LEARNING TEMPORAL SENSORIMOTOR DISTRIBUTIONS OF COMPLEX ROBOTIC MANIPULATION SKILLS WITH CONDITIONAL NEURAL PROCESSES

by

Muhammet Yunus Şeker

B.S., Computer Engineering, Boğaziçi University, 2017

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2020

ACKNOWLEDGEMENTS

I would like to thank my advisor Assist. Prof. Emre Uğur for his support and guidance throughout my master's study. His endless support, advice, encouragement, and motivation helped me a lot to be able to complete this master thesis work. My sincere gratitudes to Prof. Lale Akarun and Assist. Prof. Barış Akgün for kindly accepting to be in my thesis jury.

I would like to thank to my friends Mert İmre for his incredible contributions on sleepless nights, and Mert Tiftikci for delivering us delicious night snacks. My sincerest gratitude to Ahmet Ercan Tekden, Melisa İdil Şener, Tuluhan Mete Akbulut, Tuna, Han, Salih, Meral, Safa Andaç, Erhan Girgin, Hakan Girgin, and Serkan Buğur for their endless friendship and support anytime, anywhere. I feel privileged to be able to work with the members of the CoLoRs research group. Many thanks to Alper Ahmetoglu, Ece Suzan Ada, Hamit Basgol, Utku Bozdogan for the deep academic brainstorms, and joyful frozen chai tea latte breaks. Last but not least, I would like to thank to my dearest high school friends, Mustafa Sabanci, Ari Ali, Fatih Can, Firat Yilmaz, Mustafa Erdogan, and Mehmet Ozdemir for absolutely no reason but somehow being ended up written down here as my best friends.

I would like to thank my parents, my little baby brother, and sisters for their genuine support and encouragement during my whole life. I would have never been able to persevere in my education without them. Needless to say, many loves to my fiancée Nur for being a cute little cotton candy during the whole process of my Master's study.

ABSTRACT

LEARNING TEMPORAL SENSORIMOTOR DISTRIBUTIONS OF COMPLEX ROBOTIC MANIPULATION SKILLS WITH CONDITIONAL NEURAL PROCESSES

This thesis proposes a new framework, namely Conditional Neural Movement Primitives (CNMPs) that is a learning from demonstration framework designed as a robotic movement learning and generation system built on top of a recent deep neural architecture, Conditional Neural Processes [1] (CNPs). CNMPs extract the prior knowledge directly from the training data by sampling observations from it, and uses it to predict a conditional distribution over any other target points. CNMPs specifically learns complex temporal multi-modal sensorimotor relations with external parameters and goals; produces movement trajectories in joint or task space; and executes these trajectories through a high-level feedback control loop. Through simulations and real robot experiments, we showed that CNMPs can learn the non-linear relations between low-dimensional parameter spaces and complex movement trajectories from few demonstrations; and they can also model the associations between high-dimensional sensorimotor spaces and complex motions using large number of demonstrations. The experiments further showed that even the task parameters were not explicitly provided, the robot could learn their influence by associating the learned sensorimotor representations with the movement trajectories. The robot, for example, learned the influence of object weights and shapes through exploiting its sensorimotor space that includes proprioception and force measurements; and be able to change the movement trajectory on the fly when one of these factors were changed through external intervention.

ÖZET

KOŞULLU SİNİR SÜREÇLERİ İLE KOMPLEKS ROBOTİK YÖNETİM BECERİLERİNİN ZAMASAL SENSÖR-MOTOR DAĞILIMLARININ ÖĞRENİLMESİ

Bu tez, derin bir sinir mimarisi olan Koşullu Nöral Süreçler [1] (CNP'ler) üzerine inşa edilmiş bir robotik hareket öğrenme ve üretim sistemi olan Koşullu Nöral Hareket İlkelleri (CNMP) adında yeni bir öğrenim çerçevesi sunmaktadır. CNMP'ler, ön bilgileri, gözlemleri örnekleyerek doğrudan eğitim verilerinden çıkarır ve diğer hedef noktalara göre koşullu bir dağılımı tahmin etmek için kullanır. CNMP'ler spesifik olarak harici parametreler ve hedeflerle karmaşık zamansal çok modlu sensörimotor ilişkilerini öğrenir; eklem veya görev alanında hareket yörüngeleri üretir; ve bu yörüngeleri üst düzey bir geri besleme kontrol döngüsü ile hayata geçirir. Simülasyonlar ve gerçek robot deneyleri sayesinde, CNMP'lerin düşük boyutlu parametre uzayları ve karmaşık hareket yörüngeleri arasındaki doğrusal olmayan ilişkileri birkaç gösteriden öğrenebildikleri; ve ayrıca çok sayıda gösterim kullanarak yüksek boyutlu sensörimotor uzaylar ve karmaşık hareketler arasındaki ilişkileri modelleyebildikleri gözlenmiştir. Deneyler ayrıca görev parametrelerinin bile açıkça sağlanmadığı zaman, robotun öğrenilen sensorimotor temsillerini hareket yörüngeleriyle ilişkilendirerek öğrenebileceğini göstermiştir. Örneğin deneylerden birinde robot, propriyosepsiyonu ve kuvvet ölçümlerini içeren sensorimotor alanından yararlanarak nesne ağırlıklarının ve şekillerinin harekete etkisini öğrendi ve bu faktörlerden biri harici müdahale ile değiştirildiğinde hareket yörüngesini anında değiştirmeyi başarmıştır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS

A	Averaging layer
C_i	Corner coordinates of the ith corner
E	Parameter sharing Encoder network
F_x	X coordinate of the applied force
F_y	Y coordinate of the applied force
n_{max}	Observation hyperparameter
Q	Query network
R	Ring number
r_i	Latent space representation of observation i
r	Latent space general representation
SC_i	Shape context feature vector
$SM(t)$	Sensorimotor paramater space
t	timestep
W	Wedge number
x_i	X coordinate
y_i	Y coordinate
γ	External task parameters
μ_q	Predicted mean query value
σ_q	Predicted variance query value
ϕ	Neural network weights
θ	Neural network weights

LIST OF ACRONYMS/ABBREVIATIONS

1D	One Dimensional
2D	Two Dimensional
3D	Three Dimensional
ANN	Artifical Neural Network
CNN	Convolutional Neural Network
CNMP	Conditional Neural Movement Primitives
CNP	Conditional Neural Processes
DMP	Dynamic Movement Primitive
DOF	Degrees of Freedom
GMMs	Gaussian Mixture Models
GPU	Graphics Processing Unit
HMM	Hidden Markov Model
LfD	Learning from Demonstrations
LSTM	Long Short-Term Memory
LWR	Locally Weighted Regression
MATLAB	Matrix Laboratory
MLP	Multilayer Perceptron
PHMM	Parametric Hidden Markov Model
ProMP	Probabilistic Movement Primitives
RNN	Recurrent Neural Network
ROS	Robot Operating System
SM	Sensorimotor
SVM	Support Vector Machines
UR10	Universal Robot 10
V-REP	Virtual Robot Experimantation Platform

1. INTRODUCTION

Acquiring an advanced robotic skill set requires a robot to learn complex temporal multi-modal sensorimotor relations in connection with external parameters and goals. Learning from demonstration (LfD) framework [4] has been proposed in robotics as an efficient and intuitive way to teach such skills to the robots, in which the robot observes, learns and reproduces the observed demonstrations. During teaching the skills, how the task is influenced by different factors is generally obvious to humans, yet this knowledge is mostly hidden in the experienced sensorimotor data of the robot and difficult to extract autonomously. Development of feature extraction and learning methods that are sufficiently general and flexible for a broad range of robotic tasks still stands as an important challenge in robotics. In order to deal with large variety of tasks that are influenced by factors defined in different levels of abstractions, we require a single framework that can learn feature-movement and raw-data-movement associations from small and large number of demonstrations, respectively, automatically filtering out the irrelevant information.

Another challenge in LfD is to deal with multiple trajectories: Multiple demonstrations might be required to teach a skill either because different action trajectories are required in different situations or simply because there are multiple means to achieve the same task even in the same settings. The robot, in return, needs to capture the important characteristics in these observations coming from several demonstrations, and be able to reproduce the learned skill in new configurations reacting to unexpected events, such as external perturbations, on the fly. For this, the robot needs to develop an understanding on if and how the sensorimotor data and externally provided parameters are related to each other and to the motor commands.

While one or more of the above properties were addressed by the existing movement frameworks [5–11], none of these approaches can handle all these requirements in one framework. In this thesis, we propose Conditional Neural Movement Primitives (CNMP), a robotic framework that is built on top a recent neural network architec-

ture, namely conditional neural processes [1] (CNP), to encode movement primitives with the identified functionalities. Given multiple demonstrations, CNMP encodes the statistical regularities by learning about the distributions of the observations from reasonably few input data. Conditioning mechanism is used to predict a sensorimotor trajectory given external goals and the current sensorimotor information. Conditioning can be applied over the learned sensorimotor distribution using any set of variables such as joint positions, visual features or haptic feedback at any time point. For example for a learned grasp skill, the system might be queried to predict and generate hand and finger motion trajectory with a conditioning on the color of the object, weight measured in the wrist joint, and target aperture width for the finger. Given low- or high-dimensional input, CNMP can utilize simple MLPs or convolution operation, respectively, to encode the correlations. Such networks allow the system to automatically extract the features that are relevant to the task. Finally, the predicted trajectory is realized by generating the corresponding actuator commands. CNMP can also learn multiple modes of operations from multiple demonstrations of a movement primitive. Importantly, CNMP specifically produces movement trajectories in joint or task space, and generates the corresponding motor commands to achieve these trajectories through a high-level feedback control loop. The encoded sensorimotor representations are continuously conditioned through the feedback coming from the sensory readings, enabling the system to ensure match between predicted and actual trajectories and to respond to unexpected events in its sensorimotor space.

In Chapter 2 the work-space we have used in the experiments, and software tools that our method is coded with will be introduced. Chapter 3 provides the preliminary works we have been studying on as a starting point of this thesis. In Chapter 4, the state-of-the-art frameworks of learning from demonstration area will be introduced. The related background information that is required to fully comprehend this study will be given in Chapter 5. The proposed framework will be introduced in Chapter 6. The experiments and their setups, and results will be presented in the Chapter 7. As our final points, the discussion and the conclusion chapter will take place in the Chapter 8.

2. EXPERIMENT PLATFORM

In this chapter, All of the components that we used in this thesis is introduced with details to provide a better understanding of the experiments, environments and implementations. Section 2.1, introduces the physical components used in the study, and Section 2.2 explains the software components in detail.

2.1. Physical Components

2.1.1. UR10 Robot

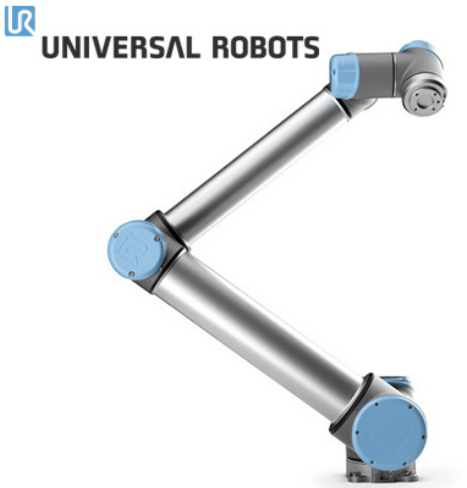

Figure 2.1. UR10 robot.

In this thesis, The UR10 robot of Universal Robots is used [12]. UR10 is a community oriented mechanical robot arm intended for tasks which require high accuracy. It can carry up to 10 kg and has 1300mm range of a work-space. The UR10 is utilized in collaborative tasks, such as, bundling, pelleting, pick and place, and packing. It has 6 degrees of freedom (DOF), implying that the arm has 6 joints. Every one of the joints of the UR10 is rotational joint. In 3D space, 6 DOF is required to have the solution to arrive at each point in the working space, and the joints of the UR10 is arranged and equipped upon this requirement. Unlike the human arms, the UR10 only has a single solution for its joints on one 3D point in the space. The UR10 robot has no kinematic redundancy for this reason.

2.1.2. 3F Robotiq Gripper

Figure 2.2. Three finger Robotiq gripper.

In our experiments, we used three fingered Robotiq gripper. The gripper is easily attached to the UR10 robot. It has internal control mechanism and low level force sensors. It is compatible for robotic research and manufacturing. Thanks to its design, it can adapt to the shape of the object and grasp it solidly. The gripper has 4 modes: pinch mode, wide mode, scissor mode and basic mode. In our experiments, we used pinch mode to grasp small objects. The gripper can be connected and controlled to ROS via a Python package very easily. By using the API programmed for the gripper, we coded a driver in Python to collect demonstrations in the real world.

2.1.3. Robotiq Force Sensor

In our real world experiment, we trained our system with force data in order to test our system with changing weights on the fly. To collect the force data while collecting demonstrations, we used Robotiq Force Sensor attached under the robot gripper, and connect the sensor to our PC using USB. Internal ROS API of the sensor allowed us to record the sensor information real time.

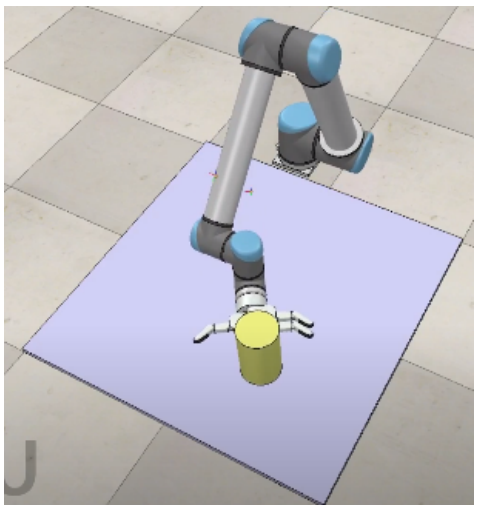

Figure 2.3. V-REP simulation environment with UR10 robot.

2.2. Software Components

2.2.1. Robot Operating System (ROS)

ROS is an open-source, meta-operating framework that allows information sharing between processes. It gives tools and libraries for getting, building, composing, and running code over different computers. Processes are ran under nodes, and ROS framework provides topics and services in order to send special types of messages between these nodes. In our experiments, we used ROS in order to communicate with the UR10 robot and the gripper. We also collect force sensor data from the robots hand via ROS system.

2.2.2. V-REP

The robot simulator V-REP, with coordinated environment, is suitable for development, and it is based on a disseminated control design. Each object/model in the simulator can be separately controlled by embedded scripts, plugins, ROS or BlueZero nodes, explicit API clients, or custom arrangements. V-REP is exceptionally flexible and perfect for multi-robot applications. Controllers can be coded in C/C++, Python, Java, Lua, Matlab or Octave.

2.2.3. Tensorflow and Keras

Tensorflow is an open-source system for machine learning and especially deep learning. Tensorflow has been created to encourage open research and experimentation. It encompasses a comprehensive, adaptable environment of instruments, libraries and community assets that lets analysts push the state-of-the-art in machine learning applications. The aim of the framework is providing an approach to machine learning where a shared worldwide knowledge is constructed. Additionally, being a high-end library of Tensorflow, Keras is an API intended for all types of people.Keras takes after best practices for reducing intellectual burden. The Framework design offers steady and basic APIs. For basic use cases, it significantly reduces the time and code spend for constructing deep learning systems with the help of broad documentations and designer guides.

2.2.4. V-REP RosInterface Plugin

V-REP has a additional plugin that is called RosInterface in order to allow developers to communicate between ROS and V-REP environments. In our experiments, we used RosInterface plugin to control the robot from outside of the simulation via ROS messages.

2.2.5. Jupyter Notebook

Jupyter notebook is a basic python editor environment where the code cells are independent from each other to allow developers to code in more flexible environments. In our implementations we used Jupyter notebook to develop our system and documented it with the help of the editor.

3. PRELIMINARY WORK

In this section, preliminary works that provide a starting point to our main study are introduced. While learning from demonstrations, for most of the basic scenarios, the robot uses only proprioception information to find a smart representation of the given movements. However, in more complex scenarios where the robot additionally has to find the relationships between its movement and the external environment, the robot has to find a way to represent the outer world data in a way that it can be used as a feature in the learning process. Thus, our preliminary work focuses on finding smart representations of environment objects, which is a widely studied topic in affordance and effect prediction.

The first study is about a geometric shape descriptor and its usage in affordance prediction scenarios. The second study is about finding smart high-level representations for the shapes in an environment that the robot tries to do effect predictions upon using sample actions.

3.1. Action-Effect Prediction Using Shape Context

3.1.1. Motivation

With the development of robotic technologies, robots are now interacting more with the outside world. In this case, the ability of robots to anticipate the effects of the actions they perform or observe on objects in the outside world can contribute to the robot's ability to plan its actions and continue to work depending on the effects of its actions. Our goal in this study is to predict the outcomes of the effects of difficult physical actions and to adjust the actions according to these predictions as the robot interacts with the environment.

The action-effect estimation can contribute to interactive learning robot applications and can be used to learn various actions and tasks. An example of this type of

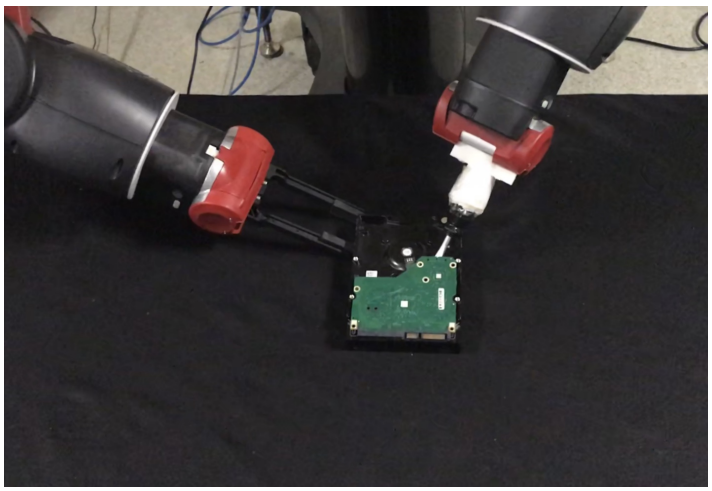

Figure 3.1. Baxter robot performing lever up action on PCB module that is top of the hard drive.

learning task is the disassembly of hard drives, one of the use cases of the IMAGINE project we are working on. The aim here is to estimate the motion of some complex objects, to use these estimates in planning to complete a more complex task, and to check whether the action has occurred as planned. In this case, the effect estimation can be done with relatively simple regression methods, or with more complex trajectory estimation methods such as Long Short-Term Memory (LSTM) [13] and Hidden Markov Model (HMM) [14].

At this point, we need a generic, comprehensive descriptor that can represent the physical properties of the interacting objects, to be provided as an input into complex trajectory estimation methods. In this section, we propose shape contexts to be used as a descriptor [15]. Shape contexts can represent the general physical information of objects, thereby enabling similar shapes with similar physical properties to be distinguished from other [16]. Therefore, when the shape context descriptor is combined with the force information contained in the actions, the estimation of the effect on the object can reach a predictable state.

In this preliminary work, force information and shape contexts are used together as feature vectors and these attributes are trained by machine learning methods. It is shown that robot can predict the effects of its actions on objects that it has not seen

before. Lever up movement is selected as an example action for this work to show that a learning model based on shape context features can successfully predict which edge of the object stays on the ground and supports the levering up action.

3.1.2. Related Work

In this subsection, we will mention two case studies which used the shape context as a descriptor. The first paper is about a topic that can be very related to the robotic applications by enhancing the interaction capabilities of the robot with the environment which is 3D object recognition. The second paper also shows that the grasping point estimation while a robot performs grasping action can be predicted successfully by using shape context descriptors. By improving the action-effect prediction, the robot can understand from which points it should grasp to get the desired effect, allowing the robot to plan its actions.

Shape Matching and 3D Object Recognition Using Shape Context. In this paper, Belongie and Malik propose a new approach to measure the similarity of objects and show that this approach can be used in 3D object recognition as a descriptor. [17]. Their similarity measurements are based on two steps: 1) Finding the possible corresponding points on two shapes. 2) From these possible corresponding points estimating an aligning transform. To solve the problem of finding corresponding points, shape context descriptor is being used on all sample points. Shape context on a reference point captures the distributions of other sample points relative to itself, so it offers discrimination between different shape characteristics. Two similar corresponding points give similar shape contexts, so it makes it possible to find optimal alignments on the correspondence points. The dissimilarity between two shapes is measured based on a term that defines the magnitude of the aligning transform and the sum of matching errors between corresponding points. They treat recognition classification as a problem of designing a framework that finds the most maximally similar object from the stored prototype objects. Experiment results show that shape context is a stable descriptor on object recognition applications.

Grasping Point Calculation Using Shape Context. The main idea in the work of Bogh and Kragic is vision-based object grasping [18]. The proposed method is based on obtaining and representing the global contour of an object via computer vision methods. The robot learns several grasping points with some example objects and the learned model is used to generate feasible grasping points on novel objects. Shape context feature descriptors are used for representing objects and grasping points. Shape context descriptors and images with grasping points marked beforehand are provided into supervised machine learning methods. This work showed that, when shape context features are combined with nonlinear classification methods, they can predict grasping points of objects accurately and robustly.

3.1.3. Method

In this work, we follow the methodology below to show that shape context features significantly improves the results in predicting action-effects of lever-up action performed by the robot. First, we created random shapes and selected random points on all edges, on which robot can perform lever-up actions. Then, all shapes and generated points are transferred into the simulation environment. On all edges of all shapes, lever-up actions are performed in the simulation environment, resulting in support points are saved, which formed our dataset. For all vertices of all shapes, shape context features are computed and included in our dataset. Lastly, we trained a learning model with our dataset, utilizing machine learning methods. We measured the performance of predicting the supporting edge which results from the lever-up action. The following subsections explain the methods in detail.

Generating Random Shapes and Force Points. To investigate the generalizability and role of the shape context on action-effect prediction, a dataset that consists of randomly generated shapes is constructed. To generate the shapes in the dataset, a circle its radius is selected randomly is drawn and the desired number of corner points are picked randomly on the circle. Corners are enumerated according to the angle between the reference frame of the circle and the vectors that are constructed separately

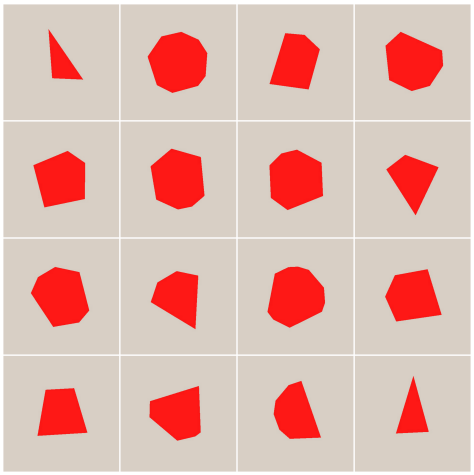

Figure 3.2. Randomly generated sample shapes.

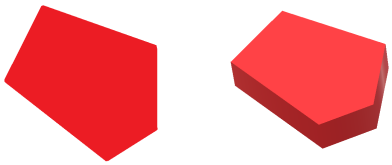

Figure 3.3. A randomly generated shape and its simulation model.

from the center of the circle to each point. Bonding the consecutively numbered corners, a polygon with the desired number of corners is constructed.(Figure 3.2). Finally, force points are also selected randomly on the random edges of the shapes.

By repeating this process for the desired number of shapes and randomly selecting corner numbers, 100 shapes, and 5 force points for each shape are generated and recorded.

Constructing a Dataset Using Simulator. Using the generated data for random shapes, we created object model files and imported them into a simulation environment (Figure 3.3). V-REP [19] with Bullet 2.83 Physics Engine is used as a simulation environment. We place the generated shape on the surface of a table created in the simulation environment, ensuring that the shape is stable on the table. An artificial force perpendicular to the surface normal of the given shape is applied from the points on the edges, which were selected after shape generation(Figure 3.4). The aim is to

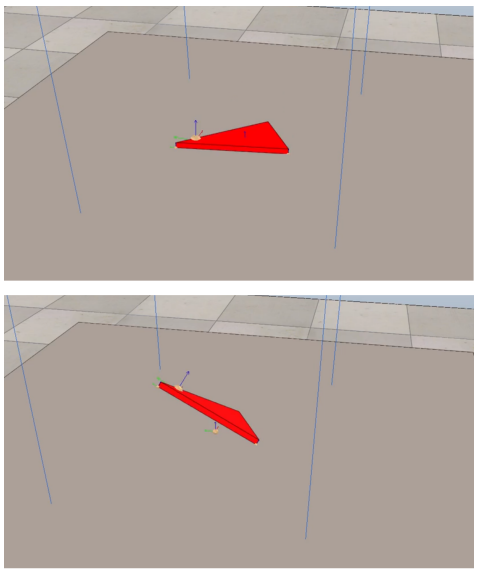

Figure 3.4. An example of lever-up experiment with a triangle.Lever-up action is performed from the indicated point of force. The edge which supports the lever-up is observed.

mimic the lever-up action that the robot performs a from an edge of the particular shape.

After applying the force, we saved the indices of the vertices that support the lever-up action, following the same vertex indexing convention in the previous section. This operation is systematically repeated for every force point of all 100 shapes, forming a dataset consisting of the shape, applied force, and a binary label for each vertex of that shape, indicating whether that vertex supported the lever-up action or not. For example, Table 3.1 shows the format of an experiment result involving a hexagon, with force applied at the point (0.04, 0.08) and vertex 2, and 3 are support vertices.

At the table, C_i represents the (x,y) coordinates of the $i'th$ corner. $(0.04, 0.08)_{C_i}$ represents the coordinates of the applied force, relative to the reference frame whose origin is the $i'th$ vertex, and the y-axis is the line from i'th vertex to (i-1)'st vertex. Since we represent the coordinates of the forces relative to the reference frame of the particular vertex, we obtain the positions invariant of shape rotation.

Table 3.1. The format of an example hexagon data.

Coordinates of the applied force	**Corner coordinates**	**Corner Labels**
$(0.04, 0.08)_{C_1}$	C_1	0
$(0.04, 0.08)_{C_2}$	C_2	1
$(0.04, 0.08)_{C_3}$	C_3	1
$(0.04, 0.08)_{C_4}$	C_4	0
$(0.04, 0.08)_{C_5}$	C_5	0
$(0.04, 0.08)_{C_6}$	C_6	0

Calculating Shape Context. Shape context is a descriptor that can represent the shapes with broad strokes and express the physical properties of them by sampling the contour points of shapes. At first, the contour points of the shape are calculated and then these points are sampled uniformly through the edges. The shape context feature is calculated based on a reference point on the contours of the shape, and in our work, these reference points are selected as corner points. After selecting the reference corner point, all vectors that start from the reference corner to all other sample points are calculated(Figure 3.5). These vectors are holding the information about the distance and the angle from the reference point to others. The fact that sample points are uniformly distributed along all contours provides that the shape context extracted from the selected corner is representing the general properties and physical characteristics of the shape according to the reference frame of that corner. As the final step, a log-polar system is located on the reference point and vectors are counted as which one of them is placed inside of which wedge and which ring of the log-polar system. In this way, a feature matrix size of RxW is extracted where R and W are ring and wedge numbers of the log-polar system respectively(Figure 3.6). The log-polar coordinate system is used when measuring shape context because it provides more detailed information for the contour points that is closer to the reference point.

To make the extracted shape context features independent from the size of the shapes, all corner points of the shapes on the dataset is scaled to a one by one window

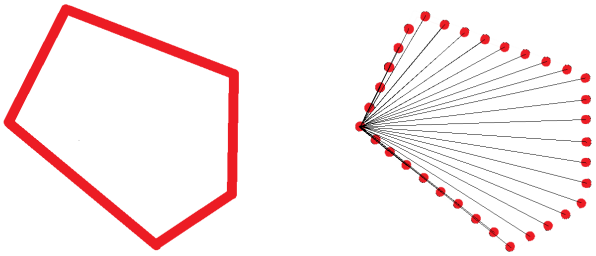

Figure 3.5. Finding the contours of shape, Sampling contour points and Constructing vectors from reference corner to all sample points.

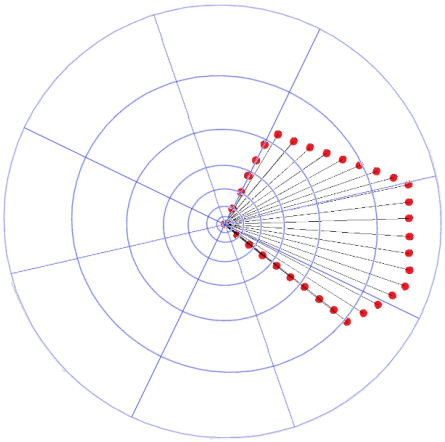

Figure 3.6. Wedges, Rings, and Shape context of a sample shape. First wedge of the log-polar coordinate system starts with the same orientation of the edge that reference corner and its next corner constructs. By doing this, we have rotation invariance.

at first. After that, each corner's reference frames are rotated so that the y-axis of the reference point aligns with the edge vector that is from the reference corner to its next corner. By this rotation, shape context features become rotation invariant. In the end, the zeroth wedge of the log-polar system is selected as the y-axis of the reference point, and shape context features are recorded to the dataset according to this measurement method.

Support Point Prediction with Machine Learning. After calculating shape context feature vectors, we set up a supervised machine learning model which predicts whether a vertex is a support point. The model takes the application point of the force and shape the context feature vector for that particular corner as an input. The labels

are binary, 1 if that particular corner is support point, 0 if not.

$$\begin{bmatrix} F_x & F_y & SC_1 & \dots & SC_n \end{bmatrix} \tag{3.1}$$

F_x is the x-coordinate of applied force, F_y is the y-coordinate of applied force, and SC_i is the respective shape context feature vector element of the corner in the investigation for the particular shape.

As a result of the learning of the model, given a force application point and shape context features for some corner, we classify whether a point is a support point or not. But the actual aim is to determine the edge where the shape is supported on lever-up action. Therefore, in the test procedure, a prediction for the support edge of a shape considered correct, if all each vertex of a particular shape is correctly classified as a support point or not. For example, assume that we have a hexagon and we found from the simulation results that the vertices 2 and 3 are the support points. We should get for the labels predicted (0,1,1,0,0,0) for the vertices. Any other combination of labels means that the prediction of the support edge is wrong. Equation 3.2 shows the mathematical representation of our model.

$$f(F_x, F_y, SC_1, SC_2, .., SC_n) = \begin{cases} 1 & , \quad f > 0.5 \\ 0 & , \quad f < 0.5 \end{cases} \tag{3.2}$$

For classification , we used Artificial Neural Networks (ANN) and Support Vector Machines (SVM). We randomly selected a subset of the shapes , used experiments related to selected shapes as training set, preserving the class ratios. Remaining experiments are used as test set.

3.1.4. Experiments

For experiments, firstly we generate 100 random shapes. The radius of the circle that we get shapes with edge numbers from three to ten is selected as 20cm. After, to make the shapes random sized, each shape's corner points are multiplied by a random float number within the range of 1 and 3. So in the end, we have 100 random sized shapes that have random edge numbers between 3 and 10.

Generated shapes are modeled and reconstructed one by one as 3D objects inside of the VREP simulator. Lever up action is simulated by applying virtual forces from the 5 random selected force points on each edge and results are recorded to the dataset. From a single shape with N edges, there are $5 * N^2$ data extracted and recorded. For example, for a triangle, 5 points are randomly selected from each edge as force points. After performing lever up the action from these total of 15 force points, status for each corner that tells this corner is a support point or not at the end of each action is extracted as data. In the end, for a triangle, we have 45 experimental data holds information about each corner with 15 lever-up actions performed. In this case, the total number of data can be calculated as $\sum_{i=1}^{S}(5N_i^2)$ where S is the total number of shapes, N_i is the edge number of the shape i.

Calculating the shape context features of the corner points, we use a log-polar coordinate system with ring and wedge numbers of 10 and 20 respectively. From the 10x20 shape context matrix, we have 200 features in total. Lastly, the feature vector is normalized as its values sum up to one.

As for corners of the shape, coordinates of force points in the experiments are also converted to the given corner's reference frame on that data. Force points' coordinates are also normalized to one, so they also become rotation and scale-invariant.

After gathering and organization of experiment results, the classification process is done via Artificial Neural Networks and Support Vector Machines. After training, the model is tested with novel shapes that were not in training set then both experiment-

based and shape-based prediction accuracies are calculated. For this dataset, ANN models were better than SVM models in terms of overall prediction performance therefore SVM results were not analyzed in this work.

For the ANN model, 50 hidden layers are used with sigmoid activation. Between class, Cross-entropy is used as the loss function. MATLAB Machine Learning Toolbox [20] implementations are used for training.

3.1.5. Experiment Results

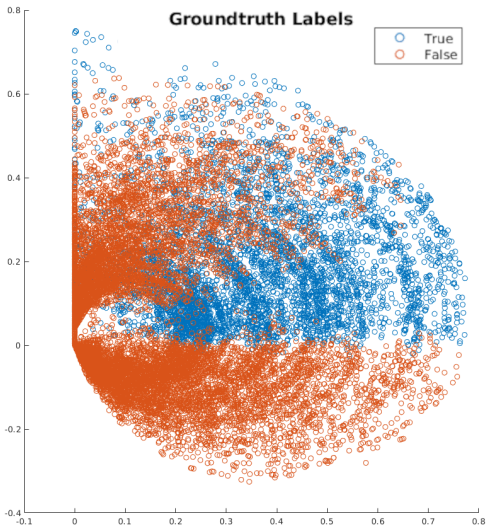

Figure 3.7. Groundtruth labels of all experiments done in simulator.

In our experiment results, our proposed shape-context based feature descriptor is compared with a baseline method. A feature vector only based on force application point is selected as the baseline method, and it is named as Force Position (ForcePose). Our proposed method is appending the shape-context features to the ForcePose, and it is named as (ShapeCon).

To investigate the impact of the ShapeCon method on recognizing the support edge, both experiment setups were trained by ForcePose features as the input, and these results are saved as a baseline. Then, the test set performance for both experiment setup is computed after training the model with ShapeCon features. We compared

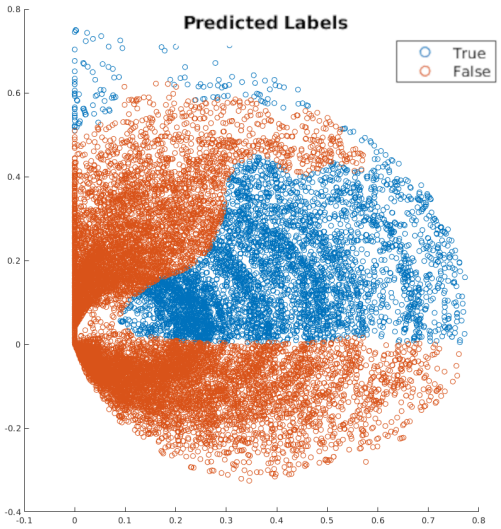

Figure 3.8. Distribution of training results when 50 hidden layer ANN trained with ForcePose features.

these results with the baseline. The performance in this experiment, calculated according to predicting all the points on a shape, i.e. correctly identifying the support edge of the shape.

Learning Data Analysis for Lever-Up Action. We visualized the learning space to investigate the separability of the samples and distribution of the force application points. Figure 3.7 shows the ground truth labels in the space of force application points. In training samples, since we include lever-up points relative to the coordinate frame of each corner, corners are at (0,0) in the space and all forces are shown relative to this point. Figure 3.8 depicts the separation of the learning space when only the ForcePose features are given as input. The experiment-based accuracy turned out to be %76, for the ForcePose model.

Effect of Increasing Amount of Training Dataset to Learning. To compare the accuracies of the two models that using only force points and using shape context with force points, two experiment methods are established. In the first experiment we divide the dataset into 10 parts, and starting from %10 percent to %90 percent we train the

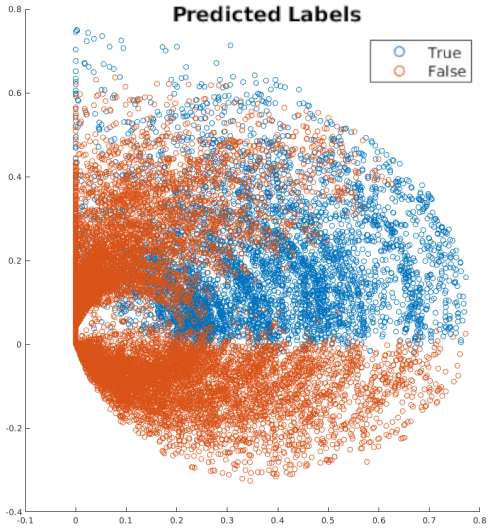

Figure 3.9. Distribution of training results when shape context features are included in training.

model with an increasing amount of data. All figures demonstrated on the paper show the accuracy of predicting all corners of a shape correctly, therefore predicting the true support edges of the shapes. Figure 3.10 shows the change in the test accuracy of the model that used only force points as input and trained by an increasing amount of train data. Figure 3.11 on the other hand, shows the change in the test accuracy of the model used shape context with force points as input and trained by an increasing amount of train data. In this experiment, we see that shape context features are improving the test accuracy around %35. Also, with the increasing amount of train data, the accuracy of the model using shape context is increased from %72 to %90.

Effect of the Polygon Type In the Training Set to Learning and Generalizability. In this experiment setup, we investigated the performance of the model when it is trained only with a single type of shape. For this purpose, we trained the network with the same type of polygons, and measure the prediction accuracy of remaining polygons. For example, the model is trained with the results obtained from all hexagons in the dataset, and the prediction accuracy for all other polygons are measured. This training procedure is done one by one for all polygon types with vertex count 3 to 10.

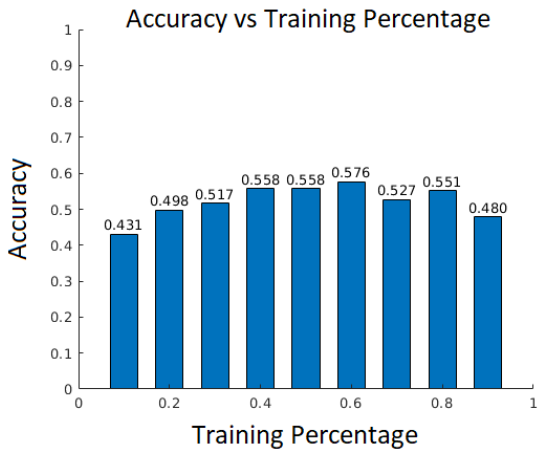

Figure 3.10. Test accuracy of the model when only force points are used as input to learning process, and its change with the increasing amount of training data.

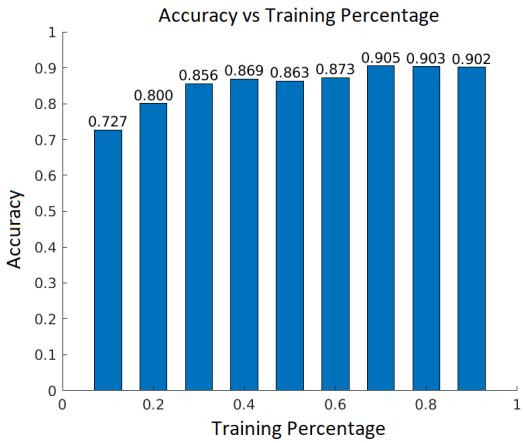

Figure 3.11. Test accuracy of the model when shape context are used with force points as input to the learning process, and its change with the increasing amount of training data.

To analyze the contribution of the polygon types to the learning process, each polygon type is given as input to the learning process one by one and tested for other polygon types. Figure 3.12 shows the test accuracy of the model using only force data as input to the learning process and when only selected polygon type is used as train data and tested on others. Also, Figure 3.13 shows the test accuracy of the model using shape context with force data as input to the learning process and when only selected polygon type is used as train data and tested on others. Comparing the results, we can say that shape context improves the generalizability of almost all polygon types.

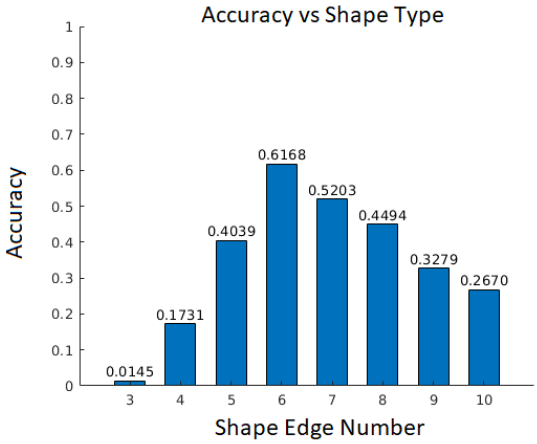

Figure 3.12. Accuracy of the model using only force data as input to learning process and when only selected polygon type is used as train data and tested on others.

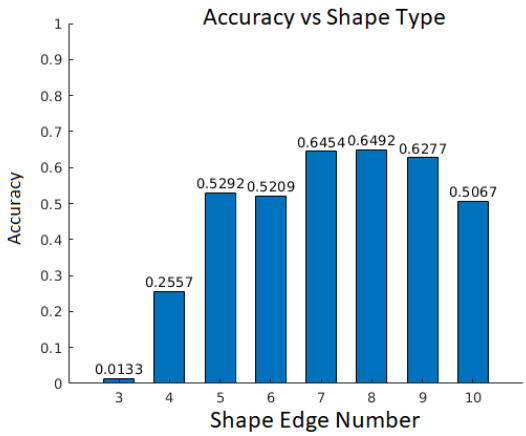

Figure 3.13. Accuracy of the model using shape context with force data as input to learning process and when only selected polygon type is used as train data and tested on others.

Although preliminary work presented here may be seen not very related with learning from demonstration area, these works helped us understand how the representations are formed using deep neural networks and allowed us to propose a more flexible and efficient learning framework which is one of the most important factors while learning from demonstrations. The rest of the thesis examines the related work done in the learning from demonstration area and introduces our system after giving background information.

4. RELATED WORK

LfD [4] has been applied to various robotic learning problems including object grasping and manipulation [7–11]. Among others, learning methods that are based on dynamic systems [21] and statistical modeling [22] have been popular in the recent years. Dynamic Movement Primitives (DMPs) [21] encode the demonstrated trajectory as a set of differential equations, and offers advantages such as one-shot learning of non-linear movements, real-time stability and robustness under perturbations with guarantees in reaching the goal state, generalization of the movement for different goals, and linear combination of parameters. The parameters of the system can be learned with different advanced algorithms such as Locally Weighted Regression (LWR) [23] and Locally Weighted Projection Regression [24]. Statistical modeling techniques, on the other hand, use probabilistic models such as Gaussian Mixture Models [7] or Hidden Markov Models [25] to encode the motion from multiple demonstrations where the statistical regularities are learned in the spatiatemporal data.

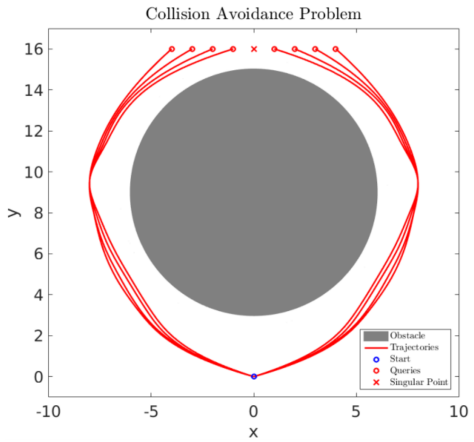

Figure 4.1. 2D obstacle avoidance task from [2].

Zhou et al. [2] used a simple 2D obstacle avoidance task to show that standard DMP approach is not designed to learn from multiple trajectories and therefore cannot encode the important parts of multiple demonstrations (Figure 4.1). In this task, the robot learned a trajectory that moves the end effector of a robot from one side of an obstacle to the other side. Various goal positions are defined and demonstrations for these goal positions are shown to the DMP model. However, the produced trajectory

fails to avoid the object in the task when the end goal position of the learned DMP model is shifted to a new goal position which is symmetrically on the other side of the object. To handle with this problem, a parametric representation relates the obstacle size and the end goal of the action should be learned. They developed a method that performs LWR by defining a new objective function dependent on the environment restrictions and parameters, and that uses a model-switching algorithm to be able to train over all the parametric space. While this is an effective method for the given task, in this thesis, we want to mix the advantages of DMP based classical approaches with the knowledge provided by statistical modelling algorithms such as the variance in the demonstrations. Ude et al. [26] used LWR and GPR methods to compute new DMP based trajectories given parameters that describe the characteristics of the action, such as the goal point. Their method could be generalized with complex trajectories for both periodic and discrete movements. Pervez et al. [27] learned a separate GMM for each demonstration and mixed the separately learned GMMs to generate trajectories for new task parameter values.

In order to enable physical collaboration, the robot is required to learn haptic feedback models that encodes the default predicted force feedback measured during execution of the corresponding skill. Memorized force and tactile profiles have already been successfully utilized in modulating learned DMPs in difficult manipulation tasks that contain high degrees of noise in perception such as grasping and in-hand manipulation of objects from incorrect positions or flipping boxes using chopsticks [28]. HMMs were used to learn multi-modal models from temperature, pressure and fingertip information for exploratory object classification tasks [29]; and PHMMs were used to learn haptic feedback trajectory models to adapt actions in response to external perturbations [30, 31]. Deep networks [32] were used to learn multi-modal models from different sensory information such as temperature, pressure, fingertip, contacts, proprioception, and speech; however these models were used only to categorize the sensory data without any effect on action execution. More recently, the same problem generalization of force/torque profiles was investigated for contact tasks [33] where these profiles were modeled by Locally Weighted Regression (LWR) which has local generalization capabilities, hence successful at the intermediate query points.

5. BACKGROUND

In this chapter, we will first explain Conditional Neural Processes, a new family of neural network models we are inspired from. Then, we will introduce Conditional Neural Movement Primitives, a neural network to encode movement primitives.

5.1. Conditional Neural Processes

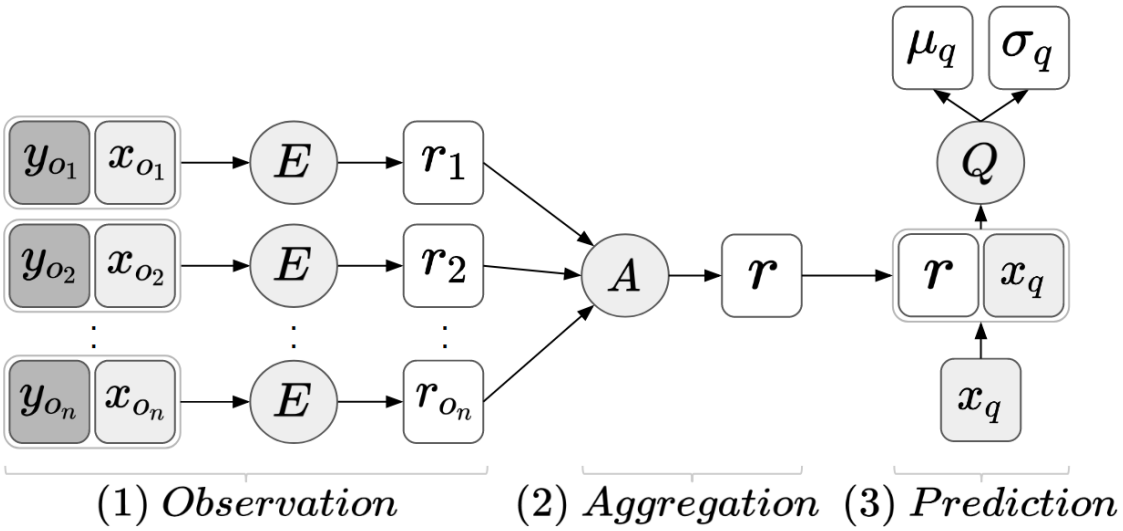

Figure 5.1. Structure of the CNP adapted from [1].

Until recently, most of the neural networks in deep learning were trained to approximate a single output function. However, when the data is a distribution which can be represented by statistical models such as Gaussian distributions, networks that approximates single functions cannot represent the underlying model. Instead the network can be modelled as a probabilistic approximator that can predict the distribution parameters, namely mean and variance, instead of producing one single output value. [1] proposed a new family of neural models that processes the training data based on Bayesian Inference principles by combining the advantages of neural networks with Gaussian Processes. The proposed method is able to extract prior knowledge directly from the training data by sampling observations from it, and use this information to predict a conditional distribution over any other target points. The general structure of the model is shown in the Figure 5.1. The model consists of three parts: (1) A

parameter-sharing Encoder Network E is used to encode all sampled observations into fixed size representations r_i in a latent space. Notice that the numbers and orders of the coming observations can change during both training and test time because empirical observations can change dynamically. (2) The individual representations are aggregated into one general representation r that covers all the observations sampled at the beginning. Therefore, according to the prior knowledge r, the model can predict a distribution over the target input/inputs by conditioning on the observations. (3) Finally, a Query Network Q produces distribution parameters (μ_q, σ_q) over the query input x_q by using the general representation r as a prior. As a summary, the CNP model can be formulated as

$$\mu_q, \sigma_q = Q_\theta \left(x_q \oplus \frac{\sum_i^n E_\phi((x_i, y_i))}{n} \right)$$

where $\{(x_i, y_i)\}_{i=0}^n$ are observation pairs sampled from a function, $f(x) = y$, E_ϕ is the encoder network with parameters ϕ, x_q is the target input, i.e., the query over which we want the model to predict a Gaussian distribution, $\oplus$ is the concatenation operator, and Q_θ is the query network with parameters θ that outputs the Gaussian distribution parameters (μ_q, σ_q). The whole model is trained together as a stochastic process where at each training iteration, n random observation pairs $(x_i, y_i)_{i=0}^{n \in [1, n_{\max}]}$ and one query point (x_q, y_q) are sampled from the training data. $n_{\max}$ is the hyper-parameter giving the maximum number of observations that can be used during the training process. At the end of each iteration, neural network parameters (θ and ϕ) of both Encoder and Query network are optimized according to the following loss function:

$$\mathcal{L}(\theta, \phi) = -\log P(y_q \mid \mu_q, \text{softplus}(\sigma_q)) \tag{5.1}$$

where μ_q and σ_q are the outputs of the CNP, y_q is the corresponding output of the x_q for this training iteration, and P is the Gaussian probability function that returns the conditional probability of the y_q for given mean and variance parameters.

6. PROPOSED SYSTEM

Conditional Neural Movement Primitives (CNMP) is a learning from demonstration framework that is designed as a robotic movement learning and generation system built on top of CNPs. It specifically produces movement trajectories in joint or task space, and executes these trajectories through a high-level feedback control loop. The encoded sensorimotor representations are conditioned externally to constrain the movement and set goals. Additionally, as detailed later, CNMP is also continuously conditioned by the current sensory readings, predicting the desired sensorimotor values based on these readings. This allows to detect unexpected events and respond to these events by changing the movement trajectory on the fly.

A CNMP is trained for each skill that is taught to the robot using one or more demonstrations in possibly different configurations. In each demonstration, the robot stores the actuator positions, the perceptual features, and the corresponding time points. Therefore each demonstration D_j is stored as a trajectory of sensorimotor features and the corresponding time points, i.e. $\{(t, SM(t))\}_{t=0}^{t=\tau}$ where $SM(t)$ corresponds to the observed sensorimotor data at time t. SM can encode any information including task and joint space positions of the robot arm and gripper, force/torque measurements, low-dimensional environment properties such as size of manipulated objects or high-dimensional world state such as top-down 2D raw image mask of the environment. In the rest of this section, SM is described as a one dimensional vector which might be flexibly referred as sensory or motor data in different times for simplicity and clarity.

CNMP training and trajectory reproduction steps are explained in Figure 6.1 and detailed in the next sections. Recall that although SM is displayed as a one dimensional vector, it is normally a multi-dimensional vector that includes any sensory or motor information. Therefore, CNMP is designed to enable learning from and constraining in any set of dimensions in the SM space.

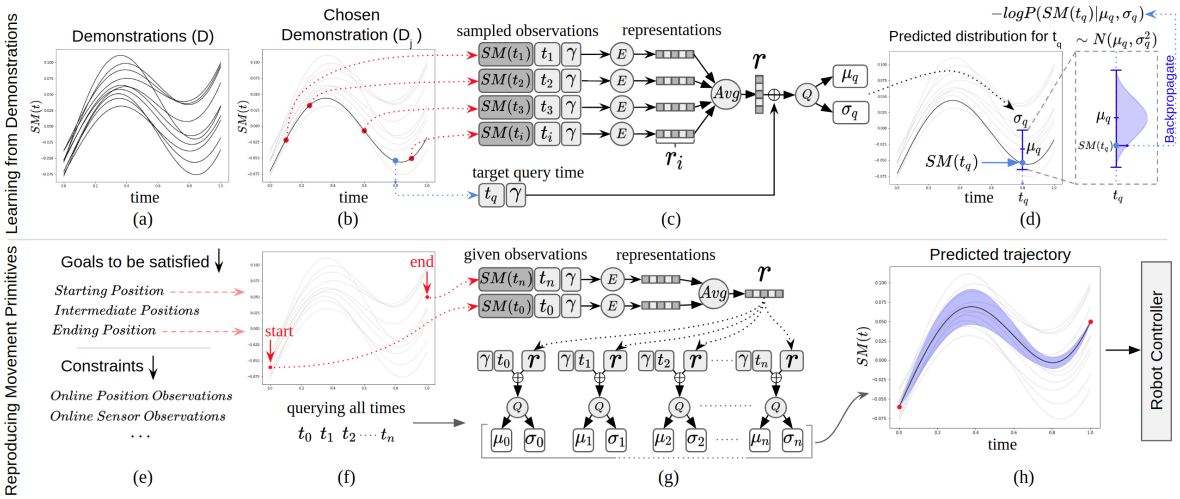

Figure 6.1. Training and test procedures of Conditional Neural Movement Primitives.

Conditional Neural Movement Primitives are explained in two parts as follows: Learning from demonstrations and reproducing movement primitives at test time .

6.1. Learning from Demonstrations Using CNMP

For each demonstration, first, time invariance is ensured by scaling the time values into [0,1]. In each training iteration, a demonstration D_j is selected randomly from a uniform distribution. From the chosen demonstration, n observation tuples $\{(t_i, SM(t_i))\}_{i=0}^{n} \in D_j$ and one target query $(t_q, SM(t_q)) \in D_j$ are sampled uniformly. n is a random number between $[0, n_{max}]$ where n_{max} is a hyper-parameter that sets the maximum number of samples collected from the chosen demonstration by CNMP. This allows our framework to be flexible over changing numbers of observations at the test time. Figure 6.1.1c illustrates the neural network operations for an example of four observation points. Each observation tuple goes through the parameter sharing Encoder Network (E) and produces the representation for the corresponding observations in a fixed sized latent space. Using a mean operator, individual representations are merged into one general representation r. The general representation r and target query time t_q are concatenated and given to the Query Network (Q) as input. As the final step of the neural network, Query Network predicts the distribution parameters (μ_q, σ_q) of SM for target time t_q. Figure 6.1.1d shows the predicted Gaussian distribution and the actual $SM(t_q)$ of the robot at the queried time. The error between the predicted

and actual SM values is calculated using the loss function in Eq. 5.1 for each query. The computed errors are back-propagated to train the weights of the network.

Task specific external parameters (γ) can also be included in encoding the movement primitive by directly concatenating with the observation and query data and providing to the Encoder and Query Networks directly. Such external parameters can be regarded as part of sensory data, therefore for simplicity we will not provide separate explanation for handling these parameters in the rest of this section.

6.2. Reproducing Movement Primitives

After learning from demonstrations, the CNMP is used to generate the SM trajectory that is predicted to satisfy the externally provided goals given the current sensorimotor information, where both goals and sensorimotor information are represented with a set of observation tuples. The external goals are set to the desired sensorimotor values at the desired time points ($\{(t_i, SM(t_i))\}$). The means of identifying and querying the goals is extremely flexible (Figure 6.1.2e). For example, the goals can refer to the starting, intermediate or final positions of the movement trajectory, and set once to generate and execute the desired trajectory. As another example, the goals can be set as the desired force/torque readings expected to be observed during movement generation. The current sensorimotor information, on the other hand, corresponds to the SM at the current time point (t_c): ($\{(t_c, SM(t_c))\}$). Continuous querying with the current SM enables the system to detect and respond to unpredicted events such as external perturbations that generates discrepancy between the actual and predicted sensorimotor values. In case such prediction error is detected, the movement trajectory changes on the fly, and executed as detailed in the next subsection.

This effectively corresponds to autonomously reacting to external perturbations exploiting the encoded multi-modal distributions[1] . Figure 6.1.2f illustrates an example

[1]The complexity of the trajectory generation is $\mathcal{O}(n_o + n_q)$ where n_o and n_q are the numbers of the observations and queried time points respectively. Single query takes 0.9 msec. with Inter Core i7 processor and GeForce GTX 1050 gpu, the robot can quickly adapt to the changes in the environment at the test time.

case where the goal is set as two test time observations that correspond to the desired start and end positions of the movement. These goal observations are given to the trained CNMP model and the general representation r is produced after the Encoding and Averaging operations in the neural network (Figure 6.1.2.g). This general representation r holds the compact information for all the given goal observations, thus, guides the Query network to produce movement primitive distributions that satisfy all the desired goals/constraints at the once. To extract the whole movement trajectory, all time steps $t_{i i=0}^{1}$ are concatenated with the pre-computed general representation r, and the set of mean and variance values for the whole trajectory is predicted by using the Query network. Figure 6.1.2h shows the predicted trajectory distribution that satisfies the given goals. The predicted trajectory is sent to the controller of the robot to be executed as detailed next.

6.3. Execution and Online Adaptation of the Generated Movement

The CNMP-driven control loop that aims to bring the robot to the sensorimotor state predicted by the CNMP in each time step is provided in Figure 6.2. In each time step, the environment is perceived through sensors of the robot, and the current SM is provided along with the externally defined goals to the CNMP for generation of the sensorimotor values that are expected to be observed to achieve the task. In an online control loop, CNMP predicts the $SM(t)_{t=t_c+1}^{t_n}$ configurations of the next time step and continuously produces motor commands to achieve the predictions. In case the predicted SM and the current SM does not match, i.e. the current SM does not match with the SM that is predicted to match with the corresponding skill, time stops advancing until the error drops below a threshold. With this constantly changing online observation loop, our system can react to the perturbations of the sensor measurements, because of the the low computation complexity of the CNMP at the test time.

Note that the structure of the CNMP does not need any strict requirements on the controller side. In its current form, the motor variables in the generated SM are set

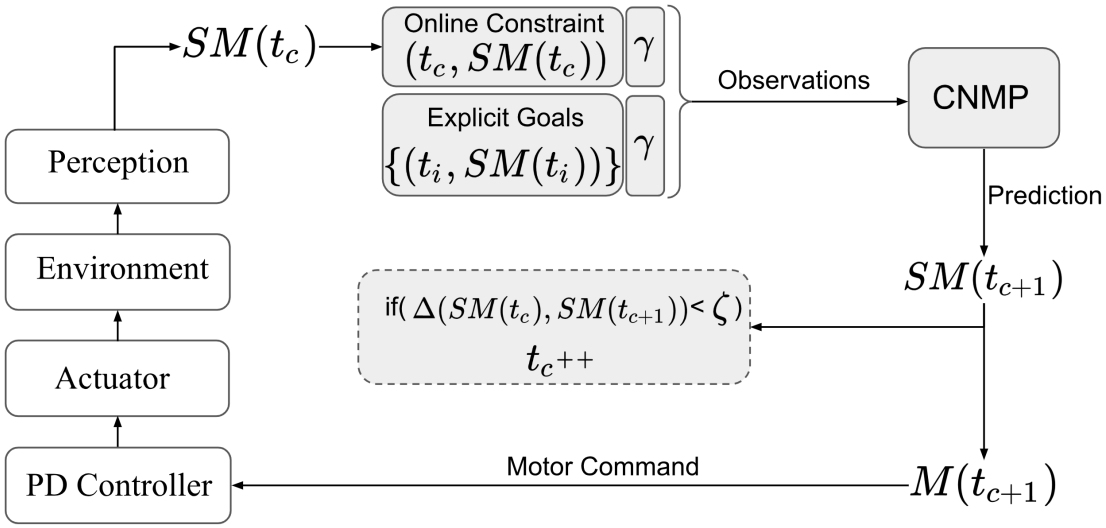

Figure 6.2. CNMP execution loop with feedback.

as the desired motor commands in each time step, and executed using a PD controller. The fact that the output of the CNMP is constructed as a distribution allows the robot to move freely within the boundaries of the predicted distribution. This flexibility can potentially be exploited to achieve the given task with alternative trajectories. Another option is to use variance information to reason about which segments of trajectory should be followed stiffly and accurately, automatically adjusting the compliance based on the variance in the distribution.

7. EXPERIMENTS AND RESULTS

The performance, capabilities, and limitations of the CNMP were studied in four simulated and two real robot experiments. Links to our implementation and data used in the experiments are provided in the corresponding sections. In Section 7.1, the CNMP and another widely used method that encodes distribution of the demonstrations, namely ProMP [6], was compared in learning of a 2D simulated obstacle avoidance task where different modes of the avoidance were demonstrated. In Section 7.2, the capacity of the CNMP to learn in high-dimensional SM was investigated in a 2D obstacle avoidance task where the system learns planning movement trajectory avoiding two obstacles with different top-down image masks of the environment. In Section 7.3, the capacity of the CNMP to adapt the visual conditionings that are changing dynamically was examined where the robot in the simulator learns planning movement trajectory avoiding two obstacles using images collected from the the environment. In Section 7.4, the performance of the CNMP in learning non-linear environment-trajectory relations in a real robot manipulator was investigated through analyzing generalization capability to environment configurations inside and outside the range of demonstrations. In Section 7.5, the capability of the CNMP model to react to external perturbations detected in the proprioception and force data was verified in the real robot exploiting the learned multi-modal information. In Section 7.6, a new type of CNMP is introduced and it is shown that CNMP can produce new types of movement primitives by blending the primitives it learned.

7.1. Learning of Different Modes of Operations of the Same Skill

The aim of this experiment is to investigate if and in which conditions the models can automatically discover the categories inherent to the demonstrated primitive; and reproduce new avoidance trajectories taking into account this knowledge. For this, the capabilities of the CNMP and ProMP models were studied in a 2D object avoidance task where demonstrated trajectories passed around either one side or the other side of the obstacle and from which side it passes was not explicitly provided to the system.

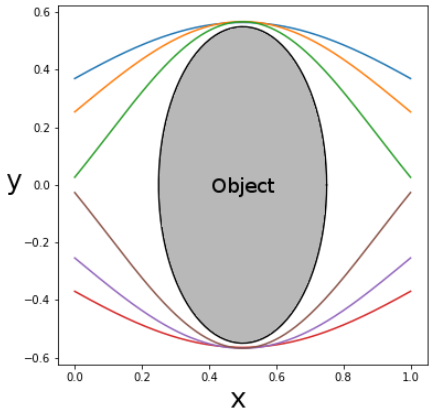

Figure 7.1. Object avoidance experiment. Based on starting position y_0, robot avoids the object from upside or downside. Around $t = 0.5$, trajectories have a strict and small variance.

The task and the demonstrated trajectories are provided in Figure 7.1. Each trajectories started from the left side (t=0,(x=0,y=y_0)), it passed around the upper or lower side of the obstacle at time point t=0.5, and ended up on the right side at a symmetric position (t=1,(x=1,y=y_0)). The 6 trajectories shown in the figure with 300 time steps were used to train both models. The ProMP model[2] was trained with 31 basis functions, setting the σ value to 0.05. The CNMP, on the other hand, was trained by sampling a number of observations $\{(t_i, (x_i, y_i))\}_{i=0}^{n\in[1,5]}$ from a randomly selected demonstration in each iteration. Encoder and Query networks were structured in 2 fully connected layers with 128 neurons each. The neural network was trained by using Adam optimizer with an empirically found learning rate of 0.0001.

After training, new avoidance trajectories were generated by setting goals, i.e. by conditioning the models at different points. The obtained results are presented in Figure 7.2. The upper and lower rows in this figure correspond to the results obtained by ProMP and CNMP, respectively. In each plot in the figure, the black dot shows the conditioned position. The black line and the shaded area correspond to the mean and variance of the predicted movement distribution computed for the corresponding conditioning. As shown in the first column, when the models were conditioned at a position

[2]Our implementation and training data for the experiments can be found in [34]. To compare our framework with ProMP, ProMP library of the Flowers INRIA Laboratory in learning and generating the movement primitives is used. See [35]

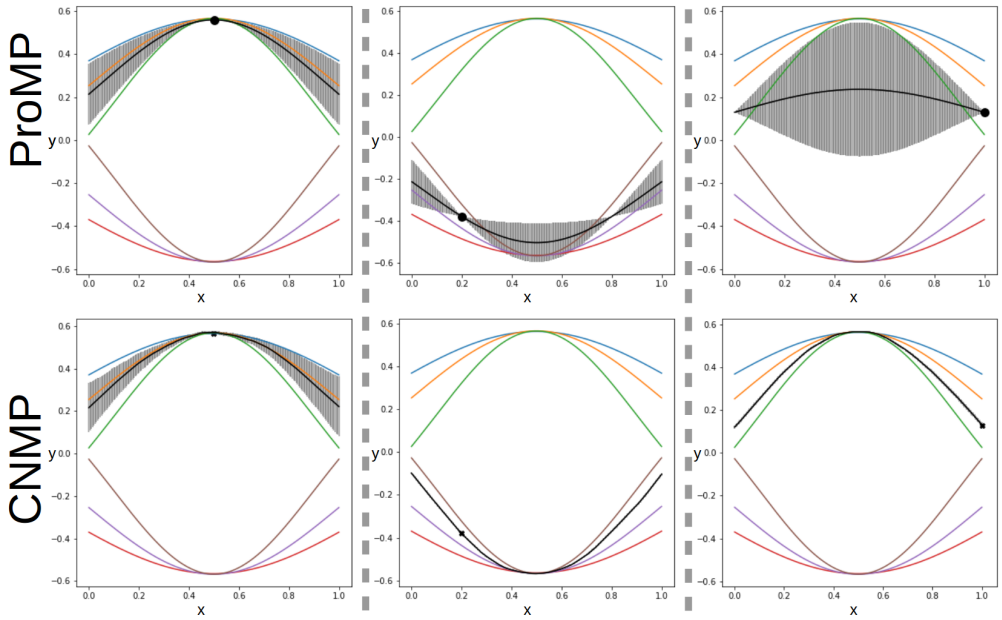

Figure 7.2. 2D object avoidance experiment and the results. The demonstrations are given with colored trajectories. The conditions and generated movement distributions are shown with the black dots and gray areas, respectively.

that ensures avoidance, both models generated distributions and trajectories that enabled the system to avoid the obstacle. On the other hand, when the conditioning was set to other positions such as to an intermediate point (t=0.2,(x=0.2,y=-0.37)) or to an end point (t=1,(x=1,y=0.13)), the ProMP started failing whereas CNMP continued generating movement trajectories that avoid the invisible obstacle. These results show that CNMP learned to produce multiple distributions by observing different categories of the same primitive from the data itself without any explicit parameterization. With this, we conclude that CNMP allows learning and encoding different modes of operation of the same movement primitive, and reliable and flexible reproduction of the trajectories automatically following one of the distinct modes.

7.2. Movement Primitive Learning in High Dimensional Sensorimotor Space from Top Down Raw Image Masks

In most of the LfD applications, the task parameters such as positions and dimensions of the objects to be handled or the target positions of the end effectors are explicitly provided for learning the associations between low-dimensional feature

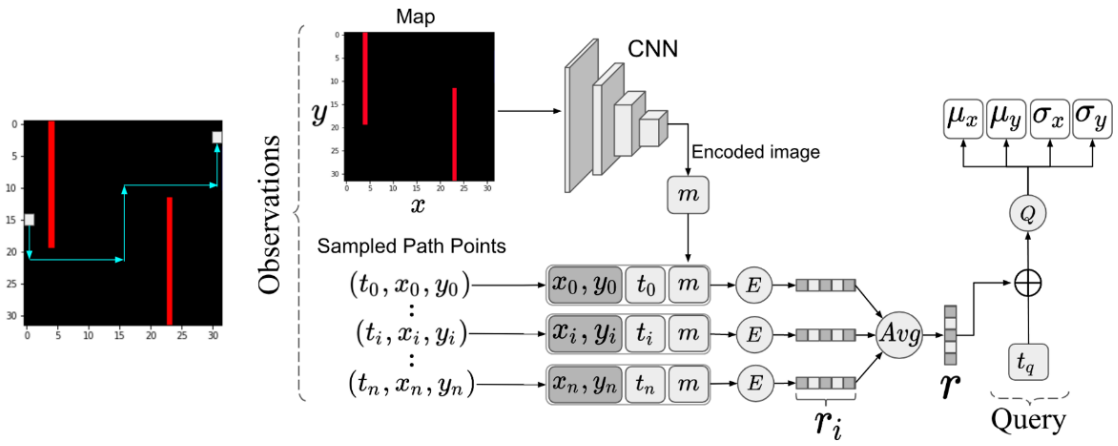

Figure 7.3. Left: The 2D obstacle avoidance setup used in learning experiments from top-down images. An example environment and demonstration trajectory were presented. Right: The structure of Convolutional CNMP.

space and motor control variables. In the previous section, we also directly provided low-dimensional 2D positions in learning such relations. However, as discussed in the Introduction Chapter, it might not always be feasible to provide the best set of low-dimensional features that well represents the task, especially when the robot encounters with novel environments and tasks. In these cases, the robot is required to automatically extract the relevant information from its high-dimensional perceptual data, possibly using large number of learning examples.

In this experiment, the capability of CNMPs in learning directly from top-down raw image masks in a simulated experiment setting is studied. Two obstacles of varying sizes were attached to the opposite sides of a 2D room at different positions as shown in Figure 7.3. For each room configuration, a motion trajectory that moves the robot from the left to the right side of the room was generated with a simple heuristic. The motion trajectories were presented as demonstrations along with the raw images to train the CNMP model. Convolution operation was included into the model (Figure 7.3) to deal with the raw images. Convolution was achieved with a neural network that consists of 4 convolution layers followed by max-pooling operation with channel numbers 8, 16, 32 and 64. At the end, SM vector was composed of 2D position, image mask encoding (m), and the corresponding time-point. In order to train the system, 1000 trajectories with random room configurations are collected.

The CNMP model, after trained with 1000 trajectories (in configurations with random start/end positions and randomly placed and sized obstacles) was queried to generate trajectories with different conditions. The conditions were encoded in SM vectors that include time, position and image information.

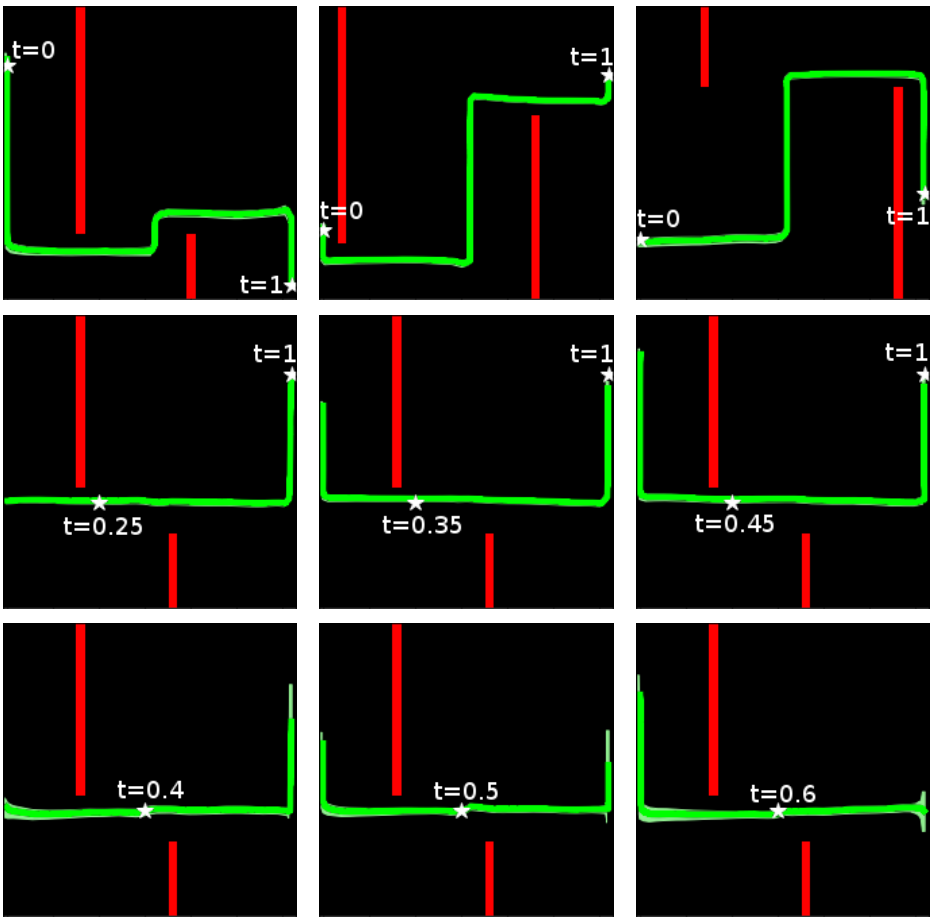

Figure 7.4. CNMP was conditioned by (first row) start and final positions, (second row) intermediate and final positions, and (third row) one intermediate position.

Figure 7.4 shows the trajectories generated by the CNMP conditioned with start and end positions, intermediate and end positions, and an intermediate position for a number of test configurations. As shown, the trained CNMP could successfully generate movement trajectories avoiding the red walls in all configurations. Interestingly, when the CNMP was provided only an intermediate pose at a specified time, it changed the start and end positions, shifting the movement if necessary (Figure 7.2). These results indicate that the CNMP was successful in learning in high-dimensional sensorimotor space and it is a promising framework for end-to-end learning of complex trajectories.

7.3. Dynamic Visual Conditioning via Movement Primitive Learning in High-dimensional Sensorimotor Space

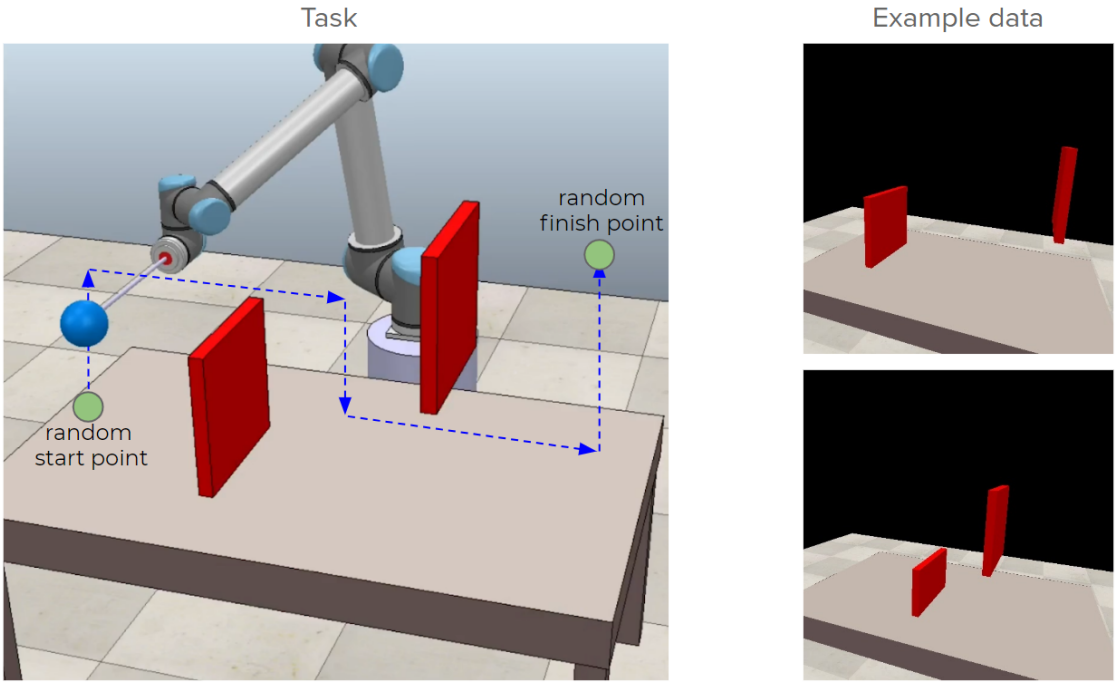

Figure 7.5. Left: Experimental Setup. Right: Two example configurations of the environment which are recorded by a vision sensor in the simulator.

This experiment has the same motivation with the previous experiment. However, in this experiment, we wanted to show that a visual conditioning can be applied to our proposed system in a scenario where the robot is actually executing the motion that is produced by the system dynamically by using a basic control loop. Specifically, aim of this experiment is to show that a robot executing the motion with our control loop can adapt to the changes in the environment in real time even the visual conditioning is dynamically changed on the fly. Figure 7.5 shows the experimental setup and example visual data recorded. The task in this experiment is same with the Experiment 7.2. At each run, the robot which has an object attached to its gripper starts from a starting point at the left side of the table, and by avoiding the red objects in the environment with a predefined heuristic shown in the figure, it finishes the movement at a point at the right side of the table. In the training process, we used the same architecture that we used in the Figure 7.3. The only differences are that in this experiment the end effector of the robot is a 3D (x, y, z) vector, and 500 trajectories are recorded.

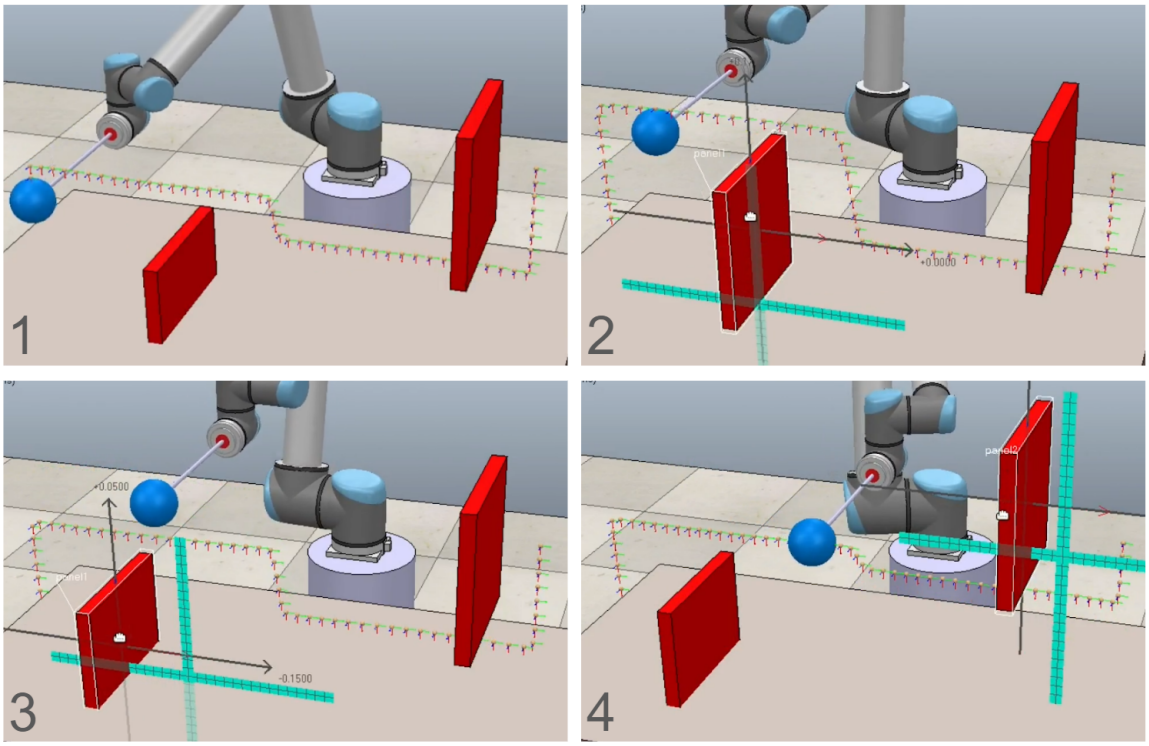

Figure 7.6. Testing our system by changing the environment dynamically on the fly.

Figure 7.6 shows the test progress while the visual condition is changed dynamically over time. In the first image, it can be seen that a trajectory which avoids the red obstacles is predicted successfully by the CNMP system. While the movement is executed, we change the position of the obstacle on the left, thus, change the visual condition of our system. It can be seen that the CNMP immediately produces a new trajectory that moves the robot up in order to avoid the obstacle. In the third image, the left obstacle is moved down and the produced trajectory immediately goes below. In the last image, it can be seen that the right obstacle is moved up and the CNMP immediately produces a new trajectory according to the new visual configuration. These results show that even in a scenario such as using a visual feature in our system which can be seen as a very hard task to handle in real time in general robotic tasks, the CNMP is capable of handling high-dimensional sensory data at real time and successfully accomplish the task.

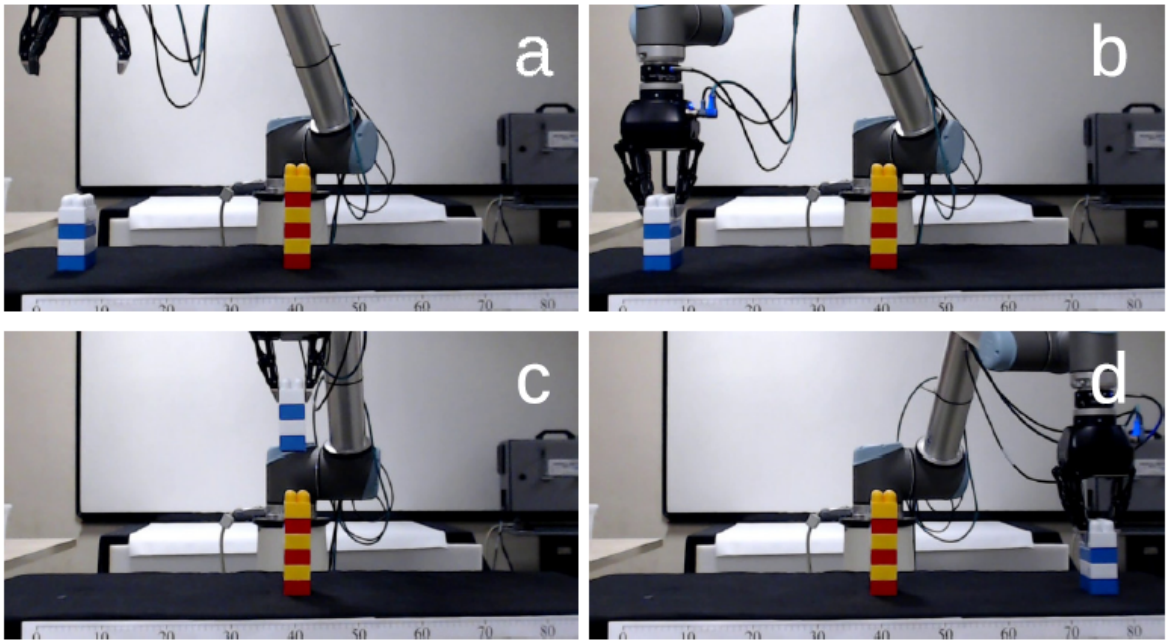

Figure 7.7. Experimental setup and snapshots from demonstrations in the obstacle avoidance task where heights were provided as parameters.

7.4. Task Parameterization and Generalization in Real Robot

This experiment investigates the capability of the CNMP in learning of non-linear environment-movement relations in an object pick and place task that requires obstacle avoidance. UR10 robotic arm equipped with a Robotiq gripper and a wrist mounted force/torque sensor was used for this purpose. After the robot was provided a number of demonstrations in different environments, we analyzed the generalization capabilities in novel configurations. The heights of the manipulated object and the static obstacle were provided as task parameters (γ) and the joint angles were included in the SM vector. The task was composed of moving the gripper to a suitable position to pick up the object according to its height first, and then moving it towards the target position following an arc avoiding the obstacle. The experimental setup and snapshots from an example demonstration is provided in Figure 7.7. CNMP was trained with 8 movement trajectories that were obtained in different object-obstacle configurations. The heights of the object and obstacle were provided as parameters: γ_o=$(2, 6)$, γ_h=$(2, 4, 6, 8)$. The non-linear relationship between joint values and γ can be observed in Figure 7.8, which provides the demonstrated trajectories in gray, especially in elbow and shoulder joints.

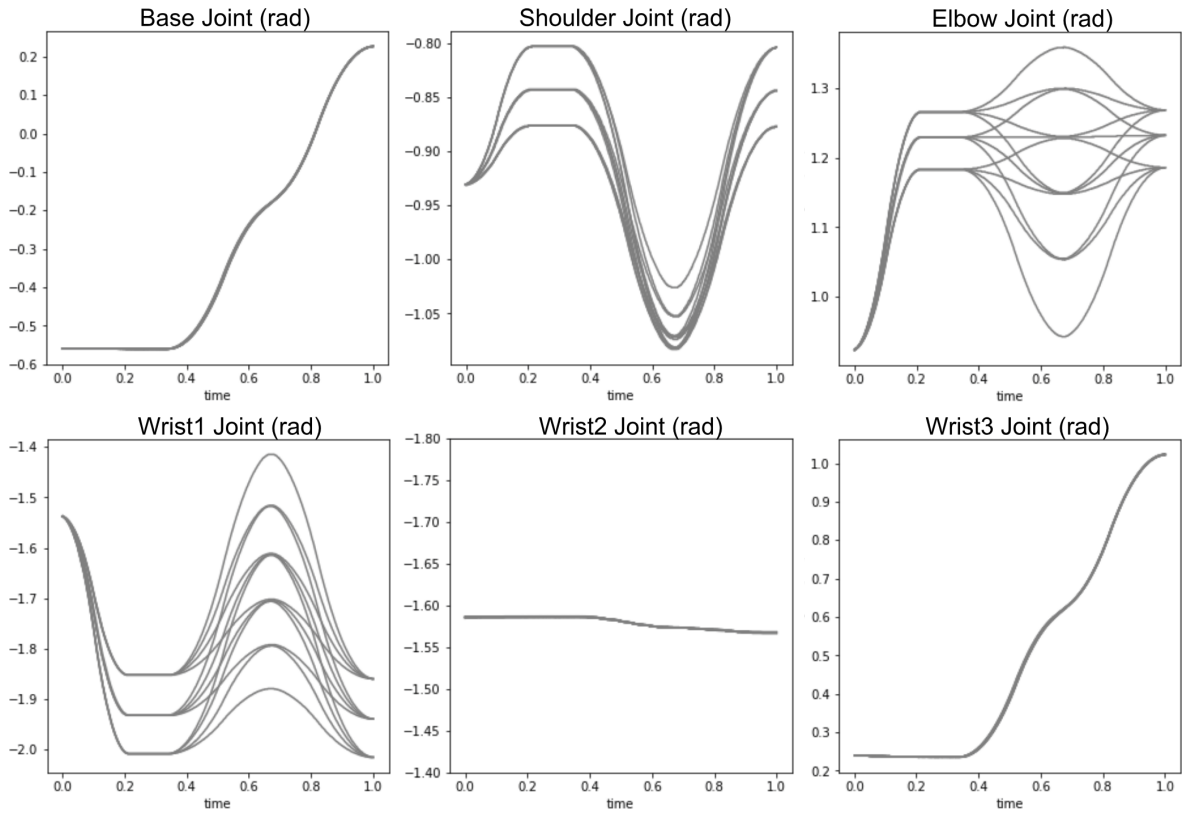

Figure 7.8. Demonstration data used for training the CNMP.

The generalization capacity of the system to novel object-obstacle settings was tested in two different conditions where the test distributions were sampled from inside and outside of the range of demonstration. Both conditions were tested with three different environments. In the interpolation condition, the robot generated a movement trajectory in the joint space successfully picking and placing the object avoiding the obstacle in all three tasks. The predicted and executed trajectories were compared against ground truth trajectories and the results of two example test cases are provided in the Figure 7.9(a,b). The gray lines indicate the demonstrations used for the training, the blue dashed line indicates the ground-truth, and the red line is the prediction of the system. The average absolute errors measured in degrees are also summarized in Table 7.1, where star indicates the novel conditions. The maximum average absolute error along the trajectories was 0.28 degrees. Considering these results, we conclude that the prediction mechanism of the CNMP can generalize to the novel situations sampled inside the training distribution. In the extrapolation cases, our motivation was to gradually increase difficulty and evaluate the limits of the system. For this purpose, the heights of the object and obstacle was gradually increased. The task

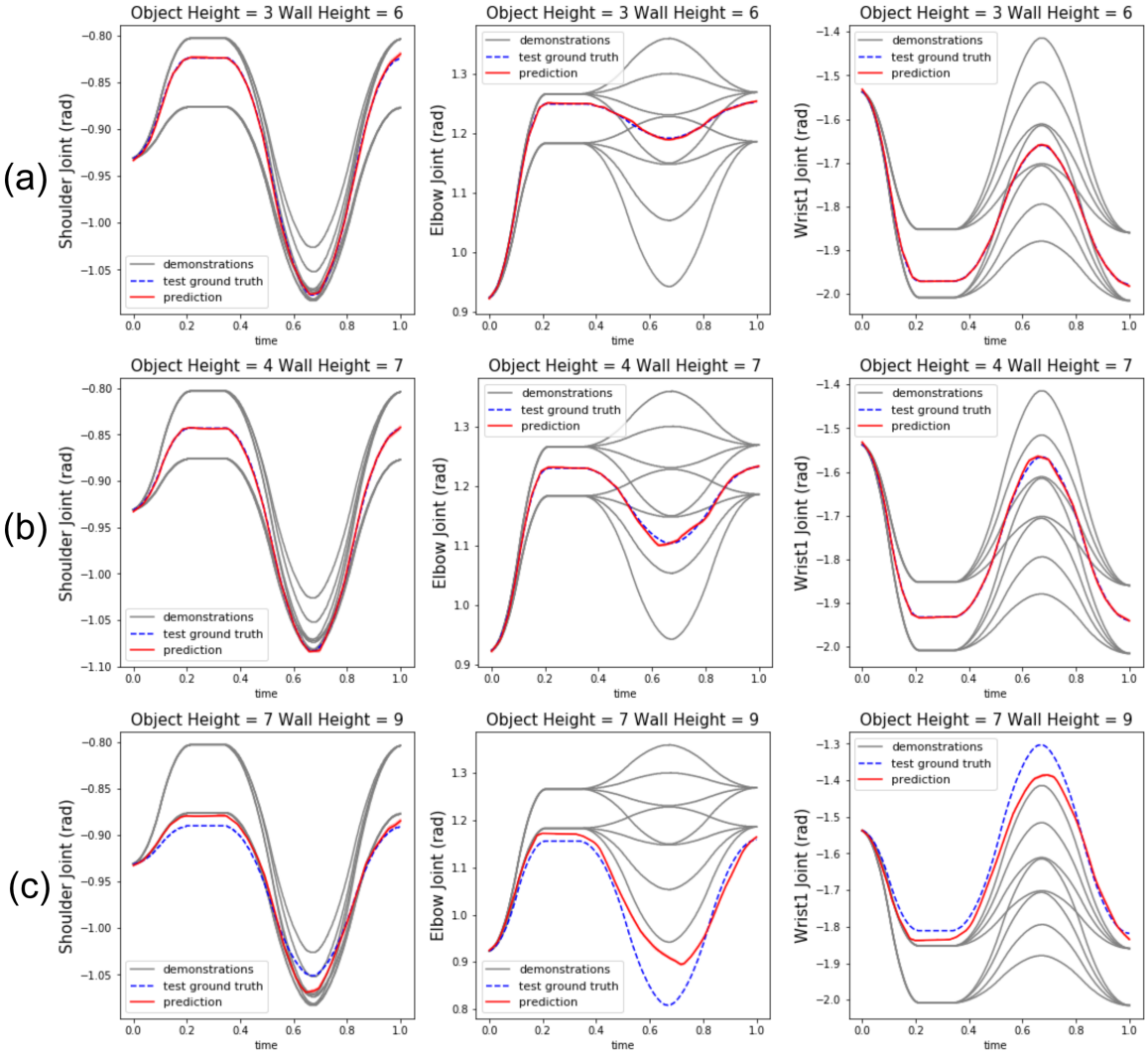

Figure 7.9. Qualitative results of selected three joints on novel interpolation (a,b) and extrapolation (c) cases.

was successfully executed in $\gamma = (7, 9)$ parameter configuration but the robot started failing when the object and wall height exceeded $\gamma = (8, 10)$. Figure 7.9(c) shows the trajectory prediction of one of the trajectories generated and Table 7.1 bottom three rows present the average absolute errors for the extrapolation cases. As the results show the performance of the trajectory prediction for extrapolation novel cases decreased significantly, especially when the extremity of the case with respect to the boundaries of the training. This is a general issue of deep neural networks that we revisit in the Conclusion Chapter.

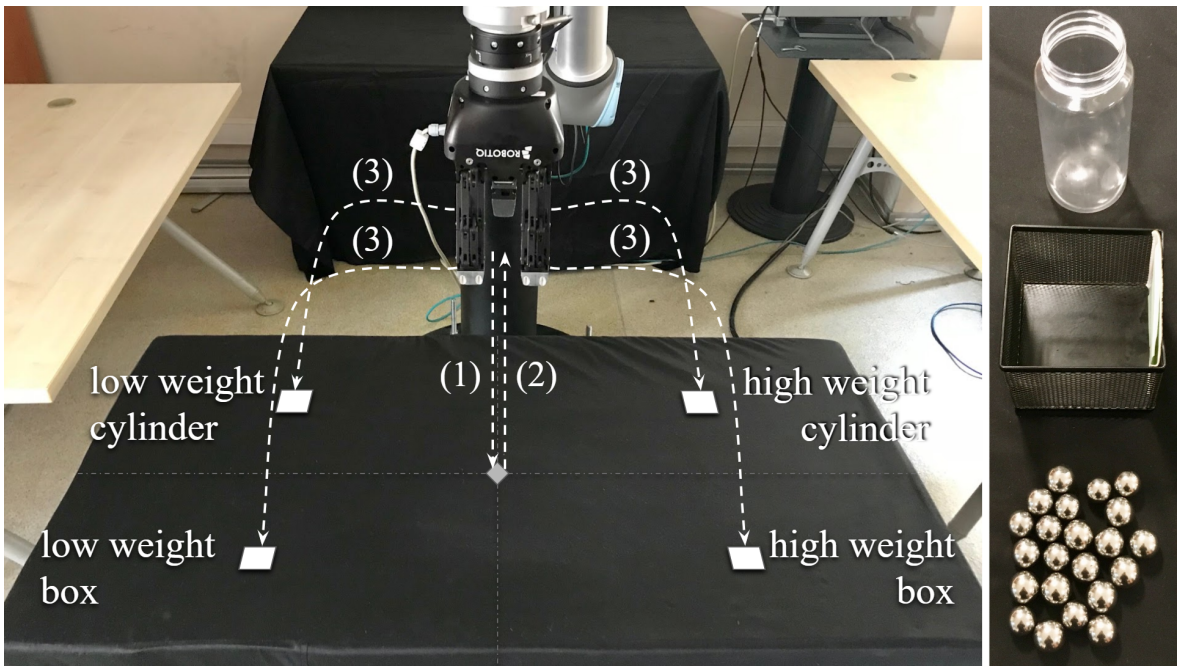

Figure 7.10. The experimental setup for pick-and-place task where the placement position depends on the weight and shape of the container. Left: the movement trajectories for different configurations. Right: Container types and marbles.

7.5. Movement Primitive Classification and Adaptation to Perturbations in Real Robot

This experiment aimed to demonstrate the movement generation capability of the CNMP taking into account the categorical semantics included in the demonstrations and its adaptation capacity in response to external perturbations. The UR10 robot arm, Robotiq gripper and force/torque sensor system was used to teach a pick and place

Table 7.1. Average joint errors (in degrees).

	Joint Errors (deg)					
Task Parameters	**Base**	**Shoulder**	**Elbow**	**Wrist1**	**Wrist2**	**Wrist3**
Obj: 3* Obs.: 3*	0.23	0.271	0.173	0.232	0.064	0.254
Obj: 3* Obs.: 6	0.158	0.167	0.145	0.163	0.061	0.126
Obj: 4* Obs.: 7*	0.099	0.101	0.269	0.280	0.055	0.093
Obj: 7* Obs.: 9*	0.719	0.895	3.569	3.687	0.156	0.665
Obj: 8* Obs.: 10*	1.038	2.325	8.879	8.094	0.360	0.830
Obj: 9* Obs: 11*	1.861	5.380	17.978	14.894	0.590	1.386

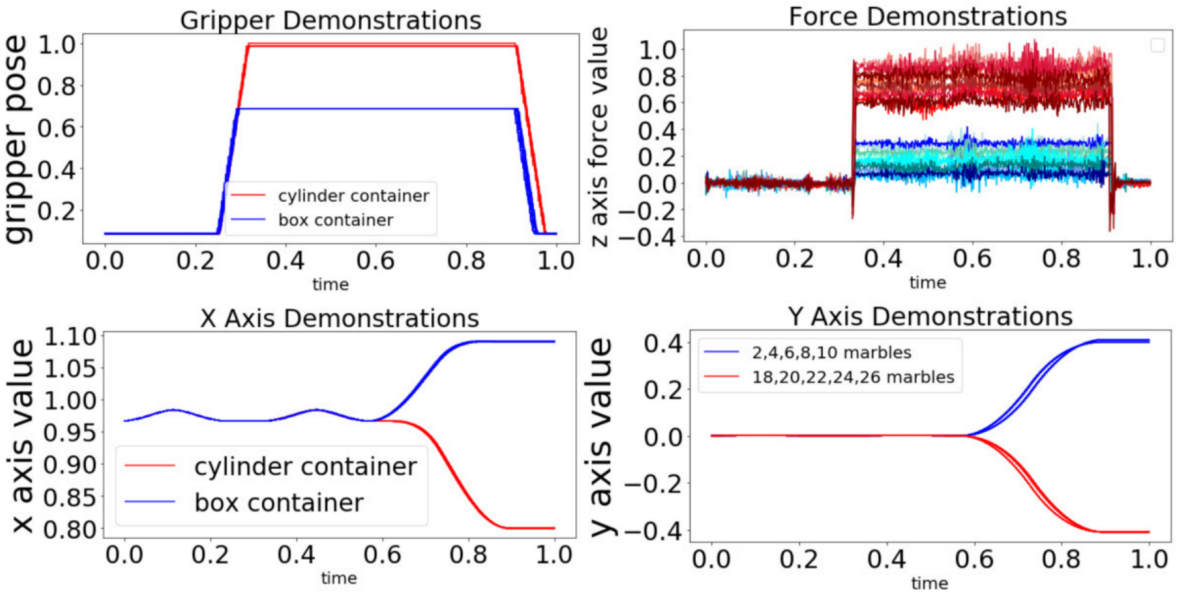

Figure 7.11. Demonstration data used for training the real robot.

task applied to a container placed on a table. The task is designed with the following schedule: The object was picked up from the middle of the table, and placed at one of the four target locations depending on the type of the container and its weight. Two different containers were used with different amounts of weights as shown in Figure 7.10, resulting in four different types (2 containers x 2 weight categories) of demonstrations. 5 different weights were used for each weight category, therefore 4x5=20 demonstrations were provided to the system for training, following the trajectories shown with the dashed lines in the same figure. SM vector included the 3d position of the gripper, the joint positions of the fingers, and the force/torque readings.

In testing, given a container (cube with a low weight) unknown to the robot, the robot started the execution without any information on the conditions and updated its predictions online. Before interacting with the container, any four target-points were equally likely for the system, therefore the variance of the prediction started high as shown in the middle-left plot in Figure 7.13. The robot next reached to the object and started interacting with it: As soon as it enclosed its fingers and started lifting the object, the received feedback from the interacted container through proprioception and force/torque measurements automatically conditioned the robot to generate a trajectory that ends up in the lower-bottom corner from a low-variance distribution (the

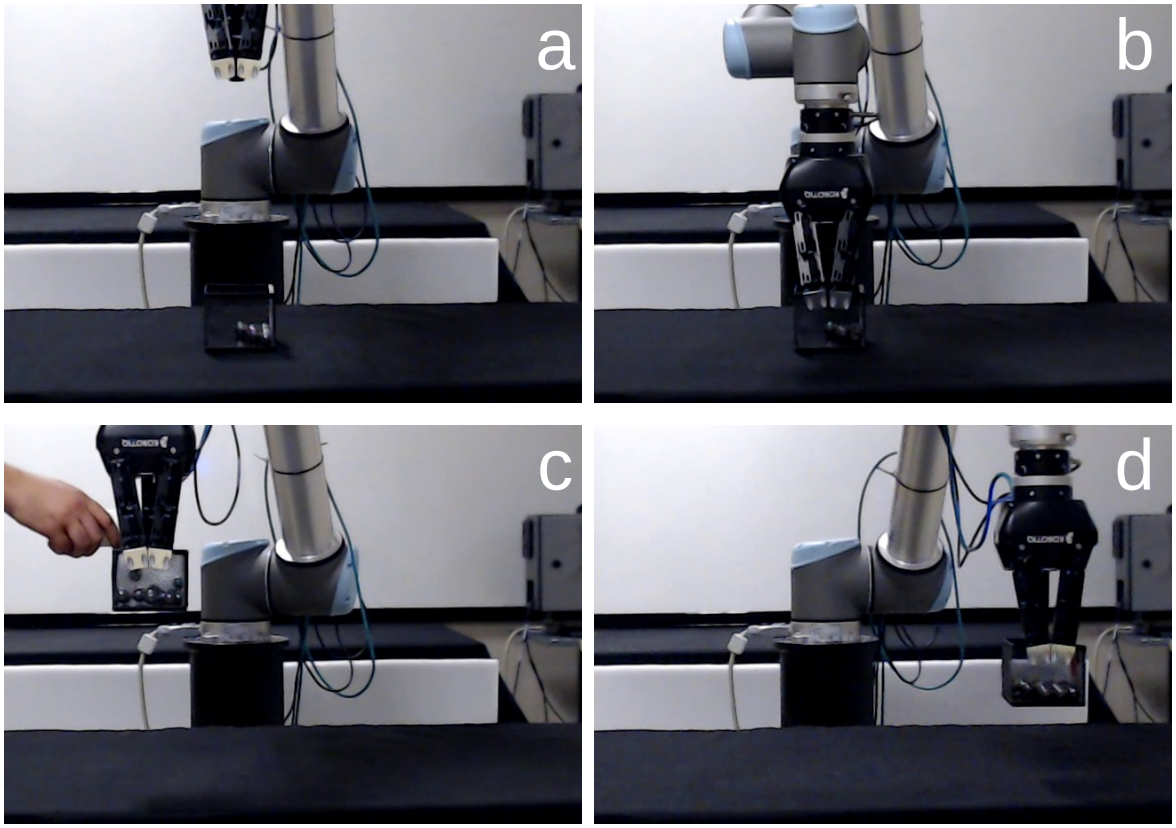

Figure 7.12. The robot generates the required movement trajectories, automatically perceiving the task-related features such as the weight and shape of the object through its sensors (proprioception and force/torque readings). Video at [3].

bottom-left plot in Figure 7.13). During the movement of the robot, we stepped in and put more marbles inside the box inside the gripper, effectively increasing the weight of the cube carried by the robot. The updated SM conditioning automatically updated the distribution of the movement trajectory, setting the final point to the bottom-right region on the table (see the bottom-right plot in Figure 7.13) and changing the motion of the end-effector towards that position through the PD controller. The increase in the time variable is suspended until the error between predicted and actual SM vanished. Finally, the cube with the increased weight was placed to the position that was demonstrated for high-weight boxes.

This experiment showed that although the task parameters (type and weight of the container) were not explicitly provided as parameters, the association between the movement trajectory and these parameters were learned in the SM space of the

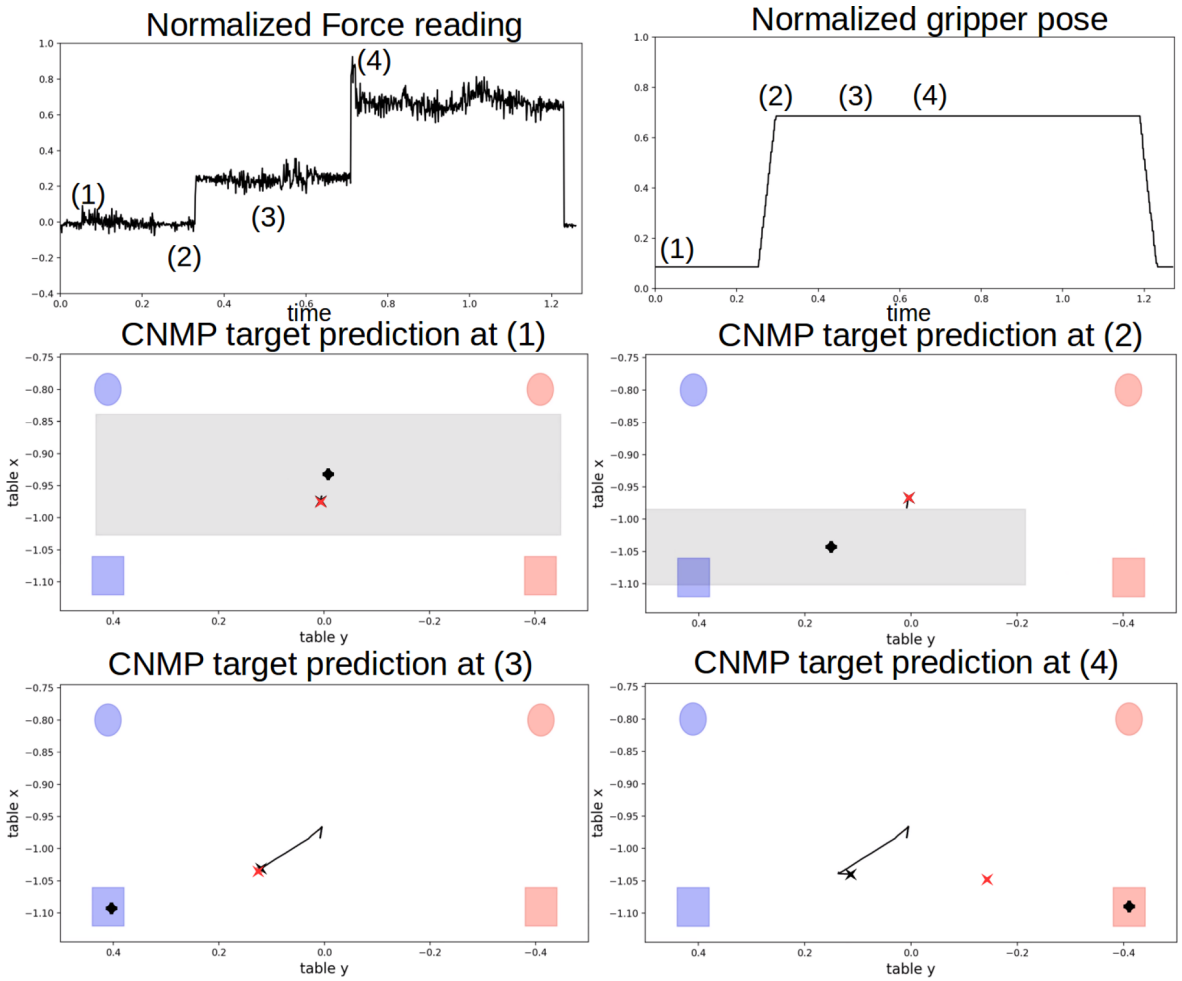

Figure 7.13. Online conditioning and adaptation to unexpected changes during the execution. Final position estimations are shown with black dots and gray shades. Container is approached (1), grasped but not lifted (2), lifted (3), made heavier (4).

robot through associating finger proprioception values and force/torque readings with the movement trajectories of different types. Furthermore, thanks to conditioning the system with the current SM in each time step, the system was shown to effectively react to changes that influence the task execution. Such a reaction was possible as the robot inferred that the target position only depended on the sensor measurements and was invariant from the current location of the robot.

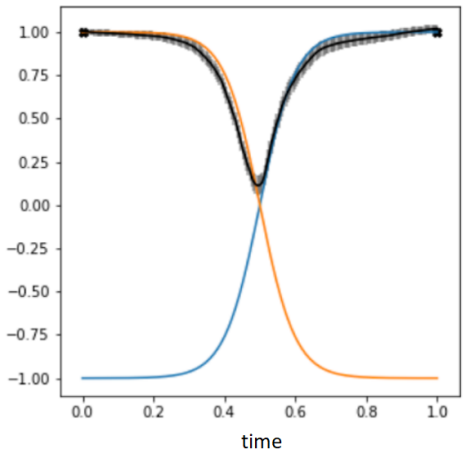

Figure 7.14. An example scenario where temporal representation weighting is needed in order to accomplish the task. Black dots are the conditioning points.

7.6. Movement Primitive Blending via Temporal Representation Weighting

In this experiment, we examined a special type of task where we introduced an alternative modification in our proposed system. In this experiment, our aim was to show that while we are conditioning our system at any time point of the execution, in some cases, some of the corresponding representations of our conditions may become more important than the others depending on the time of the execution. For example, a condition of the start of the movement could be not important anymore when the execution time continues to go far and far away, and the system might have to need to start ignoring the conditioning at the beginning and give more importance to the conditionings come after it.

In Chapter 6, we introduced an averaging layer in order to reduce the changing number of observations into one single general representation. The averaging layer basically weights all representations equally and takes the mean of all representations. However, in such cases, instead of using equal weights, system should weight the representations that are closer to the executing time that is robot is currently queried on. Figure 7.14 shows an example where there are two trajectories defined as $tanh(t)$ and $-tanh(t)$ functions. The two conditioning points are $(t = 0, y = 1)$ and $(t = 1, y = 1)$. As it can be seen in the figure, there is no such primitive in the demonstration set in

order to accomplish the task according to conditionings. However, system can switch between two primitives at the middle of the execution if the representations are locally weighted. In such system, through the middle of the execution, the importance of the condition at the beginning decreases as well as the importance of the final condition is increasing. This allows our system to produce a trajectory by switching between the demonstrations. We called this alternative version of the CNMP as Locally Weighted CNMPs.

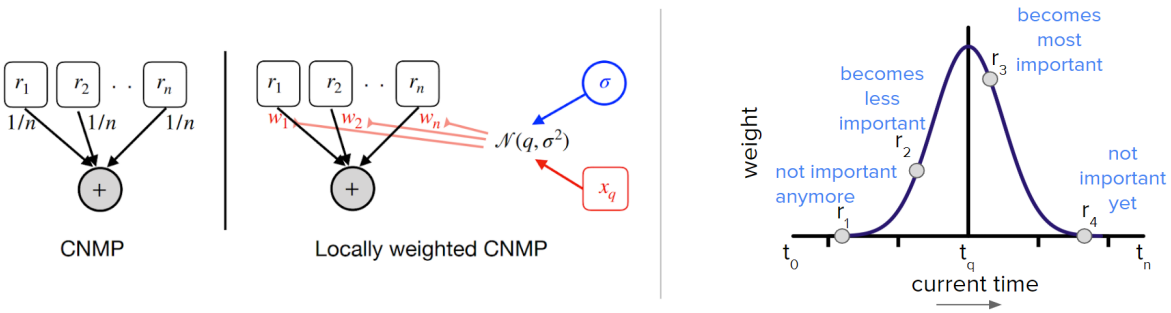

Figure 7.15. Left: Comparison of the CNMP and the Locally Weighted CNMP. Right: An example scenario where normalized Gaussian weights are used.

Figure 7.15 explains the local weighting method that we are using. Instead of weighting all of the representations $1/n$ where n is the observation number at the moment, we used normalized Gaussian weights where the mean of the Gaussian is t_q and the σ is a hyper-parameter that allows us to set the locality strength of the weighting. If the variance of the Gaussian increases, the system starts to behave more like normal CNMP because the weights of the representations become more close to being equal. On the other hand, the system starts to behave more locally weighted when the variance of the Gaussian decreases. This allows us to change the behaviour of our system on the fly.

In order to show the capabilities of the Locally Weighted CNMPs, we construct an environment where the representations lead the system to blend movement primitives and construct a new movement primitive to satisfy the conditions. The important point is we do not tell the system to switch primitives explicitly. The system automatically switches between primitives because representations encoded from observations allow our system to produce a blending movement from one primitive to another smoothly.

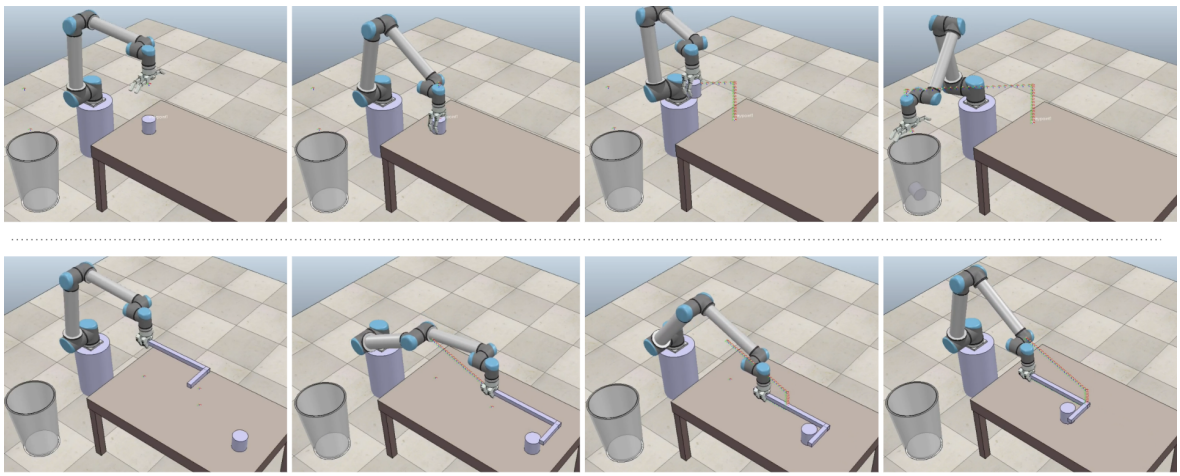

Figure 7.16. Two movement primitive demonstrations that we used in the experiment.

Figure 7.16 shows the movement primitives that is used for training our system. First row in the figure shows the movement primitive where robot reaches to the object, grasps it, carries it to the trash can and opens its gripper to throw it. The second row shows the movement primitive where the robot starts its movement grasping a tool, reaches to the object at the end of the table and pulls the object towards itself. We trained a CNMP with these two demonstrations where we recorded the 3D (x, y, z) of the end point of the robot and the 1D gripper status, $g \in [0, 1]$.

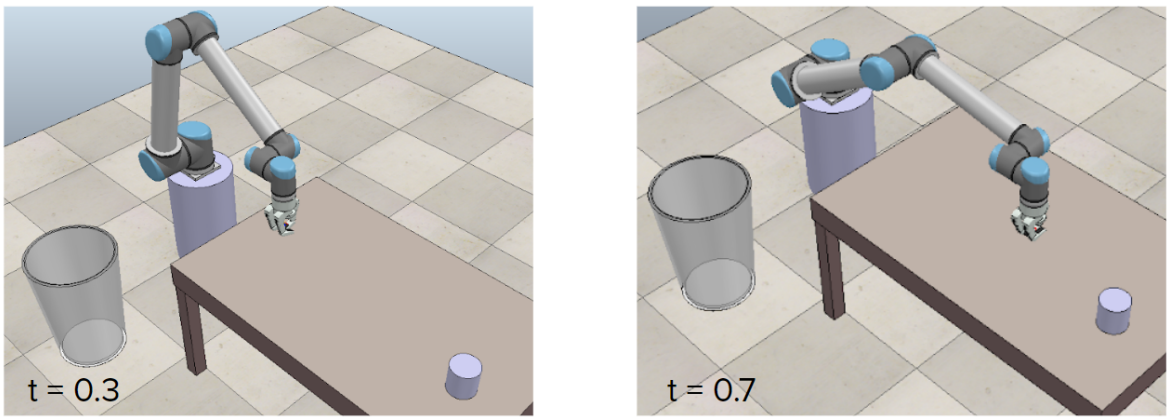

Figure 7.17. Two conditions that are used in the test time.

In the test time, we expect our system to produce a movement primitive that is not in the training set. In order to show that, instead of starting the robot grasping the tool as in the second primitive, we put the tool to the position of the object and start the robot just like in the first movement primitive. Figure 7.17 shows the visual

configurations of the conditionings that we want our system to produce a trajectory at the test time. First condition is at $t = 0.3$, positioned at in front of the table and gripper is closed. The second condition is at $t = 0.7$, positioned at the other side of the table and gripper is closed. Note that none of the demonstrations that we trained our system can satisfy these conditions by itself. However, since we are using locally weighted CNMPs, we expect our system to switch from one primitive to another automatically in order to satisfy the conditions.

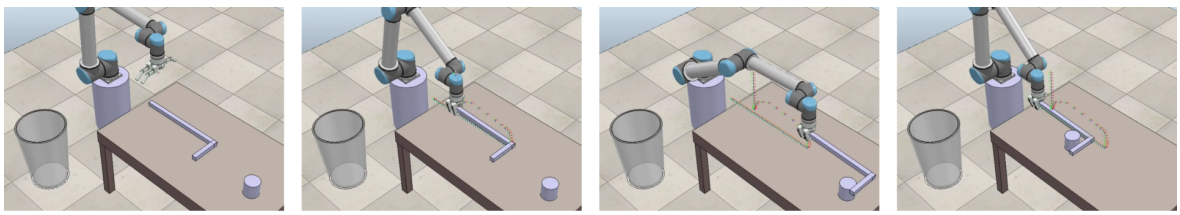

Figure 7.18. Produced trajectory using Locally Weighted CNMPs.

Figure 7.18 shows the produced trajectory using Locally Weighted CNMPs. Resulting trajectory shows that our system successfully predicts a trajectory using the first movement primitive at the start, reaching to the tool and grasping it, then switches to the second movement primitive in order to reach to the object and pull it towards itself. At the end, our system was able to produce a movement primitive that is not in the training set which is reaching to the tool on the table and then reaching to the object to pull it. These results show that our system can be very flexible and has a potential to even produce new movement primitives automatically thanks to the representation learning based on conditionings.

8. DISCUSSION AND CONCLUSION

Our CNMP method was shown to learn the non-linear relationships between environment parameters and complex action trajectories, deal with raw input through convolution operation, learn multiple modes of operations for the same primitive, and generalize to novel configurations if they are sampled from the experienced range. A wide range of relationships were shown to be automatically discovered and learned. In the first real robot pick-and-place task, the robot learned a continuous non-linear relation between the parameters and the action trajectories, whereas in the second robot experiment, it learned that containers with different weights were required to be placed in the same final positions. In the second robot experiment, the factors that influenced the task execution, such as weight and shape of the object, were never provided to the robot. Still, based on its interaction experience, the robot successfully learned to generate the required movement trajectory through exploiting the learned encoding in its low-level sensorimotor space that included proprioception and force readings. Furthermore, our continuous online conditioning with the current sensorimotor values enabled the robot to detect and respond to changes in the factors that influence task execution, on the fly. The PD controller enabled reacting to such external perturbations, however without any convergence or goal-achievement guarantees. In the future, we plan to study on mechanisms that makes search in the execution history and in the encoding space of the sensorimotor representations, and possibly rewinding the taken steps in order to successfully reach to the new environment state exploiting the learned multi-modal relations.

REFERENCES

1. Garnelo, M., D. Rosenbaum, C. Maddison, T. Ramalho, D. Saxton, M. Shanahan, Y. W. Teh, D. Rezende and S. M. A. Eslami, "Conditional Neural Processes", *ICML*, pp. 1704–1713, 2018.

2. Zhou, Y. and T. Asfour, "Task-oriented generalization of dynamic movement primitive", *IROS*, pp. 3202–3209, 2017.

3. Seker, M. Y., *CNMP execution video*, `https://youtu.be/cPKOIafOmUc`, accessed at May 2020.

4. Argall, B. D., S. Chernova, M. Veloso and B. Browning, "A survey of robot learning from demonstration", *Rob. and Auto. Sys.*, Vol. 57, No. 5, pp. 469–483, 2009.

5. Pahic, R., A. Gams, A. Ude and J. Morimoto, "Deep Encoder-Decoder Networks for Mapping Raw Images to Dynamic Movement Primitives", *ICRA*, pp. 1–6, 2018.

6. Paraschos, A., C. Daniel, J. Peters and G. Neumann, "Probabilistic movement primitives", *NIPS*, pp. 2616–2624, 2013.

7. Calinon, S., P. Evrard, E. Gribovskaya, A. Billard and A. Kheddar, "Learning collaborative manipulation tasks by demonstration using a haptic interface", *Advanced Robotics, 2009. ICAR 2009. International Conference on*, pp. 1–6, IEEE, 2009.

8. Asfour, T., P. Azad, F. Gyarfas and R. Dillmann, "Imitation learning of dual-arm manipulation tasks in humanoid robots", *International Journal of Humanoid Robotics*, Vol. 5, No. 02, pp. 183–202, 2008.

9. Pastor, P., L. Righetti, M. Kalakrishnan and S. Schaal, "Online movement adaptation based on previous sensor experiences", *IROS*, pp. 365–371, 2011.

10. Ben Amor, H., O. Kroemer, U. Hillenbrand, G. Neumann and J. Peters, "Generalization of human grasping for multi-fingered robot hands", *IROS*, pp. 2043–2050, IEEE, 2012.

11. Mühlig, M., M. Gienger and J. J. Steil, "Interactive imitation learning of object movement skills", *Autonomous Robots*, Vol. 32, No. 2, pp. 97–114, 2012.

12. Universal Robots, *UR10 Website*, `https://www.universal-robots.com/products/ur10-robot/`, accessed at May 2020.

13. Hochreiter, S. and J. Schmidhuber, "Long Short-Term Memory", *Neural Comput.*, Vol. 9, No. 8, pp. 1735–1780, 1997.

14. Rabiner, L. R., *Readings in Speech Recognition*, chap. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, pp. 267–296, Morgan Kaufmann Publishers Inc., 1990.

15. Belongie, S., J. Malik and J. Puzicha, "Shape Context: A New Descriptor for Shape Matching and Object Recognition", *Advances in Neural Information Processing Systems 13*, pp. 831–837, MIT Press, 2001.

16. Belongie, S. and J. Malik, "Matching with shape contexts", *Proceedings Workshop on Content-based Access of Image and Video Libraries*, pp. 20–26, 2000.

17. Belongie, S., J. Malik and J. Puzicha, "Shape matching and object recognition using shape contexts", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Apr 2002.

18. Bohg, J. and D. Kragic, "Grasping familiar objects using shape context", *2009 International Conference on Advanced Robotics*, pp. 1–6, June 2009.

19. E. Rohmer, M. F., S. P. N. Singh, "V-REP: a Versatile and Scalable Robot Simulation Framework", *IROS*, pp. 1321–1326, 2013.

20. *MATLAB Machine Learning Toolbox*, 2017, the MathWorks, Natick, MA, USA.

21. Schaal, S., "Dynamic movement primitives-a framework for motor control in humans and humanoid robotics", *Adaptive Motion of Animals and Machines*, pp. 261–280, Springer, 2006.

22. Calinon, S., "A tutorial on task-parameterized movement learning and retrieval", *Intelligent Service Robotics*, Vol. 9, No. 1, pp. 1–29, 2016.

23. Atkeson, C. G., A. W. Moore and S. Schaal, "Locally weighted learning for control", *Lazy learning*, pp. 75–113, Springer, 1997.

24. Vijayakumar, S. and S. Schaal, "Locally weighted projection regression: Incr. real time learning in high dimensional space", *ICML*, pp. 1079–1086, 2000.

25. Lee, D. and C. Ott, "Incremental kinesthetic teaching of motion primitives using the motion refinement tube", *Autonomous Robots*, Vol. 31, No. 2-3, pp. 115–131, 2011.

26. Ude, A., A. Gams, T. Asfour and J. Morimoto, "Task-specific generalization of discrete and periodic dynamic movement primitives", *IEEE Transactions on Robotics*, Vol. 26, No. 5, pp. 800–815, 2010.

27. Pervez, A. and D. Lee, "Learning Task Parameterized Dynamic Movement Primitives using mixture of GMMs", *Intelligent Service Robotics*, Vol. 11, pp. 61–78, 2018.

28. Pastor, P., H. Hoffmann, T. Asfour and S. Schaal, "Learning and generalization of motor skills by learning from demonstration", *ICRA*, pp. 763–768, 2009.

29. Chu, V., I. McMahon, L. Riano, C. G. McDonald, Q. He, J. Martinez Perez-Tejada, M. Arrigo, N. Fitter, J. C. Nappo, T. Darrell *et al.*, "Using robotic exploratory procedures to learn the meaning of haptic adjectives", *ICRA*, pp. 3048–3055, 2013.

30. Girgin, H. and E. Ugur, "Associative Skill Memory Models", *IROS*, pp. 6043–6048, 2018.

31. Ugur, E. and H. Girgin, "Compliant Parametric Dynamic Movement Primitives", *Robotica*, Vol. 38, pp. 457–474, 2019.

32. Droniou, A., S. Ivaldi and O. Sigaud, "Deep unsupervised network for multimodal perception, representation and classification", *Robotics and Autonomous Systems*, Vol. 71, pp. 83–98, 2015.

33. Kramberger, A., A. Gams, B. Nemec, D. Chrysostomou, O. Madsen and A. Ude, "Generalization of Orientation Trajectories and Force-torque Profiles for Robotic Assembly", *Robot. Auton. Syst.*, Vol. 98, No. C, 2017.

34. Seker, M. Y., *CNMP implementation*, `https://github.com/myunusseker/CNMP`, accessed at May 2020.

35. Flowers, I., *ProMP implementation*, `https://github.com/flowersteam/promplib`, accessed at May 2020.