elementary PNs are still defined by humans rather than learned from observation.

## 2.3 Error Recovery

The process of error recovery can be divided into error detection, error diagnosis and error correction [62]. Error detection is usually handled by some form of monitoring system [62], e.g. load monitoring using force sensors in [94] or the intelligent monitoring system including touch sensors in [36].

Error detection and correction have received significant attention in the automation literature, particularly for automated manufacturing systems (AMS) [25], [62]. Common

approaches include task planning and knowledge base methods [36], [64], [67] and graphical models such as PNs [97], [95], [25], [76]. In general, error correction can be classified into four categories [97], [62], [76]:

1. *Forward error recovery* consists of reaching a later state, i.e. error correction leads to a state that comes after the state which led to the error. This approach is most desirable, but may be difficult to implement in a fully automated system [76].

2. *Backward error recovery* consists of reaching a previous state, i.e. error correction leads to a state that occurs before the state which led to the error. This approach assumes reversibility in processes and may lead to infinite loops in the PN, though the problem of infinite loops can be avoided by preventing continued re-use of states in the backward recovery "loop" [76].

3. *Input conditioning* consists of returning to the state that led to the error via error correction.

4. *Alternate path* avoids the normal path when an error occurs and selects an alternate path that will result in error recovery.

### 2.3.1 Plan Based

Plan based approaches usually use a pre-defined language for describing sequences in the task and error recovery is achieved by constructing a knowledge base which consists of if-then rules created by a human expert [36], [67]. Matsuoka et al. [67] propose a robotic hand system capable of manipulation tasks with error recovery. The error recovery process uses object pose estimation for error detection and a pre-defined knowledge base of error correction procedures. No learning of error recovery procedures occurs.

The work of Gini and Gini [36] is an example of using a knowledge base for error recovery. This approach considers the pre-conditions and post-conditions of each action for detecting errors, thus, taking into consideration the goals of the actions. However, their approach requires a human expert to encode the error recovery rules into the knowledge base and no learning occurs from the errors encountered.

### 2.3.2 Graph Based

Error correction using PNs usually consists of adding an error recovery subnet to the normal PN. Depending on the application, the error recovery subnet may implement any one of

the four aforementioned error correction categories. Examples include PNs extended with fuzzy logic [14], [15] or timing [76] to perform error recovery for manufacturing processes via an error recovery subnet whenever a difference between the expected state and actual state is detected. However, in these works, both the nominal and error correction subnets are specified a priori and no learning occurs.

The approach in [97] considers both cases where unexpected, but known states (no error or previously seen errors) and unknown states are encountered. However, while this work proves that adding recovery subnets preserves the overall properties of the PN, it does not specify exactly how the new error recovery subnets would be constructed and added to the PN.

### 2.3.3 Social Learning Based

There is also related work addressing error recovery in robot task performance [23], [94]. For example, [23] uses PNs to model action plans for soccer playing robots that include the possibility of failed actions. Their approach of structuring the PN is different from our proposed task learning process as both states and actions are encoded as part of the place component of the PN. Furthermore, there is no learning or dynamic extension of the PN. Yamazaki et al. [94] on the other hand use manipulation behaviors consisting of behavior units for an assistive cleaning robot. Error recovery capabilities are designed into the behavior units, but no learning occurs.

Error recovery has also been addressed through human-robot interaction and social learning. For example, Lockerd and Breazeal [63] develop an approach for robot task learning through interactive demonstrations. The robot can query the human demonstrator for additional guidance when it cannot autonomously complete the task. Their approach models tasks symbolically via simple words representing the task goals and it is unclear how information for recovery from particular errors would be incorporated into the overall goal. Similarly, Nikolescu and Mataric [74], [75] develop a sequential task learning framework using directed acyclic graphs where the robot can learn from observation and query the human demonstrator to correct the task model if errors are encountered. Their approach also relies on pre-conditions and post-conditions of actions, but it requires the error correction portion to be interleaved with the task execution in order for it to be incorporated into the correct location in the sequence. In our approach, even though our examples show the error correction during task execution, the error correction portion can also be shown as a separate demonstration and still be correctly added to the PN due to the additional encoding of object states in the PN.

## 2.4 Summary

In motion modeling, ANN and stochastic approaches are unable to generalize generated motion trajectories to a particular goal position. Although extensions such as PHMMs and FHMMs exist for HMMs, they come with an added computational cost. DMPs on the other hand inherently allow parameterized variations in the goal position. As the ability to generate motion trajectories to a particular goal position is important for task imitation to be generalizable, DMPs are preferable for task based motion modeling. Furthermore, the DMP weight parameters appear to be good features for motion recognition. For task modeling, combining actions and objects into a single task model more readily allows for task imitation as opposed to only using actions or objects. Though PN models have been applied to various robot systems, they tend to be manually generated and are not automatically created from observation. Error recovery approaches exist for plan based systems, PNs and social learning systems. Error correction is not usually considered in learning by imitation systems. The existing PN error recovery approaches rely on manually specified error recovery subnets and are not automatically learned from demonstration.

# Chapter 3

# Background

The learning by imitation system implemented in this thesis involves motion recognition, object and object state recognition, as well as task modeling with error recovery. For the motion recognition component, DMPs are used to model motions and the AP clustering algorithm is used to perform motion recognition by grouping similar motions into the same cluster. For the object and object state recognition, SOMs, a type of ANN, are used to match object features to the according object or object state. Both APs and SOMs are unsupervised machine learning methods that do not require data labels during training, which may not always be readily available. Lastly, for task modeling, a PN is used to represent the sequence of both motions and object states that are recognized. DMPs, APs, SOMs and PNs are existing algorithms or modeling methods and are introduced in more detail in the following sections of this chapter.

## 3.1 Dynamic Movement Primitives (DMPs)

The DMP model is based on a dynamical system expressed as a nonhomogeneous second order linear differential equation:

$$\begin{aligned} c_3\ddot{y} + c_2\dot{y} + c_1 y &= c_1 g \\ \ddot{y} + c_2'\dot{y} + c_1'(y - g) &= 0 \end{aligned} \tag{3.1}$$

where $y$ is the current position, $g$ is the goal position, $c_2' = \frac{c_2}{c_3}$ and $c_1' = \frac{c_1}{c_3}$ are positive constants. The system in (3.1) is globally, exponentially stable, i.e. the position $y$ will exponentially converge to goal $g$ [41], [42]. However, a linear system cannot model arbitrary

complexities in the motion trajectory, hence, a nonlinear component $f$ is introduced by first rearranging the nonhomogeneous second order linear dynamical system as follows:

$$\begin{aligned}
\ddot{y} &= -c_2'\dot{y} - c_1'(y-g) \\
\ddot{y} &= c_2'(\frac{c_1'}{c_2'}(g-y) - \dot{y}) \\
\tau^2\ddot{y} &= \tau^2 c_2'(\frac{c_1'}{c_2'}(g-y) - \dot{y}) \\
\tau^2\ddot{y} &= \tau c_2'(\frac{\tau c_1'}{c_2'}(g-y) - \tau\dot{y}) \qquad (3.2)
\end{aligned}$$

where $\tau$ is a temporal scaling factor. Then, adding nonlinear component $f$:

$$\begin{aligned}
\tau^2\ddot{y} &= \tau c_2'(\frac{\tau c_1'}{c_2'}(g-y) - \tau\dot{y} + f) \\
\tau(\tau\ddot{y}) &= \tau c_2'(\frac{\tau c_1'}{c_2'}(g-y) - (\tau\dot{y} - f)) \qquad (3.3)
\end{aligned}$$

and introducing a new state and constants:

$$\tau\dot{z} = \alpha_z(\beta_z(g-y) - z), \quad z = \tau\dot{y} - f \qquad (3.4)$$

where $\alpha_z = \tau c_2'$ and $\beta_z = \frac{\tau c_1'}{c_2'}$ are time constants.

Equation (3.4) is the original, standard formulation of the DMP motion model for goal directed motions [42], [86]. In order to support any arbitrary nonlinear complexity, the nonlinear component $f$ is modeled as a normalized weighted sum of Gaussian basis functions [42], [86]:

$$f = \frac{v\sum_{i=1}^{N} w_i\Psi_i}{\sum_{i=1}^{N}\Psi_i}, \quad \Psi_i = exp\left(-h_i\left(x-c_i\right)^2\right) \qquad (3.5)$$

where $v$ is a gating term, $w_i$ are weight parameters learned during training, $N$ is the total number of basis functions used and $\Psi_i$ is a Gaussian basis function with center $c_i$ and spread $h_i$.

In order to reduce the effects of $f$ as the position $y$ approaches the goal $g$, the gating term $v$ must tend towards zero as $y$ approaches $g$. This is accomplished by modeling $v$ as a linear dynamical system:

$$\tau\dot{v} = \alpha_v(\beta_v(g_v - x) - v), \quad v = \tau\dot{x} \qquad (3.6)$$

where $x$ is a state variable that varies non-linearly with time to decouple the dependence of the nonlinear component $f$ with the progression of time in order to allow for disturbance rejection, i.e. if the motion is halted by an obstruction, the state $x$ can remain close to constant regardless of the progression of time until the obstruction is removed at which point the motion will proceed along the (time-delayed) planned trajectory [41].

The weight values of the DMP model are determined during training of the DMP, where the progression of the gating term $v$ and state $x$ are first computed using:

$$\dot{v} = \frac{\alpha_v(\beta_v(0 - x) - v)}{\tau} \tag{3.7}$$

$$\dot{x} = \frac{v}{\tau} \tag{3.8}$$

By repeatedly computing these derivatives, the gating term $v$ and state $x$ can be found via integration, given that the initial values are $v_0 = 0$ and $x_0 = 1$. Then, given the exemplar position $y$, velocity $\dot{y}$ and acceleration $\ddot{y}$ trajectories, the nonlinear component $f$ can be computed using:

$$f = \frac{\tau^2 \ddot{y}}{\alpha_z} + \tau \dot{y} - \beta_z(g - y) \tag{3.9}$$

And the basis functions can be computed using:

$$\psi_i = exp(-h_i(x_i - c_i)^2) \tag{3.10}$$

Lastly, the weights can be computed via regression methods [34] by rearranging (3.5):

$$w = \frac{f \sum_{i=1}^{N} \Psi_i}{v \sum_{i=1}^{N} \Psi_i} \tag{3.11}$$

Once the weight values $w_i$ have been found, the DMP model is trained.

The weights of the DMP indicate how each basis function $\psi_i$ should be scaled to fit to the nonlinearities in the motion trajectory. Thus, the value of the weights reflect the type of motion that the DMP was trained on. The authors of [42] and [41] have shown that the correlation $c$ between the weight values of separate DMP models trained on different motion trajectories can be used to indicate how similar the motions are. The correlation $c$ is computed using:

$$c = \frac{w_a^T w_b}{\mid w_a \mid\mid w_b \mid} \tag{3.12}$$

where $w_a$ is a vector of weights for motion $a$ and $w_b$ is a vector of weights for motion $b$. A higher correlation $c$ value indicates that the motions are more similar.

DMPs are generative models, because once trained, a starting position $y_0$ (position $y$ at time zero) and a goal position $g$ can be used to compute the corresponding $y$ values of the trajectory using (3.4). Furthermore, the $y_0$ and $g$ values are adjustable, which means the DMP can generalize the learned trajectory, i.e. the DMP can generate trajectories of a similar shape as the trajectory that it was trained on, but with different $y_0$ and $g$ values. The actual values of the various parameters in the DMP formulation are explained in more detail in the context of an implementation in Chapter 4, Section 4.4.1 and Appendix A.

## 3.2 Affinity Propagation (AP)

AP is a clustering algorithm that simultaneously considers all data points as possible candidates for being an exemplar for a cluster [31]. Real-valued messages are passed between data points to indicate the suitability of each data point to serve as a cluster exemplar. As these messages are updated iteratively, good cluster exemplars emerge along with the data points that the exemplar represents, i.e. the data points belonging to each cluster.

The input to AP is a matrix of similarities between data points, where the element $s(j, k)$ of this matrix indicates how well the candidate exemplar data point $k$ represents the data point $j$, i.e. the similarity of data point $k$ to data point $j$. Any distance metric could be used to calculate this similarity, though the negative Euclidean distance is recommended [31]. For the diagonal elements of the matrix of similarities $s(k, k)$, rather than calculating the distance of data point $k$ to itself, a "preference" value is set. The "preference" value incorporates any prior knowledge of the suitability of a particular data point $k$ for being a cluster exemplar, where a higher "preference" value indicates a greater suitability. If no prior knowledge is available, all "preference" values are set to the same value such as the median of the input similarities (for a moderate number of resulting clusters) or the minimum of the input similarities (for a small number of resulting clusters) [31].

Two kinds of messages are passed between the data points: "responsibilities" and "availabilities". The responsibility $r(j, k)$ is sent from data point $j$ to candidate exemplar data point $k$ to indicate how confident data point $j$ is in data point $k$ being its exemplar. The availability $a(j, k)$ is sent from candidate exemplar data point $k$ to data point $j$ to indicate how confident $k$ is in being a good exemplar for $j$. The steps of the AP algorithm are as follows:

1. Initialize all availabilities to zero.

2. Compute and update the responsibility values:

$$r(j,k) \leftarrow s(j,k) - \max_{k':k' \neq k} \{a(j,k') + s(j,k')\} \tag{3.13}$$

3. Compute and update the availability values:

$$a(j,k) \leftarrow \min\{0, r(k,k) + \sum_{j':j' \notin \{j,k\}} \max(0, r(j',k))\} \tag{3.14}$$

$$a(k,k) \leftarrow \sum_{j':j' \neq k} \max(0, r(j',k)) \tag{3.15}$$

4. Repeat 2) and 3) until the exemplar decisions no longer change for a number of iterations, e.g. for 10 iterations. Exemplar decisions are made for each data point $j$, by finding the value of $k$ that maximizes $a(j,k) + r(j,k)$. If $j = k$, data point $j$ is an exemplar, otherwise $k$ will indicate the exemplar for data point $j$.

The AP algorithm outlined above runs offline and has been tested by the original authors on clustering images of faces, gene data, sentences and accessibility of cities based on airline travel times between them [31].

## 3.3 Self Organizing Maps (SOM)

The SOM is a specific type of ANN developed by Kohonen in the 1980s [51] which consists of only two layers of interconnected simple units, i.e. nodes. The SOM is trained without data labels, i.e. it is an unsupervised machine learning method, and is capable of matching a particular input pattern to a specific output signal, which can be used to match a set of object or object state features to a specific output signal representing the object or object state matching those features. The two layers of nodes in a SOM are referred to as the input and output layer. Each node in the input layer is connected to every node in the output layer via a weighted connection. The number of nodes in the input layer of the SOM corresponds to the number of elements in the input patterns, i.e. dimension of the input data vector, presented to the SOM, while the number of output nodes should be at least as much as the expected number of clusters that the input data should be divided into, e.g. the number of objects or the number of object states matching the observed input features.

Table 3.1: Components of the SOM algorithm

| **Component** | **Description** |
|---|---|
| $x_{SOM}$ | Input vectors |
| $c_{SOMi}$ | Output nodes |
| $w_{SOMci}$ | Weight vectors contained in a weight matrix |
| $\alpha_{SOM}(k)$ | Learning rate where $0 < \alpha_{SOM}(k) < 1$ |
| $N_{SOMci}(k)$ | Neighborhood |

Given an input pattern, i.e. input data vector, one winning output node is selected based on the weights of the connections. During training, the SOM weights will be adjusted to ensure that the same or similar input patterns will activate the same output node. Output nodes around the winning node are considered to be part of the neighborhood of the winning node and may also be updated during training. The advantage of the SOM, apart from being an unsupervised learning method, is that the same output will be consistently picked for the same input pattern even during training, so that it can be used to incrementally learn to match observed features to object and object states.

The formal SOM algorithm for picking the winning output node requires the definitions shown in Table 3.1. The neighborhood $N_{SOMci}$ is calculated using (3.16) as specified by Kohonen [52]:

$$N_{SOMci}(k) = exp\left(\frac{-\|r_{SOMci} - r_{SOMwin}\|^2}{2 \cdot \sigma^2_{SOM}(k)}\right) \tag{3.16}$$

Where $\|r_{SOMci} - r_{SOMwin}\|$ is the Euclidean distance between the current output node and the winning output node and $\sigma_{SOM}(k)$ is the adjustable parameter of the neighborhood function.

The SOM training algorithm proceeds as follows [47]:

1. *Initialization:* All of the weights in the weight matrix are initialized to small random values. The learning rate $\alpha_{SOM}$ and the neighborhood parameter $\sigma_{SOM}$ are initialized. The initial values of $\alpha_{SOM}$ and $\sigma_{SOM}$ are tunable depending on the application.

2. *Selection:* One input vector is selected for training. The winning output node $c_{SOMi}$ is selected using the performance index $I$ as shown in (3.17).

$$I_{SOMwin} = min\|x_{SOM} - w_{SOMci}\| \tag{3.17}$$

   Where $\|\cdot\|$ represents the Euclidean distance.

3. *Weight Update:* The weight vector corresponding to the winning node and the neighboring nodes specified by neighborhood $N_{SOMci}$ are updated as shown in (3.18).

$$w_{SOMci}(t+1) = w_{SOMci}(t) + \alpha_{SOM}(k) \cdot N_{SOMci}(k) \cdot [x_{SOM} - w_{SOMci}(t)] \quad (3.18)$$

4. *Parameter Update:* After repeating steps 2) and 3) for each input vector presented, one epoch of training is complete and $\alpha_{SOM}$ and $N_{SOMci}$ are reduced using a shrinking function, for example:

$$\alpha_{SOM}(k+1) = c_{SOM1} \cdot \alpha_{SOM}(k) \quad (3.19)$$
$$\sigma_{SOM}(k+1) = c_{SOM2} \cdot \sigma_{SOM}(k) \quad (3.20)$$

   Where $c_{SOM1}$ and $c_{SOM2}$ are constants tuned depending on the application.

Steps 2) to 4) of the SOM algorithm above can be repeated until the change in weights during the weight update step is sufficiently small. However, the algorithm above can also be considered incremental [52] as the SOM is adjusted according to each new input pattern seen and will consistently activate the same output node for the same input pattern for robust recognition results.

## 3.4 Petri Nets (PN)

PNs are a tool for modeling processes including any type of information or part flow. Originally developed by Carl Adam Petri in his PhD thesis [83], PNs can be expressed in a mathematical or graphical manner and are flexible enough to accommodate processes that are sequential, parallel, synchronous, asynchronous, deterministic and/or stochastic [71], [81]. PNs have been applied in a variety of fields, including distributed software systems, manufacturing control systems, local-area networks, etc. [71]. PNs are usually generated manually and were originally used to *describe* systems, but have also become increasingly used to *analyze* the properties of systems [81].

A PN consists of two main components: places ($P$) and transitions ($T$). Places represent states or conditions in the system being modeled, while transitions represent different actions or events that may occur. The connections between the places and transitions is indicated by defining input places ($I$) and output places ($O$) for each transition, effectively defining the states or pre-conditions necessary for an action or event to take place and the states or post-conditions resulting from a particular action or event. The only connections

in PNs is between places and transitions or transitions and places, places cannot connect directly to other places without a transition in between and transitions cannot directly connect to other transitions without a place in between. Transitions without any input or output places are called *source* or *sink* transitions, respectively [71].

To describe the dynamics of a PN, places in the PN are marked with *tokens* to indicate the current state(s) or condition(s) of the system. *Bounded* PNs limit the number of tokens that a place may hold to a finite number. The tokens indicate which transitions are *enabled*, i.e. which action(s) or event(s), if any, may take place. A transition is considered enabled if all of its input places have at least one token in them. The *firing* of a transition removes one token for each input place of the transition and adds one token into each output place. For *weighted* PNs, where a weight is associated with each input place to transition connection and each output place to transition connection, the number of input place tokens required for a transition to be enabled and the number of tokens removed or added corresponds to the weights of the connections to the input and output places.

PNs may be expressed either mathematically as *PN structures* or visually as *PN graphs* [81]. It is also possible to set up PNs in a hierarchical manner, where places and/or transitions are represented by a sub-net, i.e. another PN [81]. The following subsections provide more detailed descriptions of the above mentioned points.

### 3.4.1 Petri Net Structures

The mathematical notation of a PN consists of a 4-tuple [71], [81]:

$$C = \{P, T, I, O\} \tag{3.21}$$

where

$P = \{p_1, p_2, ..., p_m\}$ is the finite set of all places $p_i$

$T = \{t_1, t_2, ..., t_n\}$ is the finite set of all transitions $t_i$

such that $P \cap T = \varnothing$ and $P \cup T \neq \varnothing$

$I(t_i)$ is the set of all input places of transition $t_i$

$O(t_i)$ is the set of all output places of transition $t_i$

Table 3.2: Example PN structure

| **Component** | | | | |
|---|---|---|---|---|
| P | = {p1, p2, p3, p4, p5} | | | |
| T | = {t1, t2, t3} | | | |
| I (t1) | = {p1} | | | $\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}$ |
| I (t2) | = {p2} | or I | = | |
| I (t3) | = {p3, p4} | | | |
| O (t1) | = {p3} | | | $\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$ |
| O (t2) | = {p4} | or O | = | |
| O (t3) | = {p5} | | | |

Input places $I$ and output places $O$ can be expressed either in basic set notation or as a matrix. The set notation lists all the input and output places for a given transition $t_i$, whereas the matrix notation has a row for each transition and a column for each place and each element is either a 1 to indicate that a place is an input or output of transition $t_i$ or a 0 indicating otherwise. An example for a specific PN is given in Table 3.2.

Tokens are assigned to places in the PN in order to indicate the state(s) or condition(s) of the system. The location of the tokens is represented via markings. A marking $\mu$ of a PN is defined as follows:

$$\mu = P \rightarrow N \tag{3.22}$$

where $N$ is the set of natural numbers $\{0, 1, 2, ...\}$. Essentially, the marking is a vector with an element for each place $p_i$ to indicate the number of tokens in that place.

PNs may be given along with an initial marking $\mu_0$ to form a 5-tuple [81]:

$$M = (C, \mu_0) \tag{3.23}$$

The firing of transitions is denoted via the next-state function $\delta$ where $\mu_{j+1} = \delta(\mu_j, t_i)$. A sequence of transition firings is denoted by $\sigma = t_i...t_n$. Computationally, $\mu_{j+1}$ may be obtained via the following:

$$\mu_{j+1} = \delta(\mu_j, t_i) \tag{3.24}$$

$$= \mu_j - I(t_i, :) + O(t_i, :) \tag{3.25}$$

where $I(t_i, :)$ is the $i^{th}$ row vector from the matrix $I$ and $O(t_i, :)$ is the $i^{th}$ row vector from the matrix $O$. This computation essentially moves tokens from the input places of a transition to the output places of the transition. It should be noted that the existence

Table 3.3: Example PN markings

| Transition Fired | Resulting Marking |
|---|---|
| - | $\mu_0 = (1, 1, 0, 0, 0)$ |
| $t_1$ | $\mu_1 = (0, 1, 1, 0, 0)$ |
| $t_2$ | $\mu_2 = (0, 0, 1, 1, 0)$ |
| $t_3$ | $\mu_3 = (0, 0, 0, 0, 1)$ |

of a valid $\mu_{j+1}$ computed using the above equation does not indicate that $t_i$ was actually enabled, as, in the case of self-loops in the PN, $O(t_i, p_j)$ would add back what $I(t_i, p_j)$ has subtracted, but this does not indicate that there was a token in $t_i$ to begin with [82]. Thus, separate checks for the existence of tokens in each input place must be implemented to determine if a transition is enabled. Table 3.3 shows an example for a sequence of transition firings and their associated markings, given an initial marking for the example PN from Table 3.2.

### 3.4.2 Petri Net Graphs

The visual representation of a PN consists of a directed bipartite graph, i.e. a graph with two types of vertices (circles for places and rectangles for transitions) where one type of vertex (place or transition) is always connected to another type of vertex (transition or place) via a directed edge. Tokens are represented by solid circles placed on top of a place. A transition is enabled when all its input places have a token in it and once a transition fires, one token for each input edge is removed from the input places and a token for each output edge is placed in the output places. Figure 3.1 shows the graph version of the example PN from from Table 3.2 and Figure 3.2 shows the addition of the initial marking $\mu_0$ from Table 3.3 to the same PN.

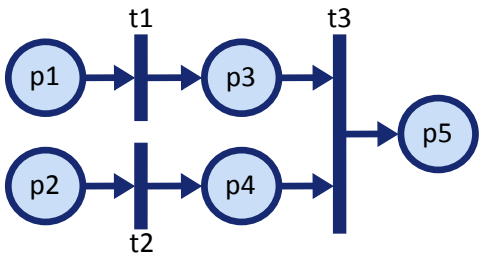



Figure 3.1: Example PN graph

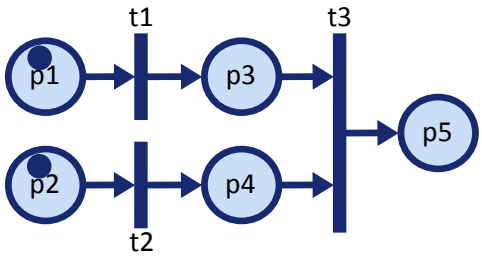



Figure 3.2: Example PN graph with marking

Because PN structures and PN graphs represent the same information, the remainder of the thesis will simply use the term PN to refer to both representations and the term *structure* or *graph* is only used when explicitly distinguishing between the two representations.

### 3.4.3 Reachability

The reachability problem in PNs pertains to whether a particular marking of the PN can be attained given an initial marking [71], [81]. Reachability in bounded PNs is determined via the construction of a reachability tree [71]. Since the work presented in this thesis only deals with bounded PNs, the discussion about reachability is limited to the use of reachability trees for bounded PNs.

The root node of the reachability tree is the initial marking. Each branch of the reachability tree is labeled by a corresponding, enabled transition which will lead to another node representing the resulting marking following the firing of the transition. The leaf nodes of the tree consist of the markings after which there are no more enabled transitions (terminal nodes) and markings which are the same as the marking of a previous node in the tree (repeated node). Care must be taken to follow only enabled transitions when constructing the reachability tree. As previously noted, the existence of a valid resulting marking $\mu_{j+1}$ does not guarantee that the transition leading to this resulting marking was enabled.

Figure 3.3 shows the reachability tree of the example PN from from Table 3.2 and Figure 3.1 given the initial marking $\mu_0$ from Table 3.3 and Figure 3.2. Note that the root node is the initial marking $\mu_0 = (1, 1, 0, 0, 0)$ and the two leaf nodes represent a terminal node $(0, 0, 0, 0, 1)$ and a repeated node $(0, 0, 1, 1, 0)$. There is no need to continue constructing the reachability tree after a repeated node, as the resulting branches already exist. The

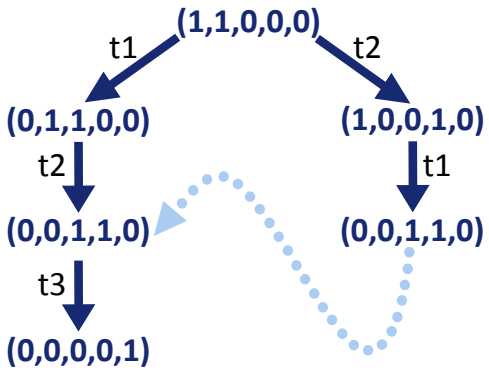



Figure 3.3: Example PN reachability tree (dotted line connects repeated nodes)

repeated node simply has to provide a link to the first occurrence of a node with the same marking.

## 3.5 Summary

DMPs model motions using a second order differential equation with an added nonlinear term. Once trained on a motion trajectory, DMPs can generate motions to different goal positions, which is desirable for task based motion learning. The weight parameters of the DMP may be used as features for motion recognition. AP is an efficient, exemplar based clustering algorithm. It uses simple iterative computations to determine cluster membership and the best representative exemplar for a cluster. Furthermore, SOMs are a two layer ANN that can consistently match an input pattern to one winning output node. Lastly, PNs can be used to model processes using places, which represent the states or conditions of a system, and transitions, which represent the actions or events of a system. A reachability tree can be constructed given a PN to reflect all the possible transitions following a given initial set of states, i.e. marking. Using this reachability tree, a path can be constructed between the given initial state and a desired final state of the system. In the following chapters, the DMP motion modeling described in this chapter will be applied to demonstrator data and clustered using the AP algorithm, while SOMs will be used to model objects and object states. Subsequently, PNs will be used to build a task model consisting of sequences of actions and objects states.

# Chapter 4

# Motion Learning[1]

The learning by imitation system proposed in this thesis includes a task model involving both actions and objects. Knowledge of actions is necessary for the robot to imitate tasks that it has learned from human demonstration. An action consists of a motion trajectory representing one or more motion primitives, which are atomic units of meaningful motion segments [86]. Learning actions corresponds to learning motion primitives, as the sequence of motion primitives can be stored in the task model to later recreate the actions to imitate the task. This is an efficient approach to learning actions since motion primitives that appear as part of a number of different actions do not have to be relearned each time, as long as the system can determine via motion recognition if a motion primitive has already been learned.

This chapter will cover the motion recognition component in the learning by imitation system as highlighted in Figure 4.1, including motion segmentation, modeling and recognition. The main focus is on the motion modeling and recognition, where DMPs are used for motion modeling and AP clustering is used for motion recognition. By clustering the DMP weight parameters, similar motions will be grouped into the same cluster. Furthermore, AP will identify a representative exemplar for each motion cluster, leading to a minimal representation of the observed motions that can also be used for motion generation. Experiments using a small and large motion data set verify the proposed approach and show that it is capable of making a coarse or fine distinction between the different types of motions in the data sets.

[1]A version of this chapter has been accepted for publication [18]: G. J. Chang and D. Kulić. Motion learning from observation using affinity propagation clustering. In *22nd International Symposium on Robot and Human Interactive Communication (RO-MAN)*, 2013, to be published.

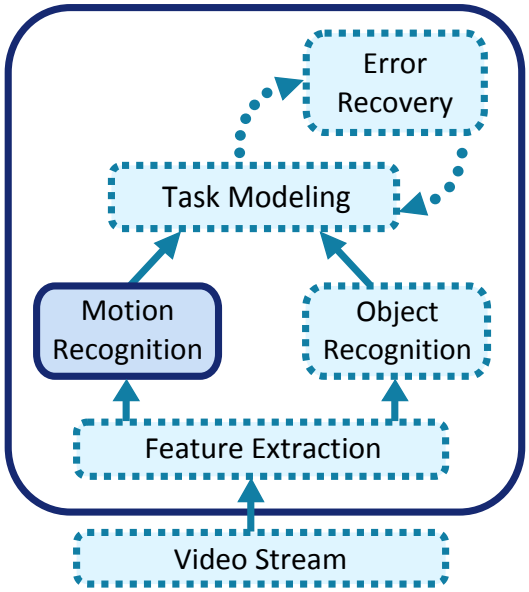



Figure 4.1: The motion recognition component in the proposed learning by imitation system

## 4.1 Motion Segmentation

A simple technique similar to zero-velocity crossings (ZVCs) is used to determine the start and end of a motion segment from continuous observation data. In ZVC [30], the velocities of a motion trajectory are used to determine zero velocity crossings, where the motion changes direction. These points are potential segmentation points.

The simple technique implemented for this thesis places a segment point whenever the displacement in one of the observed DoF switches from non-moving (zero displacement) to moving (non-zero displacement) or vice versa. This motion segmentation strategy is capable of breaking a motion trajectory into meaningful segments when one type of motion switches to another with a transition between stopping and moving or moving and stopping. To avoid extraneous segment points due to noise in the extracted motion trajectories, instead of defining non-moving as exactly zero displacement, non-moving is defined as all displacements below a certain threshold. This threshold is a small value, manually tuned depending on the amount of noise in the data and the sampling rate of the data points. The threshold that was used is 5 pixels for switching from non-moving to moving and 4 pixels from switching from moving to non-moving. Additionally, at least one frame of video

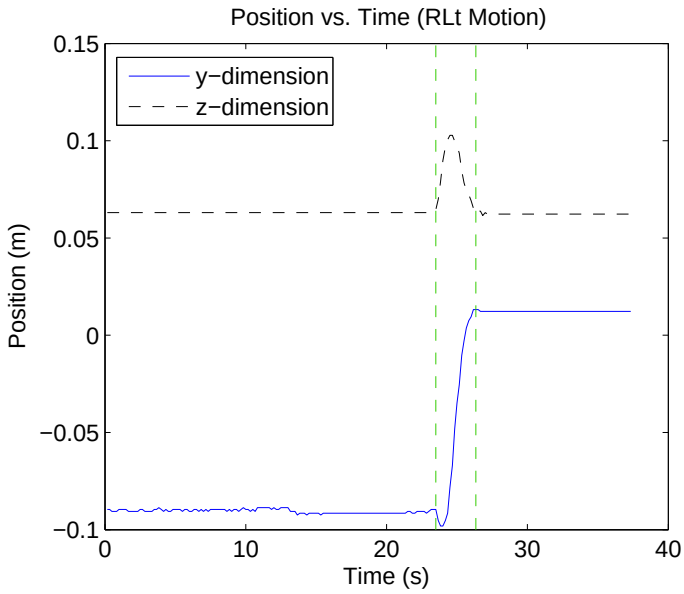



Figure 4.2: Segmentation results for right to left motion on top of another object (RLt)

has to elapse between motion segments. Figure 4.2 shows example segmentation points found in motion trajectories of the small motion data set using the proposed segmentation approach.

After motion segmentation is performed, the segments of the motion trajectory which contain no motion are discarded by setting a threshold for the minimum total distance traveled in the motion segment. Thus, only the segments of the motion trajectory that are of interest, i.e. the segments when the demonstrator is actually moving, are used for the motion modeling and clustering. This simple segmentation technique is appropriate for the low DoF motions of the small motion data set and it was chosen as motion segmentation is not the main focus of the motion recognition component of the learning system. A more sophisticated motion segmentation strategy such as [55] can be considered as part of future improvements, if necessary.

## 4.2 Motion Modeling

Motion modeling is achieved via DMPs. The DMP was chosen as the motion model of the learning system as it inherently parameterizes the goal positions of motion trajectories that

it has been trained on while allowing disturbance rejection. Furthermore, by clustering the weight parameters of the DMP, it is possible to achieve motion recognition [41], [42], [68].

For each motion primitive, a separate DMP is trained for each DoF of the motion. If the motions are characterized by Cartesian data, each axis dimension for which a motion trajectory is available is modeled as a separate DMP. If the motions are characterized by joint angle data, each joint for which motion trajectory data is available is modeled as a separate DMP. The weight values of each DMP for each DoF $w_{DoFj} = [w_1 w_2 ... w_N]$ are stacked together in one feature vector $w = [w_{DoF1} w_{DoF2} ... w_{DoFM}]'$. The $w$ feature vectors for all the motion primitives serve as the data points for clustering and are arranged into a weight matrix where the number of rows represents the total number of weights for each motion primitive, i.e. the number of features that will be used for clustering, while the number of columns represents the number of separate motion primitive, i.e. the number of data points that will be used for clustering. As motions of the same type have similar trajectories which result in similar weight $w_i$ values, clustering the feature vectors will group similar motions into the same cluster, effectively performing motion recognition.

## 4.3 Motion Recognition

Once the DMP models have been trained on motion primitive trajectories, the AP clustering algorithm is run to identify clusters of similar motion primitives. For the purposes of the learning by imitation system, identifying which cluster a motion belongs to is considered as recognizing the motion and an exemplar motion of that cluster is then used to represent that motion. AP uses a simple, yet efficient message passing algorithm for exemplar based clustering, i.e. the most representative exemplar to serve as each cluster center is identified along with the clusters. One advantage of the AP algorithm is that, unlike many other clustering algorithms, it does not require the number of expected clusters to be specified beforehand. Though, if available, prior knowledge of the number of clusters can be used to fine-tune the clustering results. Additionally, the exemplar clustering nature of the AP algorithm is very well suited for directly finding the trained motion model that best represents a group of similar motions. Instead of having to store all of the motion models in a motion group, only the exemplar motion model needs to be stored and can be used later to generate kinematically achievable motions for the entire motion group.

The AP clustering algorithm uses the negative Euclidean distance between the weight vectors of separate DMP models as input, i.e. the similarity values, and outputs the cluster that each DMP model belongs to as well as the most representative DMP model for each cluster. In order to compare the motion recognition performance of the DMP weight

parameters through AP clustering, HMMs were also trained on the large motion data set and the negative Kullback-Leibler (KL) distances between the trained HMM models [58] were used as inputs to AP. Comparable motion recognition results were obtained using DMPs and HMMs, as outlined in more detail in the next section.

For the learning system in this thesis, it is sufficient to run the AP algorithm offline each time a new motion trajectory is observed, due to the small number of motion trajectories and the efficiency of the AP algorithm. However, there are proposed adaptations of the AP algorithm that allow data streams to be clustered incrementally [96], which may be considered for future improvements of the learning system, as incremental algorithms are preferable for continuous learning in a dynamic environment [10], [56] and the proposed motion segmentation and modeling methods can already be run incrementally.

## 4.4 Experiments

The entire motion recognition system is verified experimentally on two motion data sets:

1. A **small motion data set** consisting of four different motion primitives with six exemplar motions each, resulting in a total of 24 motion exemplars. Each motion exemplar consists of 2 DoF Cartesian coordinate data extracted from video stream data using simple computer vision techniques. A separate DMP model is trained for each DoF.

2. A **large motion data set** consisting of nine different motion primitives with a number of exemplar motions each (on average 15 motion exemplars were captured for each motion primitive), resulting in a total of 137 motion exemplars. Each motion exemplar consists of 20 DoF joint angle data captured using motion capture equipment. A separate DMP model is trained for each DoF.

The small motion data set contains motions that form a part of the tasks that the task modeling portion of the learning system is tested on in the next chapter. The motion trajectories are extracted together with the object features and the computer vision techniques used for extracting the motion trajectories are described in detail with the object feature extraction in Appendix B. The extracted motion trajectories for the small motion data set are a continuous representation of the positions of the objects in the task and must first be segmented into motion primitives. The large motion data set on the other hand is obtained from [58] and has already been manually segmented. The large motion data set contains

full body human motions which are not used for the task learning system in the next chapter, as the proposed learning by imitation system does not track the pose of the human demonstrator but rather the objects and their positions in the environment. The purpose of including the large motion data set in the experiments is to illustrate the generalizability of the proposed motion recognition approach to other data sets. Moreover, comparisons of motion recognition results using the proposed DMP and AP clustering combination with HMM and AP clustering are performed on the large motion data set to further verify the validity of the proposed approach. Comparisons between AP and other commonly used clustering algorithms are also performed for both the small and large motion data set.

### 4.4.1 Motion Model Implementation

Before describing the motion recognition results for the small and large motion data set, some details about the DMP formulation need to be outlined. Details on how DMP parameters affect the motion recognition results are not available through existing DMP literature and were determined experimentally. This section outlines the main findings of the factors that affected motion recognition. Appendix A provides further insight into the detailed implementation and tuning of the parameter values of the DMP motion models for motion recognition.

For the small motion data set, the motion trajectory data $y$ contained very little noise due to the nature of the raw video data and no pre-processing was necessary. For the large motion data set several issues with the position data $y$ that can later affect motion recognition were noted:

1. **Noise:** due to the nature of the motion capture data set, there was a lot of high frequency noise in the data.

2. **Angle switches:** where joint angle values switch by 360° between consecutive data points. The cause of these joint angle switches is due to occlusion of markers during motion capture. When markers are occluded, no information for updating joint angle values is available and the joint angles are kept at their previous value, however, once the marker becomes visible again, the algorithm for computing the joint angle values from the marker positions recommences without considering the previous joint angle value, allowing rotations of 360° in between joint angle data points.

3. **Timing variability:** where the exact duration and onset of movements between different exemplars of the same motion type is offset because of human limitation in exactly repeating the same type of motion.

Through experimentation using the large motion data set it was found that shifting angles into the range of $-180°$ to $180°$ is sufficient for motion recognition, as the DMP model is able to handle noisy position data, though there is some susceptibility to noise especially towards the end of the motion trajectory. This is resolved by shifting the last Gaussian basis function slightly away from the very last data point in the motion trajectory to avoid overfitting the weight $w_i$ value that scales the Gaussian to the noise. This is further discussed in Appendix A, Section A.6.

Thus, a pre-processing step of shifting angles is applied to the large motion data set before training the DMP model. The timing variability is not a problem if an appropriate number of Gaussian basis functions is selected. The number of Gaussian basis functions affect motion recognition. If too many Gaussian basis functions are used, the differences in onsets of certain features of the movement will be captured by different (neighboring) Gaussian basis functions. Because the Gaussian basis functions are scaled by the weight values $w_i$, a different sequence of weight $w_i$ values will be obtained due to the aforementioned offsets which can cause incorrect AP clustering results. If too few Gaussian basis functions are used, the nonlinearities in the motion trajectory cannot be fully captured and recreated accurately by the DMP. Further discussion of the number of Gaussian basis functions can also be found in Appendix A, Section A.6. Future improvements can consider landmark alignment to address the timing variability problem without having to excessively tune the number of Gaussian basis functions.

Some issues were encountered in motion recognition for the small motion data set, but not the large motion data set, when using a constant value for the goal $g$. Because the large motion data set is quite noisy, the DMP model generated trajectory does not or rather cannot closely follow the acceleration profile of the exemplar motion trajectory due to the high frequency noise components in it. The small motion data set on the other hand is not very noisy, but the acceleration profile is not perfectly smooth, i.e. it did not appear to have high frequency noise components, but may have contained lower frequency noise causing variabilities such as non-zero initial acceleration values. The DMP model generated trajectory follows this acceleration profile fairly closely and it is not possible to control the initial acceleration using the standard DMP formulation [73] to ensure a continuous acceleration profile. However, by adding an extension to the DMP which changes the DMP formulation to a third order system, control can be obtained over the initial acceleration values [87], [73]. This extended DMP formulation is adapted from [87], [73]:

$$\tau\dot{z} = \alpha_z(\beta_z(r - y) - z), \quad z = \tau\dot{y} - f, \quad \tau\dot{r} = \alpha_g(g - r) \tag{4.1}$$

where $r$ is a state variable that models the approach to the goal position $g$ as a first order differential equation and $\alpha_g$ is a time constant. This formulation of the goal $g$ ensures a

continuous acceleration profile [87]. The implications of this extension and the tuning of the $\alpha_g$ constant value is described in further detail in A, Section A.4.

To train this new, third order dynamical system formulation of the DMP, the same steps as described in the previous chapter, Section 3.1, are applied with the exception that the goal state $r$ now needs to be computed in the beginning alongside gating term $v$ and state $x$:

$$\dot{r} = \frac{\alpha_g}{\tau}(g - r) \tag{4.2}$$

Again, by repeatedly computing the goal state $r$ derivatives, integration can be used to find the goal state $r$ values over time, given the initial value $r_0 = y_0$. Similar to (3.9), once the goal state $r$ is known, the nonlinear component $f$ can be computed:

$$f = \frac{\tau^2 \ddot{y}}{\alpha_z} + \tau \dot{y} - \beta_z(r - y) \tag{4.3}$$

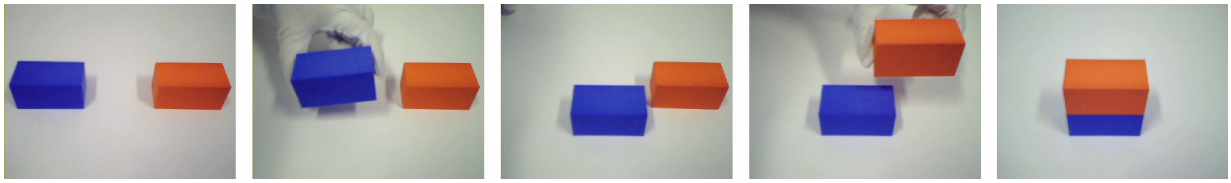
The remaining steps are the same as outlined for the standard DMP formulation described in the previous chapter, Section 3.1.

The above mentioned pre-processing steps and the extension of the DMP formulation were used to obtain the motion recognition results described in the following sections.

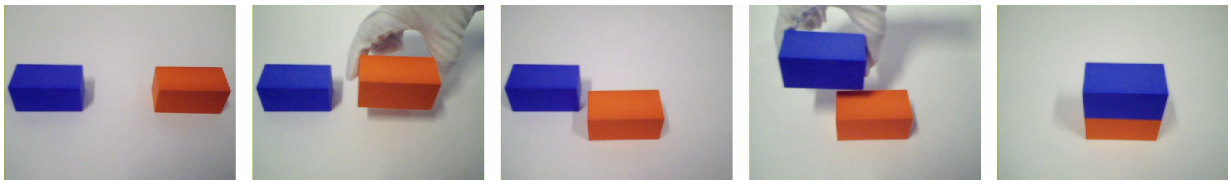
### 4.4.2 Small Motion Data Set

The small motion data set consists of 2 dimensional Cartesian coordinate positions of two primary colored blocks being stacked. The goal is for the motion recognition process to extract the four groups of motions LR (left to right motion), RLt (right to left motion on top of another block), RL (right to left motion), LRt (left to right motion on top of another block) as shown in Figure 4.3. The motions are demonstrated six times by a human and captured as video stream data by the NAO humanoid robot [4]. To properly observe the block stacking, the NAO remains stationary during video capture. The experimental setup is shown in Figure 4.4. The trajectories of the motions are extracted from the video stream data via simple computer vision techniques described in Appendix B and transformed from the (x,y) camera frame to (y,z) robot frame. Each extracted motion trajectory will contain an LR and RLt or a RL and LRt motion pair, which are segmented using the procedure described in Section 4.1.

There are a total of 24 resulting, segmented motion primitive exemplars: six instances of each of the four groups of motion were used because there are six different possible color combinations for the blocks for each group of motion, i.e. there are three ways of choosing



(a) Moving the left block (LR) then the right block on top (RLt)



(b) Moving the right block (RL) and then the left block on top (LRt)

Figure 4.3: The four possible groups of motions (LR, RLt, RL and LRt) when two blocks are stacked

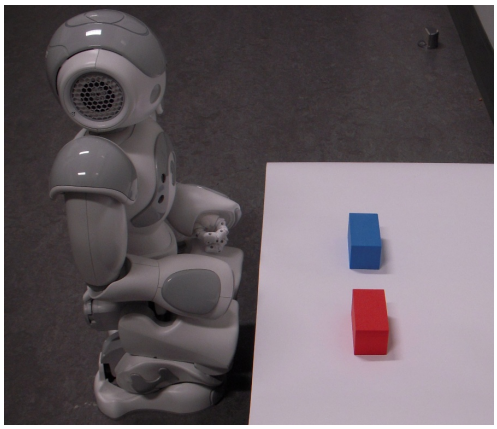


Figure 4.4: The experimental setup for capturing videos using the NAO robot

two differently colored blocks out of three possible primary block colors and two ways of arranging them. Each motion exemplar has a dimensionality of 20, consisting of 10 DMP weights for the y-dimension motion segment and 10 DMP weights for the z-dimension motion segment. The negative Euclidean distances between the sets of 20 DMP weights per motion exemplar are used as inputs to the AP clustering algorithm.

First, the AP clustering algorithm is run with different preference input values to determine the resulting number of clusters. As can be seen in Figure 4.5, by using -12.95, the recommended median of all similarities, as the preference value, only two clusters are

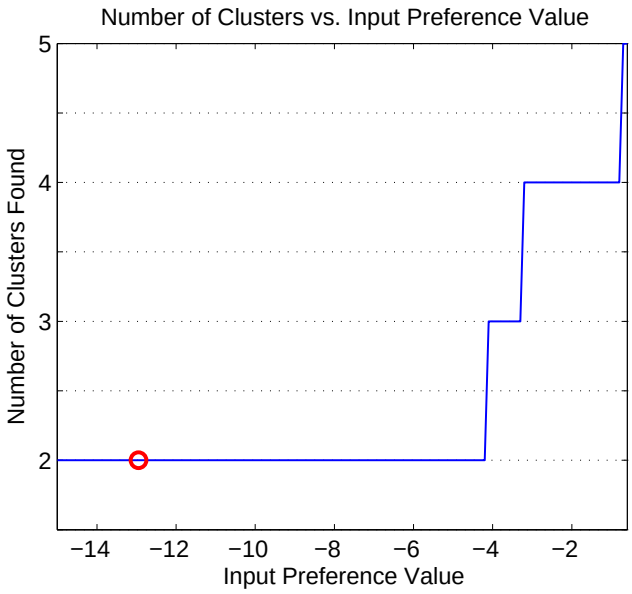



Figure 4.5: Input preference values and the resulting number of clusters for the small motion data set using DMP weights and AP (the circle marks the median of all similarities)

obtained. These two clusters cleanly divide the motions into those that move from left to right (LR and LRt) and those that move from right to left (RL and RLt). In fact, the result of two clusters can be obtained for a wide range of preference values (-170 to -4.2 sampled at step size 0.1). In order to obtain the more fine-grained division of four motion groups, the preference value must be increased to the range of -3.2 to -0.8 sampled at step size 0.1. Any preference value in this range will work for obtaining the four desired motion groups, which indicates the AP clustering results are stable for this range of preference values.

Additionally, the AP algorithm was run first on a smaller set of only four randomly selected instances of each of the four groups of motion and then again on the full data set. All motions were clustered correctly for both the smaller set and the full data set, although the representative exemplar for a cluster may change as new motions are added. This illustrates that the AP algorithm is able to robustly cluster motions even when new motions are added to the data set.

The confusion matrix of the resulting clustering of the full data set is shown in Table 4.1, where the column headings indicate the true class labels and the first set of rows indicate the clusters found by AP. All values are probabilities. As can be seen from Table

Table 4.1: Small motion data set clustering confusion matrix

| **Algorithm** | **Clusters** | **True Class Labels** | | | |
|---|---|---|---|---|---|
| | | **LR** | **LRt** | **RL** | **RLt** |
| **AP** | **LR** | **1** | 0 | 0 | 0 |
| (p = -2) | **LRt** | 0 | **1** | 0 | 0 |
| | **RL** | 0 | 0 | **1** | 0 |
| | **RLt** | 0 | 0 | 0 | **1** |
| **HAC** | **LR** | **1** | **1** | 0 | 0 |
| (single) | **RL** | 0 | 0 | **1** | 0 |
| | **RLt1** | 0 | 0 | 0 | **0.833** |
| | **RLt2** | 0 | 0 | 0 | **0.167** |
| **HAC** | **LR** | **1** | 0 | 0 | 0 |
| (complete, average, | **LRt** | 0 | **1** | 0 | 0 |
| minimum-variance) | **RL** | 0 | 0 | **1** | 0 |
| | **RLt** | 0 | 0 | 0 | **1** |
| **K-means** | **LR** | **1** | **0.167** | 0 | 0 |
| (random, uniform) | **LRt** | 0 | **0.833** | 0 | 0 |
| | **RL** | 0 | 0 | **1** | 0 |
| | **RLt** | 0 | 0 | 0 | **1** |
| **K-medoids** | **LR** | **1** | 0 | 0 | 0 |
| | **LRt** | 0 | **1** | 0 | 0 |
| | **RL** | 0 | 0 | **1** | 0 |
| | **RLt** | 0 | 0 | 0 | **1** |

4.1, the LR, LRt, RL and RLt motions were all clustered correctly by the AP algorithm.

Additional clustering results using hierarchical agglomerative clustering (HAC), K-means clustering and K-medoids clustering are provided in Table 4.1. The HAC algorithm places each data point into its own cluster and then recursively merges them until a stopping condition is reached, resulting in a hierarchy of clusters [45] from which the desired number of clusters must be selected. The single-link and complete-link variations of HAC are most commonly used [45], however, the more computationally expensive average-link and minimum-variance may provide more robust results. The K-means and related K-medoids algorithms follow a divisive approach and recursively recompute cluster centers and memberships, where the K-means uses the mathematical mean as the cluster center and the K-medoids, similar to the AP algorithm, uses a data point most representative of the cluster as the center. Both the K-means and K-medoids algorithms require the number

of clusters to be known beforehand and, as they are not guaranteed to converge to a global optimum, multiple runs must be made to obtain good results. In order to obtain comparative results with the AP algorithm, the Euclidean distance was used as the distance metric for all the comparison clustering algorithms.

All the comparison clustering algorithms, like AP, had no trouble dividing motions into those that move from left to right (LR and LRt) and those that move from right to left (RL and RLt). However, the HAC single-link approach was unable to divide the left to right motions into the respective LR and LRt clusters (i.e. at no point in the cluster hierarchy were LR and LRt cleanly placed into two clusters), instead the RLt motion was divided into two clusters, RLt1 and RLt2, as shown in Table 4.1. The RLt2 cluster only contains one RLt motion exemplar. This outlier exemplar had a slightly different y-axis trajectory: its y-axis trajectory starts off slower and then speeds up faster than the y-axis trajectories of the other RLt exemplars. This difference was enough to cause the minimum distance (i.e. the criteria used by single-link HAC) between the outlier RLt and the other RLt exemplars to be larger than the minimum distance between the LR and LRt exemplars, leading to the incorrect clustering result. As none of the other clustering algorithms use the minimum distance, they were not affected by this outlier.

Similarly, the K-means was unable to cleanly divide the left to right motions, resulting in one LRt exemplar being placed into the LR cluster. This mistake persisted even when the best result from 1000 runs of K-means was picked. The misclassified LRt exemplar had a slightly different z-axis trajectory, where the highest point in the trajectory was more pronounced than in the other LRt z-axis trajectories. Though the overall shapes and goal positions of the LRt z-axis trajectories were very similar, the difference was enough to cause the outlier LRt to be closer to the mean found for the LR motion cluster than the mean of the LRt motion cluster for the K-means algorithm. The remaining HAC algorithms and K-medoids were able to cluster the motions correctly.

The representative exemplars picked out by K-medoids for both the two and four cluster cases are the same as those picked out by the AP algorithm. To visualize the representative exemplars found, Figure 4.6 shows all the motion segments and the exemplar found by the AP and K-medoids algorithm to represent the LR cluster. The exemplar motion segment lies in between the other motion segments of the cluster and can be seen as a relatively good representation for the cluster.

The motion generation capabilities of the representative exemplars for the four different clusters found are shown in Figures 4.7 and 4.8 for the left to right and right to left motions, respectively. The trained DMP of the representative exemplar was used to generate a motion trajectory that was executed on the NAO robot using the same goal positions as

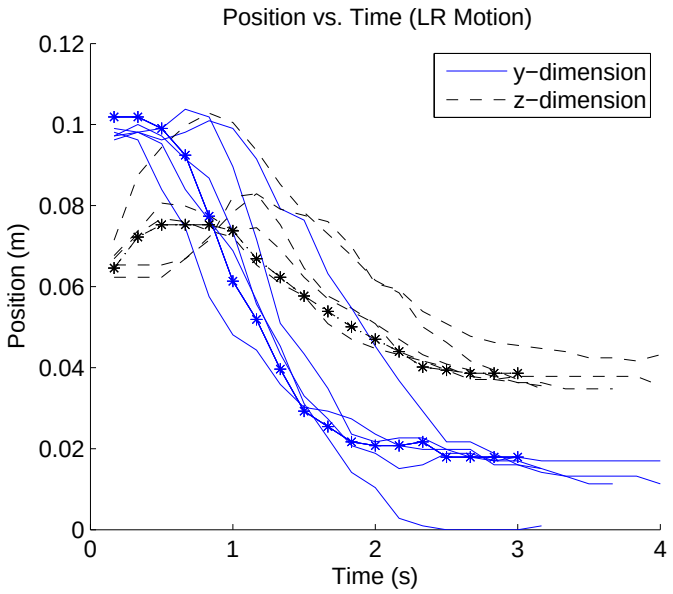
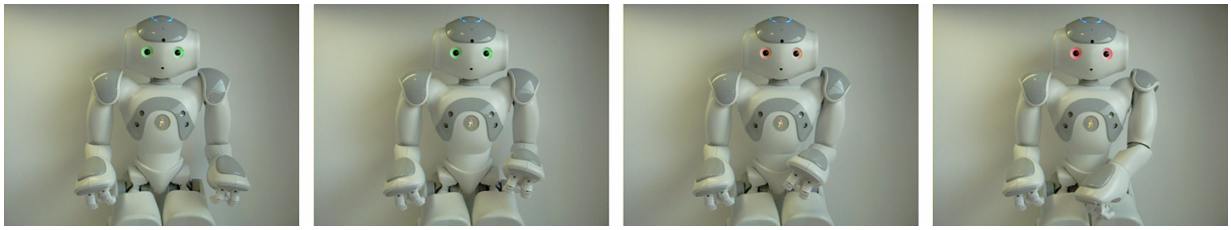



Figure 4.6: Example motion segments in the LR cluster (asterisks mark the exemplar motion found by AP and K-medoids)

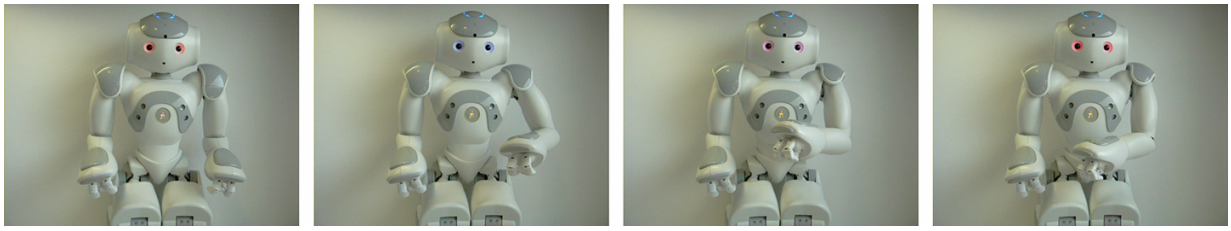
encountered in the trained trajectories. As can be seen in Figures 4.7 and 4.8, the LRt and RLt motions have a motion trajectory with a higher final position than the LR and RL motions.

### 4.4.3 Large Motion Data Set

The motion recognition system was further validated on a larger, existing motion capture data set provided in [58]. This data set consists of 9 motion groups performed by a single demonstrator (a total of 137 exemplars): walking (WA, 28 exemplars), cheering (CH, 15 exemplars), dancing (DA, 7 exemplars), kicking (KI, 19 exemplars), punching (PU, 14 exemplars), sumo leg raising (SU, 13 exemplars), squatting (SQ, 13 exemplars), throwing (TH, 13 exemplars) and bowing (BO, 15 exemplars). These full body human motions are not used for task learning in the following chapter, but their clustering result illustrates the suitability of the proposed motion recognition approach for large DoF motion data. The motion exemplars consist of time series joint angle data and are already segmented, thus, feature extraction or segmentation is not necessary for this data set. Each motion exemplar in the large motion data set consists of the joint angle data of a 20 DoF humanoid model captured at 33 frames per second. An additional feature that distinguishes this data set

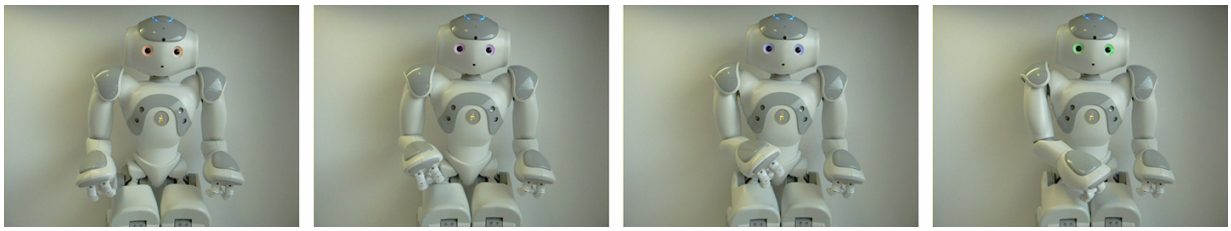


(a) LR motion

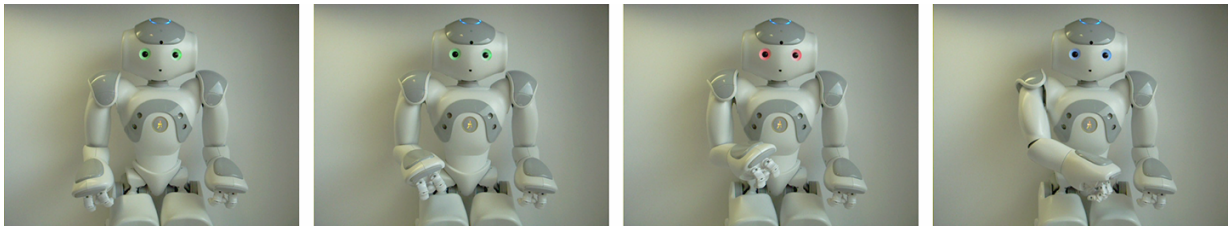


(b) LRt motion

Figure 4.7: NAO robot performing the left to right motions (LR and LRt)



(a) RL motion



(b) RLt motion

Figure 4.8: NAO robot performing the right to left motions (RL and RLt)

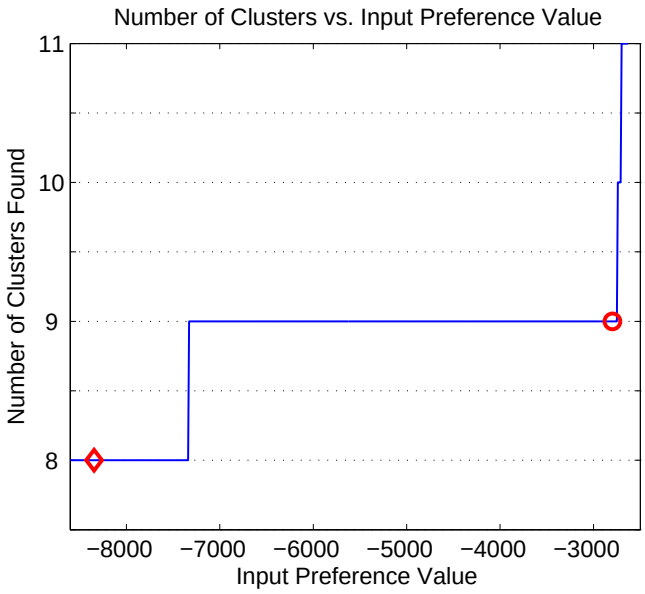



Figure 4.9: Input preference values and the resulting number of clusters for the large motion data set using DMP weights and AP (the diamond marks the minimum of all similarities and the circle marks the median of all similarities)

from the video stream data is that significantly more high frequency measurement noise was present in the data.

The motions were modeled using DMPs, which were able to handle the high frequency measurement noise in the data without additional pre-processing. A separate DMP with 5 weights was used to model each of the 20 DoF for each motion exemplar, resulting in a dimensionality of 100 for each motion exemplar. The negative Euclidean distances between DMP weight vectors were used as inputs to the AP clustering algorithm.

First, the AP clustering algorithm was run with different preference input values to determine the resulting number of clusters. As can be seen in Figure 4.9, when the preference value is set to the recommended median of all similarities, exactly 9 clusters are obtained. When the minimum of all similarities is used as the preference value, 8 clusters are obtained. A range of preference values around the recommended median and minimum will lead to the 9 and 8 cluster cases, respectively, indicating that the AP clustering results are stable. The longer "plateau" of the 9 cluster case can be seen as an indication that this is the more stable result which also corresponds with the true classes.

As in the small motion data set case, the AP algorithm was run with only about 70%

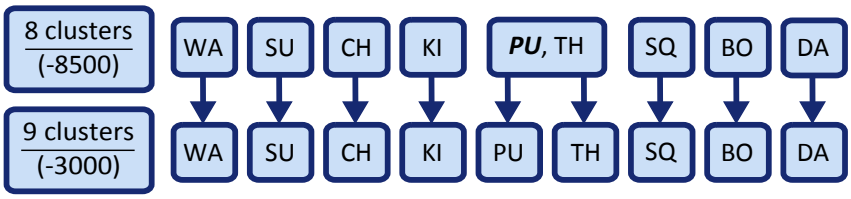



Figure 4.10: Clustering results of the large motion data set using DMPs of 5 weights and two preference values for the AP algorithm. The total number of clusters found is indicated at the beginning of each row (including the preference value used in parentheses). The types of motions contained in each cluster are indicated through their respective labels, where WA = walking, SU = sumo leg raising, CH = cheering, KI = kicking, PU = punching, SQ = squatting, TH = throwing, BO = bowing, DA = dancing. If one cluster contains more than one type of motion, the type of motion of the representative exemplar for the cluster is italicized.

of the motion exemplars randomly selected from each motion group followed by the full data set. Again, no clustering errors were observed though the representative exemplar of a cluster may change as new motions are added. This illustrates the robustness and adaptability of the AP algorithm in the case of new motions being observed.

The results of the AP algorithm clustering the full data set are shown in Figure 4.10. At 8 clusters, the punching (PU) and throwing (TH) motions are placed into the same cluster with a punching (PU) motion as the representative exemplar. At 9 clusters, punching (PU) and throwing (TH) are correctly placed into their own respective clusters, thereby resulting in the 9 motion groups expected from the data set. All the data points are assigned to clusters matching their true classes.

The grouping of throwing (TH) and punching (PU) is reasonable due to the similarity between those motions. This is also in accordance with the findings in [58]. These results indicate that the DMP weights can indeed be used as features for motion recognition and that the AP algorithm is capable of robustly clustering large motion data sets of motion.

The motion library was next clustered using HMM models. A separate 5 state HMM model is trained on each motion exemplar of the large motion data set. The negative KL distance between each trained HMM motion model is used as the distance metric for the similarity matrix used as input to the AP algorithm. Once again, the AP algorithm is first run with different preference values. As can be seen in Figure 4.11, very similar to the DMP case, by using the recommended median of all similarities as the preference value, exactly 9 clusters are obtained. When the minimum of all similarities is used as the

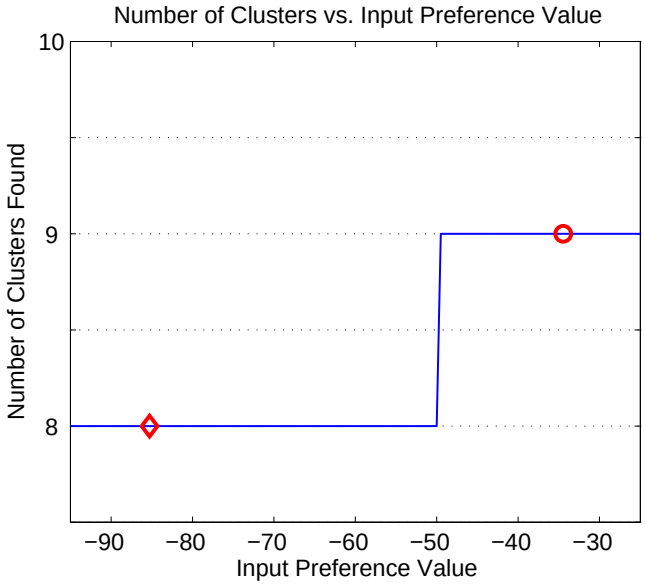



Figure 4.11: Input preference values and the resulting number of clusters for the large motion data set using HMM negative KL distances and AP (the diamond marks the minimum of all similarities and the circle marks the median of all similarities)

preference value, 8 clusters are obtained. Again, a range of preference values around the recommended median and minimum will lead to the 9 and 8 cluster cases, respectively, indicating that the AP clustering results are stable.

The results of the AP algorithm clustering the full data set trained on HMMs are shown in Figure 4.12. At 8 clusters, very similar to the DMP model clustering results, punching (PU) and throwing (TH) are placed into the same cluster, with a punching (PU) motion as the representative exemplar. At 9 clusters all motions are clustered correctly into their respective clusters. Though the DMP and HMM model clustering results are very similar, only the bowing (BO) and dancing (DA) representative exemplars are the same between the DMP and HMM model clustering results. For all other motion groups, the AP algorithm selected different representative exemplars when using the HMM model then when using the DMP model. This is likely due to the differences between the distance metrics used for the two modeling strategies.

The comparison clustering algorithms were also applied to the large motion data set. None of the variations of HAC were able to distinguish the 9 desired clusters using the DMP weights as the features. The minimum-variance method fared best and the resulting

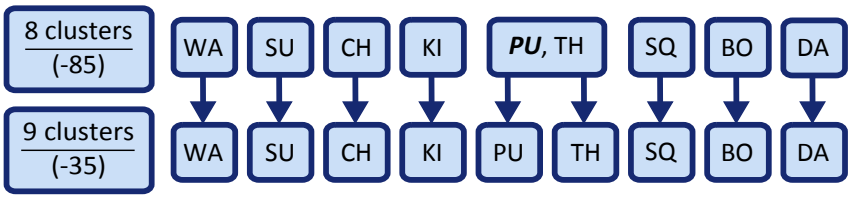



Figure 4.12: Clustering results of the large motion data set using HMMs with 5 states and two preference values for the AP algorithm. The total number of clusters found is indicated at the beginning of each row (including the preference value used in parentheses). The types of motions contained in each cluster are indicated through their respective labels, where WA = walking, SU = sumo leg raising, CH = cheering, KI = kicking, PU = punching, SQ = squatting, TH = throwing, BO = bowing, DA = dancing. If one cluster contains more than one type of motion, the type of motion of the representative exemplar for the cluster is italicized.

dendrogram can be seen in Figure 4.13.

As shown in Figure 4.13, the 8 cluster case of the HAC minimum-variance method is very similar to the AP results. However, at 9 clusters, the bowing (BO) motion was split into two clusters, while the punching (PU) and throwing (TH) motions were still in one cluster and only at 10 clusters are punching (PU) and throwing (TH) placed into their separate clusters. Thus, HAC minimum-variance never fully attains the 9 expected clusters for the large motion data set. The HAC single-, complete- and average-link performed even more poorly, with bowing (BO), kicking (KI) and cheering (CH) splitting before punching (PU) and throwing (TH) are separated. The reason why bowing (BO), kicking (KI) and cheering (CH) were placed into separate clusters may be due to the variability and potential outliers in these motions to which the HAC algorithm is sensitive.

Similar behavior is observed when clustering HMM negative KL distances using HAC single-, complete- and average-link. At 8 clusters, all HAC variations group punching (PU) and throwing (TH) into the same cluster. At 9 clusters, the kicking (KI) motion is separated into two clusters, while punching (PU) and throwing (TH) are still in one cluster. This result agrees with the findings in [58], as kicking (KI) was found to tend to be split into multiple clusters, possibly due to higher variability in the kicking motion. HAC minimum-variance cannot be used with HMM models, as it requires the use of the Euclidean distance as a distance metric.

K-means and K-medoids, on the other hand, were both successful in finding the same 8 and 9 clusters as AP using the DMP weights. However, K-means required a minimum of

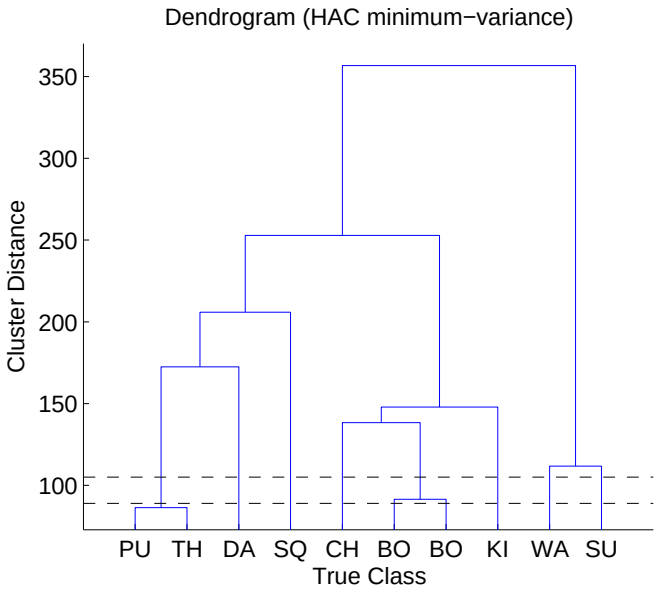
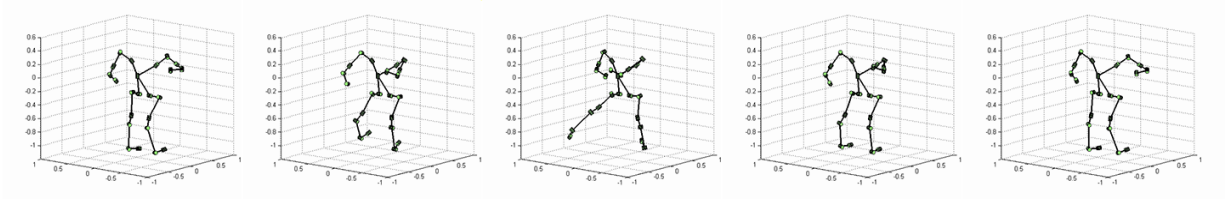


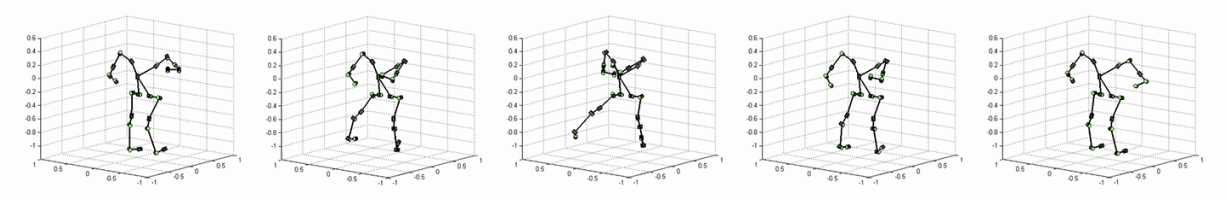

Figure 4.13: Clustering results of the large motion data set using DMPs of 5 weights and the HAC minimum-variance algorithm. The total number of clusters found is indicated by the number of branches intersecting a specific cluster distance value. The 8 and 9 cluster cases are marked with dotted lines. The types of motions contained in each cluster are indicated through their respective labels, where PU = punching, TH = throwing, DA = dancing, SQ = squatting, CH = cheering, BO = bowing, KI = kicking, WA = walking, SU = sumo leg raising.

50 runs and K-medoids required a minimum of 100 runs in order to consistently determine those clusters. K-means performed better for the large motion data set than the small motion data set as the large motion data set contains a much higher number of DoF, which leads to larger Euclidean distances between motion exemplars, which is the distance metric used by K-means to determine clusters. K-medoids again found the exact same representative cluster exemplars as AP. For the HMM model, K-means and K-medoids cannot be used, as they need to compute the distances between features and cannot directly take the negative KL distances as inputs.

Again, the motion generation capability of the representative exemplars was verified using a MATLAB visualization tool of a full-body humanoid model. This visualization tool was created based on the lower-body visualization tool by [46]. The DMP generated motions of all representative exemplars were checked against the captured motion trajectories of the raw motion capture data using this visualization tool and the DMP generated

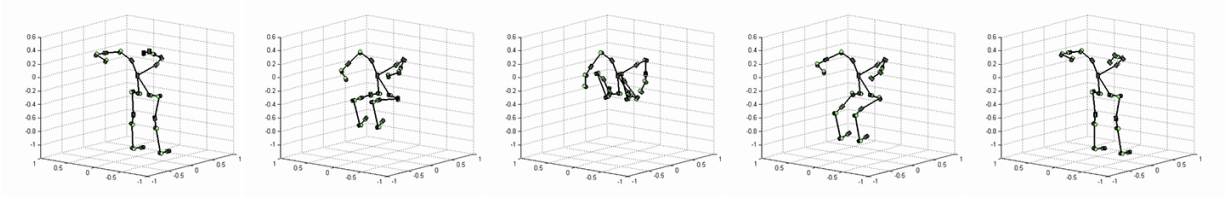


(a) Raw motion capture trajectory

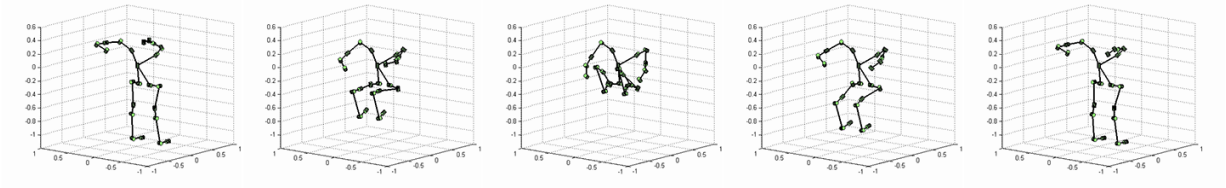


(b) DMP generated trajectory

Figure 4.14: Visualization of the representative kicking (KI) motion



(a) Raw motion capture trajectory



(b) DMP generated trajectory

Figure 4.15: Visualization of the representative squatting (SQ) motion

motions are similar to the captured motion trajectories. Example kick and squat motion visualizations comparing the raw motion capture trajectories with the DMP generated ones are shown in Figures 4.14 and 4.15, respectively. The minor differences in the visualized trajectories may be ascribed to the captured motion trajectories containing high frequency measurement noise which the DMP generated trajectories do not.

It should be noted that for this motion data set, a DMP was trained for the continuous trajectories of each DoF for performing a particular motion (e.g. walking), however, better performances may be obtained by breaking down each motion type into further primitives, e.g. in the case of walking, raising and lowering the right leg may be two motion primitives and raising and lowering the left leg may also be treated as two primitives. This way, there may be a slight increase in the computational overhead both in segmenting the trajectories and in training multiple DMPs for each DoF for each motion exemplar, but the tracking performance would improve due to the reduced amount of nonlinearities in each shorter motion primitive. There should not be any impact on the motion recognition, but this was not verified experimentally.

### 4.4.4 Discussion

The clustering results for both the small and large motion data set indicate that the weights of the DMP can indeed be used as features for motion recognition with just a simple distance metric, e.g. the negative Euclidean distance. The DMP weights can be used to successfully distinguish between different types of motion as well as indicate the amount of similarity between different types of motion, e.g. the punching (PU) and throwing (TH) motions of the large motion data set were recognized as being more similar to each other than other motions.

The experimental results also show that AP can be used for motion recognition. The algorithm can be used without any prior knowledge of the number of clusters required. AP is simple to use since its only parameter, the preference value, can be computed by simply taking the median of the all the input similarity values. This choice of preference value gives meaningful clustering results, as it was able to distinguish the left to right from the right to left motions in the small motion data set and it was directly able to find the 9 clusters matching the 9 true classes for the large motion data set. Other clustering methods, such as K-means and K-medoids, require the number of classes to be known beforehand, while HAC requires selecting the level of hierarchy.

If additional knowledge is available, the preference value of the AP algorithm may be tuned to obtain a coarse or fine division of motions. For the small motion data set, both the two and four cluster cases found using AP contain no clustering errors, while the HAC single-link variation and K-means fail to identify the more finer distinction between motions in the four cluster case. Additionally, AP works for different motion models and distance metrics, i.e. DMP with negative Euclidean distance and HMM with negative KL distance. The HAC minimum-variance, K-means and K-medoids cannot directly be used with the negative KL distance metric for HMMs.

Table 4.2: Clustering algorithm computational complexities

| **Algorithm** | **Computational Complexity** | **Source** |
|---|---|---|
| **AP** | $O(n^2m)$ | [32] |
| **HAC** | $O(n^2 \log n)$ | [45] |
| **K-means** | $O(nkl)$ | [45] |
| **K-medoids** | $O(k(n-k)^2l)$ | [13] |

The theoretical computational complexities of the clustering algorithms are shown in Table 4.2, where $n$ is the number of data points, $m$ is the number of iterations, $k$ is the number of clusters and $l$ is the number of runs. HAC has the highest computational complexity, which is further exacerbated when using the average-link or minimum-variance variations which require more computation at each step. K-means, on the other hand, appears to be the most efficient, but only if the number of runs $l$ that are needed to obtain good clustering results is low. This condition may not always be met. Similarly, AP and K-medoids look comparable in terms of computational complexity, but again this may not hold if the number of runs $l$ that K-medoids requires is high. The number of iterations $m$ for AP were found to be more or less constant for the small and large motion data sets as the large motion data set required only one more iteration than the small motion data set to obtain all the clusters.

AP finds representative exemplars for the motion cluster that can directly be used for motion generation. K-medoids is the only other clustering algorithm amongst those tested that will also find representative exemplars. The representative exemplars found by both AP and K-medoids are the same for the DMP motion models for both the video stream and large motion data set. For HMM models trained on the large motion data set, AP tended to find different representative exemplars from those found for the DMP. This may be because of the different nature of modeling, i.e. HMMs are stochastic while DMPs use a nonlinear dynamical model, combined with the different distance metrics, i.e. negative KL distance versus negative Euclidean distance. However, the ability of the AP algorithm to correctly group similar motions into the same cluster is not impeded.

Overall, DMPs are better suited for task based motion modeling than HMMs because DMPs allow an exact initial and final position to be specified. DMPs also allow distance metrics other than the Euclidean distance to be used for computing the matrix of similarities for AP clustering, while HMMs require the use of the KL distance metric. However, HMMs are better at handling noise than DMPs, as HMMs inherently assume that there is noise in the observations. Ultimately, the choice of motion modeling algorithm depends on the application.

## 4.5 Summary

The motion primitive learning and recognition approach involves motion segmentation, motion modeling and motion recognition with a focus on the latter two. DMPs are used for motion modeling and AP is used for motion recognition via clustering. The proposed approach is successfully verified on a small motion data set of 24 exemplar motions and a large motion data set of 137 exemplar motions. For the small data set, motion segmentation, DMP motion modeling and AP clustering was performed to obtain motion recognition results. For the large motion data set, DMP and HMM motion modeling was performed and the resulting AP clustering outcomes were compared. It was found that clustering DMP motion models can be sensitive to the placement of the centers of the DMP Gaussian basis functions. In particular the basis functions towards the end of the motion trajectories tend to represent less data points of the trajectory and are more susceptible to angle switches, noise and timing variabilities found in the large motion data set. The problem of angle switches in the large motion data set is addressed by shifting the angles into the correct range, while sensitivity to noise and timing variabilities in the data towards the end of the trajectories is addressed by slightly shifting the centers of the basis functions away from the very last data point. With these adjustments, the DMP and HMM motion recognition results are very similar. One advantage of using DMP motion modeling is that the parameterized goal value allows for an arbitrary goal position to be set during motion generation which is useful for manipulation tasks. The AP clustering algorithm proves to be robust in detecting meaningful clusters under a variety of conditions. One advantage of using AP is that AP finds exemplars for each cluster, thus, directly picking the representative motion for the entire cluster which can be stored and used to generate motions for the entire cluster. The results of the proposed motion learning and recognition approach are used in the following chapter for task modeling.

# Chapter 5

# Task Learning[1]

Task modeling requires a method to combine the recognized motions and object states into a coherent sequence. Based on the observation that a manipulation task consists of sequences of actions that can be divided into goal based units of (pre-conditions for an action, action, and post-conditions of an action) [39], PNs are chosen as they can efficiently model sequences of such units.

This chapter develops an algorithm for automatically creating PN task models from observations of human task demonstration(s) as highlighted in Figure 5.1. The motion recognition component uses the approach described in Chapter 4 and models only object motions. The object and object state recognition component consists of matching a set of observable features to the object or object state to be identified. In the learning system in this thesis, this is achieved using SOMs. Having recognized both motions and object states, a PN reflecting the sequence of motions and object states is automatically created. The PN training process is able to summarize multiple demonstrations into the same PN model, as previously added motions and object states can be detected when they are repeated by the human demonstrator. Furthermore, the created PN can generate sequences of motions that generalize the task and allow the robot to imitate it.

[1] A version of this chapter has been accepted for publication [19]: G. J. Chang and D. Kulić. Robot task learning from demonstration using Petri nets. In *22nd International Symposium on Robot and Human Interactive Communication (RO-MAN)*, 2013, to be published.

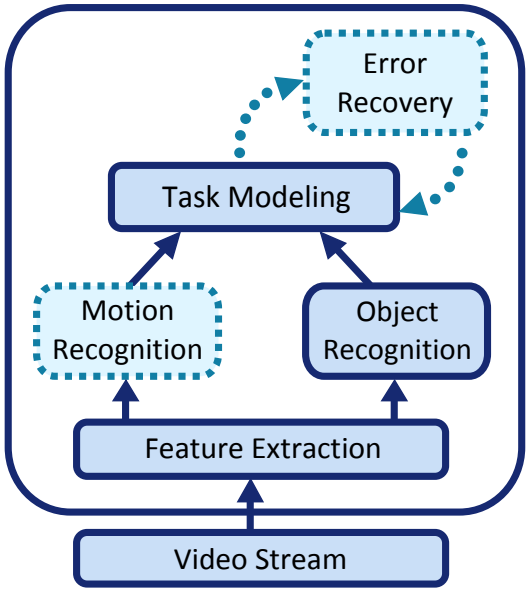



Figure 5.1: The object recognition and task learning components in the proposed learning by imitation system

## 5.1 Object and Object State Recognition

Object and object state recognition is achieved by extracting relevant features of objects shown in video stream data using simple computer vision techniques in order to match them to a particular object or object state. This concept is similar to the icons in Deacon's symbolic human language model where recognition is the process of matching the physical properties of an object to the object.

The details of the simple computer vision techniques applied in the feature extraction process are outlined in Appendix B. It should be noted that in computer vision, object recognition is a vast field of study [49]. The same is true for feature selection [37], i.e. determining which features to extract in the first place. But as neither computer vision nor feature selection is the main focus of the learning by imitation system, simple and intuitively useful features are pre-selected for extraction, such that the extraction algorithm can directly obtain the features without having to consider the feature selection problem. The pre-selected features are shown in Table 5.1. This simple approach is sufficient for the video stream data used for this thesis, but more sophisticated feature extraction methods can be considered as part of future improvements.

Table 5.1: Pre-selected features extracted from video stream data

| **Feature** | **Description** |
|---|---|
| $x$ | x coordinate of the pixel corresponding to the center of mass of the object. |
| $y$ | y coordinate of the pixel corresponding to the center of mass of the object. |
| $w$ | Maximum width of the object. |
| $h$ | Maximum height of the object. |
| $R$ | Red color value of the object. |
| $G$ | Green color value of the object. |
| $B$ | Blue color value of the object. |

Table 5.2: SOM parameter values used in implementation

| **Parameter** | **Object SOM** | **State SOM** |
|---|---|---|
| Input pattern vector size | 3 | 3 |
| Output nodes grid size | 2x2 | 5x5 |
| Initial learning rate $\alpha_{SOM}$ | 0.5 | 0.5 |
| Initial neighborhood parameter $\sigma_{SOM}$ | 0.1 | 0.2 |
| Parameter update constant $c_{SOM1}$ | 0.3 | 0.3 |
| Parameter update constant $c_{SOM2}$ | 0.5 | 0.5 |

Once the extracted features are available, two SOMs are used to perform object and object state recognition whenever all objects in the system are stationary. One SOM performs object recognition by using the red, green and blue (RGB) color value features extracted. The winning output node of this SOM will be the same for objects of the same color. Using a second SOM, the state of each object is detected using the x, y coordinates of the object as well as a flag indicating the relationship between objects in particular whether there is an object stacked on top or placed below the current object. See Appendix B for more details about this flag. Thus, whenever an object is in a particular location and has the same relationship to other objects, the same output node in the SOM will be triggered. The advantage of using SOMs is that they should support object and object state recognition even if more sophisticated features are selected for future improvements. The SOM parameter values used are listed in Table 5.2.

## 5.2 Task Modeling

To model a particular task, the places in the PN are used to represent object states, while the transitions in the PN are used to represent motions, i.e. actions taken by the human demonstrator. The sequence of object states and motions observed dictates the sequence of places and transitions of the PN. This allows PN to be automatically created. The advantage of using PNs is that PNs are flexible enough to model various task flows, including synchronous and asynchronous, parallel and serial processing. Additionally, the created PNs can be used to generate a sequence of transitions for task imitation. For the purposes of modeling, it is assumed that transitions in the PN occur instantaneously [81], even though during the real task a transition may take some time to actually occur.

### 5.2.1 Petri Net Training

The automatic PN creation algorithm proceeds as follows:

1. When all objects are stationary, the state of each object is represented using a place in the PN. If the place representing a particular state does not already exist, a new place is added to the PN to represent the state of the object. To determine if a place already exists or not a simple search through all previously seen states is performed.

2. When one or more of the objects are moving, object positions are recorded in order to obtain the trajectories of the motion.

3. When a motion has just finished, i.e. objects are no longer moving, the motion is first modeled via a DMP and then identified using AP using the motion recognition approach described in Chapter 4. If the motion that has just finished is a new, previously unseen motion, it is added as a new transition to the PN. A new motion is determined by searching through all the representative exemplars of previous motions. Currently, the proposed algorithm only considers the case when one object is moving at a time. Smaller motions of other objects are assumed to be due to occlusions when one object is moving.

4. Then, the state (i.e. place in the PN) right before the motion and the state right after the motion are checked for each object. If the states are different, the states will be connected via the transition representing the motion just observed, i.e. the state right before the motion is added to the input place(s) and the state right after the motion is added to the output place(s) of that transition. Additionally, the object that is

moved during the motion, i.e. the object undergoing the most displacement, will also have its state prior to the motion and its state right after the motion connected via the transition, regardless of whether these states are different. This will cover cases when the object's position before and after the motion is the same.

5. Return to step 1 or stop if all stationary and motion sequences in the observation have been processed.

This PN creation algorithm is able to accommodate multiple demonstrations of the same task, since the same sequence of object states and motions will be observed during the task, the PN creation algorithm will be able to recognize that the places and transitions representing the object states and motions are already present in the PN and do not need to be added. In the case that there are variations in how the same task is performed during the demonstration, the created PN will either already reflect the variation or the PN will be extended to include new ways of performing the task. The PNs created for the learning system in this thesis are 1-bounded (also referred to as *safe*), i.e. at most one token per place at any given time, and unweighted, i.e. all the weights are effectively 1.

Lastly, separate from the PN, but as part of the task model, each object that has been observed is stored along with its affordances, i.e. the actions that have been performed on each object. This is done so that it is possible to verify whether a transition can occur given a particular object for task imitation. A transition is only possible on an object if it has previously been observed being performed on the object during demonstration.

### 5.2.2 Petri Net Generation

For imitating the observed task, the created PN can be used to generate a sequence of motions. This is accomplished using pre-existing PN analysis tools as follows:

1. Use an image of the initial and desired final states of the task to be imitated as input.
2. Extract the objects and object states from the input images and convert the extracted states to a marking of the PN.
3. Using the initial marking as the root node, generate the reachability tree for the PN.
4. Using depth first tree traversal[2], go through the reachability tree to find a path

[2]Depth first tree traversal is used because it simultaneously generates a path from the root node (initial marking) to the tree node with the desired final marking, which indicates the sequence of transitions, i.e. actions, to be performed during task imitation.

between the root node (initial marking) and a tree node with the final marking. Always traverse the entire tree and store all potential paths found.

5. Select the shortest path found. If multiple shortest paths exist, pick the first one found. If additional criteria for task imitation are available, a more complex path selection system could be used.

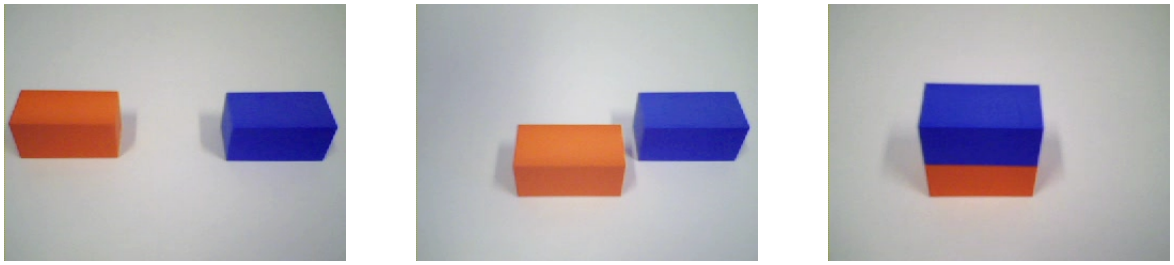
The path found via the reachability tree will indicate the sequence of transitions, and thus, the motions, required to imitate the task. The DMPs will then be used to generate the actual motion trajectories, which will be used by the robot to perform the task. It should be noted that because the reachability tree of the PN covers all the places, i.e. object states, in between the initial and final states, it should be possible to perform any sub-component of the task using a PN that is created from one demonstration of the entire task.

## 5.3 Experiments

To test the code implementation, videos were captured using the NAO humanoid robot [4] with a resolution of 320x240 pixels and an average frame rate of 6 frames per second. As the NAO humanoid robot only has one camera for observing the workspace, stereo images are not available and the distances to the blocks is fixed in the experiments. If necessary, depth information of objects could be extracted from the captured video using knowledge of the actual object size. Additionally, object pose estimation has not been implemented, as the reconstruction from monocular vision to 3 dimensional representation is not the main focus of the work in this thesis. The following two test settings are considered:

1. **2 block stacking**, where the demonstrator stacks two solidly colored blocks by first moving one block towards the center of the workspace and then the second block on top of the first block. This setting models a sequential task, as the second block cannot be stacked if the first block is not in place.

2. **3 block stacking**, where the demonstrator stacks three solidly colored blocks, whereby the first two blocks are placed side by side at the center of the workspace before the third block is stacked on top of the first two blocks. This setting models a task with both parallel and sequential elements. The order in which the first two blocks are moved towards the center of the workspace is unimportant, but the third block can only be moved after the first two blocks are present at the center of the workspace.



(a) Start (b) Intermediate step (c) Finish

Figure 5.2: 2 block stacking task

The following subsections describe in detail the exact test cases considered and the results obtained for the 2 block stacking and the 3 block stacking test settings. Appendix C describes the detailed code implementation for task learning.

### 5.3.1 2 Block Task

In this test setting two rectangular, solidly colored blocks are stacked by a human demonstrator who moves each block towards the middle of the the workspace as shown in Figure 5.2. For this test setting the PN has to be able to model a sequential task, where the order of actions matters.

A number of test cases are considered for this setting:

- **Exact Imitation**
  The purpose of this test case is to see if the PN can be used to model the stacking task and generate the sequence of actions needed to perform the task exactly as observed.
- **Subset Imitation**
  The purpose of this test case is to see if the PN can generate a sequence of actions corresponding to only part of the task that is observed.
- **Multiple Demonstrations**
  The purpose of this demonstration is to see if the code can summarize multiple demonstrations of the same task in the same PN.
- **Minor Generalization**
  The purpose of this demonstration is to see if the PN can be used to generate the

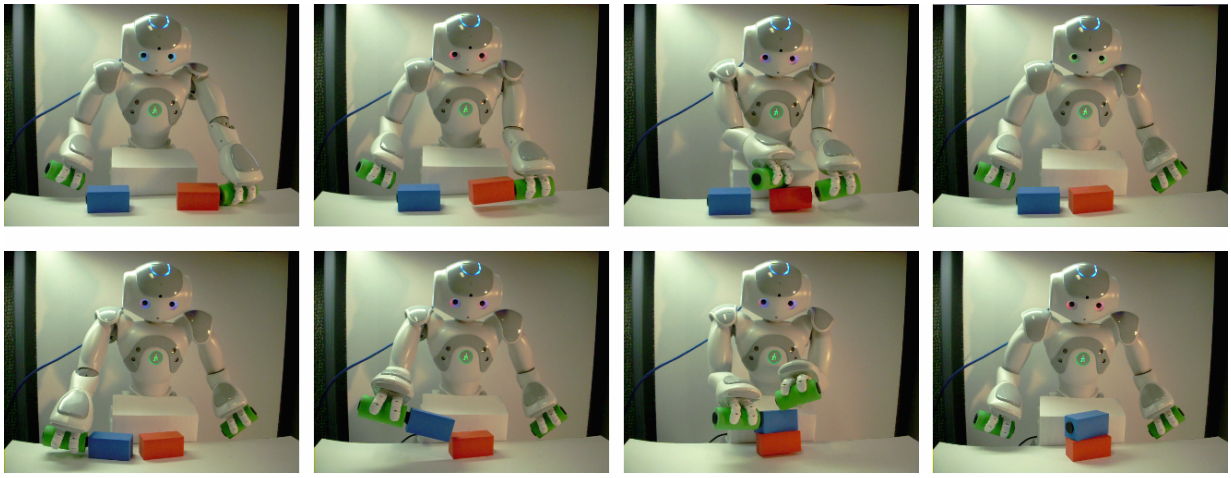


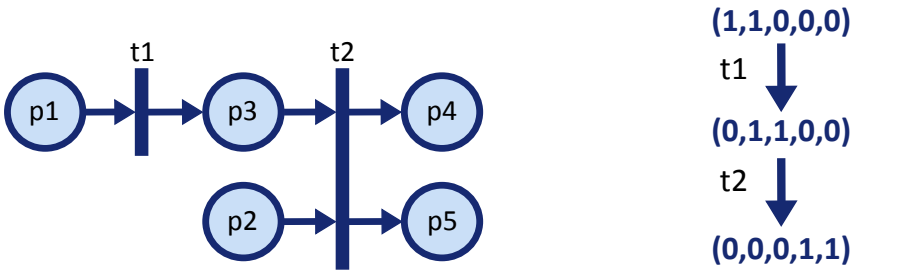
Figure 5.3: NAO robot imitating the 2 block stacking task

sequence of actions needed to perform the task with a slightly different set-up than what is observed. In this particular case, a new combination of colored blocks is the difference in the set-up.

The result of each test case is used to let the NAO robot imitate the task in simulation (for all the test cases) using the Webots [24] robot simulation environment and on actual robot hardware (for the exact imitation case). Figure 5.3 illustrates the NAO robot performing the exact imitation case. During imitation, magnets are used to allow the robot to "grasp" and "release" the blocks, as there is no actuation in the hands of the particular NAO robot used. The "grasping" motion simply consists of position interpolation from the robot's current hand position to the position of the object to be "grasped". The two dimensions of the position of the object are extracted via vision feedback and the "grasping" motion is adjusted accordingly (the third dimension, i.e. the distance from the robot to the object, is fixed due to the lack of depth information). The "releasing" motion proceeds similarly from the final object position to a pre-determined home position of the robot's hand. There are two possible ways for the robot hand to disconnect the magnets to "release" an object, one is by using the other hand of the robot to hold down the object during "releasing". The other way is to place the object against a small obstruction and pull away from it such that the obstruction holds back the object and "releases" it. It is assumed that the robot already knows how to let the magnets connect and disconnect for "grasping" and "releasing" blocks, as the only motion learned is how to move the blocks themselves.

Table 5.3: 2 block task exact imitation PN structure

| **Component** | | | |
|---|---|---|---|
| P | = {p1, p2, p3, p4, p5} | | |
| T | = {t1, t2} | | |
| I (t1) | = {p1} | or I | $= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}$ |
| I (t2) | = {p2, p3} | | |
| O (t1) | = {p3} | or O | $= \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$ |
| O (t2) | = {p4, p5} | | |




(a) PN graph (b) PN reachability tree

Figure 5.4: 2 block task exact imitation

### Exact Imitation

The purpose of this test case is to see if the PN can be used to model the stacking task and generate the sequence of actions needed to perform the task exactly as observed. The input video depicts a red block and blue block being stacked as shown in Figure 5.2. The red block starts off on the left side of the workspace as seen by the robot, while the blue block starts off on the right side of the workspace. The red block is moved from the left side towards the middle of the workspace first, before the blue block is moved from the right side towards the middle on top of the red block. The resulting PN generated from the input video is shown in Table 5.3 as a PN structure and Figure 5.4a as a PN graph.

The places and their meaning, i.e. the object states that they represent, are summarized in Table 5.4. The meaning of the transitions is as follows: $t_1$ refers to the first motion observed, which is the red block moving from the left side of the workspace towards the center and $t_2$ refers to the second motion observed, which is the blue block moving from the right side of the workspace towards the center on top of the red block. The created

Table 5.4: 2 block task PN place meanings (L = left, R = right, C = center)

| P | Object Location | | | Other Object | | |
|---|---|---|---|---|---|---|
| | L | R | C | None | Above | Below |
| p1 | ✓ | | | ✓ | | |
| p2 | | ✓ | | ✓ | | |
| p3 | | | ✓ | ✓ | | |
| p4 | | | ✓ | | ✓ | |
| p5 | | | ✓ | | | ✓ |

PN has captured the sequential nature of the task, as the second block can only be moved if the first block is already in place at the center of the workspace, i.e. place $p3$ is marked.

The initial and final images used as input show the initial and final state of the task. The resulting initial and final markings are $\mu_{initial} = (1, 1, 0, 0, 0)$ and $\mu_{final} = (0, 0, 0, 1, 1)$. The reachability tree constructed from the PN is shown in Figure 5.4b and the path between $\mu_{initial}$ and $\mu_{final}$ found through the reachability tree is $\sigma = t_1 t_2$ which exactly matches the task to be imitated. This illustrates that the PN created from observation can indeed be used by the robot to determine the correct sequence of actions to imitate the demonstrated task.

### Subset Imitation

The purpose of this test case is to show that the PN can generate a sequence of actions corresponding to only part of the task that is observed. The same PN as for the exact imitation case is used. However, the initial and final images used as input are changed. For the first subset imitation, the initial and intermediate step image (as shown in Figure 5.2) is used as input, resulting in $\mu_{initial} = (1, 1, 0, 0, 0)$ and $\mu_{final} = (0, 1, 1, 0, 0)$ being extracted. The path found through the reachability tree is $\sigma = t_1$, which is exactly the transition required to perform this part of the task. The second subset imitation shows the intermediate step and final image (as shown in Figure 5.2) as input, resulting in $\mu_{initial} = (0, 1, 1, 0, 0)$ and $\mu_{final} = (0, 0, 0, 1, 1)$ being extracted. The path found through the reachability tree is $\sigma = t_2$, which is exactly the transition required to perform this part of the task. Hence, as suggested in the proposed approach, the PN can be used to generate sub-parts of a task, having been shown only the full task.

Table 5.5: 2 block task multiple demonstration PN structure

| **Component** | | | | |
|---|---|---|---|---|
| P | = {p1, p2, p3, p4, p5} | | | |
| T | = {t1, t2, t3} | | | |
| I (t1) | = {p1} | | | $\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$ |
| I (t2) | = {p2, p3} | or I | = | |
| I (t3) | = {p3} | | | |
| O (t1) | = {p3} | | | $\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$ |
| O (t2) | = {p4, p5} | or O | = | |
| O (t3) | = {p3} | | | |

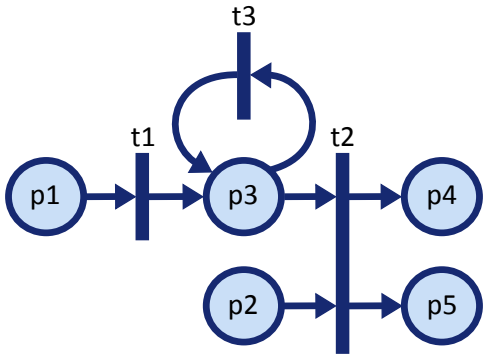



Figure 5.5: 2 block task multiple demonstration PN graph

### Multiple Demonstrations

The purpose of this test case is to demonstrate that the PN can summarize multiple demonstrations of the same task in the same model. A second video is used as input to the code. This video depicts a green block and a red block being stacked in the same fashion as in the first video. The green block starts off on the left side of the workspace as seen by the robot, while the red block starts off on the right side of the workspace. The only difference is that after moving the green block, the green block is gently tapped to straighten out its position. This tapping motion was not planned initially, but is added by the human demonstrator to better position the green block. The resulting PN generated from observing both the first and the second video is shown in Table 5.5 as a PN structure and Figure 5.5 as a PN graph.

The resulting PN is the same as the PN for the exact imitation case except for the

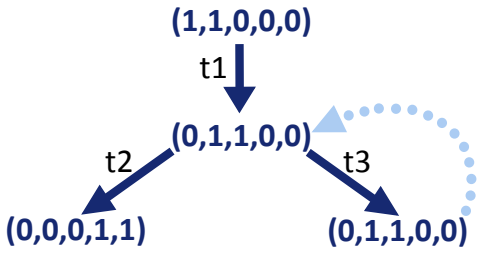


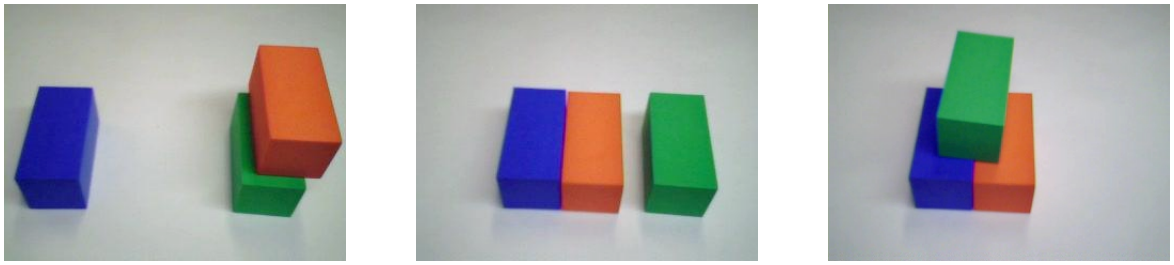

Figure 5.6: 2 block task multiple demonstration PN reachability tree

tapping motion that is added as a third transition $t_3$ in the form of a self-loop. The self-loop reflects that the tapping motion is just a minor adjustment and did not change the position of the green block sufficiently in order to cause a change in the state detected. It is clear that the two demonstrations of the task are summarized successfully into the same PN which reflects the underlying structure of the task being modeled while taking minor variations, such as the extra tapping motion, into consideration.

The initial and final images used as input show the initial and final state of the task in the second video. The resulting initial and final markings are $\mu_{initial} = (1, 1, 0, 0, 0)$ and $\mu_{final} = (0, 0, 0, 1, 1)$. The reachability tree constructed from the PN is shown in Figure 5.6 and the path between $\mu_{initial}$ and $\mu_{final}$ found through the reachability tree is $\sigma = t_1 t_2$ which exactly matches the task to be imitated. Even with multiple demonstrations, the PN can be used to find the correct sequence of actions for the robot to imitate the task. It should be noted that the tapping motion, though observed and modeled in the PN, did not affect the path found through the reachability tree, since the tapping motion does not really affect obtaining the final goal of stacking the blocks. This indicates that this approach to task modeling allows the human demonstrator to act more naturally, as some variations due to spontaneous adjustments by the human demonstrator do not affect learning of the overall task.

### Minor Generalization

The purpose of this demonstration is to see if the PN can be used to generate the sequence of actions needed to perform the task with a slightly different set-up than what is observed. In this particular case, a new combination of colored blocks is the difference in the set-up. The same PN as was generated for the multiple demonstrations case is used. However, the initial and final images used as input show a combination of a green block on the left and a blue block on the right. Even though this combination was never observed,



(a) Start (b) Intermediate step (c) Finish

Figure 5.7: 3 block stacking task

the previous videos did show that the green block affords being moved into the middle of the workspace from the left and the blue block affords being moved into the middle of the workspace from the right on top of another block. As previously mentioned, the observed affordances of objects are stored and used to keep track of which actions can be performed on which objects. Using the initial and final images of the green and blue block combination, the initial and final markings that are extracted are $\mu_{initial} = (1, 1, 0, 0, 0)$ and $\mu_{final} = (0, 0, 0, 1, 1)$. The path found via the PN generation algorithm is $\sigma = t_1 t_2$ which is the expected sequence of transitions to perform the desired task. Thus, the PN is able to generalize to new object combinations.

### 5.3.2 3 Block Task

In this test setting, three rectangular, solidly colored blocks are stacked by a human demonstrator who placed two of the blocks side by side in the middle of the workspace first, before placing the third block on top of the first two blocks as shown in Figure 5.7. For this test setting the PN has to be able to model a combination of a parallel (concurrent) and a sequential task, i.e. the order of actions for placing the first two blocks can be interchanged, but the action of placing the third block must come last.

A number of test cases are considered for this setting:

- **Exact Imitation**
  The purpose of this test case is to see if the PN can be used to model the stacking task and generate the sequence of actions needed to perform the task exactly as observed.

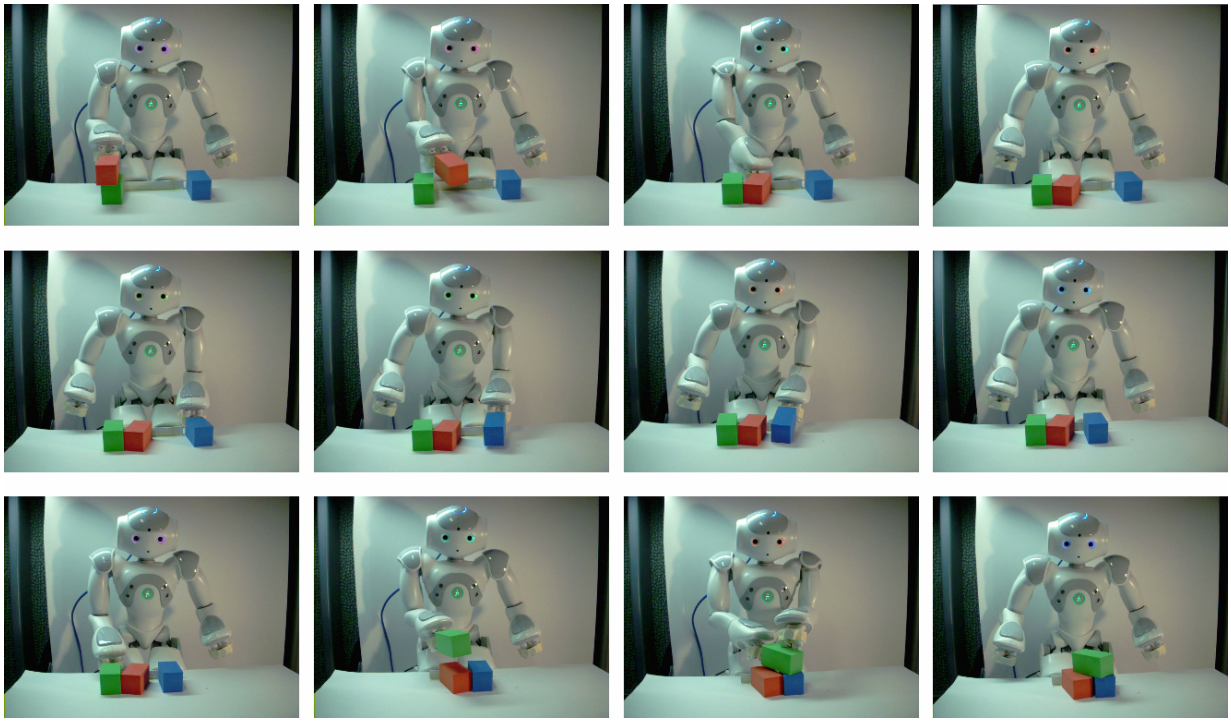


Figure 5.8: NAO robot imitating the 3 block stacking task

- **Subset Imitation**
  The purpose of this test case is to see if the PN can generate a sequence of actions corresponding to only part of the task that is observed.

- **Multiple Demonstrations**
  The purpose of this demonstration is to see if the code can summarize multiple demonstrations of the same task in the same PN.

- **Minor Generalization**
  The purpose of this demonstration is to see if the PN can be used to generate the sequence of actions needed to perform the task with a slightly different set-up than what is observed. In this particular case, a new configuration of the same set of three colored blocks is the difference in the set-up.

As with the 2 block stacking task, the results of each test case are used to let the NAO robot imitate the task in simulation using Webots [24] and the exact imitation task is tested on the NAO robot hardware as shown in Figure 5.8. The same assumptions for

Table 5.6: 3 block task exact imitation PN structure

| **Component** | | | |
|---|---|---|---|
| P | = {p1, p2, p3, p4, p5, p6, p7, p8, p9} | | |
| T | = {t1, t2, t3} | | |
| I (t1) | = {p1, p2} | | |
| I (t2) | = {p3} | or I = | $\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$ |
| I (t3) | = {p4, p5, p6} | | |
| O (t1) | = {p4, p5} | | |
| O (t2) | = {p6} | or O = | $\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$ |
| O (t3) | = {p7, p8, p9} | | |

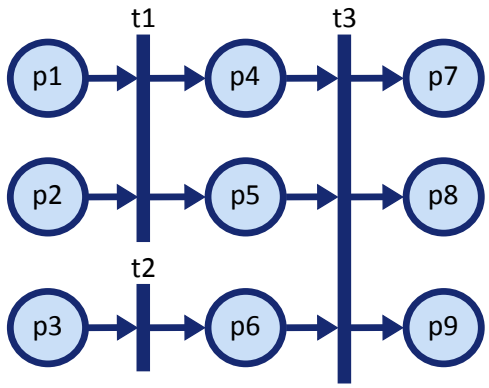



Figure 5.9: 3 block task exact imitation PN graph

"grasping" and "releasing" the blocks as for the 2 block setting apply. The NAO robot is unable to perfectly align the blocks due to joint limitations. This is corrected in Chapter 6 by treating the resulting block positions as an error case and letting the robot perform appropriate adjustments such as tapping blocks into alignment.

### Exact Imitation

The purpose of this test case is to see if the PN can be used to model the stacking task and generate the sequence of actions needed to perform the task exactly as observed. The input video depicts a blue, red and green block being stacked as shown in Figure 5.7. The blue block is initially on the left side of the workspace, while the red block is placed on top of the green block at the right side of the workspace. The red block is moved first in

Table 5.7: 3 block task place meanings (L = left, R = right, C = center)

| P | Object Location | | | | | Other Object | | |
|---|---|---|---|---|---|---|---|---|
| | **L** | **R** | **CL** | **CR** | **C** | **None** | **Above** | **Below** |
| p1 | | ✓ | | | | | | ✓ |
| p2 | | ✓ | | | | | ✓ | |
| p3 | ✓ | | | | | ✓ | | |
| p4 | | | | ✓ | | ✓ | | |
| p5 | | ✓ | | | | ✓ | | |
| p6 | | | ✓ | | | ✓ | | |
| p7 | | | | ✓ | | | ✓ | |
| p8 | | | | | ✓ | | | ✓ |
| p9 | | | ✓ | | | | ✓ | |

the video towards the middle of the workspace, then the blue block is moved towards the middle as well and placed to the left of the red block. Lastly, the green block is stacked on top of the blue and red blocks. The resulting PN generated from the input video is shown in Table 5.6 as a PN structure and Figure 5.9 as a PN graph. As can be seen, the created PN correctly reflects that the moving of the red and blue blocks are parallel actions and their order of occurrence can be interchanged without affecting the end result.

The places and their meanings, i.e. the object states that they represent, are summarized in Table 5.7. The meaning of the transitions is as follows: $t_1$ refers to the first motion observed, which is the red block moving from the right side of the workspace towards the center, $t_2$ refers to the second motion observed, which is the blue block moving from the left side of the workspace towards the center next to the red block and $t_3$ refers to the third motion observed, which is the green block moving from the right side of the workspace on top of the blue and red blocks. As can be seen from Figure 5.9, the created PN correctly reflects that the moving of the red and blue blocks are parallel actions and their order of occurrence can be interchanged without affecting the end result.

The initial and final images used as input show the initial and final state of the task. The resulting initial and final markings are $\mu_{initial} = (1, 1, 1, 0, 0, 0, 0, 0, 0)$ and $\mu_{final} = (0, 0, 0, 0, 0, 0, 1, 1, 1)$. The reachability tree constructed from the PN is shown in Figure 5.10 and two paths are found through the reachability tree for the given $\mu_{initial}$ and $\mu_{final}$ markings: $\sigma_1 = t_1 t_2 t_3$ which exactly matches the demonstrated task and $\sigma_2 = t_2 t_1 t_3$ which matches a possible alternative sequence of performing the task.

From this result, it can be seen that the PN not only correctly reflects parallel actions,

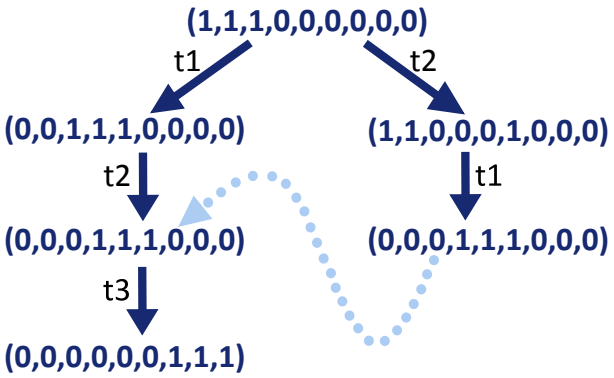



Figure 5.10: 3 block task exact imitation PN reachability tree

but it is also possible to find variations in the sequence of actions for performing a task, even though only one of the possible sequences is observed in demonstration. Currently, the code is set up to select the first shortest path found, i.e. $\sigma_1 = t_1 t_2 t_3$, but if additional information about the desirability of one sequence versus another were available, it would be possible for the algorithm to be modified to take the additional information into consideration.

### Subset Imitation

The purpose of this test case is to show that the PN can generate a sequence of actions corresponding to only part of the task that is observed. The same PN as for the exact imitation case is used. However, the initial and final images used as input are changed. Apart from the initial and final steps, there are three intermediate steps. The first intermediate step is when the red block is moved first and placed at the center of the workspace. The second intermediate step is a variation of the first intermediate step, where the blue block is moved first and placed at the center of the workspace. The third intermediate step is when both the blue and red blocks are at the center of the workspace as shown in Figure 5.7. This gives rise to a number of possible combinations for the initial and final desired markings, which are summarized in Table 5.8.

All of the resulting paths found through the reachability tree for each of the initial and final marking combinations shown in Table 5.8 result in the expected sequence of transitions $\sigma$. As before, when multiple shortest paths arise, the first one found it picked. These results show that the PN can be used to generate sub-parts of the task when only the full task has been observed, even for tasks with both parallel and sequential components.

Table 5.8: 3 block task subset imitation results

| **Initial Image** | **Final Image** | **Initial Marking** | **Final Marking** | **Path** |
|---|---|---|---|---|
| Start | Intermediate 1 | $(1,1,1,0,0,0,0,0,0)$ | $(0,0,1,1,1,0,0,0,0)$ | $t1$ |
| Start | Intermediate 2 | $(1,1,1,0,0,0,0,0,0)$ | $(1,1,0,0,0,1,0,0,0)$ | $t2$ |
| Start | Intermediate 3 | $(1,1,1,0,0,0,0,0,0)$ | $(0,0,0,1,1,1,0,0,0)$ | $t1t2$ or $t2t1$ |
| Intermediate 1 | Intermediate 3 | $(0,0,1,1,1,0,0,0,0)$ | $(0,0,0,1,1,1,0,0,0)$ | $t2$ |
| Intermediate 1 | Finish | $(0,0,1,1,1,0,0,0,0)$ | $(0,0,0,0,0,0,1,1,1)$ | $t2t3$ |
| Intermediate 2 | Intermediate 3 | $(1,1,0,0,0,1,0,0,0)$ | $(0,0,0,1,1,1,0,0,0)$ | $t1$ |
| Intermediate 2 | Finish | $(1,1,0,0,0,1,0,0,0)$ | $(0,0,0,0,0,0,1,1,1)$ | $t1t3$ |
| Intermediate 3 | Finish | $(0,0,0,1,1,1,0,0,0)$ | $(0,0,0,0,0,0,1,1,1)$ | $t3$ |

### Multiple Demonstrations

The purpose of this test case is to demonstrate that the PN can summarize multiple demonstrations of the same task in the same model. A second video is added depicting the alternate sequence of stacking the blocks, i.e. the initial and final block positions are the same, but the blue block is moved towards the middle of the workspace first, before the red block is moved. The resulting PN after showing both videos is the same as for the exact imitation case, except that a different motion is chosen as the exemplar for moving the third block. This demonstrates that multiple demonstrations of tasks with parallel and sequential components can be successfully summarized in the same PN. The created PN captures the fact that the underlying structure of the demonstrated task is the same despite variations in the sequence of the parallel components over multiple demonstrations.

Again, the initial and final images used as input show the initial and final state of the task. The resulting initial and final markings are $\mu_{initial} = (1,1,1,0,0,0,0,0,0)$ and $\mu_{final} = (0,0,0,0,0,0,1,1,1)$. Two paths are found through the reachability tree for the given $\mu_{initial}$ and $\mu_{final}$ markings: $\sigma_1 = t_1t_2t_3$ and $\sigma_2 = t_2t_1t_3$ which match the two observed variations of performing the task. This result illustrates that even with multiple demonstrations of a task with parallel and sequential components, the PN can be used to determine the sequence of actions needed for the robot to perform the task.

### Minor Generalization

The purpose of this demonstration is to see if the PN can be used to generate the sequence of actions needed to perform the task with a slightly different set-up than what is observed. The desired initial and final images for this test case are the same as for the exact imitation case, except that the red and green blocks have switched places, i.e. the initial image shows the green block on top of the red block, while the final image shows the green block beside the blue block and the red block on top. This is a configuration that has not been previously observed. In order for the learning system to recognize that the green block affords being placed at the center of the workspace and that the red block affords being placed on top of the green and blue blocks at the center of the workspace, two additional videos are recorded depicting these particular object affordances. Both of the two additional videos show three blocks being stacked as before.

The first video starts off with a red block on the left side and a green block on top of a blue block on the right side. The red block is first moved towards the center of the workspace before the green and blue blocks are moved such that the green block is beside the red block and the blue block is stacked on top of the green and red blocks. In this video, it is shown that the green block affords being moved towards the center of the workspace. The second video, starts off with a green block on the left side and a blue block on top of a red block on the right side. The blue block is moved towards the center of the workspace first, before the green and red blocks are moved beside the blue block and on top of the blue and green blocks, respectively. This video shows that the red block affords being moved on top of two blocks at the center of the workspace.

The PN from the multiple demonstrations case is further trained on these two additional videos. The resulting PN after showing all videos is the same as for the exact imitation case, except that a different motion is chosen as the exemplar for moving the second and third block.

Using the initial and final images as previously described, the initial and final markings extracted are $\mu_{initial} = (1, 1, 1, 0, 0, 0, 0, 0, 0, 0)$ and $\mu_{final} = (0, 0, 0, 0, 0, 0, 0, 1, 1, 1)$. The paths found via the PN generation algorithm is $\sigma_1 = t_1 t_2 t_3$ and $\sigma_2 = t_2 t_1 t_3$ which is the expected sequence of transitions to perform the desired task. Thus, the PN is able to generalize to new object configurations, as long as the object affordances have been previously observed.

## 5.4 Summary

An approach to automatically create PNs from videos of human demonstrations of tasks is proposed, where object and object state information is obtained via simple vision extraction techniques and SOMs. Motion primitives are modeled and clustered using DMPs and the AP algorithm, as described in Chapter 4. Having recognized motions and object states, transitions and places are automatically created for these motions and object states, respectively. To form a PN, the created transitions and places are automatically connected based on the order in which the motions and objects are observed. The created PN may be used to determine a path between an initial and final (desired) marking and can be used both for action sequence recognition and generation. The created PN can successfully capture sequential and parallel tasks and allows the detection of sub paths and alternate paths with alternate action sequences. Furthermore, multiple demonstrations depicting the same underlying task lead to the same PN being generated with variations successfully incorporated. The PN path generation can also be applied to situations slightly different from the demonstration, i.e. when a new combination of known objects is given. Dynamical reconstruction of the reachability tree is considered in the case of failed actions and unexpected situations as part of the error recovery process described in the next chapter.

# Chapter 6

# Error Recovery[1]

In order for the learning by imitation system to allow the robot to robustly operate within a dynamic human environment, the ability to handle errors is required. Robust operation is achieved via error recovery to allow the robot to continue task performance in the event of unexpected situations. Error recovery capabilities are useful for handling issues that may not be fully anticipated by the human demonstrator before the robot attempts to imitate the task [85].

The previous chapter covered task learning by automatically learning PN task models from observation of demonstrated tasks. This chapter extends the learned task models to allow the learning by imitation system to automatically recover from error conditions, i.e. any unexpected situations. The error recovery component highlighted in Figure 6.1 includes the ability to dynamically adjust the generated sequence of actions for performing a task when an unexpected, yet known situation is encountered. This is achieved by re-generating the reachability tree and finding a new path through it. Additionally, the PN task model is automatically extended with an error recovery subnet to incorporate human demonstrated error recovery procedures for unexpected and unknown situations. The proposed error recovery component is tested on the 3 block stacking task, during which unexpected, but known situations are created by the human demonstrator and unexpected, unknown situations arise from the robot's own limitations in aligning the blocks. Using the error recovery capabilities requires little effort on the part of the human demonstrator.

[1] A version of this chapter has been submitted for publication [17]: G. J. Chang and D. Kulić. Robot task error recovery using Petri nets learned from demonstration. 2013, submitted for publication.

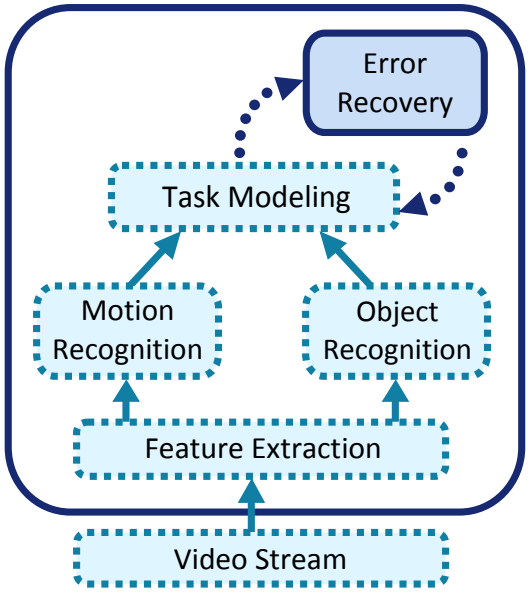


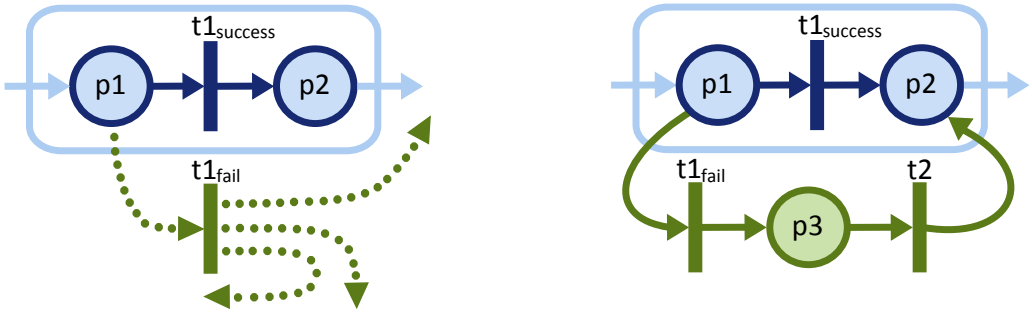

Figure 6.1: The error recovery component in the proposed learning by imitation system

## 6.1 Petri Net Extension

The PN task model framework described in Chapter 5 is extended with error recovery capabilities to allow the robot to automatically detect and recover from error conditions. Error conditions are detected during task imitation by checking the state of the objects after performing each action corresponding to a transition in the PN and also before the very first action is taken (to ensure the initial state is correct). If the states of the objects in the environment differ from the expected states based on the output place(s) reached by the transitions in the path or the place(s) in the initial marking, an error condition is detected.

In order for the PN to handle error conditions, all transitions in the PN are implicitly assumed to be able to fail during task imitation. Implicit here refers to the fact that while all transitions may be able to fail and result in any possible error state afterwards, this information is not explicitly stored in the PN. The success or failure of a transition in the PN is determined by the resulting output place of the transition as shown in Figure 6.2a:

1. A *successful* transition is one through which the desired output place is reached.
2. A *failed* transition is one through which any place other than the desired output




(a) Implicit transition fail

(b) Learned error correction

Figure 6.2: Example of failing PN transition

place is reached, i.e. the reached output place is unexpected based on the path found through the PN reachability tree. Two types of output places can be reached via a failed transition:

(a) A *known* place. In this case, it is possible to directly recover from the error by regenerating the reachability tree using the current marking for the PN and finding the paths (if any) that will lead to the final desired state.

(b) An *unknown* place. In this case, it is not possible to directly recover from the error. Instead, the robot should try to learn from a human demonstrator how to recover from the error encountered, so that it can handle the error in the future.

In order to learn how to recover from errors when the robot reaches an unknown place after a failed transition, the robot prompts the human demonstrator asking if it can learn to correct the error. The system relies on the human demonstrator for making the decision of whether the robot can learn to correct the error or if the robot should ignore the error and continue with the task. An approach to automatically detect whether an error can be corrected by the robot was not implemented in this thesis. Errors that the robot can correct must fulfill the following conditions:

1. The object states must still be visible to the robot, i.e. the robot can still sense all the objects.

2. The robot must be able to do the motion needed to correct the error, i.e. the motion must not involve impossible configurations or unreachable positions for the robot.

If the robot can correct the error, a video of the human demonstrator performing the motion for error correction is recorded. The demonstrated error correction should lead from an unknown state to a known state, i.e. existing place in the PN. From the recorded video, the robot obtains the unexpected and previously unknown error states, the motion performed by the human demonstrator and the resulting, known states. Using this information and knowing which transition led to the error states, the PN is extended by adding an arc from the known place(s) that precede the failed transition to the failed transition and from the failed transition to the new place(s) representing the error state. The new error state place(s) then have an arc leading to a new transition representing the error correction motion which then connects back to a known place in the PN. Appendix D provides the pseudocode for how failed transitions are added into the PN during PN training. Additionally, an example is shown in Figure 6.2b, where $p_3$ is the new error state place and $t_2$ is the new error correction transition learned. Figure 6.2b illustrates error correction according to the forward error recovery category introduced in Chapter 2, Section 2.3. Forward error recovery is preferable [76] and our approach does not run into difficulties with finding a forward error recovery path, since a human will demonstrate the necessary corrective action. Algorithm 6.1 outlines the main steps of task execution including the error recovery process.

Once the error recovery procedure has been incorporated into the PN, the robot can directly recover from the error the next time that it is encountered, as the error states are now part of the known states. Thus, if the error states are unexpectedly encountered again, the reachability tree will be regenerated to find a path starting from the place representing that error state and leading to the final desired state of the task. This reachability tree regeneration and path finding can be repeated multiple times during task imitation, if multiple errors are encountered.

The ability of the robot to adjust for both known and unknown output places allows for a collaborative approach between human and robot to complete the task. For example, if the human helps to move objects towards their final state, the robot will be able to adjust its next action based on the resulting states in the environment.

## 6.2 Experiments

To demonstrate the proposed error recovery approach, the NAO humanoid robot [4] is used to perform the 3 block stacking task for which it has learned a PN task model based on human demonstration in the task learning experiments of the previous chapter, Section 5.3.2. In the task, a blue, red and green block are stacked as shown in Figure 5.7 in the

**Algorithm 6.1** Task execution including error recovery

```
1: generate reachability tree
2: find path to final desired state
3: error detection: detect states
4: if environment states are as expected then
5:    if reached final desired state then exit
6:    else take next action in path and return to line 3
7:    end if
8: else error correction:
9:    if environment states are known then
10:      regenerate reachability tree using PN
11:      find path to final desired state
12:      if path exists then return to line 3
13:      else no path exists output warning and exit
14:      end if
15:   else environment states are unknown
16:      prompt human demonstrator for help
17:      if can learn to correct error then
18:         capture video of human demonstration
19:         extract object, object states and motions
20:         extend PN and return to line 3
21:      end if
22:   end if
23: end if
```

previous chapter. The blue block starts on the left side (L) of the workspace as seen by the robot, while the red block starts on top of the green block on the right side (R). The red block is moved first towards the center right (CR) of the workspace, then the blue block is moved towards the center left (CL), i.e. next to the red block. Finally, the green block is stacked at the center (C) on top of the red and blue blocks. This task contains both a parallel (concurrent) and a sequential component, i.e. the order of actions for placing the first two blocks can be interchanged, but the action of placing the third block must come last. The resulting learned PN for this task is shown in Figure 5.9 in the previous chapter.

The two types of error states handled by the proposed error recovery algorithm are *known error states* and *unknown error states*. The experimental results are obtained by letting the NAO robot imitate the task and adapt its behaviour according to the states reached so that appropriate error recovery is performed.

Table 6.1: Valid markings of 3 block task PN (L = left, R = right, C = center)

| **Valid Marking** | **Block Positions** |
|---|---|
| $\mu_1 = (1,1,1,0,0,0,0,0,0)$ | Initial state of stacking task |
| $\mu_2 = (0,0,1,1,1,0,0,0,0)$ | Red at CR, green at R, blue at L |
| $\mu_3 = (1,1,0,0,0,1,0,0,0)$ | Red at R, green at R, blue at CL |
| $\mu_4 = (0,0,0,1,1,1,0,0,0)$ | Red at CR, green at R, blue at CL |
| $\mu_5 = (0,0,0,0,0,0,1,1,1)$ | Final state of stacking task |

### 6.2.1 Known Error State

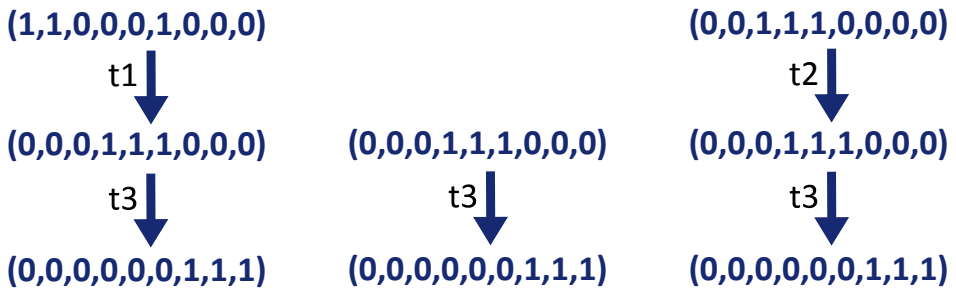
Given the PN in Figure 5.9, five possible valid markings of the PN exist as shown in Table 6.1. Valid markings involve all three objects in physically possible states for which the algorithm can find a path to the final desired marking.

Thus, given any valid initial marking, there will be one correct output marking and four unexpected markings including the initial marking itself. These four unexpected markings correspond to the possible, known error states.

The algorithm should be able to recover from any of the known error states by finding a path through the reachability tree from the marking representing the error state to the final desired marking. The reachability tree is regenerated each time, as it is not assumed that the previously generated reachability tree contains all the valid markings already.

The output marking of $t_1$ (moving the red block) is used to demonstrate recovery from known error states. The initial marking is $\mu_1 = (1,1,1,0,0,0,0,0,0)$. The expected, correct output marking following the execution of $t_1$ should be $\mu_2 = (0,0,1,1,1,0,0,0,0)$. However, a human demonstrator will change the outcome to the known error marking $\mu_1$, $\mu_3$, $\mu_4$ or $\mu_5$, i.e. the human demonstrator will move the blocks while the robot is performing the motion for $t_1$ so that the resulting state is unexpected but known. For each of these unexpected, known error states, the robot correctly determines that it has to use the reachability tree to find a new path leading to the desired final goal:

1. $\mu_1$**:** For this unexpected output marking, the reachability tree remains the tree shown in Figure 5.10. The path that the algorithm finds is $\sigma = t_1 t_2 t_3$, which is the correct sequence of actions to reach the final goal and the robot is subsequently able to finish performing the task successfully. (The algorithm actually also finds the alternate path $\sigma = t_2 t_1 t_3$, but since no additional information is available to distinguish between the two paths, the first path found is selected.) This result represents the error of the red block not having moved even though the robot tried to move it via $t_1$. It shows




(a) $\mu_3$ case (b) $\mu_4$ case (c) $\mu_2$ case

Figure 6.3: New reachability trees

that the robot is able to recognize that the action $t_1$ needs to be repeated and thus, the robot is able to recover from the error.

2. $\mu_3$**:** For this unexpected output marking, a new reachability tree as shown in Figure 6.3a is constructed from the PN and the path found is $\sigma = t_1 t_3$. This situation illustrates the case where the blue block is unexpectedly moved to the center left, while the red block is not moved even though it is expected to move to the center right. The robot switches to following the alternate sequence where $t_3$ follows $t_1$ to successfully complete the task.

3. $\mu_4$**:** For this unexpected output marking, a new reachability tree as shown in Figure 6.3b is constructed from the PN and the path found is $\sigma = t_3$. This case illustrates the situation where the blue block is unexpectedly moved towards the center when the robot is moving the red block, the robot recognizes and chooses to only perform the last action $t_3$ to successfully finish stacking all three blocks.

4. $\mu_5$**:** For this unexpected output marking, the task stacking is completed and the robot realizes that no reachability tree needs to be constructed and no more actions are necessary.

The cases of $\mu_4$ and $\mu_5$ as unexpected output markings simulates cases where a human decides to help the robot progress in the task and the robot correctly recognizes that some of the blocks have unexpectedly been moved to their final states and no more actions need to be performed on them. This could be used for human and robot collaborative task execution.

In order to verify that the algorithm is able to recover if $\mu_2$ is the unexpected output marking and also to demonstrate that the algorithm is able to handle not only unexpected

output markings, but also unexpected initial markings, the robot is presented with the red block already moved at the beginning of the task resulting in the initial marking of $\mu_2$, even though the robot is expecting $\mu_1$. In this case, the robot also successfully recognizes the error and regenerates the new reachability tree as shown in Figure 6.3c from the PN and finds the resulting path $\sigma = t_2 t_3$. Thus, $\mu_2$ is also one of the recoverable known error states.

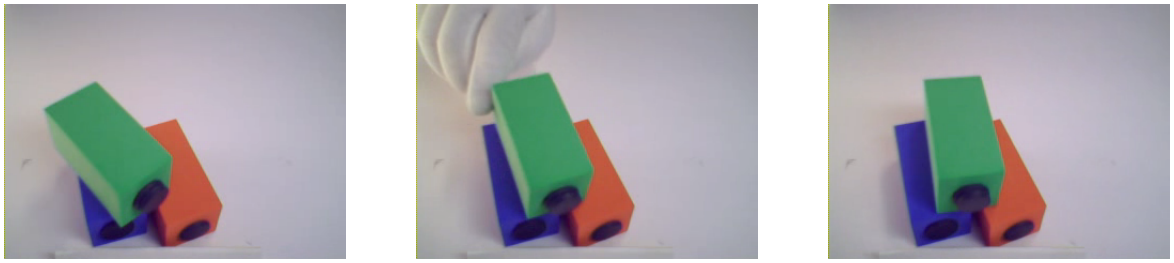
Finally, even though all the above results are given for $t_1$, the same error recovery capabilities apply to the output markings of $t_2$ and $t_3$. All possible unexpected output markings are also tested for $t_2$ and $t_3$ and the robot is able to successfully adjust the sequence of actions to be performed. The resulting reachability trees and paths are as described above for each of the unexpected output markings for $t_1$.

Furthermore, the robot can handle multiple errors during task imitation as the proposed algorithm simply checks whether the states are as expected after each transition (and also for the initial set of states) and regenerates the reachability tree and path as needed. To demonstrate this for the 3 block stacking task with the path $\sigma = t_1 t_2 t_3$, the outcome of $t_2$ (moving the blue block) is changed from $\mu_4$ to $\mu_3$, i.e. the red block is moved back to its starting position at the right of the workspace. The robot correctly regenerates the reachability tree (same as the one in Figure 6.3a) and finds the path $\sigma = t_1 t_3$. Next, when the robot performs $t_3$ (moving the green block), the outcome is changed from $\mu_5$ to $\mu_4$, i.e. the green block is moved back to its starting position at the right of the workspace. Again the robot correctly regenerates the reachability tree (same as the one in Figure 6.3b) and finds the path $\sigma = t_3$. Thus, the robot is able to correctly finish the task despite errors encountered after both $t_2$ and $t_3$ during task imitation.

### 6.2.2 Unknown Error State

For the 3 block stacking task, when placing the green block on top of the red and blue blocks an "unknown" error state may be encountered. This is due to the joint limitations of the NAO robot which result in the green block being placed on the red and blue blocks at an angle. This misalignment of the green block causes occlusion of the blue block which results in different states being detected by the algorithm for the green and blue blocks. The algorithm does tolerate some minor misalignments in block placement, however, the misalignment of the green block can exceed this tolerance.

This situation can serve as an example of an unknown error state as the robot does not know by itself how to correct the misalignment of the green block. When the green block is improperly aligned, the robot prompts for help, asking if this is an error that it



(a) Unknown error state (b) Human correction (c) Known state

Figure 6.4: Human demonstrator corrects green block misalignment in 3 block stacking task

can learn to correct. The human demonstrator lets the robot know that it can correct this problem and the robot records a video of the human demonstrator showing how to tap the green block into alignment as shown in Figure 6.4. Once the green block has been aligned properly, the robot is able to recognize the states and knows that the task has been completed.

Object state and motion information is then extracted from the recorded video and

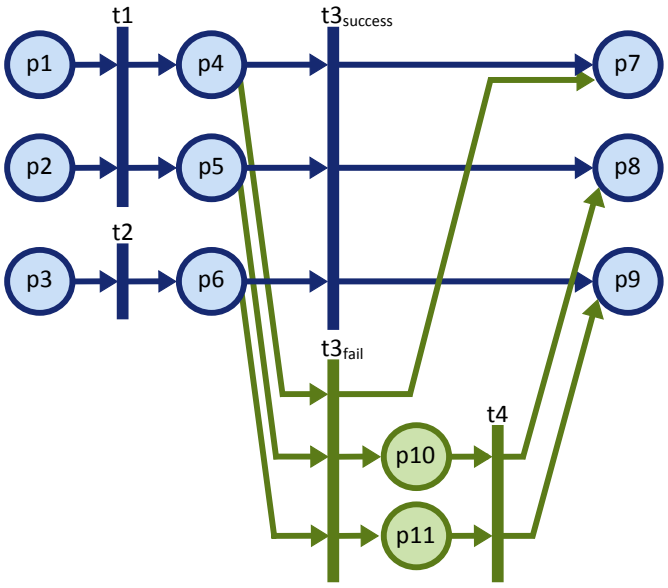



Figure 6.5: PN with learned error correction

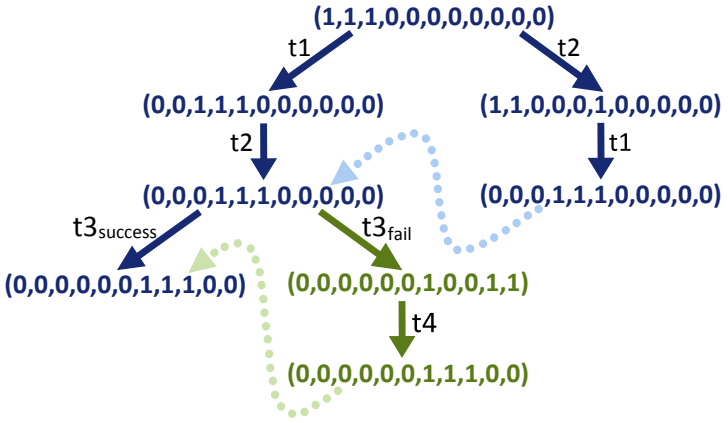



Figure 6.6: PN reachability tree with learned error correction

added to the PN resulting in the modified PN shown in Figure 6.5. The failed transition along with the resulting new place representing the error state and new transition representing the error correction motion leading back to a known state have been included in the PN. The resulting PN reachability tree is shown in Figure 6.6. The branch containing the error correction is longer than the direct path to the final state, thus the robot will not pick that path from the paths found unless the error is encountered. If additional information is provided, e.g. that failed transitions should be avoided, the robot can reject paths containing failed transitions when no error states are encountered.

If the robot encounters the error state of the misaligned green block again during task imitation, the error state will be a known error state, which will be resolved by regenerating the reachability tree and searching for a path starting from the marking representing the error state. Figure 6.7 depicts the NAO robot autonomously correcting the alignment of the green block position after having stacked it on top of the red and blue blocks. The NAO robot has previously learned how to correct the error from a human demonstrator.

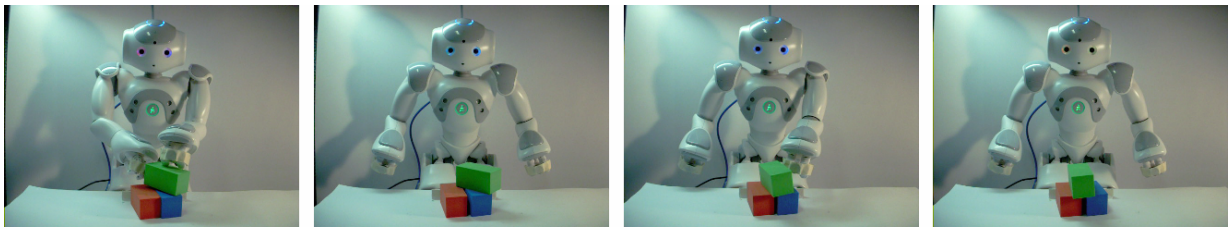


Figure 6.7: NAO robot performing error correction action after stacking the green block

## 6.3 Summary

Automatic error recovery in the learning by imitation system is achieved by checking the object states in the environment and adjusting the sequence of actions or extending the PN task model when needed. The error recovery component allows for direct error recovery from unexpected yet known states via regeneration of the PN reachability tree and finding a new path. Further, new error recovery strategies can be learned via observation of human demonstration for unexpected and unknown states. This error recovery approach results in minimal effort for the human demonstrator to teach the robot and only pertinent error recovery strategies will be learned by the robot. The proposed approach also holds potential for collaborative human robot task completion as the robot is able to dynamically adjust its actions in the case that the human helps to complete part of the task.

# Chapter 7

# Conclusions and Future Work

This thesis presented a learning by imitation system that uses DMPs and APs for motion learning, automatic generation of PNs for task learning from human demonstration and dynamic generation of action sequences and automatic extension of the PNs for error recovery during task execution. This learning by imitation system consists of a hierarchical setup, where motions, objects and object states are recognized at the lower levels of the hierarchy. The higher levels of the hierarchy combine the recognized motions and objects into a coherent task model. Experiments using the NAO humanoid robot were conducted and the robot is able to successfully learn to imitate block stacking tasks.

The motion learning approach consists of motion segmentation, motion modeling and motion recognition via clustering. Motion trajectories are segmented into motion primitives whenever the velocity of the motion switches from moving to non-moving. The motion primitives are then trained on a third order dynamical system version of the DMP. DMPs are well suited for motion modeling in manipulation tasks, as they can be used to robustly generate motion trajectories to an arbitrary goal position. The weight parameters of the DMPs are used for motion recognition by clustering them using AP. The AP algorithm is easy to use and efficient. Moreover, AP is an exemplar based clustering algorithm and will find a motion primitive exemplar in each cluster of motions to represent that type of motion. The motion recognition approach was able to correctly recognize motions for a small dataset with 24 motion exemplars consisting of four types of block stacking motions. Depending on how the AP preference value is tuned, a coarse or fine division of the motion types can be achieved. The motion recognition approach was also able to correctly recognize the nine types of different full body motions in a larger data set of 137 motion exemplars. The AP algorithm performs robustly and efficiently when compared to other common clustering appoaches. AP can work with both DMPs or HMMs to cluster

motions with comparable results. HMMs were not used for the learning by imitation system as they do not directly parameterize goal positions in learned motion trajectories.

The task learning approach models tasks as a sequence of actions and object states. This sequence is encoded in a coherent model using PNs. Simple computer vision techniques are used to extract object features from video stream data along with object motion trajectories. The object features are used for object and object state recognition via two SOMs. Each SOM provides a single output signal for a set of object features, thus, identifying the object or object state. The object states observed in the task represent pre- or post-conditions of actions and are modeled using places in the PN. The actions themselves are represented using transitions in the PN. Based on the sequence in which object states and actions are observed in a human demonstration of the task, the places and transitions in the PN, as well as their connecting arcs, are automatically generated. The task modeling approach was successfully tested on a sequential 2 block stacking task and a 3 block stacking task with sequential and parallel components. The PNs were able to capture the underlying structure of the task from a single demonstration, as well as summarize multiple demonstrations of the task. A PN reachability tree was constructed to find a path indicating the sequence of actions needed to enable exact imitation of the task, imitating a sub-component of the task and for performing the task with minor generalizations, such as a different combination or configuration of objects than what was observed during the human demonstration. Testing was performed using simulations for all the test cases and an actual NAO humanoid robot for the exact imitation case to ensure that the learned PN model can be used for imitating the block stacking tasks.

To further improve the robustness and usability of the task learning approach, error recovery capabilities were added. An error is any unexpected situation encountered during task execution. By checking the object states in the environment against the expected object states based on the path generated via the PN reachability tree, errors can be diagnosed. For known error states, the PN reachability is regenerated and a new path to the desired final state is found. For unknown error states, the human demonstrator is prompted for assistance. If the robot is physically able to perform the error correction action, it can learn to do so by observing the human demonstrator and adding a forward error recovery extension to the learned PN model. This novel dynamic adjustment and automatic learning of the error recovery procedure is tested on the NAO humanoid robot for the 3 block stacking task. The NAO humanoid robot is able to successfully adapt the actions it needs to take in case of known errors and it is able to learn error recovery procedures from human demonstration. The proposed error recovery requires minimal effort from the human demonstrator and improves the employability of the overall learning by imitation system.

## 7.1 Future Work

Various possibilities for future improvements to the proposed learning by imitation system have been identified throughout the thesis. They are listed in more detail in the following sections along with additional recommended improvements.

### 7.1.1 Motion Learning

For motion learning, the motion tracking, segmentation, modeling and recognition components can all be improved. Currently, the system only tracks object positions for motion modeling during tasks. However, tracking the object pose and joint angles of the human demonstrator are necessary for learning how to autonomously grasp and release objects. Tracking this additional information may be considered in the future.

Even though the simple motion segmentation approach proposed in Chapter 4 was sufficient for the motion trajectories extracted during the 2 block and 3 block stacking task in this thesis, more sophisticated motion segmentation techniques would contribute to the robust performance of the proposed learning by imitation system for tasks involving more complex motion sequences. Motion segmentation approaches either use a template based approach, where segmentation is achieved by comparing observed motion trajectories to previously created motion primitive templates [6], or unsupervised learning, where no prior knowledge is assumed [55]. Since unsupervised learning methods are preferred in the learning by imitation system in this thesis, the approach in [55] could be considered for adaptation in the future. In [55], the Shi and Thomasi algorithm [88] is used to identify features to track in a monocular video containing full body human motions. The optical flow of these features is estimated and clustered to determine regions of motion and their corresponding segmentation points.

Although the DMP generated trajectories worked well for the 2 block and 3 block stacking tasks, their accuracy can be further improved. For example, by adding landmark alignment, such as dynamic time warping (DTW) [48], [43], the major features in the motion primitive trajectories can be better captured by the same DMP weight parameter when training the DMP. This allows more DMP weights to be used, which will improve the ability of the DMP to capture details of the motion primitive trajectory during motion generation. Also, by using more recent re-formulations of the DMP [40], [80], the resemblance of the DMP generated trajectories with the original motion primitive trajectories that the DMP was trained on can be further improved. Further extensions to the DMP, such as obstacle avoidance capabilities [79], [40], may also be considered in the future.

The AP algorithm is employed offline in this thesis, but incremental algorithms are preferable for continuous learning in a dynamic environment [10], [56]. An incremental AP clustering approach has been proposed in [96] for data stream clustering. The approach places new data points into existing clusters found by AP if they are close enough to the cluster exemplar, otherwise they are placed into a reservoir. When the reservoir reaches a certain size, the AP algorithm is re-run on all data points. The idea of comparing new data points to existing cluster exemplar data points may be used to run AP in a more incremental fashion for the motion recognition system. Though the development of a fully incremental AP clustering approach that does not need to be re-run on all existing data points may also be considered for future work. Moreover, different distance metrics for computing the similarity values for the AP algorithm can be investigated to determine the effect of the choice of distance metric on the clustering results. Beyond improvements to the AP algorithm itself, additional experimental tests on a data set that includes multiple human demonstrators may be considered to provide more insight into the robustness of the proposed motion recognition approach. Each data set used in this thesis only contained motions of one human demonstrator, but the learning system should allow the robot to robustly recognize the same motion primitives when performed by different human demonstrators.

### 7.1.2 Task Learning

Even though the object and object state recognition approach in this thesis worked for the primary colored objects in the 2 block and 3 block stacking task, a wider range of objects and object states should be recognizable by the learning by imitation system. Object and object state recognition can be improved by using better features, which may require more sophisticated computer vision techniques. More features could be defined for the objects, such as orientation, size, shape etc. which allows additional information such as depth and object pose to be extracted. The actual feature selection techniques may be applied to select the best combination of features for object and object state recognition. Alternatively, automatic feature selection methods, such as Shi and Thomasi [88] or scale invariant feature transforms (SIFT) points [65], may be considered. Furthermore, the standard SOMs used for object recognition do not have a flexible structure, i.e. the total number of output nodes has to be set beforehand. If the exact number of expected output clusters is not known beforehand, it is possible to simply select a larger number of output nodes than what the approximate number of output clusters is expected to be. However, there are also extensions of the SOM that allow the SOM to grow over time, e.g. the growing SOM (GSOM) can be initialized with just a small number of output nodes and

rows or columns of output nodes are added dynamically as needed [27]. This extension may be added to the learning by imitation system to better handle an arbitrary number of objects.

Additionally, future improvements to the PN reachability tree usage may be considered. Most PN analysis approaches make use of the reachability tree and it has been shown that the reachability problem is decidable, though possibly computationally expensive [71], [81]. This may limit the usefulness of PNs in practice, though the PNs in this thesis are simple enough such that the computational complexity of the reachability problem is not an issue. The question whether one reachability tree is a subset of or equal to that of another reachability tree is shown to be an undecided problem, i.e. no algorithm could solve the problem in general. Furthermore, the lower bound of the computational complexity of the reachability problem has been shown to be exponential time-hard and exponential space-hard, i.e. the amount of time and memory space is at least an exponential function of the PN input description length [71], [81], [82]. To avoid constructing and searching through large reachability trees, the PN task models can have a hierarchical setup for larger, more complex tasks. The efficiency of constructing and searching through a single, large PN reachability tree versus a hierarchy of smaller PN reachability trees can be compared and analyzed as part of future work to determine if this hierarchical setup is beneficial. This analysis can be used to develop criteria for determining the cost of learning tasks of different complexities. This cost evaluation can be used to form a more theoretical framework for predicting the outcomes of task learning.

### 7.1.3 Error Recovery

A forward error recovery approach is applied in this thesis, as it is preferable to other approaches [76]. Forward error recovery is efficient as it allows the task to advance to the next set of object states after an error has been encountered. However, there may be instances where other error recovery approaches, such as backward error recovery and input conditioning, are necessary. For example, in a task where several objects are assembled together, an error may only be resolved by disassembling everything and re-starting the assembly, e.g. backward error recovery, or by disassembling the last object added and trying again, e.g. input conditioning. Thus, future work may consider including these alternate error recovery approaches. This would require a method for identifying when to apply each type of error recovery. Additionally, future work may consider a more theoretical framework for automatically determining if error correction may be learned for unknown error states rather than relying on the human demonstrator's judgement. This may require the definition of the regions of state space where error correction is not possible.

# References


[1] D. Aarno and D. Kragic. Motion intention recognition in robot assisted applications. *Robotics and Autonomous Systems*, 56(8):692–705, 2008.

[2] E. E. Aksoy, A. Abramov, J. Dörr, K. Ning, B. Dellen, and F. Wörgötter. Learning the semantics of object-action relations by observation. *International Journal of Robotics Research*, 30(10):1229–1249, September 2011.

[3] E. E. Aksoy, B. Dellen, M. Tamosiunaite, and F. Wörgötter. Execution of a dual-object (pushing) action with semantic event chains. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 576–583, 2011.

[4] Aldebaran Robotics. Home - corporate - aldebaran robotics. `http://www.aldebaran-robotics.com/`, 2012. [Online; Accessed 22-February-2013].

[5] A. Alissandrakis, C. L. Nehaniv, and K. Dautenhahn. Imitation with alice: learning to imitate corresponding actions across dissimilar embodiments. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 32(4):482–496, 2002.

[6] N. Alves and T. Chau. Vision-based segmentation of continuous mechanomyographic grasping sequences. *IEEE Transactions on Biomedical Engineering*, 55(2):765–773, 2008.

[7] D. C. Bentivegna, C. G. Atkeson, and G. Cheng. Learning tasks from observation and practice. *Robotics and Autonomous Systems*, 47(2-3):163–169, 2004.

[8] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, New York, NY, USA, 2006.

[9] C. Breazeal and B. Scassellati. Robots that imitate humans. *Trends in Cognitive Sciences*, 6(11):481–487, 2002.

[10] S. Calinon and A. Billard. Incremental learning of gestures by imitation in a humanoid robot. In *Proceedings of the ACM/IEEE International Conference on Human-Robot Interaction*, pages 255–262, 2007.

[11] S. Calinon and A. Billard. What is the teacher's role in robot programming by demonstration? - toward benchmarks for improved learning. *Interaction Studies. Special Issue on Psychological Benchmarks in Human-Robot Interaction*, 8(3):441–464, 2007.

[12] S. Calinon, F. Guenter, and A. Billard. On learning, representing, and generalizing a task in a humanoid robot. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 37(2):286–298, 2007.

[13] D. Cao and B. Yang. An improved k-medoids clustering algorithm. In *International Conference on Computer and Automation Engineering*, volume 3, pages 132–135, 2010.

[14] T. Cao and A. C. Sanderson. Sensor-based error recovery for robotic task sequences using fuzzy Petri nets. In *Robotics and Automation*, volume 2, pages 1063–1069, May 1992.

[15] T. Cao and A. C. Sanderson. Modeling of sensor-based robotic task plans using fuzzy Petri nets. In *Computer Integrated Manufacturing and Automation Technology*, pages 73–80, October 1994.

[16] S. K. Chalup. Incremental learning in biological and machine learning systems. *International Journal of Neural Systems*, 12(6):447–465, 2002.

[17] G. J. Chang and D. Kulić. Robot task error recovery using Petri nets learned from demonstration. 2013, submitted for publication.

[18] G. J. Chang and D. Kulić. Motion learning from observation using affinity propagation clustering. In *22nd International Symposium on Robot and Human Interactive Communication (RO-MAN)*, 2013, to be published.

[19] G. J. Chang and D. Kulić. Robot task learning from demonstration using Petri nets. In *22nd International Symposium on Robot and Human Interactive Communication (RO-MAN)*, 2013, to be published.

[20] C. Chao and A. L. Thomaz. Timing in multimodal turn-taking interactions: Control and analysis using timed Petri nets. *Journal of Human-Robot Interaction*, 1(1):4 – 25, 2012.

[21] M. H. Choi and W. W. Lee. Quantitative evaluation of an intuitive teaching method for industrial robot using a force/moment direction sensor. *International Journal of Control, Automation, and Systems*, 1(3):395–400, 2003.

[22] D. Coman, A. Petrisor, A. Ionescu, and M. Florescu. Role selection mechanism for the soccer robot system using Petri net. In *The International Conference on "Computer as a Tool" (EUROCON)*, pages 662–667, September 2007.

[23] H. Costelha and P. Lima. Modelling, analysis and execution of robotic tasks using Petri nets. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1449–1454, November 2007.

[24] Cyberbotics Ltd. Webots: Robot simulator. `http://www.cyberbotics.com`, 2011. [Online; Accessed 18-May-2011].

[25] M. d. Jeng. Petri nets for modeling automated manufacturing systems with error recovery. *Robotics and Automation*, 13(5):752–760, October 1997.

[26] T. W. Deacon. Symbols aren't simple. In *The Symbolic Species: The Co-evolution of Language and the Human Brain*, pages 69–101. W. W. Norton & Company, New York, New York, USA, 1997.

[27] M. Dittenbach, A. Rauber, and D. Merkl. Uncovering hierarchical structure in data using the growing hierarchical self-organizing map. *Neurocomputing*, 48(1-4):199 – 216, 2002.

[28] P. F. Dominey, G. Metta, F. Nori, and L. Natale. Anticipation and initiative in human-humanoid interaction. In *Proceedings of the IEEE International Conference on Humanoid Robots*, pages 693–699, 2008.

[29] E. Erdemir, C. B. Frankel, S. Thornton, B. Ulutas, and K. Kawamura. A robot rehearses internally and learns an affordance relation. In *7th IEEE International Conference on Development and Learning*, pages 298–303, 2008.

[30] A. Fod, M. J. Mataric, and O. C. Jenkins. Automated derivation of primitives for movement classification. *Autonomous Robots*, 12(1):39–54, 2002.

[31] B. J. Frey and D. Dueck. Clustering by passing messages between data points. *Science*, 315:972–976, 2007.

[32] B. J. Frey and D. Dueck. Supporting online material for clustering by passing messages between data points. *Science*, 315, 2007.

[33] H. Friedrich, S. Münch, R. Dillmann, S. Bocionek, and M. Sassin. Robot programming by demonstration (rpd): Supporting the induction by human interaction. *Machine Learning*, 23(2-3):163–189, 1996.

[34] A. Gams and A. Ude. Generalization of example movements with dynamic systems. In *9th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 28–33, 2009.

[35] C. Geib, K. Mourão, R. Petrick, N. Pugeault, M. Steedman, N. Krueger, and F. Wörgötter. Object action complexes as an interface for planning and robot control. In *Proceedings of the Humanoids Workshop: Towards Cognitive Humanoid Robots*, 2006.

[36] M. Gini and G. Gini. Towards automatic error recovery in robot programs. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence - Volume 2*, pages 821–823. Morgan Kaufmann Publishers Inc., 1983.

[37] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, March 2003.

[38] D. Herzog, V. Krueger, and D. Grest. Parametric hidden Markov models for recognition and synthesis of movements. Technical report, Aalborg University Copenhagen, 2008.

[39] J. Hoey, T. Plötz, D. Jackson, A. Monk, C. Pham, and P. Olivier. Rapid specification and automated generation of prompting systems to assist people with dementia. *Pervasive and Mobile Computing*, 7(3):299–318, 2011.

[40] H. Hoffmann, P. Pastor, D.-H. Park, and S. Schaal. Biologically-inspired dynamical systems for movement generation: Automatic real-time goal adaptation and obstacle avoidance. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2587–2592, 2009.

[41] A. J. Ijspeert, J. Nakanishi, and S. Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1398–1403, 2002.

[42] A. J. Ijspeert, J. Nakanishi, and S. Schaal. Learning attractor landscapes for learning motor primitives. In *Advances in Neural Information Processing Systems*, pages 1523–1530. MIT Press, 2003.

[43] W. Ilg, G. H. Bakir, J. Mezger, and M. A. Giese. On the representation, learning and transfer of spatio-temporal movement characteristics. *International Journal of Humanoid Robotics*, 01(04):613–636, 2004.

[44] M. Ito and J. Tani. On-line imitative interaction with a humanoid robot using a dynamic neural network model of a mirror system. *Adaptive Behavior*, 12(2):93–115, 2004.

[45] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, September 1999.

[46] S. Kajita. *Humanoid Robot*. Ohmsha, 2005. In Japanese.

[47] F. O. Karray and C. D. Silva. *Soft Computing and Intelligent Systems Design*. Pearson Education Limited, 2004.

[48] Eamonn J. Keogh and Michael J. Pazzani. Derivative dynamic time warping. In *In First SIAM International Conference on Data Mining (SDM)*, 2001.

[49] H. Kjellström, J. Romero, and D. Kragić. Visual object-action recognition: Inferring object affordances from human demonstration. *Computer Vision and Image Understanding*, 115:81–90, 2011.

[50] J. Kober, K. Mülling, O. Krömer, C. H. Lampert, B. Schölkopf, and J. Peters. Movement templates for learning of hitting and batting. In *IEEE International Conference on Robotics and Automation*, pages 853–858, May 2010.

[51] T. Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, September 1990.

[52] T. Kohonen. The self-organizing map. *Neurocomputing*, 21(1-3):1–6, 1998.

[53] G. Konidaris, S. Kuindersma, R. Grupen, and A. Barto. Robot learning from demonstration by constructing skill trees. *The International Journal of Robotics Research*, 31(3):360–375, 2012.

[54] V. Krüger, D. Herzog, S. Baby, A. Ude, and D. Kragic. Learning actions from observations. *IEEE Robotics Automation Magazine*, 17(2):30–43, 2010.

[55] D. Kulić, D. Lee, and Y. Nakamura. Whole body motion primitive segmentation from monocular video. In *IEEE International Conference on Robotics and Automation*, pages 3166–3172, 2009.

[56] D. Kulić and Y. Nakamura. Incremental learning and memory consolidation of whole body human motion primitives. *Adaptive Behavior - Animals, Animats, Software Agents, Robots, Adaptive Systems*, 17(6):484–507, December 2009.

[57] D. Kulić and Y. Nakamura. Incremental learning of human behaviors using hierarchical hidden Markov models. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4649–4655, 2010.

[58] D. Kulić, W. Takano, and Y. Nakamura. Incremental learning, clustering and hierarchy formation of whole body motion patterns using adaptive hidden Markov chains. *The International Journal of Robotics Research*, 27(7):761–784, 2008.

[59] Y. Kuniyoshi, M. Inaba, and H. Inoue. Learning by watching: extracting reusable task knowledge from visual observation of human performance. *IEEE Transactions on Robotics and Automation*, 10(6):799–822, December 1994.

[60] M. K. Lee, S. Kiesler, J. Forlizzi, S. Srinivasa, and P. Rybski. Gracefully mitigating breakdowns in robotic services. In *5th ACM/IEEE International Conference on Human-Robot Interaction*, pages 203–210, 2010.

[61] P. Lima, H. Gracio, V. Veiga, and A. Karlsson. Petri nets for modeling and coordination of robotic tasks. In *IEEE International Conference on Systems, Man, and Cybernetics*, volume 1, pages 190–195, October 1998.

[62] Peter Loborg. Error recovery in automation - an overview, 1994.

[63] A. Lockerd and C. Breazeal. Tutelage and socially guided robot learning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004.

[64] L. S. Lopes and L. M. Camarinha-Matos. Towards intelligent execution supervision for flexible assembly systems. In *IEEE International Conference on Systems, Man, and Cybernetics*, volume 2, pages 1225–1230, October 1996.

[65] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[66] T. Lozano-Perez. Robot programming. *Proceedings of the IEEE*, 71(7):821–841, 1983.

[67] T. Matsuoka, T. Hasegawa, and K. Honda. A dexterous manipulation system with error detection and recovery by a multi-fingered robotic hand. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 1, pages 418–423, 1999.

[68] F. Meier, E. Theodorou, F. Stulp, and S. Schaal. Movement segmentation using a primitive library. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3407–3412, 2011.

[69] D. Milutinovic and P. Lima. Petri net models of robotic tasks. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 4, pages 4059–4064, 2002.

[70] S. Münch, J. Kreuziger, M. Kaiser, and R. Dillmann. Robot programming by demonstration (rpd) - using machine learning and user interaction methods for the development of easy and comfortable robot programming systems. In *Proceedings of the 24th International Symposium on Industrial Robots*, pages 685–693, 1994.

[71] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541 – 580, April 1989.

[72] Y. Nagai. From bottom-up visual attention to robot action learning. In *8th IEEE International Conference on Development and Learning*, Shanghai, China, 2009.

[73] B. Nemec and A. Ude. Action sequencing using dynamic movement primitives. *Robotica*, 30:837–846, 8 2012.

[74] M. N. Nicolescu and M. J. Matarić. Learning and interacting in human-robot domains. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 31(5):419–430, 2001.

[75] M. N. Nicolescu and M. J. Matarić. Natural methods for robot task learning: Instructive demonstrations, generalization and practice. In *Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 241–248, 2003.

[76] N. G. Odrey and G. Mejia. An augmented Petri net approach for error recovery in manufacturing systems control. *Robotics and Computer-Integrated Manufacturing*, 21(4-5):346–354, 2005.

[77] T. Ogata, S. Sugano, and J. Tani. Open-end human robot interaction from the dynamical systems perspective: Mutual adaptation and incremental learning. *Advanced Robotics*, 19(6):435–444, 2004.

[78] P.-Y. Oudeyer, F. Kaplan, and V. V. Hafner. Intrinsic motivation systems for autonomous mental development. *IEEE Transactions on Evolutionary Computation*, 11(2):265–286, April 2007.

[79] D.-H. Park, H. Hoffmann, P. Pastor, and S. Schaal. Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields. In *8th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 91–98, 2008.

[80] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal. Learning and generalization of motor skills by learning from demonstration. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 763–768, 2009.

[81] J. L. Peterson. Petri nets. *ACM Computing Surveys*, 9(3):223–252, September 1977.

[82] J. L. Peterson. *Petri Net Theory and The Modeling of Systems*. Foundations of Philosophy Series. Prentice-Hall, 1981.

[83] C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Darmstadt University of Technology, Germany, 1962.

[84] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[85] J. Saunders, C. L. Nehaniv, K. Dautenhahn, and A. Alissandrakis. Self-imitation and environmental scaffolding for robot teaching. *International Journal of Advanced Robotic Systems*, 4(1):109–124, 2007.

[86] S. Schaal. Dynamic movement primitives - a framework for motor control in humans and humanoid robotics. *Adaptive Motion of Animals and Machines*, pages 261–280, 2006.

[87] S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert. Learning movement primitives. In *International Symposium on Robotics Research*. Springer, 2004.

[88] J. Shi and C. Tomasi. Good features to track. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 593–600, 1994.

[89] D. Summers-Stay, C. L. Teo, Y. Yang, C. Fermüller, and Y. Aloimonos. Using a minimal action grammar for activity understanding in the real world. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4104–4111, October 2012.

[90] W. Takano and Y. Nakamura. Humanoid robot's autonomous acquisition of proto-symbols through motion segmentation. In *6th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 425–431, 2006.

[91] E. Ugur, E. Sahin, and E. Oztop. Self-discovery of motor primitives and learning grasp affordances. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3260–3267, 2012.

[92] I. S. Vicente, V. Kyrki, and D. Kragic. Action recognition and understanding through motor primitives. *Advanced Robotics*, 21(15):1687–1707, 2007.

[93] A. D. Wilson and A. F. Bobick. Parametric hidden Markov models for gesture recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(9):884–900, 1999.

[94] K. Yamazaki, R. Ueda, S. Nozawa, Y. Mori, T. Maki, N. Hatao, K. Okada, and M. Inaba. System integration of a daily assistive robot and its application to tidying and cleaning rooms. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1365–1371, October 2010.

[95] D.-S. Yim and T. A. Barta. A Petri net-based simulation tool for the design and analysis of flexible manufacturing systems. *Journal of Manufacturing Systems*, 13(4):251–261, 1994.

[96] X. Zhang, C. Furtlehner, and M. Sebag. Frugal and online affinity propagation. In *Conférence francophone sur l'Apprentissage*, 2008.

[97] M.-C. Zhou and F. DiCesare. Adaptive design of Petri net controllers for error recovery in automated manufacturing systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(5):963–973, September/October 1989.

[98] V.A. Ziparo, L. Iocchi, P. U. Lima, D. Nardi, and P. F. Palamara. Petri net plans. *Autonomous Agents and Multi-Agent Systems*, 23:344 – 383, 2011.

# Appendices

# Appendix A

# Dynamic Movement Primitives Implementation

This appendix describes in detail the DMP parameter values used for motion modeling in the learning by imitation system proposed in this thesis. The DMP parameters are those given in the DMP formulation in chapter 3, section 3.1. The effects of these parameters on the motion recognition results were determined experimentally and further analyzed theoretically when needed. The parameters discussed are the motion trajectory data $y$, the temporal scaling factor $\tau$, the time constants $\alpha_z$ and $\beta_z$, the goal position $g$, the gating term $v$ and the Gaussian basis functions $\psi_i$.

## A.1 Motion Trajectory Data, $y$

Given an exemplar motion trajectory, the position $y$ will be known and the velocity $\dot{y}$ and acceleration $\ddot{y}$ can be computed through differentiation. It should be noted that the differentiation process amplifies any noise present in the position $y$ data. Furthermore, the authors of [42] recommend shifting the exemplar motion trajectory such that the initial position value $y_0$ is zero, i.e. let $y = y - y_0$. Shifting the initial position value $y_0$ can be done without loss of generality as a constant offset can be added during actual motion execution to let the motion start at a specific non-zero position. This was done for all the subsequent DMP experiments.

## A.2 Temporal Scaling Factor, $\tau$

For training the DMP formulation in (3.4), the temporal scaling factor $\tau$ should be:

$$\tau = T = n \cdot dt \tag{A.1}$$

where $T$ is the duration of the motion computed by taking the product of the total number of data points $n$ in a particular motion trajectory and the time step size $dt$ between data points. Note that given the sampling rate $f_s$ (in $Hz$) at which the motion data was captured, $dt$ (in $s$) can be easily computed using:

$$dt = \frac{1}{f_s} \tag{A.2}$$

Once the DMP model has been trained on an existing exemplar motion trajectory, $\tau$ can be adjusted to set the duration of the motions generated by the trained DMP model.

It should be noted that alternate DMP formulations exist, e.g. in [50] where $\tau$ is the reciprocal of the value defined in (A.1) or in [41] where $\tau$ is left out altogether and no adjustments can be made to the duration of the DMP generated motion trajectories.

## A.3 Time Constants, $\alpha_z, \beta_z$

The time constants $\alpha_z$ and $\beta_z$ are chosen for critical damping of the nonhomogeneous second order linear differential equation (i.e. when the nonlinear component $f = 0$) [87] by first rearranging (3.4) as shown:

$$\begin{aligned} \tau\dot{z} &= \alpha_z(\beta_z(g - y) - z), \quad z = \tau\dot{y} \\ \tau^2\ddot{y} &= \alpha_z\beta_z g - \alpha_z\beta_z y - \alpha_z\tau\dot{y} \\ \ddot{y} + \frac{\alpha_z}{\tau}\dot{y} + \frac{\alpha_z\beta_z}{\tau^2}y &= \frac{\alpha_z\beta_z}{\tau^2}g \end{aligned} \tag{A.3}$$

This equation is in the form of the standard second order dynamical system:

$$\ddot{x} + 2\zeta\omega_0\dot{x} + \omega_0^2 x = c \tag{A.4}$$

where $\zeta$ is the damping factor, $\omega_0$ is the natural frequency of the system and $c$ is a constant. For a critically damped system, $\zeta = 1$. By comparing the terms in (A.3) and (A.4) for the critically damped case:

$$\omega_0^2 = \frac{\alpha_z\beta_z}{\tau^2} \tag{A.5}$$

$$2\omega_0 = \frac{\alpha_z}{\tau}$$
$$\omega_0^2 = \frac{\alpha_z^2}{4\tau^2} \tag{A.6}$$

Equating (A.5) with (A.6):

$$\frac{\alpha_z \beta_z}{\tau^2} = \frac{\alpha_z^2}{4\tau^2}$$
$$\beta_z = \frac{\alpha_z}{4} \tag{A.7}$$

Thus, only the $\alpha_z$ value needs to be tuned for the DMP model, as the $\beta_z$ value can be found based on the $\alpha_z$ value according to (A.7) for the critically damped case. The critically damped case is used because, in theory, it affords the fastest response time without oscillations which is desirable for tracking the exemplar motion trajectory.

The $\alpha_z$ value is found via trial and error. Note that the same $\alpha_z$ value is used for training the DMP model and generating motions using the trained DMP model. For both the small and large motion data set $\alpha_z = 15$ is used as this is the highest $\alpha_z$ value before instabilities, such as oscillations, were observed in the DMP model generated trajectories.

## A.4 Goal Position, $g$

In the DMP formulation in (3.4), the goal $g$ is a constant value [42], [86]. Since the entire exemplar motion trajectory $y$ was shifted by the initial position $y_0$, the goal value $g$ accordingly is equivalent to $y_{end} - y_0$ during training of the DMP model. Once the DMP model has been trained, the goal $g$ can be adjusted to set where the DMP generated motion trajectories should end. For the DMP implementation used in this thesis, an extension to the DMP formulation in chapter 3, section 3.1 is added as described in chapter 4, subsection 4.4.1. This extension models the goal $g$ as a first order system with a constant $\alpha_g$.

One main issue encountered in implementing this extension to the DMP is how to set the $\alpha_g$ value. The authors of [87] suggest using a value of $\alpha_g = \frac{\alpha_z}{2}$. The reasoning can be traced by rewriting the DMP formulation in (4.1) in state space form for the case when the nonlinear component $f = 0$:

$$\dot{z} = \frac{\alpha_z \beta_z}{\tau} r - \frac{\alpha_z \beta_z}{\tau} y - \frac{\alpha_z}{\tau} z, \quad \dot{y} = \frac{z}{\tau}, \quad \dot{r} = -\frac{\alpha_g}{\tau} r + \frac{\alpha_g}{\tau} g \tag{A.8}$$

Let

$$\begin{aligned}
x_1 = y \quad &\implies \dot{x_1} = \dot{y} = \frac{z}{\tau} = x_2 \\
x_2 = \dot{y} = \frac{z}{\tau} \quad &\implies \dot{x_2} = \ddot{y} = \frac{\alpha_z \beta_z}{\tau^2} r - \frac{\alpha_z \beta_z}{\tau^2} y - \frac{\alpha_z}{\tau^2} z \\
&\qquad\qquad = \frac{\alpha_z \beta_z}{\tau^2} x_3 - \frac{\alpha_z \beta_z}{\tau^2} x_1 - \frac{\alpha_z}{\tau} x_2 \\
x_3 = r \quad &\implies \dot{x_3} = \dot{r} = -\frac{\alpha_g}{\tau} x_3 + \frac{\alpha_g}{\tau} g
\end{aligned} \tag{A.9}$$

Rewriting in matrix form:

$$\dot{x} = Ax + Bu$$

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ -\frac{\alpha_z \beta_z}{\tau^2} & -\frac{\alpha_z}{\tau} & \frac{\alpha_z \beta_z}{\tau^2} \\ 0 & 0 & -\frac{\alpha_g}{\tau} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \frac{\alpha_g}{\tau} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \tag{A.10}$$

Taking the eigenvalues of A:

$$\begin{aligned}
det(A - \lambda I) &= 0 \\
\left(-\frac{\alpha_g}{\tau} - \lambda\right)\left(-\lambda\left(-\frac{\alpha_z}{\tau} - \lambda\right) + \frac{\alpha_z \beta_z}{\tau^2}\right) &= 0 \\
\left(\frac{\alpha_g}{\tau} + \lambda\right)\left(\lambda^2 + \frac{\alpha_z}{\tau}\lambda + \frac{\alpha_z \beta_z}{\tau^2}\right) &= 0
\end{aligned} \tag{A.11}$$

For the critically damped case when $\beta_z = \frac{\alpha_z}{4}$:

$$\left(\lambda + \frac{\alpha_g}{\tau}\right)\left(\lambda + \frac{\alpha_z}{2\tau}\right)^2 = 0 \tag{A.12}$$

When $\alpha_g = \frac{\alpha_z}{2}$ according to the suggestion of the the authors of [87]:

$$\left(\lambda + \frac{\alpha_z}{2\tau}\right)^3 = 0 \tag{A.13}$$

There does not appear to be any particular motivation for the choice of $\alpha_g = \frac{\alpha_z}{2}$, other than that it is convenient for all three eigenvalues and hence the poles of the system to be at $-\frac{\alpha_z}{2\tau}$ and in fact, this choice of eigenvalues/poles was found to result in motion recognition errors in both the small and large motion data set.

The reason for the errors is that by placing all three poles of the system at the same location in the complex left half plane, a third order system is established. The step response of third order systems may have extra nonlinearities or oscillations not present in first order or second order system step responses. These nonlinearities or oscillations may have added variability to the weight parameters of the DMP which are used for motion recognition, thus, leading to motion recognition errors.

By moving the pole resulting from the $\frac{\alpha_g}{\tau}$ term in (A.12) towards the imaginary axis, the pole will become dominant and hence the overall system can be reduced to a first order system. Through experimentation, it was found that by setting $\alpha_g = \frac{\alpha_z}{9}$, the pole has been moved sufficiently towards the imaginary axis and no more motion recognition errors are found in the small motion data set. Some motion recognition errors continue to persist in the large motion data set, but they are due to other factors discussed in section A.6 of this chapter. For all subsequent experiments, $\alpha_g = \frac{\alpha_z}{10}$ was used to add a small safety factor to the experimentally found $\alpha_g = \frac{\alpha_z}{9}$ value. Note that these values are in close accordance with the rule of thumb that a factor of five will cause a pole to become dominant, i.e. $-\frac{\alpha_z}{10}$ is five times larger than $-\frac{\alpha_z}{2}$.

From the above discussion it can also be noted that the choice of the $\alpha_z$ value affects the settling time of the system, where larger $\alpha_z$ values result in faster settling times. The previously stated $\alpha_z = 15$ value works with the system when $\alpha_g = \frac{\alpha_z}{10}$. In an attempt to further improve the settling time, it was also attempted to move the pole resulting from the $\frac{\alpha_g}{\tau}$ term in (A.12) away from the imaginary axis so that the resulting system is a third order system with two dominant poles at $-\frac{\alpha_z}{2}$. However, it was found that the $\alpha_z$ value would have to be decreased in that case to maintain stability and the overall resulting settling time was not much improved. Thus, the system was kept at what is a third order system with one dominant pole at $-\frac{\alpha_z}{10}$.

## A.5 Gating Term, $v$

The purpose of the gating term $v$ is to ensure that the effect of the nonlinear component $f$ vanishes at the end of the movement, so that the overall system still converges to the goal $g$ [42], [86]. Assuming that the weights $w_i$ are bounded, it can actually be shown that the overall system asymptotically converges to goal $g$ [42], [86]. To model the gating term $v$, another second order linear dynamical system is used:

$$\tau \dot{v} = \alpha_v(\beta_v(g - x) - v), \quad v = \tau \dot{x} \tag{A.14}$$

where $x$ is a state variable that varies exponentially from 1 to 0 [34]. $g$ is the goal value of state $x$, i.e. 0 in this case. $\alpha_v$ and $\beta_v$ are time constants, where $\alpha_v = \alpha_z$ and $\beta_v = \beta_z$. This choice of $\alpha_v$ and $\beta_v$ matches the progression of the gating term values $v$ to that of the position values $y$, i.e. the gating term $v$ will approach zero as the position value $y$ approaches its goal. $\tau$ is the same temporal scaling factor as before.

Note that state $x$ varies nonlinearly with time $t$. This is to decouple the dependence of the nonlinear component $f$ with the progression of time in order to allow disturbance rejection [41]. State $x$ is described as a phase variable as it reflects the phase of the motion, e.g. beginning or end, and it acts as a substitute for time so that the nonlinear component $f$ is considered only dependent on state $x$ and independent from time $t$, which means that $\dot{f} = 0$. The assumption that $\dot{f} = 0$ simplifies training the DMP model.

## A.6 Gaussian Basis Functions, $\psi_i$

The Gaussian basis functions $\psi_i$ are strongly related to the weight values $w_i$ used for motion recognition. There are two sets of parameters concerning the basis functions $\psi_i$. One is the number of basis functions $N$ and the other is the set of parameters determining the exact location and shapes of the basis functions - the center $c_i$ and spread $h_i$. These two sets of parameters are discussed in more detail in the following subsections.

### A.6.1 Number of Basis Functions, $N$

Ideally, there should be one basis function $\psi_i$ for each Gaussian-shaped nonlinearity in the trajectory[1]. However, it would be difficult to predict the exact number of such nonlinearities and also to perfectly align the center $c_i$ and spread $h_i$ with the nonlinearities without manually adjusting the values for each motion exemplar, since there is temporal and spatial variation between each exemplar motion trajectory even when the underlying type of motion is the same. Thus, the number of basis functions used should be set to slightly higher than what is ideally necessary. The exact value of $N$ would have to be determined by trial and error.

If a high value of $N$ is used, i.e. a value much higher than the ideal number of basis functions, the accuracy of tracking the exemplar motion trajectory improves, but there may be overfitting to variability and noise in the exemplar motion trajectory, which may

[1]Nonlinearity here refers to valid nonlinear components of the trajectories not including noise.

cause problems in motion recognition. This was mainly observed in the large motion data set where an increase of $N$ from 5 to 7 improved the tracking accuracy of the trajectory, but an increase of $N$ from 7 to 10 introduced errors in motion recognition, as temporal variabilities within exemplars of the same motion type were reflected in the weights of the DMP. For example, if a certain movement starts earlier in some exemplars of a particular motion, given a high enough $N$, a different basis function located earlier in the progression of time substitute $x$ may be used for those exemplars to represent that movement. This means that the weights for that basis function will be different from the other exemplars of the same motion type for which a basis function located later in the progression of time substitute $x$ was used, thus, leading to problems in motion recognition.

If a low value of $N$ is used, i.e. a value close to the ideal number of basis functions, tracking of the exemplar motion trajectory may be poor and features of the different motion types will be lost. This would again cause problems in motion recognition, as different motion types that share certain similarities, e.g. a punching and a throwing motion, may be recognized as the same motion since not sufficient distinguishing features were picked up by the weights of the DMP. Moreover, a lower value of $N$ may lead to conflicts when a fast movement occurs towards the end of the motion trajectory, as the DMP attempts to both track the trajectory of the fast movement while still reaching the goal $g$ of the overall trajectory. In general, the DMP generated trajectories in that case will show relatively poor tracking of the trajectory of the fast movement towards the end of the motion trajectory (i.e. slower with less amplitude than the exemplar motion trajectory used for training) while potentially falling short of reaching the goal value $g$.

In general, there appears to be a range of values of $N$ for which motion recognition will work. It is recommended to choose the highest $N$ value within that range for best tracking performance. For the small motion data set, $N = 10$ was used during the experiments and for the large motion data set, $N = 5$ is used.

### A.6.2 Center and Spread, $c_i$, $h_i$

The center $c_i$ and spread $h_i$ values are fixed based on a heuristic [34], [41]. The center $c_i$ is modeled by an exponential function to match the exponential progression in state $x$:

$$c_i = (1 + \frac{\alpha_z}{2} t_i) exp(-\frac{\alpha_z}{2} t_i) \tag{A.15}$$

where $t_i$ is a discrete time value progressing linearly from 0 to 1 in $N$ steps. The centers $c_i$ determine the placement of the basis functions, which is very important for motion recognition.

Due to the exponential distribution of the $c_i$ values, more basis functions are placed towards the end of the motion trajectory. This allows greater accuracy for reaching the goal position $g$, but it also means that each basis function represents the nonlinearities of fewer data points in the trajectory towards the end of the trajectory. This leads to a higher susceptibility to variations such as noise or the previously mentioned angle switch. In particular when those variations occur in the very last data point of the trajectory which is aligned with the center of the last basis function and the weights of the basis function will be fitted to the noise or angle switch, causing problems with motion recognition. Furthermore, the placement of the basis function centers is sensitive to variations in timing, especially towards the end of the motion trajectory. Again, the reason is because towards the end of the trajectory, each basis function represents fewer data points of the motion trajectory.

This was observed in the large motion data set, where the first motion exemplar trajectory (representing a walking motion) had an angle switch at the very last value of the motion trajectory and that particular motion exemplar would always be placed in a cluster by itself during motion recognition. Moreover, there were three motions from another motion type (representing a cheering motion) in the large motion data set which displayed noise in the very last value of the motion trajectory and those three motion exemplars would be placed into a separate cluster from the other motion exemplars of the same motion type. Lastly, three motion exemplars of yet another motion type (representing a bowing motion) had a relatively fast movement towards the end of the trajectory and there was temporal variation in when this fast movement would start, i.e. for three of the motion exemplars the fast movement would start about twice as early as the rest. This also caused those three motion exemplars to be placed into a separate cluster from the other motion exemplars of the same motion type.

In order to address the problem of the last basis function overfitting to angle switches, noise and temporal variabilities, the centers of the basis functions are shifted slightly, by shifting the $t_i$ time values to progress linearly from 0 to 0.9999 in $N$ steps instead of from 0 to 1. This effectively shifts the center of the last basis function such that it is no longer exactly aligned with the last data point in the exemplar motion trajectory. The last data point is still covered by the basis function, just not by the center of the basis function. Theoretically, this will impact reaching the goal value $g$ perfectly, however, as the effect of the nonlinear component is reduced by the gating term towards the end of the trajectory, the impact should not be large and no significant difference in reaching the goal value is observed in the DMP generated trajectories when the basis function centers are shifted. There is more impact on reaching the goal position $g$ from fast movement towards the end of the trajectory as discussed in the previous subsection. Overall, the shifting of the basis function centers is found to remove the motion recognition problem caused by

noise and temporal variation, but not the angle switch. A combination of angle shifting as previously described and shifting the basis function removes all the motion recognition problems encountered with the large motion data set and is used in all further experiments.

The spread of the basis functions is computed such that there is a slight overlap between each basis function and their immediate neighboring basis functions:

$$h_i = \frac{0.5}{(0.55(c_i - c_{i+1}))^2} \tag{A.16}$$

The $h_i$ value is analogous to the inverse of the variance of a Gaussian normal distribution. By increasing the 0.55 value in (A.16), the variance of the basis function would be increased and there would be more overlap between basis functions and their immediate neighboring basis functions. This should improve problems with timing variabilities between different motion exemplars of the same motion type, however, from experimental results when the 0.55 value is increased to 0.8 the improvement is found to be minor, i.e. not enough to remove problems during motion recognition. Thus, the value of 0.55 is retained during all further experiments.

# Appendix B

# Object Feature Extraction Implementation

Feature extraction, i.e. obtaining the values of relevant object features from video stream data, is necessary for both the motion and object recognition portions of the learning by imitation system. For motion recognition, the extracted object positions form the motion trajectories used for motion segmentation, modeling and recognition. For object and object state recogntion, various extracted features are used. Video stream data is used because both motions and object features can be extracted at once. The feature extraction algorithm involves the following simple computer vision techniques:

1. *Channel Separation:* First of all, each incoming frame of the video stream is separated into single channels. For a RGB video there are three channels, each corresponding to the color values (ranging between 0 and 255) of either red, green or blue. Separating a RGB video frame will result in three separate frames where objects containing the colors red, green or blue stand out.

2. *Channel Subtraction:* Secondly, for detecting objects that are purely red, green or blue. In this step, the frames containing two of the color channels are added and then subtracted from the third color channel in order to let objects of the third color stand out even further. For example, for the red channel $red_{new} = red_{channel} - (green_{channel} + blue_{channel})$ will let red objects stand out further, as objects that contain colors other than red will appear in the green and blue channels as well and consequently be subtracted. This subtraction is performed for each color channel, resulting in three separate frames in which objects that are purely red, green or blue will stand out.

3. *Thresholding:* Then, to convert each of the three separate frames from the previous step into a binary representation, i.e. purely black and white image. This is achieved through the operation shown in (B.1).

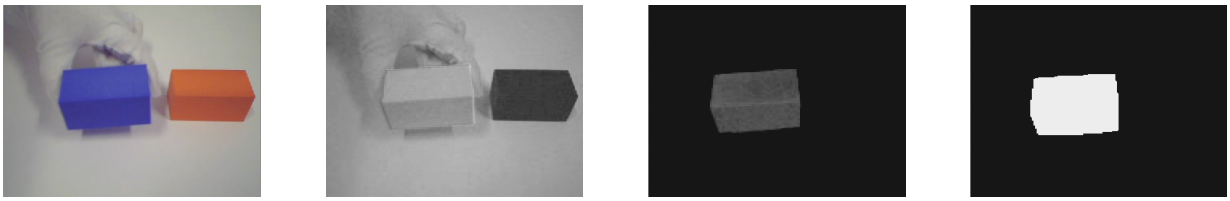
$$p_{new}(x,y) = \begin{cases} 255 & \text{if } p_{old}(x,y) > \text{threshold} \\ 0 & \text{otherwise} \end{cases} \tag{B.1}$$

   Where $p$ refers to the pixel at coordinates $x$ and $y$.

4. *Contour Extraction:* Next, all the pixels along the contours are extracted in each of the three frames that have been converted to a binary representation.

5. *Bounding Rectangle Extraction:* This step will determine the smallest rectangles that will bound each of the contours found in the previous step. These rectangles are treated as the objects found and the coordinates of their center of mass ($x$ and $y$ in Table 5.1) and their width and height ($w$ and $h$ in Table 5.1) are easily determined. The color of the object bounded by the rectangle is simply the channel color of the frame ($R$, $G$ or $B$ in Table 5.1) that the rectangle was found in.

Thus, the outputs of the feature selection process are the values of the pre-selected features for each of the objects found in each frame. Using the $x$, $y$, $w$ and $h$ values extracted, a flag indicating whether there is an object stacked on top or placed below the current object can be computed. The difference between the $x$ and $y$ coordinates of two stacked objects should not exceed the sum of half of the width of each object and the sum of half of the height of each object, respectively. Figure B.1 illustrates in detail the results of each step of the feature extraction process.



(a) Original video frame

(b) Blue channel

(c) Subtracting green and red channels

(d) Thresholding

Figure B.1: Example video frame of a 2 block stacking task and the process of extracting the blue block

# Appendix C

# Petri Net Task Learning Implementation

The implementation of the learning by imitation system uses a combination of C++ and Matlab code. The C++ code allows for faster computations and easier interfacing with the actual robot, whereas the Matlab code allows for easier coding and debugging. Figure C.1 below shows the code setup, including how the C++ and Matlab code interact. Italicized text denotes a Matlab data structure in Figure C.1.

Each separate block in Figure C.1 represents a stand-alone executable. Initially, the C++ code (part 1) will take a video as input file and extract from it the objects (by finding an object ID using the object SOM), states of objects (by finding a state ID using the state SOM), positions of objects (in $m$) and a flag indicating whether an object is moving or not for each frame of the video. The C++ code (part 2) will take two images as input and extract from them the objects and their states. This extracted information is stored in files (extractedData.txt and extractedTask.txt) that will be used as input to the Matlab code. The Matlab code (part 1 - training) will create a PN based on the extracted information, as well as determining the initial and final (desired) marking. The PN, initial and final markings are then used as input to the Matlab code (part 2 - generation) to construct a reachability tree and find a path in the reachability tree between the initial and final marking. This path is stored in a file that serves as input to the C++ code (part 3) which parses the path and generates the motions necessary for the robot to perform the task.

The following sections give the pseudocode for the Matlab PN training and generation implementation.

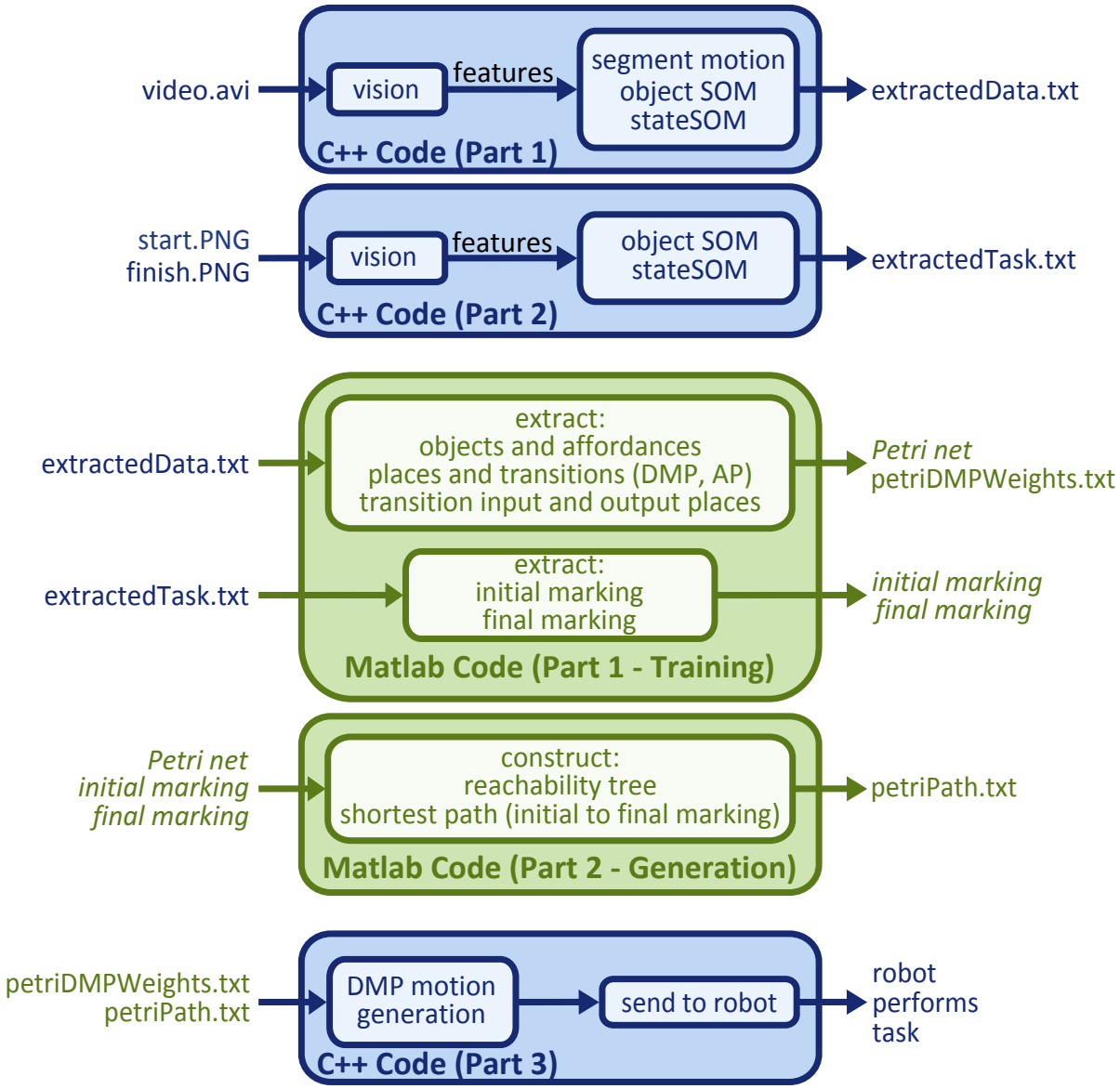



Figure C.1: Overview of the task learning code

## C.1 Petri Net Training

Algorithm C.2 describes the Matlab code (part 1 - training) which takes in the files extractedData.txt and extractedTask.txt as input and outputs a PN structure along with the markings $\mu_{initial}$ and $\mu_{final}$ stored as Matlab matrices and vectors. Furthermore, the code stores the objects and their affordances in a separate structure and outputs the file petriDMPWeights.txt containing the weights of the DMP model. For the case of multiple video demonstrations, multiple extractedData<*>.txt files with different file endings <*> can be generated and used as input to the Matlab code (part 1 - training).

**Algorithm C.2** PN training

```
1: parse PN from extractedData.txt:
2: store object ID from extracted data (frame 1) in object struct
3: for all frames do
4:    store state ID of current object from extracted data (current frame)
5:    for all objects do
6:       if none of the objects are moving then
7:          if object was previously moving with max. displacement then
8:             linear interpolation of motion trajectory
9:             train DMPs (one for each of the two DoF) with 10 weights and α_z = 15
10:            stack DMP weights of each degree of freedom into one vector
11:            stack DMP weights of all motions seen so far into one matrix
12:            if too few (<7) clustering data points[1] then
13:               repeat DMP weights and add random noise
14:               run AP algorithm on DMP weights
15:            else
16:               run AP algorithm on DMP weights
17:            end if
18:            for all t_i in T do
19:               if AP returns updated cluster exemplar then
20:                  update t_i value in T
21:                  update object affordance in object struct
22:               end if
23:            end for
```

[1]It is difficult to determine clusters of data points when there are only a few data points available, so the available data points are repeated once each with a small amount of noise (in the order of $10^{-3}$).

```
            if new transition $t_i$ then
              for all objects do
                if state change and/or actual motion then
                  add transition $t_i$ to $T$
                  add transition $t_i$ to object affordance
                end if
              end for
            else
              for all objects do
                if state change and/or actual motion then
                  if new input place $p_i$ and/or new output place $p_j$ then
                    add $p_i$ to $I$ and/or $p_j$ to $O$
                  end if
                end if
              end for
            end if
          end if
          if current state ID is not in $P$ then
            add current state ID as place $p_i$ to $P$
          end if
        else if object is moving then
          store object trajectory
        else
          clear object trajectory
        end if
      end for
    end for
convert $P$ and $T$ from set to matrix notation (store both notations)
parse $\mu_{initial}$ and $\mu_{final}$ from extractedTask.txt:
match state IDs to places
if transition is enabled and an object affordance then
  set places $p_i$ in $\mu_{initial}$ or $\mu_{final}$ to 1
end if
write DMP weights to petriDMPWeights.txt:
for $t_i$ in $T$ do
  write DMP weights, centers and spread to file
end for
```

As can be seen from Algorithm C.2, extracted object IDs, state IDs and motions are used to create a PN. From the first video frame, each object ID is stored in an object structure, which will later store the affordances (motions) associated with each of the object IDs. Each time a new object state is observed in the video, the state ID of the object is added to $P$ as a new place $p_{n+1} = state\ ID$. By storing this state ID, the system is able to determine if the objects in the environments are in the correct state during task imitation. This is necessary for error detection later. A new object state is simply a state ID not already present in $P$. Each time a new motion is observed, it is added to $T$ as transition $t_{n+1} = motion\ index$. A new motion is simply a motion index not already present in $T$. Each new motion is also added to the object structure as an affordance of the according object $object(o_i) = motion\ index$. Storing the motion index, allows the correct set of DMP weights to be used later for motion generation during robot task imitation. Additionally, the state ID immediately preceding the transition $t_{n+1}$ will be stored as an input place in $I(t_{n+1}) = state\ ID$ and the state immediately following will be stored as an output place in $O(t_{n+1}) = state\ ID$. Each time a previously seen motion has a new representative cluster exemplar, the motion index of the transition $t_i$ will be updated to the motion index of this new motion exemplar. The object affordance will also be updated. Additionally, if new preceding or following states are observed for previously seen motions, the new states are added as input and output places to $I(t_i)$ or $O(t_i)$, respectively. Again, a previously seen motion is a motion whose motion index is already in $T$ and a new preceding or following state is a state ID not already in $I(t_i)$ or $O(t_i)$, respectively. This process effectively constructs a PN that reflects the sequence of actions for performing the demonstrated task. Lastly, given a desired initial and final place, an initial and final marking $\mu_{initial}$ and $\mu_{final}$ is determined.

## C.2 Petri Net Generation

Matlab code (part 2 - generation) generates the reachability tree and finds a path between $\mu_{initial}$ and $\mu_{final}$. A PN serves as input to the code along with $\mu_{initial}$ and $\mu_{final}$ for the desired initial and final marking. The initial marking $\mu_{initial}$ is used to create the root node of the reachability tree. The **addNode** function shown in Algorithm C.3 is then called to construct the reachability tree recursively, taking into account the two types of leaf nodes: repeated and terminal. Once the reachability tree is constructed, it is stored as a Matlab data structure and the **findPath** function shown in Algorithm C.4 is called to determine the path(s), if any, between $\mu_{initial}$ and $\mu_{final}$ via depth first traversal of the reachability tree. The shortest path is then selected from the path(s) found. If there are

multiple paths with the same length, the first one found is used. The shortest path is then written to the file petriPath.txt, to be used as input to the C++ code (part 3). It should be noted that $\mu_{initial}$ and $\mu_{final}$ need not be the actual initial and final states of the human demonstration, as any place in the PN can be picked as $\mu_{initial}$ and any place reachable from $\mu_{initial}$ can be picked as $\mu_{final}$. Thus, the flexibility and modeling power of the PN should allow the robot to do more than just exactly imitate a task.

**Algorithm C.3** PN generation **addNode** function

```
1: function addNode
2: for all t_i in T do
3:    if t_i is enabled then
4:       compute resulting μ from t_i firing
5:       add node for resulting μ
6:       if added node is a repeated node then
7:          link to first node with same marking
8:       end if
9:       if added node is not a repeated node then
10:         call addNode to recurse
11:      end if
12:   else
13:      if no child nodes found then
14:         set parent node to terminal node
15:      end if
16:   end if
17: end for
18: end function
```

**Algorithm C.4** PN generation **findPath** function

```
1: function findPath
2: if current node marking is μ_final then
3:    return
4: end if
5: if current node is a repeated node then
6:    follow link to first node with same marking
7:    continue
8: end if
9: if current node is a terminal node then
10:   return
11: else
12:   for all child nodes do
13:      add transition to path
14:      call findPath to recurse
15:   end for
16: end if
17: end function
```

# Appendix D

# Petri Net Error Recovery Implementation

In order to support extensions to the PN for error recovery, the proposed PN training Algorithm C.2 for the Matlab code (part 1 - training) described in Appendix C, Section C.1 is extended with the additional code shown in Algorithm D.5. The learning by imitation system keeps track of the object states in the environment during task performance, so the state(s) right before the transition is known when a failed transistion is encountered. A failed transition is stored with a flag indicating that it had failed. The subsequent error state(s) and the motion for error correction is extracted from the video recording of the human demonstration along with the resulting known state(s) following the error correction. The above information is stored in the extractedData.txt file to be used by the PN training algorithm. The extension to the PN algorithm serves the purpose of acknowledging the failed transition so that subsequent error state(s) can be added as output place(s) to this transition without having to train a new DMP model on the failed transition. Furthermore, to allow for very short error correction motions, i.e. motions that contain fewer data points than the 10 DMP weights used, a separate DMP for motion generation is trained using a smaller number of weights, i.e. 5 weights, to prevent the DMP weights from overfitting to the noise in the data and generating noisy motion trajectories. For motion recognition the number of DMP weights used remains 10. The pseudocode in Algorithm D.5 replaces lines 7 to 9 in the task learning PN training Algorithm C.2.

**Algorithm D.5** PN error recovery

```
1: if previous transition t_i failed then
2:    if t_i is a known transition then
3:       assign a new transition t_j for the failed t_i transition
4:       set the input places of t_j to the input places of t_i
5:       for all objects do
6:          if state change and/or actual motion then
7:             add current state p_j to output places of t_j
8:          end if
9:       end for
10:   else
11:      output warning that failed transition is unknown
12:   end if
13: else
14:   if object was previously moving with max. displacement then
15:      linear interpolation of motion trajectory
16:      if too few (<10) data points in motion trajecetory then
17:         train DMPs (one for each of the two DoF) with 4 weights and α_z = 5
18:         store the trained DMP weights for motion generation
19:      end if
20:      train DMPs (one for each of the two DoF) with 10 weights and α_z = 15
```