IMPLEMENTATION OF A CLOSED-LOOP ACTION GENERATION SYSTEM ON A HUMANOID ROBOT THROUGH LEARNING BY DEMONSTRATION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

DORUK TUNAOĞLU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

SEPTEMBER 2010

Approval of the thesis:

IMPLEMENTATION OF A CLOSED-LOOP ACTION GENERATION SYSTEM ON A HUMANOID ROBOT THROUGH LEARNING BY DEMONSTRATION

submitted by **DORUK TUNAOĞLU** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Adnan Yazıcı
Head of Department, **Computer Engineering** _____

Asst. Prof. Dr. Erol Şahin
Supervisor, **Computer Engineering Department, METU** _____

Examining Committee Members:

Prof. Dr. Göktürk Üçoluk
Computer Engineering Department, METU _____

Asst. Prof. Erol Şahin
Computer Engineering Department, METU _____

Asst. Prof. Sinan Kalkan
Computer Engineering Department, METU _____

Asst. Prof. Uluç Saranlı
Computer Science Department, Bilkent University _____

Assoc. Prof. Osman Parlaktuna
Electrical and Electronics Engineering Dept., Eskişehir Osmangazi Univ. _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: DORUK TUNAOĞLU

Signature :

ABSTRACT

IMPLEMENTATION OF A CLOSED-LOOP ACTION GENERATION SYSTEM ON A HUMANOID ROBOT THROUGH LEARNING BY DEMONSTRATION

Tunaoğlu, Doruk

M.S., Department of Computer Engineering

Supervisor : Asst. Prof. Dr. Erol Şahin

September 2010, 61 pages

In this thesis the action learning and generation problem on a humanoid robot is studied. Our aim is to realize action learning, generation and recognition in one system and our inspiration source is the mirror neuron hypothesis which suggests that action learning, generation and recognition share the same neural circuitry. Dynamic Movement Primitives, an efficient action learning and generation approach, are modified in order to fulfill this aim. The system we developed (1) can learn from multiple demonstrations, (2) can generalize to different conditions, (3) generates actions in a closed-loop and online fashion and (4) can be used for online action recognition. These claims are supported by experiments and the applicability of the developed system in real world is demonstrated through implementing it on a humanoid robot.

Keywords: Action Generation, Generalization, Humanoid Robot, Dynamic Movement Primitives

ÖZ

GÖSTERİM YOLUYLA ÖĞRENEN BİR KAPALI-DEVRE HAREKET YARATMA SİSTEMİNİN İNSANSI BİR ROBOTTA GERÇEKLEŞTİRİLMESİ

Tunaoğlu, Doruk
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Yrd. Doç. Dr. Erol Şahin

Temmuz 2010, 61 sayfa

Bu tezde, insansı bir robot üzerinde hareket öğrenme ve yaratma problemi çalışılmıştır. Amacımız hareket öğrenme, yaratma ve tanıma işlerini ortak bir sistem üzerinden gerçekleştirmektir ve esin kaynağımız bu işlerin beyinde aynı sinir sistemi üzerinden yapıldığını iddia eden ayna nöron hipotezidir. Bu amaca ulaşmak için, verimli bir hareket öğrenme ve yaratma mekanizması olan Dinamik Hareket İlkelleri üzerinde değişiklikler yapılmıştır. Geliştirdiğimiz sistem (1) çoklu gösterimlerden öğrenebilmekte, (2) değişik durumlar için genelleme yapabilmekte, (3) hareketleri kapalı-devre ve eş-zamanlı olarak yaratmakta ve (4) eş-zamanlı hareket tanıma için kullanılabilmektedir. Bu iddialar deneylerle desteklenmiş ve geliştirilen sistemin gerçek dünyada kullanılabilirliği insansı bir robot üzerinde uygulama yapılarak gösterilmiştir.

Anahtar Kelimeler: Hareket Yaratma, Genelleme, İnsansı Robot, Dinamik Hareket İlkelleri

For Absent Friends

ACKNOWLEDGMENTS

Firstly, I would like to express my gratitude to my supervisor Erol Şahin for supporting and guiding me in my graduate studies. Also, I am thankful to him for providing me an excellent research environment, KOVAN Laboratory, where I had the opportunity to work with a humanoid robot and other high-tech devices.

I am very lucky to have met with Barış Akgün in our research lab. Apart from being a perfect colleague, he is a great ally and a true friend from the heart whom I encounter very rarely in my life. *Blonde commander*, I hope we will meet some day soon.

I would like to thank all my research partners in KOVAN: Hande Çelikkanat for helping me and cheerfully mocking me even when she is busy, Fatih Akgün for his calm advices about life and research, İlkay Atıl for the joy that he spreads out and for long philosophical discussions, Nilgün Dağ for being the mother of the laboratory and Sinan Kalkan for his helpful comments. In addition, I have to express my gratitude to Erhan Öztop and Emre Uğur for taking care of us during our one month visit to Advanced Telecommunications Research Institute at Japan with Nilgün.

My dearest close friends, I thank to you all for being with me in all times. You showed me your love in the light of the stars when the dawn seemed forever lost and I hope you will remember me when the dark night seems endless.

My special thanks goes to our iCub humanoid robot who allowed me to possess his mind and body. Without his support, this thesis could not be completed.

My beloved family, I could not express how lucky I am for having you. I am grateful to mom and dad for trusting and supporting me even when they disagree with me and raising me freely, and to my dearest sister for actually being a real friend, in fact a ridiculously funny one. My aunt, uncles, grandmothers and my little cousins; I love you from the heart. A wise man once said “Fate chooses our relatives, luckily we can choose our friends.” and I have to admit that fate has treated me fairly.

I would like to acknowledge the support of TÜBİTAK (The Scientific and Technological Research Council of Turkey) BİDEB 2210 graduate student fellowship and this thesis is supported by TÜBİTAK under the project with id 109E033 and European Commission under the ROSSI project with id FP7-216125.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

The progressive nature of robotics reveals new areas of usage for robots everyday. One such area is the use of robots at homes and offices where they have to both work alone and in co-operation with humans. This is a vast research topic where diverse areas such as knowledge representation, computer vision and intention understanding intersect. Action learning and generation is one of these areas where development of robust algorithms with good generalization performance is crucial.

Advances in humanoid robots have enabled roboticists to study and implement human-like motion on robots. Furthermore, the rise in the number of cognitive and neural studies on humans and monkeys is providing us valuable information about how "nature" deals with the problem of action learning and generation.

Mirror neurons, present in the F5 area of the premotor cortex of macaque monkeys, are such an inspiration source for this thesis. These neurons fire both during the recognition and the generation of goal/object directed actions [5, 6] which suggests that action learning, recognition and generation share the same neural circuitry. In addition, functional magnetic resonance imaging (fMRI) studies on humans support the existence of a mirror neuron system in human brain [7, 8].

In this thesis, we study the action learning and generation problem on a humanoid robot. Demonstrations of different actions by a human actor are used for training our system. Afterwards, the system is tested in different conditions to assess its performance in terms of robustness and generalization. Finally, the generation system is implemented on a humanoid robot to perform learned actions in real world.

1.1 Terminology

Some terms have to be clarified before going into further discussion:

- **Action** is defined as a bodily behavior performed by an agent. An action is goal/object directed if it is performed on an object (i.e. Grasping a cup is object directed whereas walking is not).
- **Action Generation** means generating the appropriate motor commands for performing an action.
- **End-effector** is the body part of the actor which interacts with the object (i.e. hand and fingers for grasping, foot for kicking).
- **Trajectory** describes how the end-effector goes from an initial state to the final state during an action (i.e. reaching to the right of the object can be performed in many different ways).
- **Task Space** is the space where the end-effector is controlled. For instance the choice of controlling the Cartesian position of the hand or controlling the joint angles of the arm defines different task spaces for controlling the same end-effector.
- **Motor Space** is the space where the robot recognizes the commands to move its body parts. A robot's mechanical structure defines the motor space. For most of the humanoid robots, motor space is either joint velocities or joint accelerations.
- **Inverse Model** calculates the appropriate motor commands to achieve a desired state in the task space. For instance, inverse model answers the question "Given the desired position of the robot's hand, what must be the joint angles?".
- **Forward Model** calculates the state in the task space given the state in the motor space. For instance, forward model answers the question "Given the joint angles of a robot, what is the position of its hand?".
- **Learning by demonstration** is concerned with teaching robots actions from demonstrations. Some alternatives to learning by demonstration are: Hand coding actions, optimizing a criterion while performing an action (i.e. minimum jerk) and learning actions by motor babbling.

- **Closed-Loop Generation** considers the current state of the end-effector while generating a new motor command. For instance, filing a cup with water while observing the water level is a closed-loop execution.

- **Open-Loop Generation** does not consider the current state of the end-effector or assumes that desired state is reached while generating a new motor command. For instance, filing a cup with water with your eyes and ears are closed is an open-loop execution because you will stop filing without observing the current state of the cup (you have to make an assumption about it most probably considering time).

1.2 Aim of the Thesis

In this thesis, our aim is to develop an action generation system which can learn from demonstration and which is suitable for action recognition. This aim is inspired by mirror neuron hypothesis which suggests that action learning, generation and recognition shares the same neural circuitry. Furthermore, an action generation system should satisfy other requirements so that it can be implemented on a real system: First of all, the system should be able to learn from demonstrations since hand-coding and tuning of actions for different objects and conditions are tedious and time consuming tasks. Second, the system should generalize the action that it has learned since an action can not be demonstrated for all different conditions. Furthermore, online or fast offline generation is desired so that the robot can interact with the environment in real time[1]. The system should be robust so that perturbations such as noisy sensor readings do not affect the result of an action. Correspondence problem, defined as mapping a demonstrated action to the robot's motor space, has to be solved in order to achieve learning by demonstration. Using a humanoid robot highly eases the correspondence problem since it shares a similar body structure with a human. Finally, an action recognition system (preferably online for online interaction) should be built upon the generation system easily.

To summarize, *action generation requirements* that should be satisfied are:

- **Learning by Demonstration:** An action should be learned from human demonstra-

[1] Online generation means that the next state of the system is calculated from current conditions, whereas in offline generation whole trajectory is generated before starting the action.

tions in order to avoid hand-coding every action and tuning its parameters for different conditions.

- **Generalization:** A learned action should be applied to different objects in different conditions without demonstrating the trajectory for all cases since it is impossible to demonstrate an action for all different cases.

- **Fast Generation:** Action generation should be fast so that the robot can interact with humans and environment in real time. Online or fast offline generation can be used for this purpose.

- **Robustness:** A robot should be able to cope with the perturbations that may occur during action generation and these perturbations should not affect the result of an action.

- **Correspondence:** A robot should map an observed action to its task space so that it can learn and generate the same action.

- **Action Recognition:** An action recognition system should be implemented on top of the action generation system without requiring much effort. Furthermore, online recognition is more desirable than offline recognition since effective interaction with a human requires intention understanding before an action is completed.

Note that fast generation, preferably online, highly affects the robustness of the system since robustness requires generating new commands periodically to deal with perturbations.

We reviewed various action generation algorithms and decided to modify Dynamic Movement Primitives (DMPs) [9, 10] in order to develop a system that satisfies the above requirements.

In another thesis [11], which is done in parallel with this thesis, we have studied and implemented an online action recognition system on top of the system that is developed in this thesis. This thesis and [11] complement each other and they should be considered together to understand how the same system is used in action learning, generation and recognition.

This thesis is mainly supported by TÜBİTAK (The Scientific and Technological Research Council of Turkey) under the project with id 109E033 which addresses the question of how concept formalization and tool use can be developed on humanoid robots.

Another source of support for this thesis is the European Union Framework 7 Project ROSSI

[12] which addresses the question of how the possibility of communication between agents (i.e. humans and robots) is affected by differences in sensorimotor capacities. Action generation is one of the components of the large architecture of the ROSSI project which also includes affordance learning, intention understanding and sensorimotor grounding of robotic conceptualization and language use (verbal labeling of objects and actions).

The rest of the thesis is organized as follows: Chapter 2 presents the state of the art for action learning and generation. Chapter 3 analyzes Dynamic Movement Primitives, one of the surveyed approaches, in detail and describes how we modified them in order to overcome some of their shortcomings. Chapter 4 focuses on the experimental platform, describing the humanoid robot, motion capture system and other components used in the experiments. Experimental setup, experiments performed using this setup and the results of them are provided and discussed in chapter 5. Finally, chapter 6 concludes the thesis and informs the reader about future work.

CHAPTER 2

RELATED WORK

In this chapter we review some of the approaches in the literature for action generation and make a comparison between them. See [2] and [13] for a detailed review of the state of the art for action learning, generation and recognition.

2.1 The Mimesis Model

The mimesis[1] model is an action learning, generation and recognition model which abstracts body motions as symbols [1, 14]. These symbols denote different body states (also called as motion elements) such as walk, rest and squat. In the mimesis model, an action is represented by a Hidden Markov Model which is a probabilistic model between hidden states and observable motion elements as seen in figure 2.1.

A Hidden Markov Model (HMM) is a statistical model which calculates the probabality of a sequence of states given the sequence of observations and vice versa. In a HMM, states are not directly observable but they affect the observations. Probability of transition between states is called transition probabilities (TP) and observation of an output given the state is called the output probabilities (OP). Depending on the application, TP and OP can be defined by a programmer or can be learned with an algorithm.

Action learning in the mimesis model corresponds to estimating TP and OP given an observed action. Assuming that motion elements are known, which can be learned from observations as well, observed action is divided into non-overlapping windows with same size and each

[1] Mimesis is an ancient Greek word and it carries a wide range of meanings which include imitation, representation and mimicry.

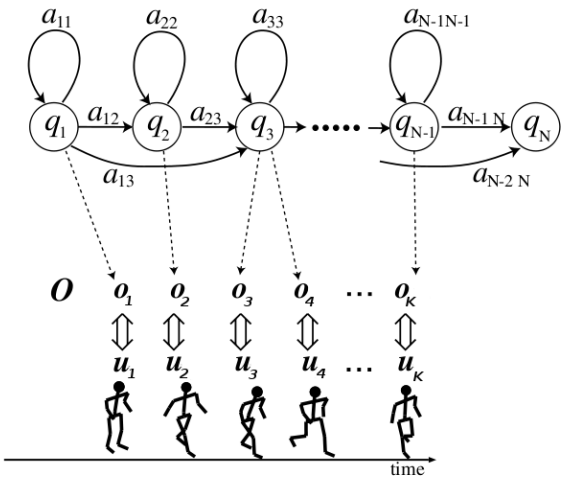


Figure 2.1: Elements of the mimesis model. q and **o** represent the states and observations respectively. a represents the transition probabilities and dotted lines shows the output probabilities (Not labeled with a symbol). **u** shows the observed motion elements. Figure taken from [1] with the kind permission of the corresponding author.

window is labeled with the motion element that best describes the samples in it. The index of the motion element (j) that is assigned to a window with the observed body state θ is calculated as:

$$j = \arg\max_i \frac{\exp\left[-(1/2)(\theta-\mu_i)^T\Sigma_i^{-1}(\theta-\mu_i)^T\right]}{\sqrt{det\ \Sigma_i}}, \tag{2.1}$$

where μ_i and Σ_i are the mean vector and covariance matrix of the i^{th} motion element. A sequence of motion elements $u_k = [u_{k_1}\ u_{k_2}\ \ldots\ u_{k_T}]$ is obtained by applying this operation for each window. Finally, TP and OP are calculated for the observed action using the Baum-Welch algorithm [15] which is an expectation maximization algorithm. In this context, different TP-OP pairs describe different actions.

Action generation in the mimesis model corresponds to making stochastic transitions between states until the end state is reached. Steps followed for action generation are as follows:

1. Let the starting state be q_1 and let motion elements sequence $O = \emptyset$.

2. Decide on the next state q_n stochastically considering the current state and TP of the generated action.

3. Decide on the output (motion element) o_n stochastically considering q_n and OP of the generated action.

4. Add the new motion element to the sequence. $O := [O\ o_n]$.

5. If the end state q_N is not reached, return to step 2. Otherwise, continue.

6. Transform the sequence of motion elements into continuous joint angles and convert them into appropriate motor commands.

Action recognition in the mimesis model is performed by calculating the probability of actions given an observation sequence. Then, R(O) is defined as the recognition confidence to have an analytical measure for recognition:

$$R(O) = \log \frac{\max_i\ P(O \mid ACT_i)}{second_j\ P(O \mid ACT_j)}, \tag{2.2}$$

where O is the sequence of observations and ACT_i is the i^{th} action. If $R(O)$ is above a threshold then the action that gives maximum probability for the observation sequence is recognized (ACT_i). Overall architecture of the mimesis model is shown in figure 2.2.

2.2 Recurrent Neural Networks with Parametric Biases

Recurrent Neural Networks with Parametric Biases (RNNPB)[16, 17] is an action learning, generation and recognition model which is built on top of a Jordan-Type recurrent neural network. A new unit called *parametric bias* (PB) vector, which can be used as input or output according to the context, is added to the network in order to change the behavior of the network when different actions are generated.

Jordan-Type recurrent neural networks simulate dynamical systems by using the past values of the hidden layer as additional inputs. This operation is done through context units by feeding the previous context output as context input to the network as seen in figure 2.2. Context units can be thought as the state of the network and by feeding the previous state as input to the system, dynamical systems are simulated.

PB vector in RNNPB defines the characteristics of the dynamical system simulated by the network. Different trajectories can be generated by changing the PB vector and a PB vector uniquely represents an action. Note that different actions have different PB vectors but share the same network weights. This is useful for incremental learning since similar actions can be learned by updating the PB vector of an already learned one if network weights are stabilized.

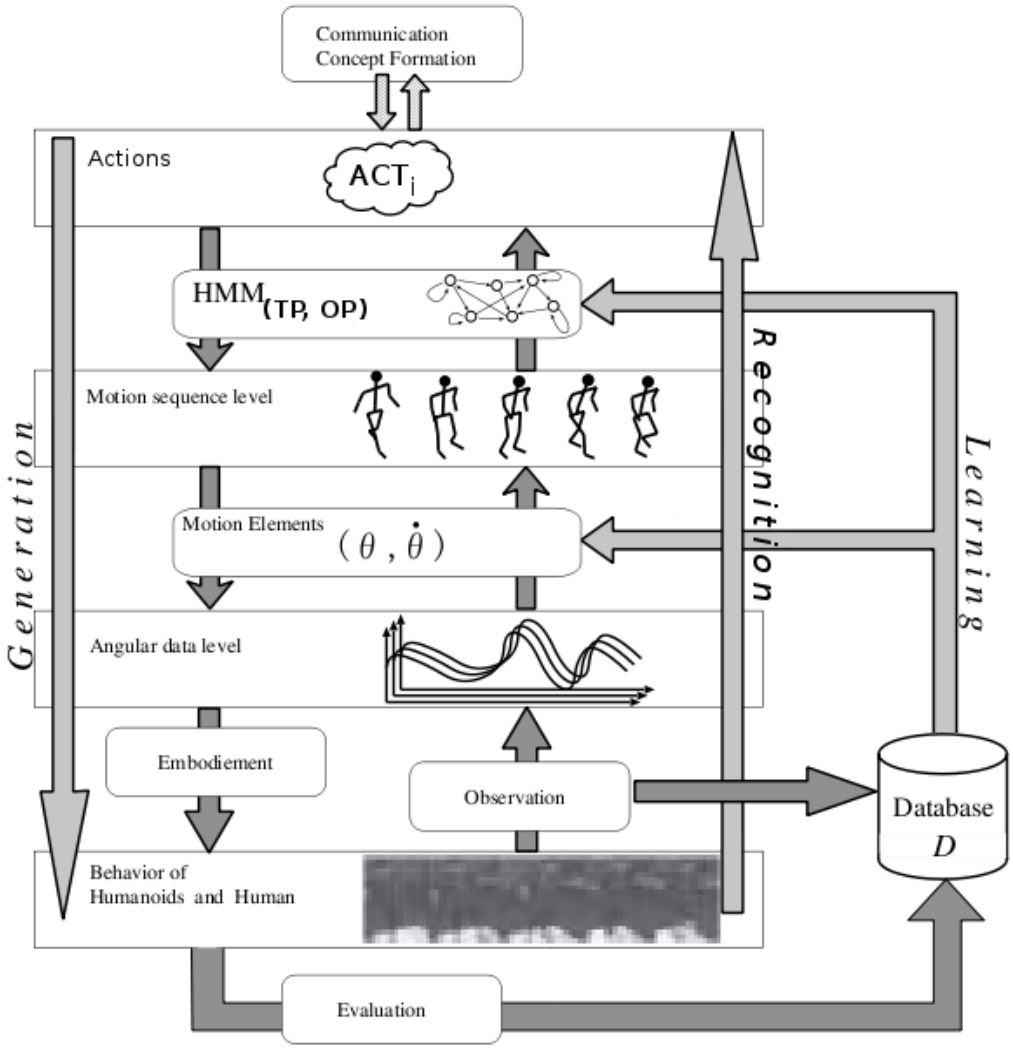


Figure 2.2: Overall architecture of the mimesis model. Figure taken from [1] with the kind permission of the corresponding author.

Action generation in RNNPB is done by generating a target sensorimotor state given the PB vector of the action and the current sensorimotor state as seen in figure 2.3(a). Difference between the motor components of the target and current state are used to generate appropriate motor commands. In the figure, sensory component of predictions are directly fed back as sensory input to the system which corresponds to open-loop execution. For closed-loop execution, actual sensory readings can be used.

Learning by demonstration in RNNPB is done by estimating the PB vector and updating the network weights given an observed action. During learning, current and next sensorimotor state of a demonstration are presented to the network as input and output respectively. According to the prediction error, network weights and the PB vector are updated as seen in figure 2.3(b). These updates are done using back-propagation through time algorithm (not shown in the figure) and they continue until the PB vector and the network weights converges.

Action recognition in RNNPB is done by estimating the PB vector given the sensorimotor data of an observed action. PB vector is estimated by updating it according to the prediction error (without updating the network weights) as seen in figure 2.3(c). After PB vector converges to a value, it is compared with the PB vectors in the reportoire to recognize an aciton.

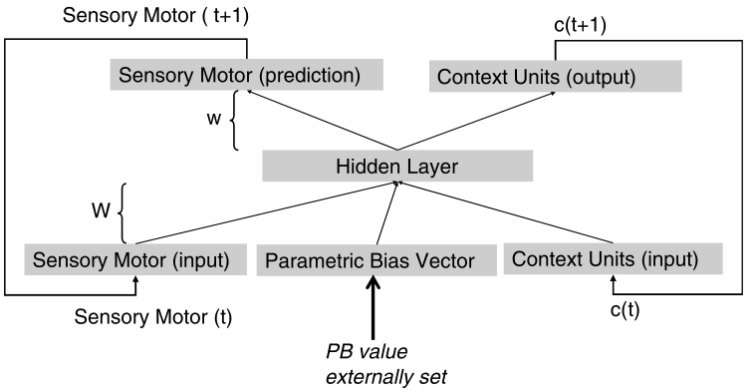
2.3 Modular Selection and Identification for Control

Modular selection and identification for control (MOSAIC) [18, 19] is an action learning, generation and recognition approach that uses multiple controller-predictor (CP) pairs to control an end-effector where the controllers are inverse models and the predictors are forward models. Generated motor command is defined as the weighted sum of controllers. The main idea is to assign a higher weight to a controller if prediction performance of its pair is high for the current state and a lower weight otherwise. Time evolution of these weights are called the responsibility signals since they define the responsibilities of controllers in generating motor commands. Figure 2.4 shows the flow diagram of MOSAIC in control mode.

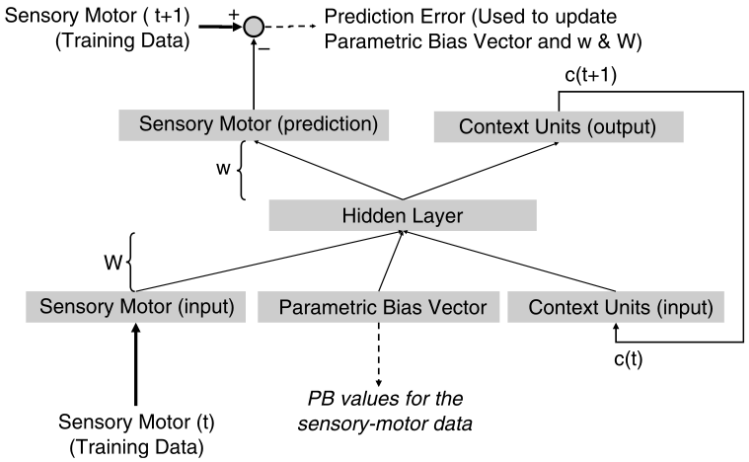
The predictors of MOSAIC predict the next (sensorimotor) state given the current state x_t and the motor command u_t:

$$\hat{x}^i_{t+1} = \phi(w^i_t, x_t, u_t), \tag{2.3}$$

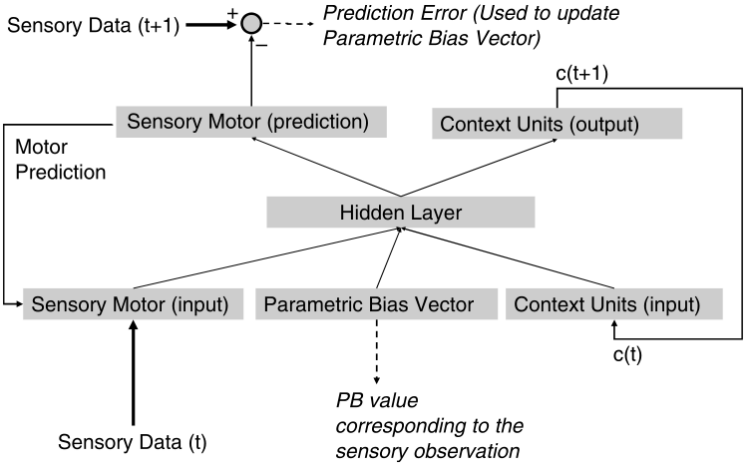
where $\phi(.)$ is a function approximator (i.e. a neural network) and w^i_t are the parameters of



(a) Action Generation



(b) Learning by Demonstration



(c) Action Recognition

Figure 2.3: Action generation, learning and recognition using RNNPB. Figures taken from [2] with the kind permission of the corresponding author.

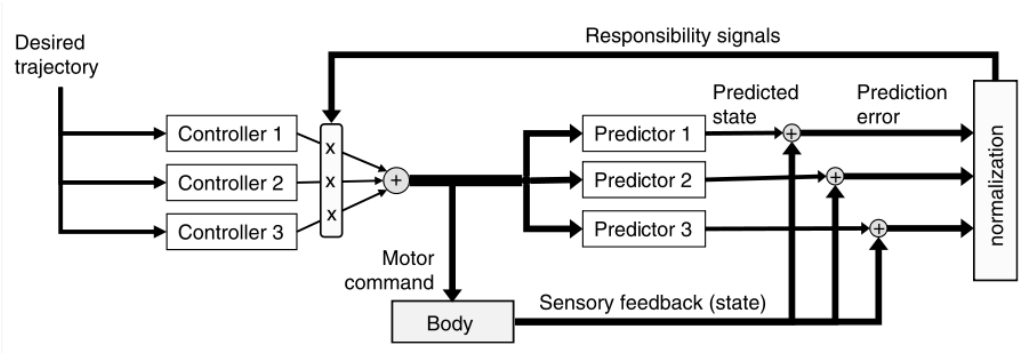


Figure 2.4: MOSAIC in control mode. Generated motor command is the weighted sum of the outputs of controllers where these weights are calculated according to the prediction errors. Figure taken from [2] with the kind permission of the corresponding author.

ϕ. These predicted states are compared with the actual state x_t to calculate prediction errors. Then these errors are normalized with a softmax function to obtain the responsibility signals:

$$\lambda_t^i = \frac{err_t^i}{\sum_{j=1}^{N} err_t^j}, \tag{2.4}$$

$$err_t^i = \exp\left(-\frac{(x_t - \hat{x}_t^i)^2}{\sigma}\right), \tag{2.5}$$

where N is the number of CP pairs, λ_t^i is the i^{th} CP pair's responsibility signal and σ is a scaling parameter.

The controllers of MOSAIC generate a motor command u_t given the current state x_t and the desired state x_{t+1}^*:

$$u_t^i = \psi(v_t^i, x_{t+1}^*, x_t), \tag{2.6}$$

where $\psi(.)$ is a function approximator and v_t^i are the parameters of it. Note that a controller is tightly coupled with its predictor pair where they are inverse functions of each other:

$$\hat{x}_{t+1}^i = \phi\left(w_t^i,\ x_t,\ \psi(v_t^i, \hat{x}_{t+1}^i, x_t)\right), \tag{2.7}$$

$$u_t = \psi\left(v_t^i,\ \phi(w_t^i, x_t, u_t),\ x_t\right). \tag{2.8}$$

To complete the system, the total motor command is defined as the sum of the outputs of controllers weighted by the responsibility signals:

$$U_t = \sum_{i=1}^{N} \lambda_t^i\, u_t^i = \sum_{i=1}^{N} \lambda_t^i\, \psi(v_t^i, x_{t+1}^*, x_t). \tag{2.9}$$

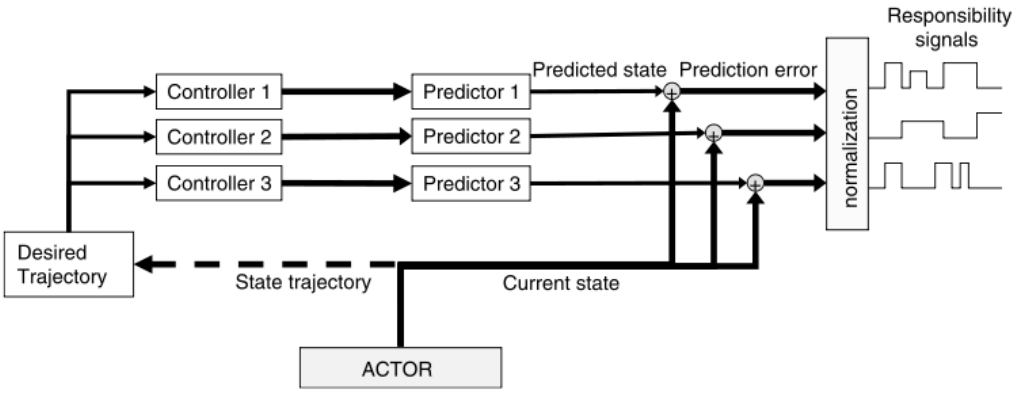


Figure 2.5: MOSAIC in learning/recognition mode. Observed trajectory is treated as the desired trajectory and the resulting responsibility signals are stored as the parameters of observed action for learning. For recognition, responsibility signals can be compared with the ones in the repertoire. Figure taken from [2] with the kind permission of the corresponding author.

There are two phases of learning in MOSAIC. The first phase is to train the CP pairs in order to increase the accuracy of the controllers and predictors. In this phase, responsibility signals are used as learning rates so that a controller's parameters are updated more if its predictor pair works with high accuracy. Parameters of predictors are updated in a similar fashion just after this step. Controllers and predictors are trained in a loop which continues until prediction errors gets below a threshold.

After CP pairs are trained, the second phase is learning the responsibility signals for an action. For each sample of a demonstrated action, responsibility signals are calculated and stored in memory as the signature of that action as shown in figure 2.5.

There are two ways of generating an action in MOSAIC: If desired trajectory is generated by an external system then it can be given directly to the system in figure 2.4. If trajectory generation is expected from MOSAIC, then the learned responsibility signals are used as controller weights. In this mode, prediction errors are not used for updating the responsibility signals.

Responsibility signals in MOSAIC are used for several purposes in [19]: Training the CP pairs, learning a demonstrated action after the training, generating trajectories and adjusting the weights of controllers when desired trajectory is given. Furthermore, responsibility signals can be used for action recognition as follows: An observed trajectory can be treated as the desired trajectory as in figure 2.5. Then responsibility signals of that observed trajectory can

be compared with the ones in the repertoire to recognize the action.

2.4 Dynamic Movement Primitives

Dynamic movement primitives (DMPs) [9, 10] are smooth trajectory generators which define a dynamical system for controlling a variable.

A DMP generates a trajectory $\vec{x}(t)$ through defining a differential equation for $\ddot{\vec{x}}(t)$. This equation is inspired from a mass-spring-damper (MSD) system perturbed by an external force [10]:

$$\ddot{\vec{x}} = \underbrace{\mathbf{K}(\vec{g} - \vec{x}) - \mathbf{D}\dot{\vec{x}}}_{\text{Canonical Part}} + \underbrace{\mathbf{K}\vec{f}(s, \mathbf{W})}_{\text{Non-linear part}} . \tag{2.10}$$

In equation 2.10, $\vec{x}$ is the vector of controlled DOFs. $\vec{g}$ is the goal and defined as the desired value of $\vec{x}$. $\mathbf{K}$ is a diagonal matrix where K_{ii} is the spring constant for the i^{th} DOF. Similarly, $\mathbf{D}$ is a diagonal matrix with damping coefficients. Mass is taken as one and thus not represented in the equation. $\vec{f}(\cdot)$ is the perturbing function where $\mathbf{W}$ represents the parameters of it. According to the $\vec{f}(.)$ formulation $\mathbf{W}$ can be a vector or a matrix. s is the one dimensional phase variable which is shared by all DOFs of $\vec{x}$. The canonical part corresponds to the MSD system and the non-linear part corresponds to the perturbing force.

Equation 2.10 generates the necessary accelerations so that $\vec{x}$ converges to $\vec{g}$. The trajectory for $\vec{x}$ can be generated by double integrating the accelerations obtained from this equation. Note that $\vec{x}$ is the state in the task space and variables of it are the variables of the task space. Actions can be generated on a robot by converting the accelerations in task space into accelerations in motor space. If joint angles are controlled as in [9] then there is no need for conversion, if Cartesian positions are controlled as in [10] an inverse model can be used to convert Cartesian accelerations to joint accelerations.

The canonical part of the system (MSD system) is a well known and widely studied system in physics which has a global asymptotic stable equilibrium point at $\{\vec{x}, \dot{\vec{x}}\} = \{\vec{g}, \vec{0}\}$. In other words, the system converges to the goal point with zero velocity as time goes to ∞. Moreover, if we ensure $D = 2\sqrt{K}$ then the system becomes critically damped and converges to the goal point as quickly as possible without generating oscillations.

The canonical part of the system generates a simple line trajectory between the starting point

and the goal point in the task space. To generate complex trajectories, the system is perturbed by $\vec{f}(\cdot)$ which is learned from demonstrations.

$\vec{f}(\cdot)$ is chosen as a sum of basis functions with respect to the phase variable s in [9, 10]. Let f_k and $\vec{w}^k$ be the k^{th} element of f and k^{th} row of $\mathbf{W}$ respectively, then:

$$f_k(s, \vec{w}^k) = \frac{\sum_{i=1}^{N} \psi_i(s)\, w_i^k}{\sum_{i=1}^{N} \psi_i(s)}\, s, \tag{2.11}$$

where $\psi_i(s)$ is the i^{th} basis function, w_i^k is the weight of the i^{th} basis function and N is the number of basis functions. As basis functions, Gaussian functions with center c_i and bandwidth h_i are used:

$$\psi_i(s) = \exp\left(-h_i\,(s - c_i)^2\right). \tag{2.12}$$

The phase variable s is used for synchronizing all DOFs. It decays from one to zero as an action unfolds:

$$s_0 = 1, \tag{2.13}$$

$$\dot{s} = -\alpha s, \tag{2.14}$$

$$\implies s = \exp\left(-\alpha(t - t_0)\right), \tag{2.15}$$

where α can be calculated by imposing $s = s_f$ when a convergence criterion is met (i.e. $s = 0.001$ when action is 95% converged).

With the addition of the non-linear part, the system is no longer guaranteed to converge to the goal point. However, convergence of the system is theoretically proved as follows: The non-linear part decays to zero as $s \to 0$ for finite $\mathbf{W}$ (Equation 2.11) and $s \to 0$ as $t \to \infty$ (Equation 2.15). Thus, the non-linear part will decay to zero as $t \to \infty$ and the canonical part will take the system to the goal point if $\|\mathbf{W}\| < \infty$.

Learning by demonstration using DMPs are achieved as follows: Given a recorded trajectory $\vec{x}(t)$, $\vec{f}(\cdot)$ has to be learned which means that $\mathbf{W}$ has to be estimated. First, each element of $\vec{f}(t)$ is calculated by leaving $\vec{f}$ alone in equation 2.10:

$$f_i(t) = \frac{\ddot{x}_i(t) - K_{ii}\,(g_i - x_i(t)) + D_{ii}\dot{x}_i(t)}{K_{ii}}. \tag{2.16}$$

Then $s(t)$ is calculated from equation 2.15. Since $s(t)$ and $\vec{f}(t)$ are calculated and s is a monotonically decreasing function between 1 and 0; $f(s)$ in $s \in (0, 1]$ is known. As the final step, least squares regression (Appendix A) is applied to estimate $\mathbf{W}$ given the basis functions.

2.5 Discussion

In this section we compare the aforementioned approaches considering the *action generation requirements*.

2.5.1 Learning by Demonstration

All of the mentioned approaches support learning by demonstration. The mimesis model, RNNPB and MOSAIC learn from recordings of joint angles but they can be extended to learn from Cartesian positions as well. DMPs can learn from any set of observed variables as long as there exists a forward and an inverse model for the variable choice. DMPs have been used with joint angles [9] and Cartesian positions [10] in previous works.

Another comparison for learning by demonstration can be made considering the number of demonstrations used in learning: Multiple or single. All of the mentioned approaches use one-shot learning since:

- motion elements, transition probabilities and output probabilities in the mimesis model,
- parametric bias vector in RNNPB,
- responsibility signals in MOSAIC,
- weight vectors in DMPs

are learned from a single demonstration and there are no means of learning the mapping between different conditions and these parameters in any of the approaches. On the other hand, these approaches can benefit from multiple demonstrations in different ways: Averaging the parameters to reduce the effects of noise, selecting the demonstration which optimizes a criterion or treating each demonstration as a new action. Note that this is not truly learning

from multiple demonstrations since novel conditions are only used for optimization instead of learning.

2.5.2 Generalization

The mimesis model learns the critical points (motion elements) of an action and how to make transitions between these critical points without learning how these elements should be changed according to context. Thus they cannot generalize for object directed actions (i.e. grasping, pushing) since they have no means of changing motion elements when the position and orientation of an object is changed.

RNNPB does not provide any means of goal setting since it does not take the goal as input. As a result, it cannot generalize for different conditions and it is not useful for object directed actions. A remedy for that situation can be to embed the goal in the parametric bias vector.

MOSAIC is poor in terms of generalization because it does not learn how to generate new trajectories for different conditions. Instead, it only learns how to generate motor commands in order to imitate the observed trajectory. To implement goal setting in MOSAIC, responsibility signals should be learned and generalized with respect to the goal position which does not seem as a simple task.

DMPs allow goal setting by an explicit variable (g). Furthermore they can cope with goal changes during action execution since they generate actions online (g can be changed during action execution).

2.5.3 Fast Generation

The mimesis model generates actions offline because the HMM of an action is simulated until it reaches to the end state before starting an action. In other words, all motion elements and the corresponding motor commands are calculated before an action starts. In [1, 14] it is shown that offline generation of an action with the mimesis model takes approximately a second. This can be considered as fast or slow generation according to the context.

RNNPB, MOSAIC and DMPs generate actions online since they generate a motor command considering the current sensorimotor state.

2.5.4 Robustness

The mimesis model generates actions offline and this is a drawback for the robustness of the system as mentioned in section 1.2. If offline generation can be completed before a perturbation causes major errors, then the system is robust. For instance, suppose that we push a robot while it walks. If it can generate new commands before it falls down then the system is robust, otherwise it is not. This is the reason why one second for offline generation can be considered as fast or slow according to the context.

RNNPB and MOSAIC generate actions online in a completely closed loop fashion and thus they are robust.

DMPs generate actions online as well, but they can be considered only as "half robust" since the non-linear part of DMPs works in an open-loop fashion. In other words, the non-linear part cannot cope with perturbations since it does not take the current sensorimotor state into account while generating new commands.

2.5.5 Correspondence

Corrsepondence problem is an relatively easy problem when:

1. The actor and the observer shares a similar body structure.

2. Directly controllable parameters of the body structure (i.e. joint angles) are used in action generation.

Using a humanoid robot satisfies the first requirement while using joint angles satisfies the second. Thus, the correspondence problem is easily solved in the mimesis model, RNNPB and MOSAIC since they use joint angles for observation and implement their approaches on humanoid robots. The same fact holds for DMPs when they are used for learning from joint angles and an inverse - forward model pair can be used to solve the correspondence problem when Cartesian positions are used for learning.

2.5.6 Action Recognition

All of the mentioned approaches support action recognition. However, DMPs are limited to offline recognition since some parameters are estimated from the convergence time of an action. Other approaches can be used for online recognition if they meet some requirements, but this has not been tested or mentioned in the cited papers. Each of these approaches can be used for online recognition if:

- recognition confidence (Equation 2.2) in the mimesis model calculated from partial observations is sufficient for confidence to exceed the threshold,
- parametric bias vector in RNNPB converges before an action finishes,
- partial comparison of responsibility signals in MOSAIC is sufficient for recognition.

Note that these comments are solely based on our understanding of these approaches (except the comments on DMPs) and they do not reflect the views of the authors.

2.6 Pros and Cons of Surveyed Approaches

The aim of this thesis is to develop an action generation system which:

- Learns from demonstrations (preferably from multiple demonstrations),
- Generalizes to different conditions,
- Generates motor commands in closed-loop (thus online and robust),
- Solves the correspondence problem,
- Can be used for online action recognition.

According to the aforementioned comparisons which are also summarized in table 2.1, we decided to build our system on top DMPs since they satisfy most of the *action generation requirements*. The most dominant advantage of DMPs that affected our choice was explicit goal setting since we could not come up with an easy way to add goal setting to other approaches.

Table 2.1: Comparison of Different Action Generation Approaches

	Learning by Demonstration	**Generalization**	**Fast Generation and Robustness**	**Correspondence**	**Action Recognition**
The Mimesis Model	- Joint angles - One-shot learning	- No explicit goal setting - Poor generalization	- Offline generation in one second - Fast/slow generation and thus robustness depends on the context	No correspondence problem (using joint angles)	Online recognition is possible (not implemented)
RNNPB	- Joint angles - One-shot learning	- No explicit goal setting - Poor generalization	- Fast and robust (online and closed-loop)	No correspondence problem (using joint angles)	Online recognition is possible (not implemented)
MOSAIC	- Joint angles - One-shot learning	- No explicit goal setting - Poor generalization	- Fast and robust (online and closed-loop)	No correspondence problem (using joint angles)	Online recognition is possible (not implemented)
DMPs	- Joint angles or Cartesian positions - One-shot learning	- Explicit goal setting - Good generalization	- Fast but “half” robust (online but “half” closed-loop)	An inverse model have to be used if Cartesian positions are used	Offline recognition (implemented and tested)

CHAPTER 3

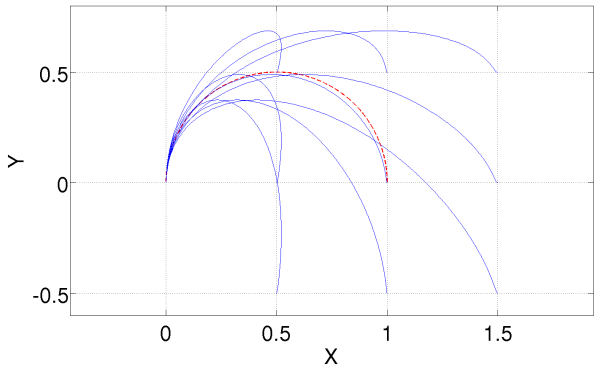
ACTION GENERATION

As explained in section 2.6, we decided to build our system on top of DMPs. In order to fully satisfy the *action generation requirements*, we extended the definition of DMPs to overcome their shortcomings.

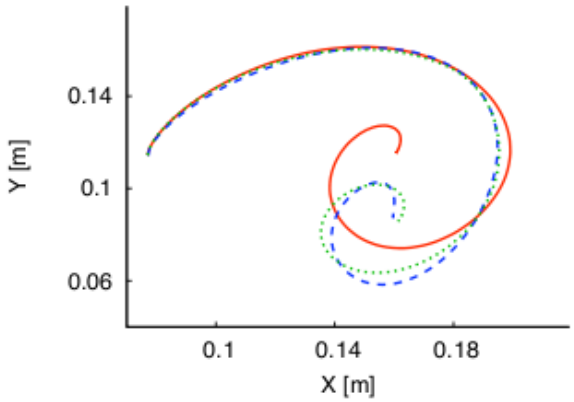
3.1 Advantages of Dynamical Movement Primitives

Some advantages of the DMPs can be summarized as follows:

- They can generate any smooth trajectory [10].
- They converge to the goal point for finite weight vectors.
- They allow the setting of the goal point explicitly, a feature that is crucial for applying the same action in different conditions.
- They can generalize to different conditions as seen in figure 3.1.
- They are invariant under affine transformations [10] such as the rotation and the translation of the coordinate system. This ensures the generation of the same action when the coordinate system changes but the relation between the starting point and the goal point remains the same.
- They allow superposition of new terms since they are defined as an explicit equation which consists of sum of different terms. For instance, in [10] additional terms that model human obstacle avoidance are superposed to the DMPs and used for obstacle avoidance on a robot arm.



(a) Goal adaptation of DMPs for simulated data. The red dashed curve is the demonstrated trajectory and the blue curves are the trajectories generated by DMPs.



(b) Goal adaptation of DMPs for a human demonstration. The red curve is the demonstrated trajectory, blue dashed curve is the trajectory of the human for the new goal and the green dotted curve is the one generated by DMPs. Figure taken from [10] with the kind permission of the corresponding author.

Figure 3.1: Generalization Performance of DMPs

- They can be used for offline action recognition by comparing the weight vectors of the demonstrated action with the vector of the actions in the repertoire.

3.2 Drawbacks of Dynamical Movement Primitives

The canonical part of DMPs runs in closed loop since it uses position ($\vec{x}$), velocity ($\dot{\vec{x}}$) and goal ($\vec{g}$) in its calculations without explicitly using time. On the other hand, the non-linear part does not use any of these variables and it depends only on s which is directly related to time (Equation 2.15). In other words, the non-linear part runs in open-loop and this causes

some problems for learning and generation:

- The non-linear part cannot react to perturbations on the end-effector. Figure 3.2 shows the effect of a perturbation (stopping the end-effector) applied to the end-effector: The non-linear part decays to zero in time and the trajectory expected from the system cannot be followed.

- The non-linear part applies the same accelerations to the system with respect to time irrespective of the initial and the current states. We tested this argument as follows: First a trajectory is learned with DMPs. Then new trajectories are generated by selecting points on this trajectory (x and $\dot{x}$) as initial conditions while leaving g constant. As seen from figure 3.3, DMPs cannot follow the same trajectory although the initial sensorimotor state of an action is equal to one of the sensorimotor states of the demonstrated action. In other words, DMPs generate different motor commands for the same sensorimotor states according to the starting time of an action.

- The behavior of the non-linear part, which only depends on time, is learned from a single demonstration of an action and it may change totally if another demonstration of the same action is used for learning. For instance, suppose that you are reaching to the right of an object. If you start from a point that is on the left side of the object you have to give a curvature to your action in order to avoid hitting the object. On the other hand, if you start from its right side then you reach directly to the object. If the former action is demonstrated to DMPs then they will generate similar curvatures even if an action starts from the right side. If the latter action is demonstrated to the DMPs then they will learn to reach directly to the object and they will hit the object in some cases. In this example, DMPs can not generalize to different conditions unless they treat the two different demonstrations of the same action as different actions.

DMPs are used for offline recognition since current definition of DMPs require whole motion to be observed (including the starting point) before any recognition has been done. Mathematically, α in equations 2.14 and 2.15 has to be estimated from the convergence time of the action. Furthermore, offline recognition may not be performed if starting time of an action (which is an intrinsic parameter of the actor) is not observed or estimated by an observer.

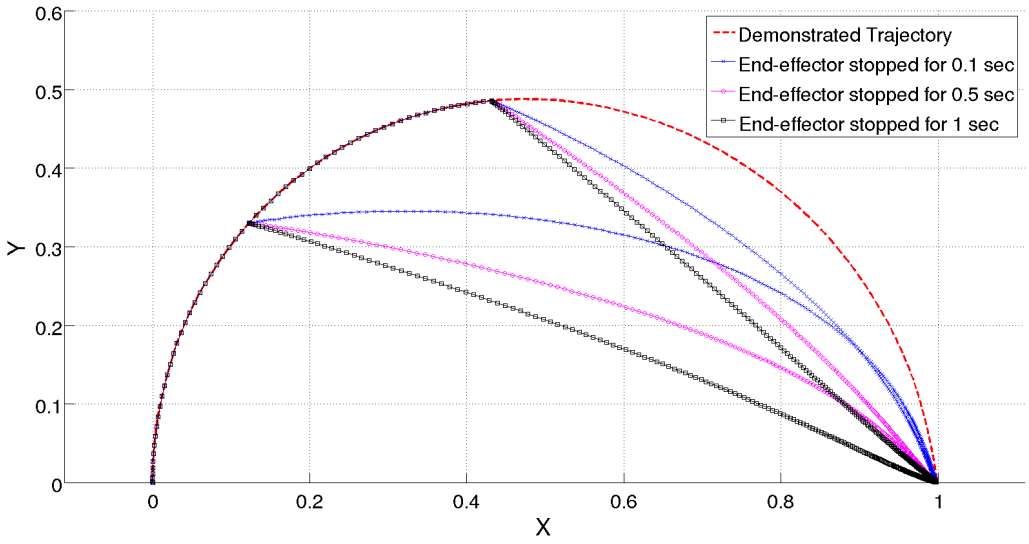


Figure 3.2: Effects of stopping the end-effector for different durations on the generated trajectory.

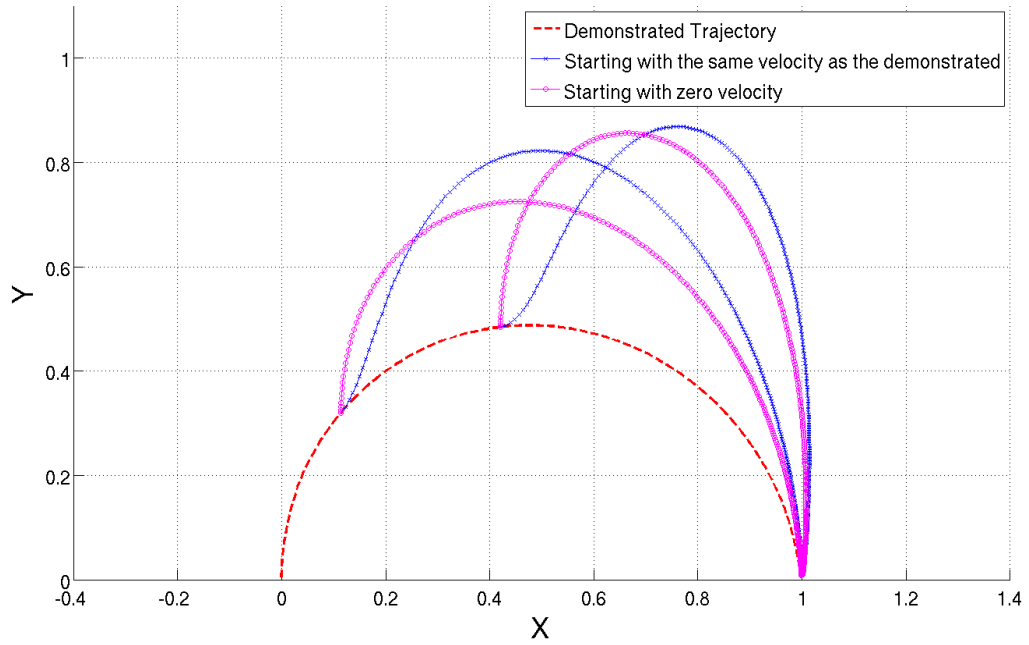


Figure 3.3: Different trajectories generated by taking different points on the demonstrated trajectory as starting points.

3.3 Extending Dynamical Movement Primitives

We extended the DMPs in order to satisfy *action generation requirements* and replaced the non-linear part that runs in open-loop with a new part that runs in closed-loop.

One approach for making the non-linear part run in closed-loop is to change the phase variable to $\frac{g-x}{g-x_0}$(ratio of current distance to the goal to the initial distance) for each DOF seperately as in [9]. This choice makes the non-linear part closed-loop while conserving the generalization properties of DMPs. On the other hand, it does not generate any motion for a DOF when $g = x_0$ for that DOF[1]. Furthermore, it is not useful for action recognition since the phase variable depends on x_0, an intrinsic parameter of an actor, which may not be observed by the observer.

Another choice of phase variable is $\frac{g-x}{x_{MAX}}$ (ratio of distance to the goal to a constant maximum distance) for each DOF seperately. This choice solves the recognition problem since the phase variable does not depend on the intrinsic parameters of the actor. However, it still cannot generate any motion when $g = x_0$. In addition to that, its generalization performance is poor since the non-linear part does not generate any acceleration for unobserved $(g - x)$ values which corresponds to unobserved values of the phase variable.

There are other choices for the phase variables as well (in fact infinitely many): $\frac{\|\vec{g}-\vec{x}\|}{\|\vec{g}-\vec{x_0}\|}$, $\frac{\|\dot{x}\|}{\dot{x}_{MAX}}$, $|\frac{g-x}{g-x_0}|$, etc. Among these choices, one of them may satisfy all our requirements. However, even if we come up with an appropriate phase variable for an action, we may need to find another phase variable for a different action since the choice of the phase variable determines how an action should be generalized to different conditions given a demonstration. For instance $|\frac{g-x}{g-x_0}|$ treats the point on both sides of the goal in the same way since absolute value is used and $\frac{\|\vec{g}-\vec{x}\|}{\|\vec{g}-\vec{x_0}\|}$ treats the points with equal distance to the goal in the same way since norm of the vector distance is used.

Instead of defining a phase variable explicitly, one may suggest that the phase of the system should be learned with respect to the current state of the system (goal, position, velocity and etc.). However, this approach suffers from the lack of appropriate target values for the phase variable (i.e. what should be the phase variable for different actions). For instance consider you are reaching to the right of an object: If your hand is to the right of the object you will

[1] i.e. Height remains same when an object is lifted from the table and put to another place on the table again.

reach directly (the non-linear part and thus the phase variable will be approximately zero), but if you are reaching from the left side of the object you will trace a curved trajectory (the non-linear part and thus the phase variable will decay from some value to zero). Thus, this approach is not realistic since there is not any explicit definition of what should be learned by the system.

Considering these facts we believe that the non-linear part has to be directly learned from the current state of the system (goal, position, velocity and etc.) instead of defining a new phase variable or learning a mapping from the current state to the phase variable.

We modified the DMPs so that our system runs in closed-loop and generalizes to different conditions by learning from multiple demonstrations. The non-linear part and the phase variable are modified while leaving the canonical part as it is since it satisfies our requirements. Instead of constraining the system with a one dimensional phase variable, we replaced it with a state vector $\vec{z}$. In this new system, called *the modified DMPs*, equation 2.10 is replaced with:

$$\ddot{\vec{x}} = \mathbf{K}(\vec{g} - \vec{x}) - \mathbf{D}\dot{\vec{x}} + \mathbf{K}\vec{f}(\vec{z}, \mathbf{W}). \tag{3.1}$$

To allow online recognition without knowing t_0 or x_0, we restricted $\vec{z}$ to include only the variables which can be calculated from the current state. We chose the goal oriented position and velocity of all DOFs ($(g - x)$ and $\dot{x}$ for each DOF) as variables of $\vec{z}$ since these are the variables used by the canonical part of the DMPs. We decided to include the position and velocity of all DOFs in $\vec{z}$ so that all DOFs share the same state vector:

$$\vec{z} = [(g_1 - x_1) \quad \dots \quad (g_N - x_N) \quad \dot{x}_1 \quad \dots \quad \dot{x}_N]^T \tag{3.2}$$

where x_1 through x_N are the controlled variables. Note that other variables that do not violate the restriction on $\vec{z}$ can be used as well.

$\vec{f}(\cdot)$ has to be redefined since we replaced the one dimensional phase variable with a multi-dimensional state vector. Instead of extending the linear basis function model in one dimension to multi-dimensions, we used a neural network for $\vec{f}(\cdot)$ calculation since neural networks are universal function approximators [20]. We used a single network for each action where the input and the output of the network are $\vec{z}$ and $\vec{f}$ respectively as seen in figure 3.4. In this new definition $\mathbf{W}$ represents the weights of the neural network.

Neural network of an action is trained from a demonstration $\vec{x}(t)$ as follows:

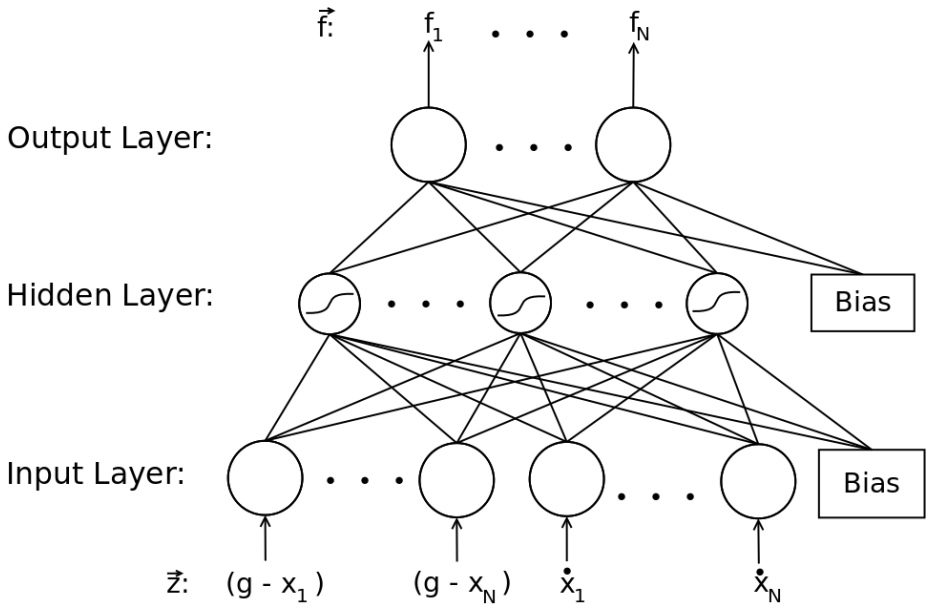


Figure 3.4: Feed forward neural network used as the non-linear part of the modified DMPs.

1. $\vec{f}(t)$ is obtained by calculating f for each dimension from equation 2.16.
2. $\vec{z}(t)$ is calculated from equation 3.2.
3. The network is trained using $\vec{z}$ and the corresponding $\vec{f}$ values as inputs and target values respectively.

For learning from multiple demonstrations, all $\vec{f}$ and $\vec{z}$ values calculated from different demonstrations are used at once for training the network. Note that neural networks support incremental learning and different demonstrations can be used to incrementally train the network instead of training it at once.

The overall architecture of the modified DMPs are shown in figure 3.5 and we can justify the advantages of the modified DMPs as follows:

- The non-linear part now works in a closed-loop fashion since $\vec{z}$ contains the positions and velocities of the controlled DOFs and it does not contain any variable that can not be calculated solely from the current state.
- System is not highly affected by perturbations such as stopping the end-effector since the non-linear part works in a closed-loop fashion. Furthermore, starting an action from

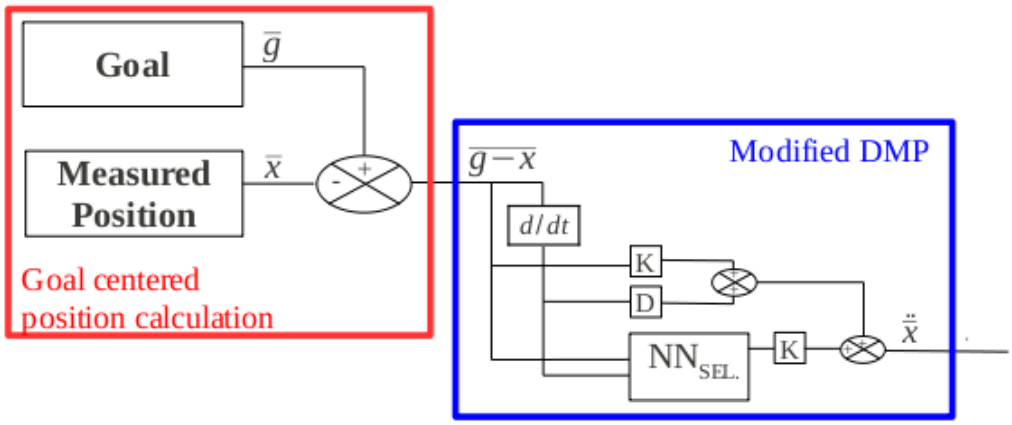


Figure 3.5: Overall architecture of the modified DMPs. $NN_{SEL.}$ is the neural network of the selected action.

an exact same sensorimotor state chosen from the demonstrated trajectory, will generate exactly the same trajectory since $\vec{z}$ of the starting point of the generated trajectory is equal to the chosen point of the demonstrated trajectory.

- An online recognition system can be built easily on top of the modified DMPs since $\vec{z}$ can be calculated from the current state only. A simple realization of such a system can be as follows: Starting from the first observed point of an action, observer generates a different trajectory in his mind by considering each action and object pair. Then he compares the observed trajectory with the ones generated as the action unfolds. The action-object pair that gives the smallest error can be recognized if the error is below a threshold.

- The modified DMPs learns from multiple demonstrations since each demonstration spans a different part of the space defined by the state vector $\vec{z}$.

The modified DMPs do not theoretically guarantee the convergence of an action since randomly chosen neural network weights may result in divergent actions[2]. This is a drawback of our approach but we are expecting the system to converge to the goal point when it is trained by sufficiently large number of demonstrations.

[2] The non-linear part may suppress the canonical part which may take the system to an irrelevant point instead of the goal point.

CHAPTER 4

EXPERIMENTAL PLATFORM

This chapter explains the devices that are used in the experiments: The humanoid robot which we implemented and tested our system on and the motion capture device that is used for recording the actions.

4.1 The iCub Robot Platform

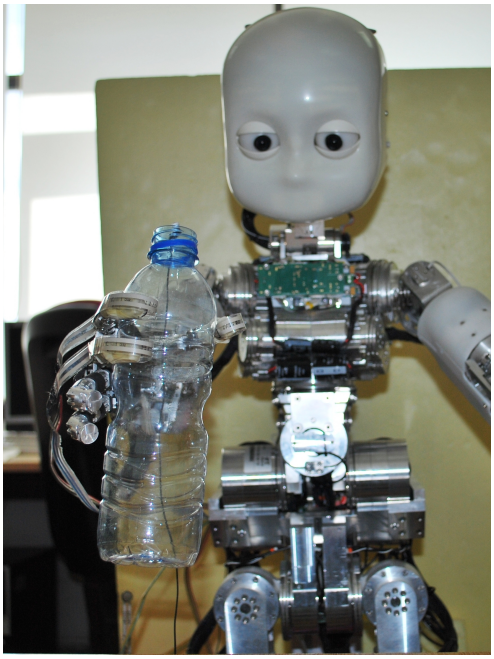
The iCub robot platform [21] (Figure 4.1) is developed as part of the EU project RobotCub [22] in order to study cognition through a humanoid robot of a child size.

The iCub robot has 53 degrees of freedom (DOF) in total with 3 for the head, 3 for both of the eyes, 7 for each arm, 9 for each hand, 3 for the torso and 6 for each leg. Its hands are dexterous and objects can be manipulated with it. We fixed our iCub robot from its waist as seen in figure 4.1(b) because we are not working on locomotion.

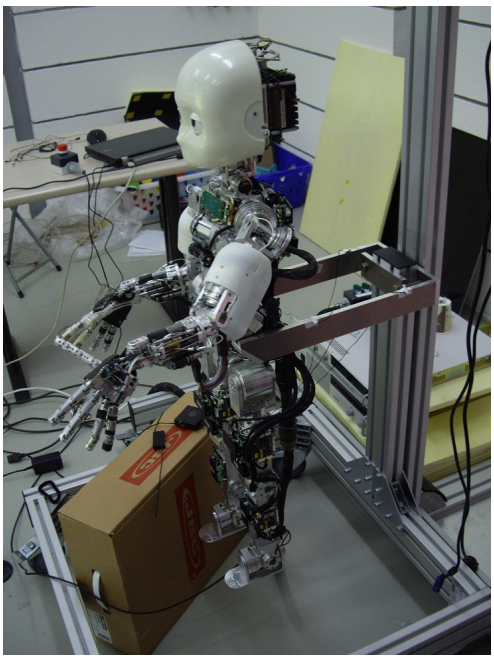
The iCub robot has one motor per DOF. Its head, eyes and hands are driven by brushed DC motors and the rest of the body parts which require stronger motors are driven by brushless DC motors. Angular positions of joints can be obtained through the encoders placed on the motors. PID control running at 1000 HZ is employed on the motors.

The iCub platform has one camera in each eye with a resolution of 640*480 which supports upto 60 frames per second. It has an inertial sensor in the head to measure orientation of the head with respect to the gravitational force. A compass in the head gives the direction with respect to the magnetic field of the earth. There is a microphone on each ear.

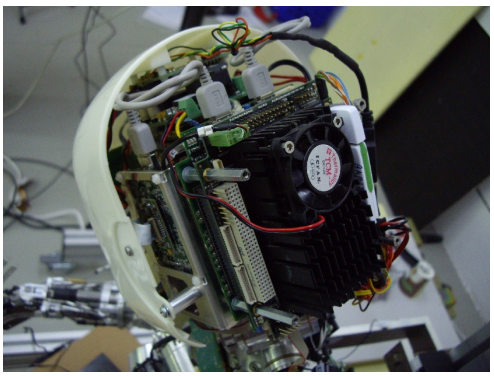
The iCub robot has an emotion interface which can be used for giving social cues. Eye-



(a) Front view of the iCub robot.



(b) Side view of the iCub robot. It is fixed from its waist.



(c) iCub's head from behind. PC104 in its head is also shown.

Figure 4.1: The iCub Humanoid Robot Platform

brows and mouth are simulated through LEDs underneath the skin and eyelids can be moved. Through this interface face expressions like sad, happy, and angry can be simulated and actions like eye blinking can be performed.

An Intel based dual core PC104 runs in iCub's head as the main processor. This processor uses Linux Debian operating system and it is used for communicating with motor controllers, sensors and the emotion interface. Main aim of this processor is to achieve communication between the robot and the other computers. Note that processing of information and calculation of new commands are done on other computers and then sent to PC104.

To achieve communication between the main processor and the other computers *Yet Another Robotics Platform* (YARP) [23] is used. YARP is an open software platform which implements network communication protocols. It aims to increase the use of reusable software in robotics. YARP is an easy to use software which allows a programmer to use the modules written by other programmers without struggling with details.

The hardware and the software of the iCub robot platform are distributed openly in order to increase collaboration between partners all around the world.

4.1.1 KDL: Kinematics and Dynamics Library

We are controlling the end-effector of the robot in the Cartesian space instead of the joint space for various reasons: (1) Our motion capture system records Cartesian positions. (2) It is easier for us to work and comment on Cartesian trajectories instead of joint trajectories. As a result, we need an inverse model for generating appropriate motor commands given the desired Cartesian state. Furthermore, we need a forward model for calculating the Cartesian state given a sensorimotor state.

Kinematics and Dynamics Library (KDL) developed by Orocos [24] is used as the forward and inverse model in our system. KDL calculates the forward and inverse mapping between the task space and the motor space given the mechanical structure of the system (joint definitions and limb lengths). The forward model is obtained by symbolically calculating the translation and the rotation from one joint to the next until the end-effector is reached. Then inverse model is obtained by taking the inverse of the forward model.

There is one constraint for KDL to function correctly: The controlled system must not contain a loop considering the joints and limbs (i.e. starting from a joint you should not be able to return to the same joint while tracing a limb only once) and there is not any loop in the mechanical structure of the iCub robot. KDL requires Denavit-Hartenberg parameters of a system for its calculations and these parameters for the iCub robot and an explanation of Denavit-Hartenberg parameters are given in Appendix B.

KDL does not check for singular points which causes some problems during action generation. An alternative to KDL is the forward/inverse kinematics library for iCub (iKin) [25]. However, iKin did not support inverse velocity calculations at the time that this thesis is done and in the future iKin can be used instead of KDL since it is a specific library writen for the iCub robot which also considers singularities.

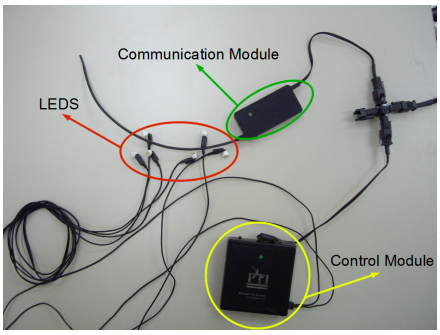
4.2 The Motion Capture System: VisualEyez

We used the motion capture system VisualEyez 4000 (VZ 4000) developed by Phoenix Technologies Incorporated [26] for tracking the end-effector of the actor and the object position. The system consists of a tracker, markers (LEDs), a control module and a communication module as shown in figure 4.2. Markers are attached to the points of interest and the tracker measures the Cartesian position of markers in space. Control module and the communication module of the markers are used for synchronization between the tracker and the markers.

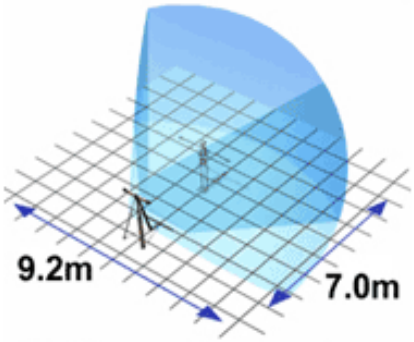
The system has to be calibrated first to obtain Cartesian positions in a well-defined Cartesian space. Three markers are sufficient for calibration: One at the origin, one at the X direction and one at X-Y plane. In our coordinate frame X-axis, Y-axis and Z-axis correspond to the left, up and outwards directions of iCub respectively. We chose the origin as the center of the waist of the iCub but the choice of origin does not affect the performance of the system since goal centered positions are used ($g - x$) instead of absolute positions (x).

VZ 4000 can track upto 512 markers and has a capture area of 7m * 9.2m as shown in figure 4.2(c). The positions of markers can be obtained with 0.5mm accuracy. About 4000 3D data points can be sampled per second which corresponds to 1000 FPS for 4 markers, 500 FPS for 8 markers and etc. These features of VZ 4000 are well above our requirements which will be explained in chapter 5.

(a) The Tracker



(b) Markers



(c) Capture Area. Figure taken from [26].

Figure 4.2: The Components and the Capture Area of the Motion Capture System

CHAPTER 5

EXPERIMENTS AND RESULTS

In this chapter, we analyze the behavior of the modified DMPs and compare them against the DMPs. We demonstrate that:

- The modified DMPs assure convergence when there are sufficient number of demonstrations.
- They can generalize to different conditions.
- They perform slightly better than DMPs.
- They are robust against perturbations such as stopping the end-effector.
- They can be used for online action recognition.

Furthermore, we implemented our system on a humanoid robot to demonstrate that it works on an embodied system in an online fashion.

5.1 Setup and Recorded Actions

Our setup consists of the humanoid robot iCub, a human demonstrator, an object hanging in the air and the motion capture system as shown in figure 5.1. We placed a marker on the right wrist of the human to track the position of the end-effector. We chose wrist instead of hand in order to minimize the occlusion problem of the motion capture system. Also we placed a marker on top of the object to track the object.

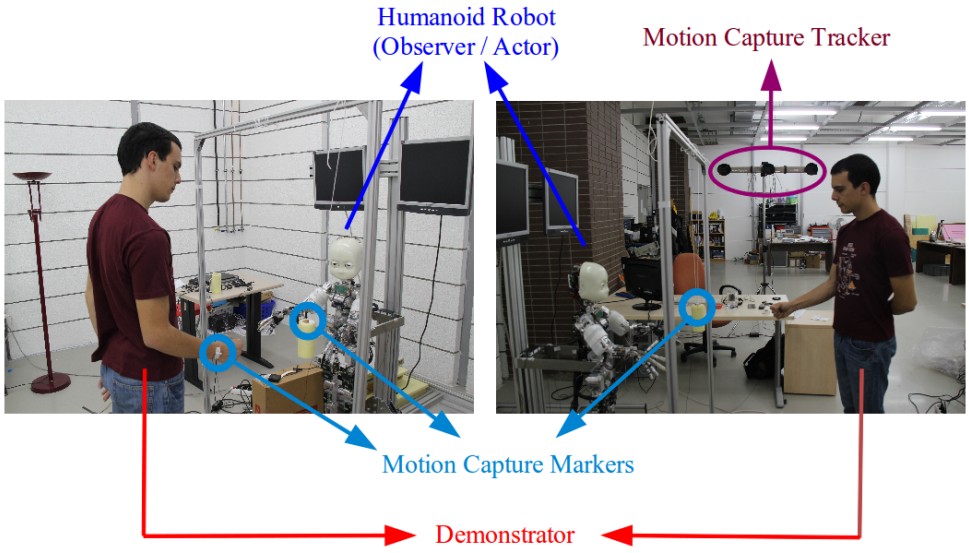


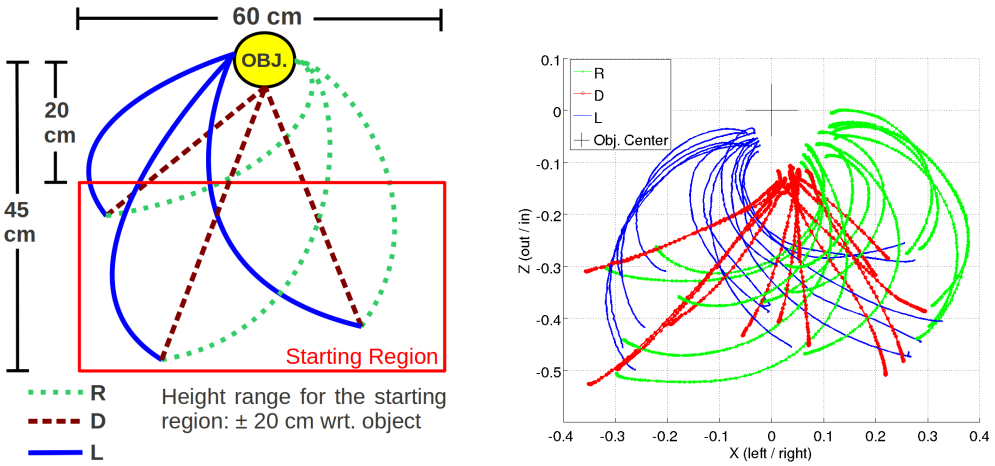
Figure 5.1: Setup used in the experiments

We defined three actions that can be applied on the object (Figure 5.2):

- R: Reaching the object from right
- D: Reaching the object directly
- L: Reaching the object from left

We recorded demonstrations of each action starting from a 3D region as shown in figure 5.2(a) in order to form the training set. This region is empirically maximized until the point where actions can not be performed easily and it is defined as: ± 30cm in the left/right direction with respect to object, ± 20cm in the up/down direction with respect to object and 20cm to 45cm away from the object. As starting points of the training data, 3 points are selected for each dimension (limit points and the medium point) which results in 27 points in total. An action is repeated 5 times from the same starting point. Thus, we recorded 135 trajectories for an action totaling in 405 recordings for all actions.

We recorded demonstrations of each action 50 times in order to form the test set. This time we selected the starting points randomly while remaining inside the 3D region mentioned above. A different demonstrator performed the actions while obtaining the test data. All actions (including the ones in the training set) are recorded with a frame rate of 100 frames per second.



(a) Illustration of defined actions

(b) A subset of the recorded trajectories

Figure 5.2: Defined and recorded actions on the setup

The motion capture system only records positions. Thus, velocities and accelerations are obtained by simple differentiation. We truncated recordings from both sides by applying an absolute velocity threshold (defined empirically as 0.05m/s) in order to discard the motionless and noisy part at the beginning and at the end of each recording.

Human motion is noisy and differentiation increases the effect of noise. This effect becomes severe during $f(\cdot)$ calculation (Equation 2.16) and thus $f(\cdot)$ becomes non-smooth and highly noisy. We applied an ideal low-pass filter at 10Hz on the calculated velocities to overcome this problem. Then, we calculated filtered positions and accelerations from these filtered velocities by simple integration and differentiation. This filtered data is used while calculating $f(\cdot)$ which is treated as target value during network training. Conversely, neural networks benefit from noisy input and we used non-filtered positions and velocities to obtain the state vector $\vec{z}$ which is used as input during network training.

Note that different demonstrations of the same action may end in different positions with respect to the object as seen in figure 5.2. As a result, instead of taking the object center as goal point in our experiments, we calculated the offset for the goal point with respect to object for each action. These offsets are calculated by treating the final point of each trajectory as the goal point and averaging these values among all recordings for an action. These offsets are added to the object center to obtain the goal point during action generation.

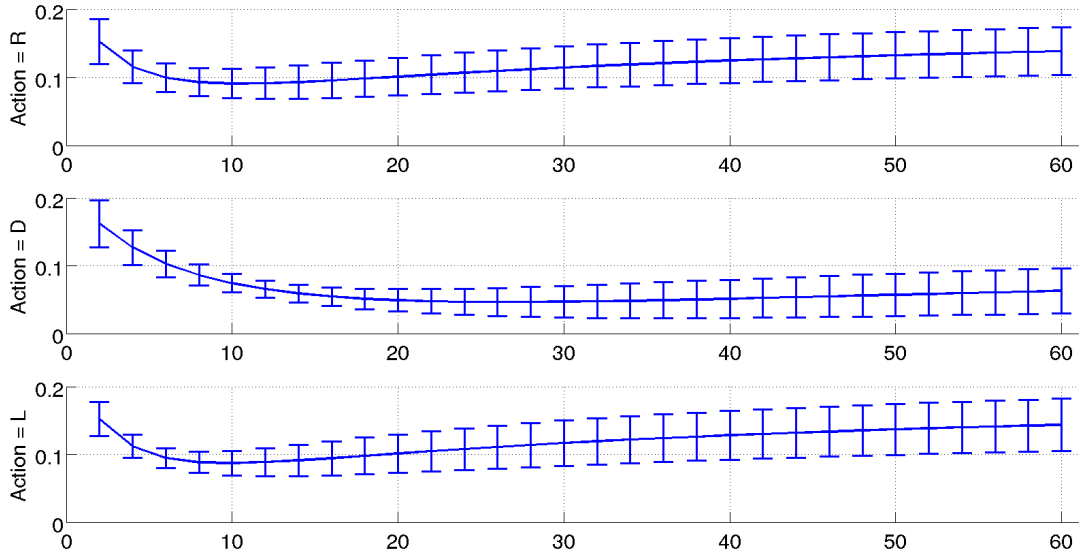


Figure 5.3: Results of the optimization procedure for K values. X axis is the tested K values and Y axis is the average work required from the non-linear part for one sample of a trajectory. Optimized K values are 10, 26 and 10 for R, D and L respectively.

5.2 Estimation of K and D

Selection of **K** and thus **D**, directly affects the performance of DMPs since it changes the amount of work that should be carried out by the non-linear part. With too low **K** and **D** values the canonical part will generate a slow motion and the non-linear part will learn most of the action from scratch which will decrease robustness. On the other hand, with too high values the canonical part will generate a very fast motion and the non-linear part will learn how to slow down the system instead of learning the complex trajectory. As a result, we need an algorithm to estimate or optimize **K** and **D** to increase system performance.

There is not any definition for analytically estimating **K** and **D** for the DMPs. Authors assigned values for these matrices in in [9, 10] without giving any reasons. We decided to select these values by minimizing the work that has to be carried out by the non-linear part ($\|\mathbf{K}\vec{f}(\cdot)\|$) during learning. We assumed that K and D values between different DOFs are constant in an action and thus we selected **K** and **D** as multiples of identity matrix. **K** and **D** are optimized using the training set and the details of this optimization procedure are given in algorithm 1. Results of the optimization algorithm are shown in figure 5.3 and optimized K values are 10, 26 and 10 for R, D and L respectively. These results show that humans perform direct reaching faster than reaching from left or right.

Algorithm 1 Optimization of **K** and **D**

$MinWork \leftarrow \infty$

$K_{OPT} \leftarrow 0$

$K \leftarrow 2$

while $K \leq 60$ **do**

  $D \leftarrow 2\sqrt{K}$

  $\mathbf{K} \leftarrow K * \mathbf{I}$

  $\mathbf{D} \leftarrow D * \mathbf{I}$

  $TotalWork \leftarrow 0$

  for All trajectories in training set **do**

    Calculate $\vec{f}(\cdot)$ using equation 2.16

    $WorkNL \leftarrow$ Average of $\|\mathbf{K}\ \vec{f}(\cdot)\|$ for all sample points

    $TotalWork \leftarrow TotalWork + WorkNL$

  end for

  if $TotalWork < MinWork$ **then**

    $MinWork \leftarrow TotalWork$

    $K_{OPT} \leftarrow K$

  end if

  $K \leftarrow K + 2$

end while

return K_{OPT}

5.3 Neural Network Training and Selection

We used fully connected feed-forward neural networks with bias as the network model of the modified DMPs. We used only one hidden layer which is a common approach in neural network literature. We trained the neural networks using Levenberg-Marquart (LM) back-propagation algorithm [27]. Although one iteration of LM algorithm is computationally more complex than traditional gradient-descent algorithms, LM converges faster than them by significantly decreasing the number of iterations.

We trained networks with different hidden layer sizes in order to decide on the size of the hidden layer. Then, these networks are used to generate trajectories using the initial conditions from the training set. Finally, the network that minimizes the MSE between the actual and the generated trajectories is selected as the neural network of the demonstrated action. We repeated this procedure five times for each hidden layer size since training performance of neural networks depends on initial weights of the network. Details of this procedure are given in algorithm 2.

We used the Neural Network Toolbox™ of Matlab® for implementing the neural network and the LM algorithm. Also we implemented a simple neural network library for C++ which is used for robot demonstrations[1]. This implementation only includes the simulation of a neural network and it does not support any training. Our C++ library uses the network weights obtained through Matlab®.

5.4 Convergence Test

We found the minimum number of demonstrations that are required for convergence since the convergence of the modified DMPs is not theoretically guaranteed. We trained the system as explained in section 5.3 using training sets of varying size. Then, we generated trajectories for each action using 1000 different starting points selected randomly from the aforementioned starting region (initial velocities are taken as zero). We recorded the number of diverged actions for each training set size where a generated action is considered as diverged if the distance between the end-effector and the goal point is higher than 5cm after an execution of

[1] Communication between a computer and the iCub platform is held through YARP which supports C++.

Algorithm 2 Neural Network Selection

$Nets \leftarrow$ {NULL, NULL, NULL}
for $Action$ = R, D or L **do**
  $MinError \leftarrow \infty$
  for $HidSize$ = 5 to 15 **do**
    for $trainNo$ = 1 to 5 **do**
      $Net \leftarrow$ New neural network with hidden layer size = $HidSize$
      $Net \leftarrow$ Train Net using LM
      $Error \leftarrow 0$
      for All trajectories of the current action in training set **do**
        $\vec{X}_{TRA}(t) \leftarrow$ Next trajectory in the training set
        $\vec{X}_{GEN}(t) \leftarrow$ Generate a trajectory using $\vec{X}_{TRA}(0)$ as the initital condition
        $Error \leftarrow Error+$ Mean value of $\|\vec{X}_{TRA}(t) - \vec{X}_{GEN}(t)\|$
      end for
      if $Error < MinError$ **then**
        $MinError \leftarrow Error$
        $Nets[Action] \leftarrow Net$
      end if
    end for
  end for
end for
return $Nets$

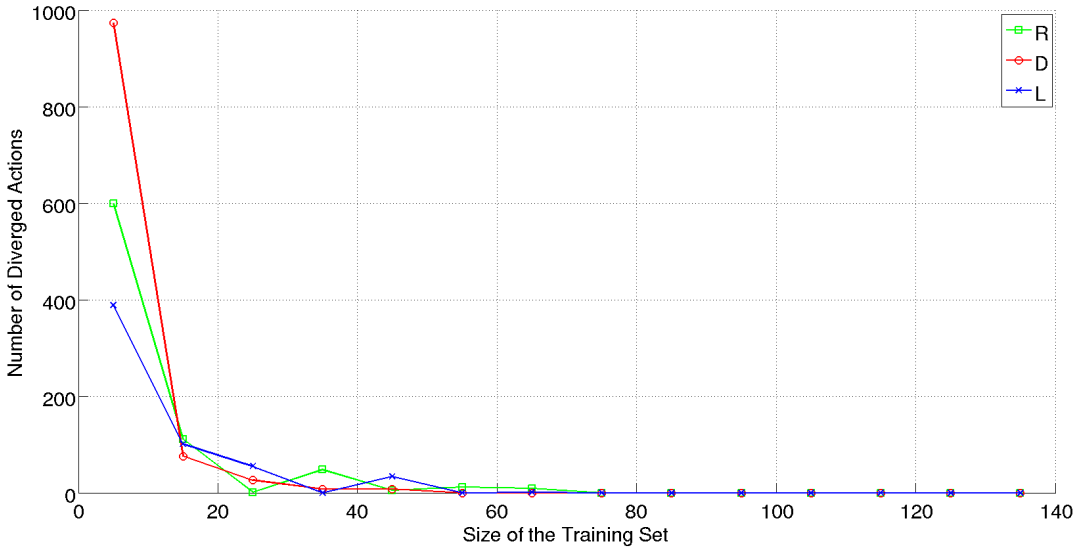


Figure 5.4: Number of diverged actions vs. training set size. For each action 1000 random starting points are used to test convergence where a generated action is considered as diverged if end-effector's distance to the goal is higher than 5cm after an execution of 1.5 seconds.

1.5 seconds. 5cm is selected as an acceptable error limit (as an ad-hoc approach output of the network can be set to 0 within that limit and the system will converge to the goal point) and 1.5 seconds is selected as a time limit since human demonstrations in the training set last between 1-1.4 seconds.

The number of diverged actions is zero for training sets with size greater than equal to 75 as shown in figure 5.4. Our training set consists of 135 demonstrations for each action which is much greater than the critical threshold of 75. Thus, we empirically shown that our system learns to converge to the goal when number of demonstrations is sufficiently high.

5.5 Generalization Performance

In this section we compare the generalization performance of DMPs and modified DMPs by conducting two experiments. In the first experiment, generalization performance is tested by taking random starting points and in the second experiment test set is used to make a comparison between human demonstrations.

DMPs learn an action from a single demonstration while the modified DMPs use multiple demonstrations for learning. To compensate for this difference, we used two different ap-

proaches while training the DMPs:

1. **Optimization:** First, a DMP is trained for each trajectory in training set. Then it is used to generate trajectories for all of the trajectories in the training set and MSE between actual and generated trajectories are calculated. The DMP that minimizes the overall error is selected as the learned one.

2. **Averaging:** A DMP is trained for each trajectory in training set and average of all DMP weights are used as the final DMP weights.

We trained modified DMPs and DMPs with training sets of different sizes in order to assess the generalization performance of them. After training, different trajectories are generated by selecting random starting points from the starting region (initial velocities are taken as zero). We used three different sizes of training data for this experiment and defined them as 15, 75 and 135. 15 is selected as a small set, 75 is selected as the point where modified DMPs assure convergence and 135 corresponds to using the whole training set. Results of this experiment for modified DMPs, DMPs obtained by optimization and DMPs obtained by averaging are shown in figures 5.5, 5.6 and 5.7 respectively. We leave the judgement about generalization performance to the reader since there is not any available data for comparison in this experiment.

In the second experiment, we used the whole training set for training the modified DMPs and the DMPs. After training has been completed, we generated trajectories by using first points of the trajectories in the test set as initial conditions. Then, we calculated the MSE between actual and generated trajectories. As seen from figure 5.8 and table 5.1, modified DMPs perform slightly better than DMPs. Although there is not any significant improvement in terms of MSE, the modified DMPs run completely in closed-loop whereas the non-linear part of the DMPs runs in open-loop.

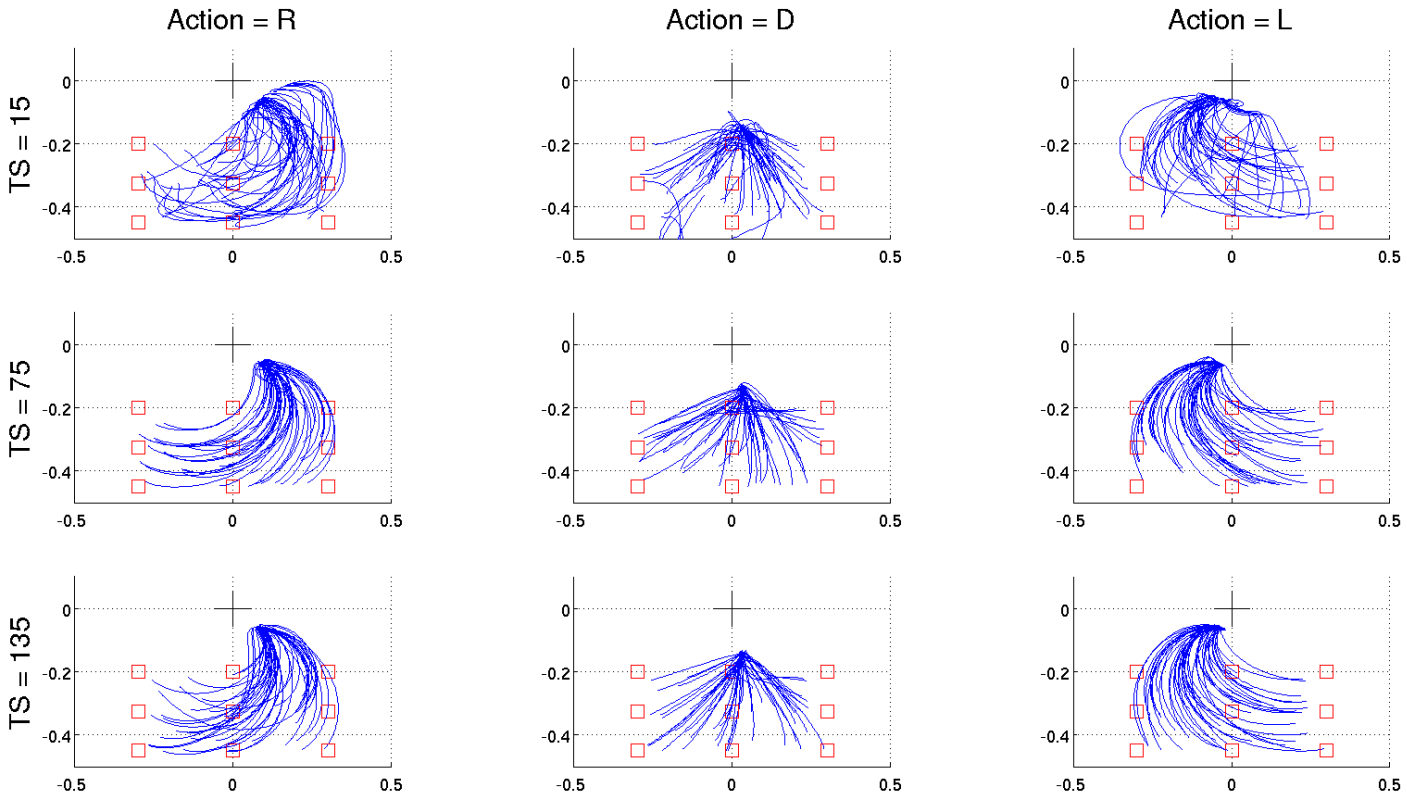


Figure 5.5: Top view of the generated trajectories using modified DMPs. TS is the size of the training set, black plus sign shows the object center and red squares shows the starting points for the demonstrations in the training set.

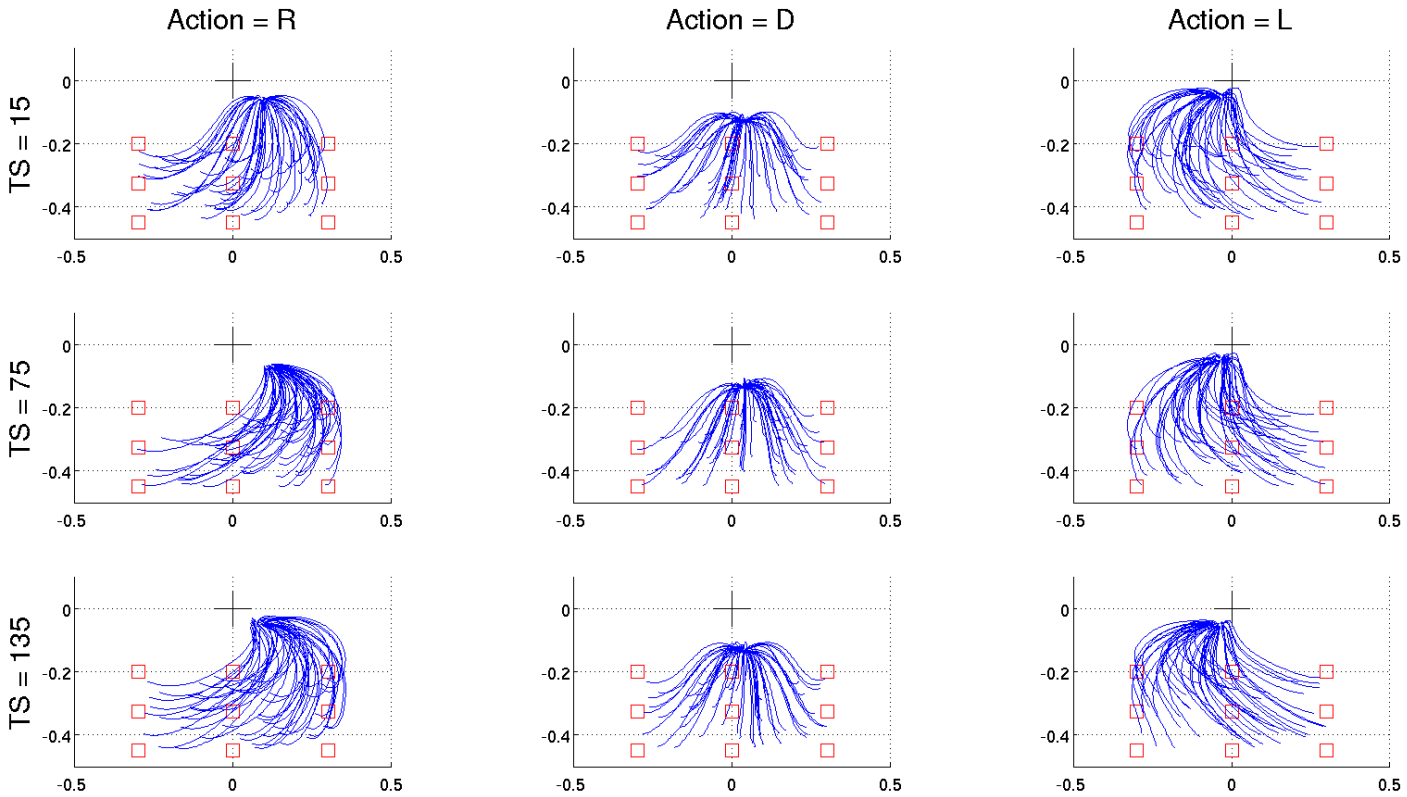


Figure 5.6: Top view of the generated trajectories using DMPs obtained by optimization. TS is the size of the training set, black plus sign shows the object center and red squares shows the starting points for the demonstrations in the training set.

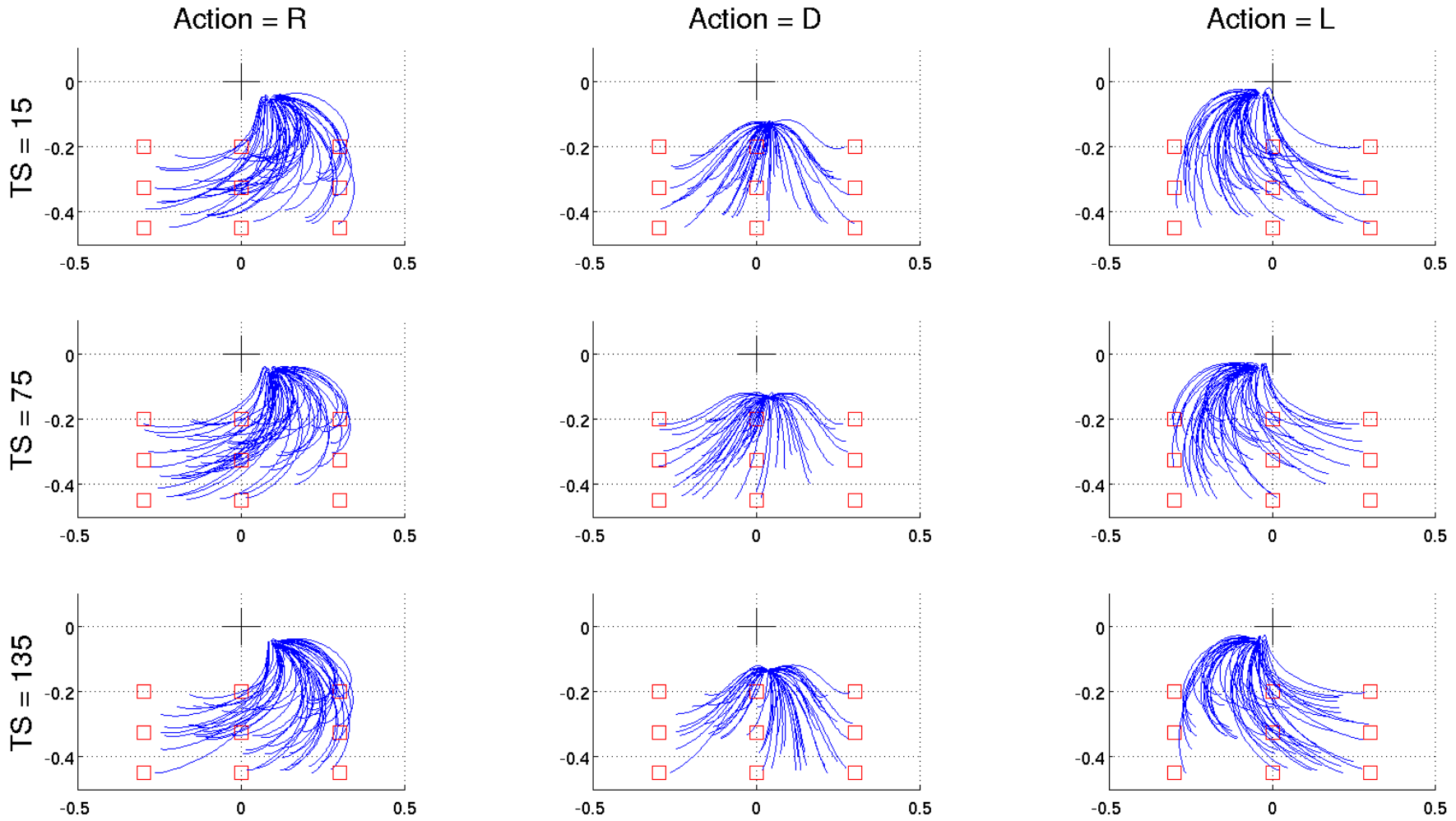


Figure 5.7: Top view of the generated trajectories using DMPs obtained by averaging. TS is the size of the training set, black plus sign shows the object center and red squares shows the starting points for the demonstrations in the training set.

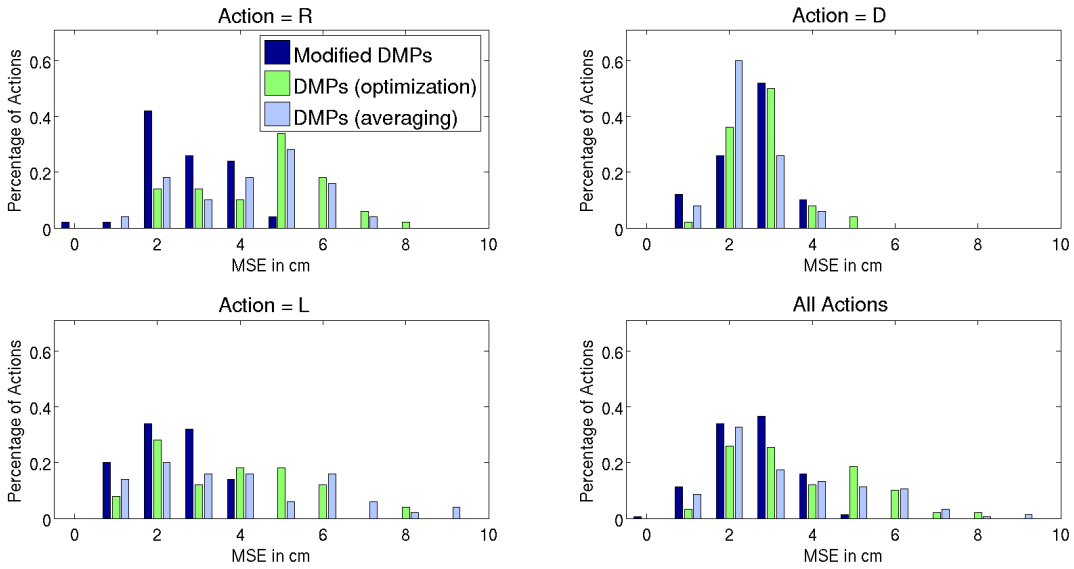


Figure 5.8: Comparison of the DMPs and the modified DMPs using the test set. Histogram of the MSEs are shown which are divided into bins of size 1 cm.

Table 5.1: Comparison of the DMPs and the modified DMPs using the test set. Average of MSEs and their standard deviations are given (in cm).

	Modified DMPs	**DMPs (Optimization)**	**DMPs(Averaging)**
Action = R	**3.33 ± 0.99**	5.14 ± 1.70	4.76 ± 1.7
Action = D	3.07 ± 0.78	3.27 ± 0.71	**2.83 ± 1.7**
Action = L	**2.92±0.90**	4.17 ± 1.76	4.39 ± 2.18
All Actions	**3.09±0.91**	4.19 ± 1.65	3.99 ± 1.85

5.6 Robustness Test

Recall that, in section 3.2 we have demonstrated the drawbacks of DMPs by making two experiments: In the first experiment (Figure 3.2), we have stopped the end-effector for different durations and demonstrated the problems that occur because of the open-loop nature of the non-linear part of the DMPs. We have made the same experiment on the modified DMPs by using the whole training set for training. Since the modified DMPs does not explicitly depend on time, the duration of stopping the end-effector does not affect the trajectory. Thus, we did not test different durations but tested different stopping points. As seen from figure 5.9, the modified DMPs can cope with a perturbation such as stopping the end effector.

In the second experiment (Figure 3.3), we have selected points on a demonstrated trajectory and then generated new trajectories by taking that point as the starting point. For the starting velocity we have tested two options: Using the velocity on the selected point in the demonstrated trajectory or using zero velocity. As seen from figure 5.10, the modified DMPs generate exactly the same trajectory when original velocity on the point is used and this is an expected result since the state vector is same for the same point and the same velocity. In addition, selecting zero velocity corresponds to stopping the end effector on that point which is exactly the same experiment with the previous one.

5.7 Online Action Recognition

Implementing an online action recognition is not in the scope of thesis and in section 3.3 we justified how the modified DMPs can be used for online action recognition. In addition, in [11, 28] we implemented an online recognition system on top of the modified DMPs. In these works, we achieved a 90% success rate using two objects and the three actions that are defined in section 5.1. Furthermore, the system recognized an action before 30% of the action is completed on average.

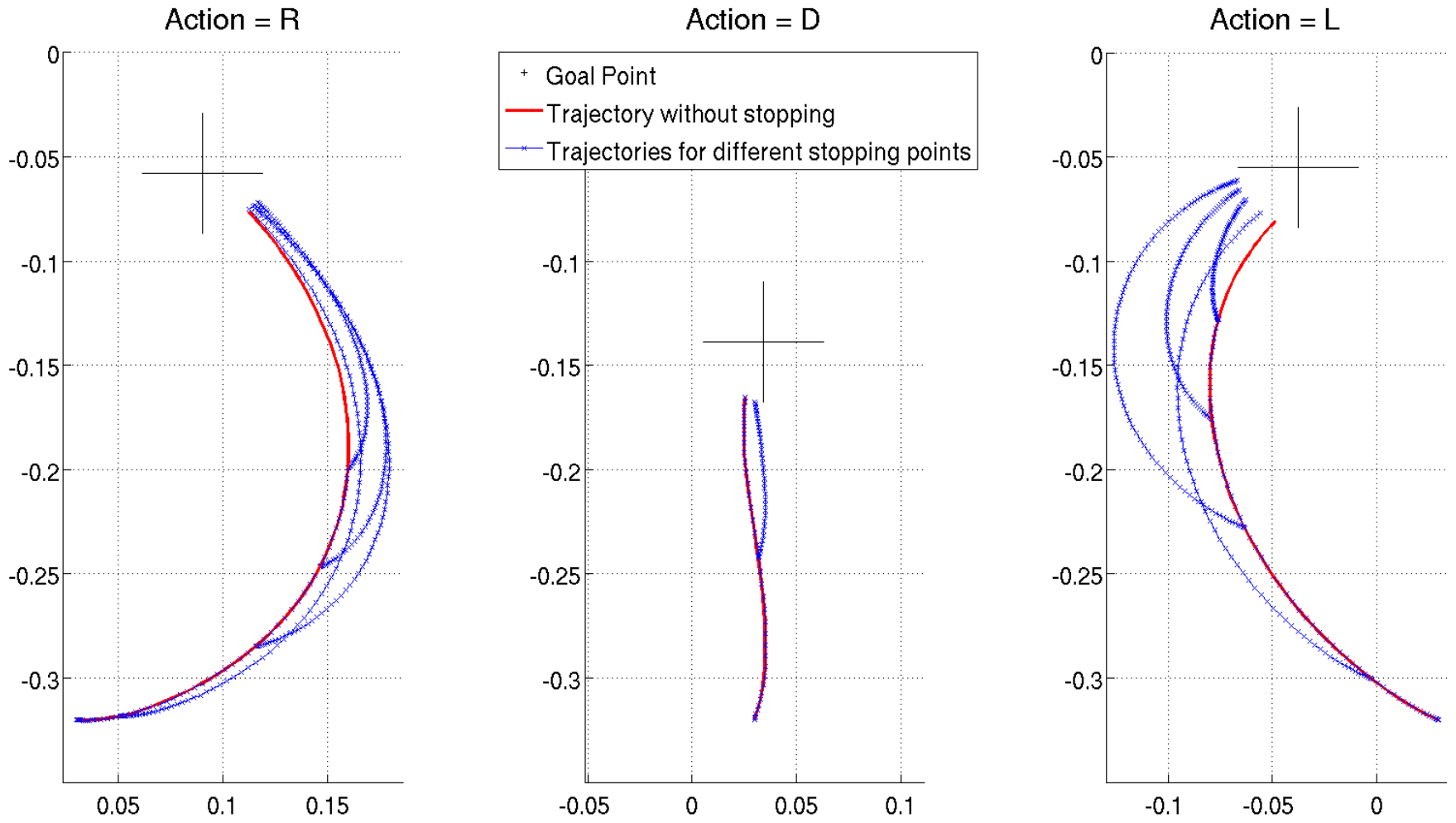


Figure 5.9: Effects of stopping the end-effector for the modified DMPs. Top view of the trajectories are shown.

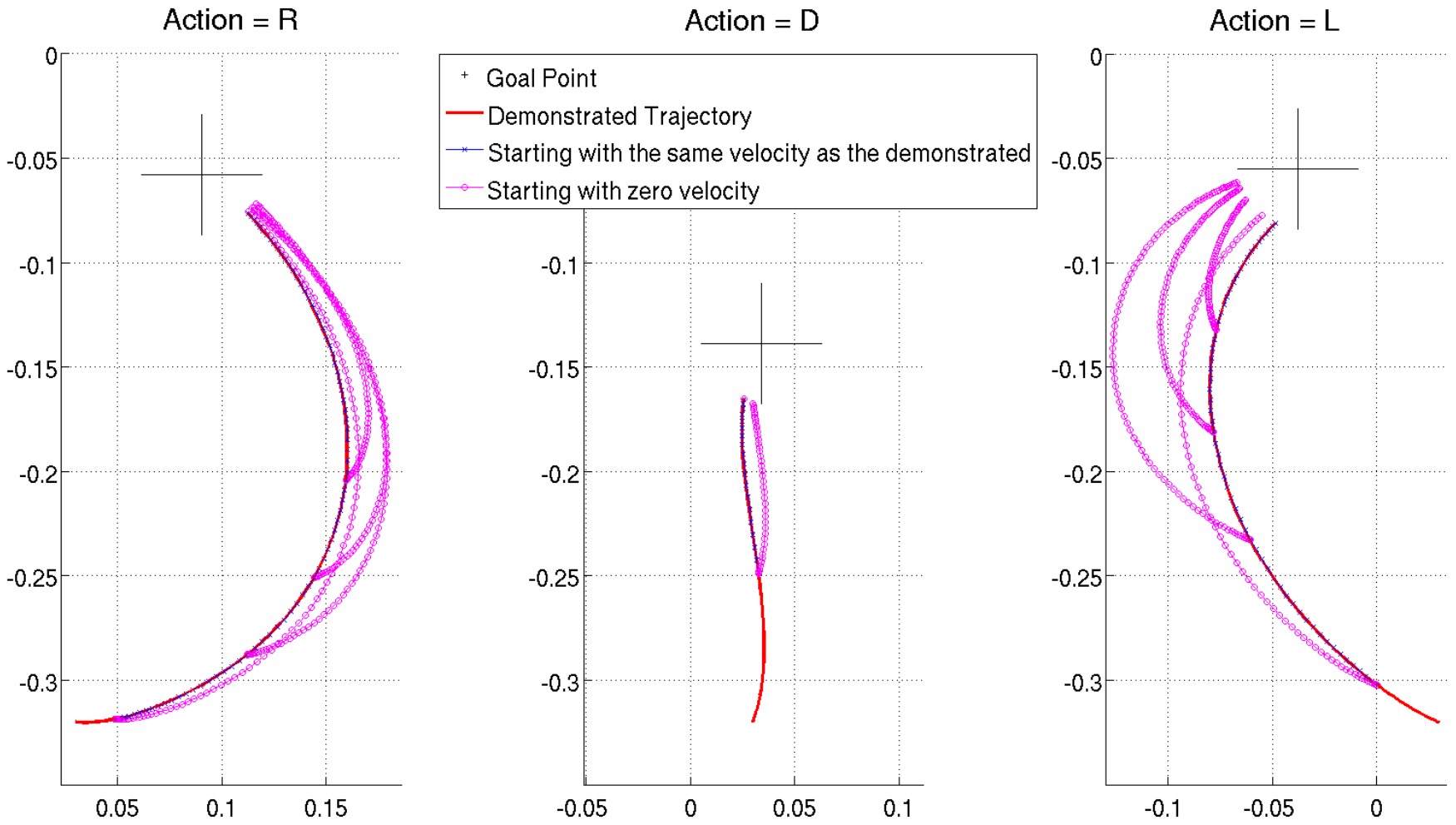


Figure 5.10: Different trajectories generated using the modified DMPs and taking different points on the demonstrated trajectory as starting points. Top view of the trajectories are shown.

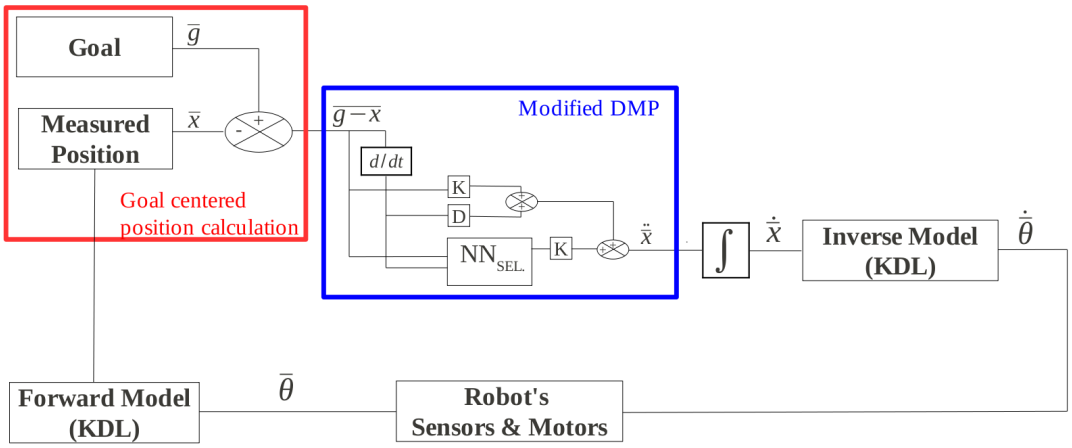


Figure 5.11: Action generation architecture implemented on the iCub platform. NN_{SEL} is the neural network of the selected action and $\dot{\bar{\theta}}$ (joint velocities) is sent to the actuators as motor commands.

5.8 Robot Demonstration

We implemented the overall system on the iCub humanoid robot platform in order to show that our system can be used on a real robot. The modified DMPs generate Cartesian accelerations as control law but the iCub platform does not support joint accelerations as motor commands. To remedy that, Cartesian velocities are obtained from simple integration and then they are converted to joint velocities using KDL as shown in figure 5.11.

To solve the correspondence problem, we have scaled the Cartesian positions (thus velocities are scaled as well) by considering the ratio of the arm length of the demonstrator to the arm length of the iCub robot which is equal to 1.5. With this approach, the system scales for different robot sizes and it can be implemented on a microscopic or a giant robot without any further operations. Note that we used closed loop control running with a period of 10ms while generating an action.

As can be seen from figure 5.12, our system succesfully generates different actions while applying online closed-loop control.

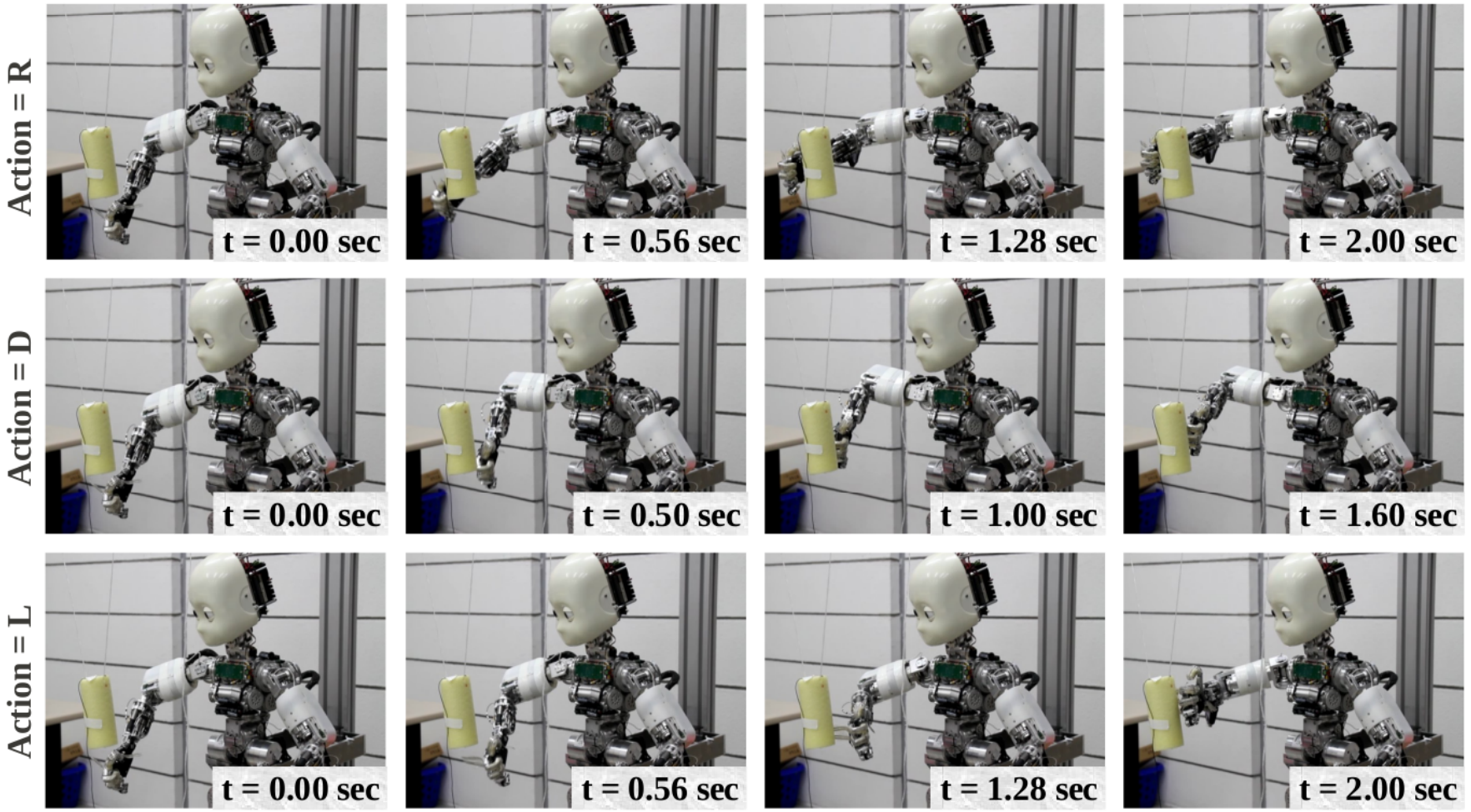


Figure 5.12: Generated actions with the iCub robot using the modified DMPs. Each row shows a different action. The first column and the last column show the beginning and the end of an action respectively and the intermediate columns show the steps in between.

CHAPTER 6

CONCLUSION

In this thesis, we presented an action generation system which can learn from multiple demonstrations, which can generalize to different conditions, which generates actions in an online and closed-loop fashion and which can be used for online action recognition. Also, we showed that our system is suitable for real systems by implementing it on a real robot.

The generation architecture is built on top of the DMPs where we modified them to overcome some of their shortcomings. To do so, first we presented an analysis of DMPs and then we reformed parts of the DMPs that were identified as the source of the drawbacks. Differences between the modified DMPs and the DMPs (as well as other surveyed approaches) are shown in table 6.1. The modified DMPs do not guarantee convergence theoretically and we accept it as a major drawback.

The main difference between the modified DMPs and the DMPs is the choice of the phase variable. We have replaced the one dimensional phase variable that is running in open-loop with a multi-dimensional state vector that is running in closed-loop. Moreover, this state vector is not dependent on intrinsic parameters of an actor and it can be calculated solely from the current state. Thus, the modified DMPs run completely in closed-loop fashion, can learn from multiple demonstrations and can be used for online action recognition whereas the DMPs can not.

Experimental results show that the modified DMPs learn to converge to the goal for sufficiently large training sets. Furthermore, they can generalize to different conditions and they perform slightly better than the DMPs. Although there is not a huge improvement in terms of system performance, the modified DMPs outperform the DMPs by learning from multiple demonstrations, being completely closed-loop and supporting online action recognition.

The modified DMPs support mirror neuron hypothesis since the same architecture can be used for action learning, generation and recognition. However, further work has to be done in order to support this hypothesis strongly.

6.1 Future Work

Our system should be revised and changed accordingly in order to prove convergence theoretically. Using spherical coordinates instead of Cartesian coordinates and forcing the derivative of the radius to be negative can be a solution for this problem.

The canonical part of DMPs and the modified DMPs converges to the goal exponentially with respect to time. However, we observed a sigmoidal convergence in Cartesian positions on human recordings (not shown in a figure). Modifying the canonical part to obtain a sigmoidal behavior may decrease the work of the non-linear part which may lead to better generalization performance.

We used a motion capture system to track the object and the hand of the human demonstrator. This restricts us to remain in the workspace of the motion capture system and this approach is not suitable for mobile robots. Instead, a vision based system should be used so that the robot solely functions using the sensors and actuators on its body.

We tested our system with simple reaching behaviors in this thesis. However, interaction with the real world requires more complex tasks such as grasping, pushing and lifting an object. Finger angles and orientation of the hand should be controlled along with the position of the hand to perform such tasks. Means of inserting these new variables into the system should be analyzed and tested in order to perform more complex tasks.

In our opinion DMPs are suitable for generating trajectories when the goal is known (such as reaching to some point of the object) and thus they are suitable for controlling the hand trajectory and orientation while grasping. However, they are not suitable to implement control laws when the goal is not known beforehand (i.e. stopping motion according to the tactile or visual sensors). As a result, to realize complex actions such as grasping through learning rules like “close the fingers until they touch the object” either a different architecture or means of guessing and/or dynamically changing goal should be studied.

There is not much research in the literature that study the relationship between object attributes and action characteristics (i.e. how to perform pushing considering the object geometry and weight) although there are many works that study the effect of a given/learned action with respect to object properties [29, 30]. Combining these approaches and addressing the question of how an action should be performed on an object to achieve a desired effect can be an interesting research topic.

Table 6.1: Comparison of modified DMPs and the surveyed approaches. Properties of modified DMPs that are superior over DMPs are shown in **bold**.

	Learning by Demonstration	**Generalization**	**Fast Generation and Robustness**	**Correspondence**	**Action Recognition**
The Mimesis Model	- Joint angles - One-shot learning	- No explicit goal setting - Poor generalization	- Offline generation in one second - Fast/slow generation and thus robustness depends on the context	No correspondence problem (using joint angles)	Online recognition is possible (not implemented)
RNNPB	- Joint angles - One-shot learning	- No explicit goal setting - Poor generalization	- Fast and robust (online and closed-loop)	No correspondence problem (using joint angles)	Online recognition is possible (not implemented)
MOSAIC	- Joint angles - One-shot learning	- No explicit goal setting - Poor generalization	- Fast and robust (online and closed-loop)	No correspondence problem (using joint angles)	Online recognition is possible (not implemented)
DMPs	- Joint angles or Cartesian positions - One-shot learning	- Explicit goal setting - Good generalization	- Fast but "half" robust (online but "half" closed-loop)	An inverse model have to be used if Cartesian positions are used	Offline recognition (implemented and tested)
The Modified DMPs	- Joint angles or Cartesian positions - **Learns from multiple demonstrations**	- Explicit goal setting - Good generalization	- Fast and **robust** (online and **closed-loop**)	An inverse model have to be used if Cartesian positions are used	**Online recognition** (implemented and tested)

REFERENCES

[1] T. Inamura, I. Toshima, H. Tanie, and Y. Nakamura, "Embodied Symbol Emergence Based on Mimesis Theory," *The International Journal of Robotics Research*, vol. 23, no. 4-5, pp. 363–377, 2004.

[2] E. Oztop, M. Kawato, and M. Arbib, "Mirror neurons and imitation: A computationally guided review," *Neural Networks*, vol. 19, pp. 254–271, 2006.

[3] *iCub's Mechanic Structure*, http://eris.liralab.it/wiki/ICubForwardKinematics. Last visited: August 2010.

[4] *Denavit-Hartenberg Parameters*, http://en.wikipedia.org/wiki/Denavit-Hartenberg-_Parameters. Last visited: August 2010.

[5] G. Rizzolatti, L. Fadiga, V. Gallese, and L. Fogassi, "Premotor cortex and the recognition of motor actions," *Cognitive brain research*, 1996.

[6] V. Gallese, L. Fadiga, L. Fogassi, and G. Rizzolatti, "Action recognition in the premotor cortex," *Brain*, 1996.

[7] G. Buccino, F. Binkofski, G. Fink, and L. Fadiga, "Action observation activates premotor and parietal areas in a somatotopic manner: an fMRI study," *European Journal of Neuroscience*, 2001.

[8] G. Buccino, F. Binkofski, and L. Riggio, "The mirror neuron system and action recognition," *Brain and Language*, 2004.

[9] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Learning attractor landscapes for learning motor primitives," in *Advances in Neural Information Processing Systems* (S. Becker, S. Thrun, and K. Obermayer, eds.), vol. 15, pp. 1547–1554, MIT-Press, 2003.

[10] H. Hoffman, P. Pastor, D.-H. Park, and S. Schaal, "Biologically-inspired dynamical systems for movement generation: Automatic real-time goal adaptation and obstacle avoid-

ance," in *2009 IEEE International Conference on Robotics and Automation*, pp. 2587–2592, May. 2009.

[11] B. Akgün, "Action recognition through action generation," Master's thesis, Middle East Technical University, 2010.

[12] *EU FP7 Project, ROSSI*, www.rossiproject.eu. Last visited: August 2010.

[13] V. Kruger, D. Kragic, A. Ude, and C. Geib, "The meaning of action: A review on action recognition and mapping," *Advanced Robotics*, vol. 21, no. 13, pp. 1473–1501, 2007.

[14] B. Janus and Y. Nakamura, "Unsupervised probabilistic segmentation of motion data for mimesis modeling," in *Proceedings of 12th International Conference on Advanced Robotics, 2005 (ICAR '05)*, pp. 411–417, July 2005.

[15] L. E. Baum, T. Petrie, G. Soules, and N. Weiss, "A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains," *The Annals of Mathematical Statistics*, vol. 41, no. 1, pp. 164–171, 1970.

[16] J. Tani, "Learning to generate articulated behavior through the bottom-up and the top-down interaction processes," *Neural Networks*, vol. 16, pp. 11–23, 2003.

[17] J. Tani and M. Ito, "Self-organization of behavioral primitives as multiple attractor dynamics: A robot experiment," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 33, pp. 481–488, 2003.

[18] D. Wolpert and M. Kawato, "Multiple paired forward and inverse models for motor control," *Neural Networks*, vol. 11, pp. 1317–1329, 1998.

[19] M. Haruno, D. M. Wolpert, and M. M. Kawato, "Mosaic model for sensorimotor learning and control," *Neural Computation*, vol. 13, no. 10, pp. 2201–2220, 2001.

[20] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, p. 251, 1991.

[21] G. Sandini, G. Metta, and D. Vernon, "The icub cognitive humanoid robot: An open-system research platform for enactive cognition," in *50 Years of Artificial Intelligence*, pp. 358–369, Springer Berlin / Heidelberg, 2007.

[22] *EU FP7 Project RobotCub*, http://www.robotcub.org. Last visited: August 2010.

[23] *Yet Another Robotics Platform*, http://eris.liralab.it/yarpdoc/what_is_yarp.html. Last visited: August 2010.

[24] *Kinematics and Dynamics Library*, http://www.orocos.org/kdl. Last visited: August 2010.

[25] *iCub Manual*, http://eris.liralab.it/wiki/Manual. Last visited: August 2010.

[26] *VisualEyez 4000: Motion Capture System*, http://www.ptiphoenix.com/VZmodels.php. Last visited: August 2010.

[27] M. Hagan and M. Menhaj, "Training feedforward networks with the Marquardt algorithm," *IEEE transactions on Neural Networks*, vol. 5, no. 6, pp. 989–993, 1994.

[28] B. Akgün, D. Tunaoğlu, and E. Şahin, "Action recognition through an action generation mechanism." Manuscript submitted for publication.

[29] E. Uğur, E. Şahin, and E. Öztop, "Affordance learning from range data for multi-step planning," in *Proceedings of the Ninth International Conference on Epigenetic Robotics (Epirob'09)*, vol. 146, pp. 177 – 184, Nov. 2009.

[30] P. Fitzpatrick, G. Metta, L. Natale, S. Rao, and G. Sandini, "Learning about objects through action - initial steps towards artificial cognition," in *Proceedings of IEEE International Conference on Robotics and Automation, 2003 (ICRA '03)*, vol. 3, pp. 3140 – 3145, Sep. 2003.

[31] C. M. Bishop, *Pattern Recognition and Machine Learning*, ch. 3, pp. 137 – 177. Springer Science+Business Media, LLC, 2006.

[32] M. W. Spong and M. Vidyasagar, *Robot Dynamics and Control*. New York : John Wiley and Sons, 1989.

APPENDIX A

LEAST SQUARES REGRESSION

Least squares regression (LSR) is a technique to estimate the coefficients of linear summation of basis functions by minimizing the mean squared error (MSE) between the target values and the weighted sum of basis functions. In this appendix, how to apply LSR is explained without giving any proof. Details and proofs about this technique can be found in [31].

The variables that are necessary to apply LSR are: Input values $\vec{x}_1$ to $\vec{x}_N$, target values o_1 to o_N and basis functions $\Psi_1(\vec{x})$ to $\Psi_M(\vec{x})$ where N is the number of samples and M is the number of basis functions. The LSR problem is defined as finding the weight vector $\vec{w}$ that minimize the MSE:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(o_i - f_i)^2, \tag{A.1}$$

$$f_i = \sum_{j=1}^{M} w_j\, \psi_j(\vec{x}_i), \tag{A.2}$$

where f_i is the output of the system for the i^{th} input sample and w_j is the j^{th} coefficient of $\vec{w}$. To find the coefficients of the basis functions, N-by-M design matrix have to be calculated first:

$$\Phi = \begin{bmatrix} \psi_1(\vec{x}_1) & \psi_2(\vec{x}_1) & \dots & \psi_M(\vec{x}_1) \\ \psi_1(\vec{x}_2) & \psi_2(\vec{x}_2) & \dots & \psi_M(\vec{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \psi_1(\vec{x}_N) & \psi_2(\vec{x}_N) & \dots & \psi_M(\vec{x}_N) \end{bmatrix}. \tag{A.3}$$

Then $\vec{w}$ that minimize the MSE can be calculated from:

$$\vec{w} = (\Phi^T\Phi)^{-1}\Phi^T\vec{t}, \tag{A.4}$$

where target vector $\vec{t}$ is composed of target values:

$$\vec{t} = [o_1 \;\; o_2 \;\; \dots \;\; o_N]^T. \tag{A.5}$$

APPENDIX B

MECHANICAL STRUCTURE OF the iCub Robot's BODY

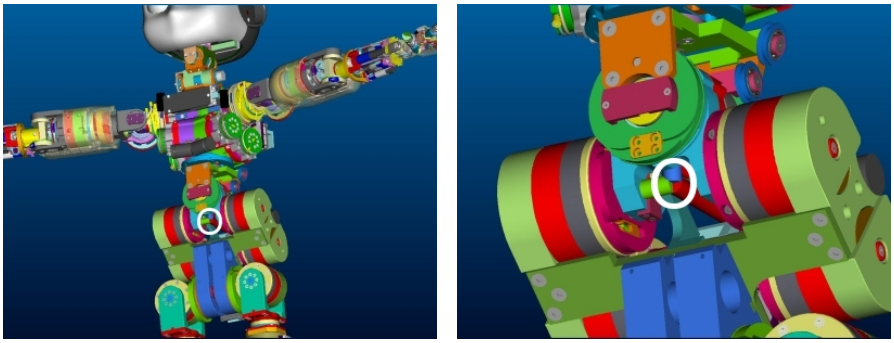
The iCub robot's mechanical body structure is defined in [3] using Denavit-Hartenberg parameters (D-H parameters) which is a widely used convention for describing the relationship between joints that are connected with rigid links [32].

First task in choosing D-H parameters is selecting a root reference frame which is an arbitrary choice. The root reference frame for the iCub robot is selected as:

- **Origin** is a point on the axis of the rotation for torso pitch and it is defined as the middle point between the two legs as shown in figure B.1.
- **X axis** points to the left of the robot.
- **Y axis** is parallel to gravity but in the upwards direction.
- **Z axis** points outwards with respect to the robot and it is selected according to the right hand rule.

After a root reference frame is defined, reference frame of the n^{th} joint is defined using the reference frame of the previous joint as follows:

- $\mathbf{Z}_n$ **axis** is defined as the axis of rotation for a revolute joint and the axis of translation for a prismatic joint.
- $\mathbf{X}_n$ **axis** is parallel to the common normal: $X_n = Z_{n-1} \times Z_n$. If Z axes are parallel then there is no common normal and this case is explained after defining D-H parameters.
- $\mathbf{Y}_n$ **axis** is selected according to the right hand rule after X_n and Z_n axes are defined.



(a) (b)

Figure B.1: Origin of the root reference frame for the iCub robot is shown with a white circle. Figure taken from [3].

- **Origin**$_n$ is defined as the intersection point of the rotation/translation axis of the joint and X_n axis passing from Origin$_{n-1}$.

Note that origin of the reference frame of a joint does not necessarily have to coincide with the origin of the joint.

After all reference frames are defined, D-H parameters define a relationship between each reference frame and the previous one with respect to it by using four parameters:

- d is the distance of current origin to the previous one in the direction of previous Z axis. d defines the translation of current joint with respect to previous one in the direction where the previous joint cannot move the current one (Z axis of the previous one).
- r is the perpendicular distance of current origin to the previous Z axis. If the previous joint is a revolute joint then r is the radius of rotation. In some definitions a is used instead of r which may be confused with α.
- θ is the rotation of the X axis from previous frame to the current one along the previous Z axis.
- α is the rotation of the Z axis from previous frame to the current one along the previous X axis.

An example of D-H parameters between two consecutive joints are shown in figure B.2 and an explanatory video can be found in [4]. Finally, D-H parameters for the iCub robot's right wrist and left wrist are given in tables B.1 and B.2 respectively.

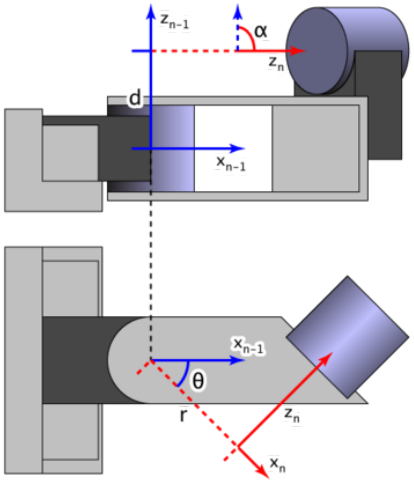


Figure B.2: Example of D-H parameters between two joints. Figure taken from [4].

Table B.1: D-H Parameters for the iCub robot's right wrist

Link No	Joint Name	d (in mm)	r (in mm)	θ (deg)	α (deg)
0	Torso Yaw	0	32	84	90
1	Torso Roll	0	0	39	90
2	Torso Pitch	-143.3	-23.264	59	90
3	Shoulder Pitch	-107.74	0	-95	90
4	Shoulder Roll	0	0	160.8	-90
5	Shoulder Yaw	-152.2	0	100	-90
6	Elbow	0	15	106	90
7	Origin of the Wrist	-137.3	0	X	X

Table B.2: D-H Parameters for the iCub robot's left wrist

Link No	Joint Name	d (in mm)	r (in mm)	θ (deg)	α (deg)
0	Torso Yaw	0	32	84	90
1	Torso Roll	0	0	39	90
2	Torso Pitch	-143.3	23.264	59	-90
3	Shoulder Pitch	107.74	0	-95	-90
4	Shoulder Roll	0	0	160.8	90
5	Shoulder Yaw	152.2	0	100	-90
6	Elbow	0	-15	106	90
7	Origin of the Wrist	137.3	0	X	X