ACTION RECOGNITION THROUGH ACTION GENERATION

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

BY

BARIŞ AKGÜN

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

AUGUST 2010

Approval of the thesis:

ACTION RECOGNITION THROUGH ACTION GENERATION

submitted by **BARIŞ AKGÜN** 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. Dr. Erol Şahin
Computer Engineering Department, METU _____

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

Asst. Prof. Dr. Onur Tolga Şehitoğlu
Computer Engineering Department, METU _____

Asst. Prof. Dr. Didem Gökçay
Informatics Institute Department, METU _____

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: BARIŞ AKGÜN

Signature :

ABSTRACT

ACTION RECOGNITION THROUGH ACTION GENERATION

Akgün, Barış
M.S., Department of Computer Engineering
Supervisor : Asst. Prof. Dr. Erol Şahin

August 2010, 50 pages

This thesis investigates how a robot can use action generation mechanisms to recognize the action of an observed actor in an on-line manner i.e., before the completion of the action. Towards this end, Dynamic Movement Primitives (DMP), an action generation method proposed for imitation, are modified to recognize the actions of an actor. Specifically, a human actor performed three different reaching actions to two different objects. Three DMP's, each corresponding to a different reaching action, were trained using this data. The proposed method used an object-centered coordinate system to define the variables for the action, eliminating the difference between the actor and the robot. During testing, the robot simulated action trajectories by its learned DMPs and compared the resulting trajectories against the observed one. The error between the simulated and the observed trajectories were integrated into a recognition signal, over which recognition was done. The proposed method was applied on the iCub humanoid robot platform using an active motion capture device for sensing. The results showed that the system was able to recognize actions with high accuracy as they unfold in time. Moreover, the feasibility of the approach is demonstrated in an interactive game between the robot and a human.

Keywords: action recognition, humanoid robot, learning from demonstration

ÖZ

HAREKET YARATMA MEKANIZMALARIYLA HAREKET TANIMA

Akgün, Barış
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Yrd. Doç. Dr. Erol Şahin

Ağustos 2010, 50 sayfa

Bu tezde, bir robotun kendi hareket yaratma mekanizmalarını kullanarak, gözlemlediği bir hareketi eş zamanlı olarak (henüz hareket bitmeden), tanıması incelenmiştir. Bu bağlamda, taklit için geliştirilen bir hareket yaratma mekanizması olan Dinamik Hareket İlkelleri (DHİ) gözlemlenen hareketi tanımak için değiştirilmiştir. Bir insan, iki nesne üstünde üç farklı uzanma hareketi göstermiştir. Farklı hareketlere tekabül eden üç farklı DHİ, bu veriler kullanılarak eğitilmiştir. Önerilen metotta, hareket yaratma mekanizmalarının değişkenleri nesne tabanlı bir koordinat düzleminde tanımlanmıştır. Bu sayede hareketi yapan ile gözlemleyen arasındaki farklılıklar ortadan kaldırılmıştır. Metodun sınanması sırasında, robot öğrendiği DHİ'ler ile kafasında hareketler yaratarak, bunları gözlemlediği hareketle kıyaslamıştır. Yaratılan ve gözlemlenen hareket yörüngeleri arasındaki farktan tanıma sinyalleri hesaplanarak bunların üstünden tanıma yapılmıştır. Önerilen metot, bir hareket izleme sistemi kullanılarak, iCub isimli insansı robotun üstünde denenmiştir. Sonuçlar, sistemin eş zamanlı çalışarak yüksek başarıya ulaştığını ortaya koymuştur. Buna ek olarak, hareket tanıma metodunun uygulanabilirliği, robotla insan arasında oynanan bir oyunla gösterilmiştir.

Anahtar Kelimeler: hareket tanıma, insansı robot, gözlemden öğrenme

To beloved mom and dad

ACKNOWLEDGMENTS

I wish to express my gratitude to my advisor Erol Şahin for his invaluable support and guidance and also for making me a member of KOVAN family and providing an excellent research environment and opportunities.

I am thankful to Doruk Tunaoğlu for being the perfect research partner, a comrade in arms and a close friend. I wouldn't have been able to complete this thesis without his support.

I would like to thank all the KOVAN members who have been very supporting; Fatih Gökçe for being a close friend and supporter, Hande Çelikkanat for making us smile with her cheerful laughter and for encouraging me to start using Linux, İlkay Atıl who I couldn't think of anybody else to spend my break times, Nilgün Dağ for letting me have all the snacks at her drawer, Emre Uğur for helping me adjust KOVAN when I first came. I would also like to thank Güven İşcan and Kadir Fırat Uyanık for the discussions and the good times. I want to salute Mustafa Parlaktuna, the new "makina" guy. I can never repay his help during the after defence preparations of this thesis. I am also thankful to Sinan Kalkan, for his comments and support.

I can never thank my family enough for supporting and enduring me. Mom and dad, without your support, I wouldn't be able to make it. Dear sister, grandparents, cousins, aunts and uncles, thanks for bearing with me, I know I am not the best at keeping in touch.

Another special thanks goes to my dear Nazlı Dönmezer, who makes everything more meaningful and shines in my heart like a bright star. I also wouldn't forget my old friends but the list is too long to write and I wouldn't know where to start.

I would like to acknowledge the support of TÜBİTAK BİDEP 2228 graduate student fellowship and this study was funded by TÜBİTAK (The Scientific and Technological Research Council of Turkey) under the Project numbered 109E033 and by European Commission under the ROSSI project(FP7-216125).

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

Robotics is on the verge of explosion. With constant developments in processors, actuators, sensors and power sources, researchers and engineers are developing more and more complicated robots and algorithms. Robots are slowly getting out of laboratories and factory floors and making their way towards our daily lives. Many humanoids and service bots have been developed during the past decade. However, there are still challenges ahead that need to be addressed before robots can be fully integrated.

Action recognition is one of these challenges. Briefly, action recognition refers to understanding what another agent, a human or another robot, is doing. This is important for interaction, cooperation and even communication which are essential if robots are to enter human environments.

Nature might have already provided an elegant way for action recognition through *mirror neurons*. These neurons, discovered in the area F5 (and some other areas) of the premotor cortex of macaque monkeys, fire both during the generation and the observation of goal-directed actions [3, 4]. Grasping is a prominent example of these actions. Grasping related mirror neurons of a macaque monkey are active both when the monkey grasps a piece of food and when a human or another monkey does the same thing. In this context, goal-oriented means that the action is directed towards an object, which is the food in the previous example. These neurons do not respond when the observed agent tries to grasp the air where there is no object.

There have been findings of a mirror neuron system in humans as well [5]. This system acts similar to mirror neurons found in macaque monkeys. However, it also responds to non-goal-oriented actions. This system is composed of different parts of the human brain. It is called a

system since investigation of individual neurons on human subjects is very restricted.

The dual response characteristics of mirror neurons is attributed to an action recognition hypothesis; action generation and recognition shares the very same neural circuitry and that the motor system is activated during action recognition. Refer to [6] for a review on experimental data supporting this hypothesis, both on primates and humans.

By taking inspiration from mirror neurons, this thesis proposes an online action recognition scheme using an action generation mechanism. Action generation mechanisms are used to *imagine* what the remaining part of the observed action could be given current observations and the action recognition includes comparing these to the observed action. Online action recognition refers to making a decision about the observed action as early as possible, before that action ends.

This thesis is conducted as a part of a larger architecture which entails action learning, generation and recognition, developed within the ROSSI Project [7] and the TÜBİTAK Project numbered 109E033. This architecture will be explained in sufficient detail for understanding the recognition scheme.

1.1 What is Action and Action Generation?

The word action has a very broad meaning in robotics literature which ranges from simple motions (e.g. pick up) to high-level activities (e.g. assemble). This thesis assumes the former definition which entails the motions of short-duration. Reaching a cup in the vicinity is a good example.

Actions usually have a goal which could be set according to the environment. The cup defines the goal of the reaching in the previous example. As a counter example, take an action defined as reaching forward. In such an action definition, there is no explicit way to set where exactly to reach. This thesis will deal with goal-directed actions.

Simple actions are usually associated with open-loop control where there is no sensory feedback. The word behavior is generally used to describe closed-loop control, i.e., that use sensory feedback. However, there is no clear distinction between the two. In this thesis, actions are assumed to be closed-loop and the terms action and behavior may be used interchangeably.

To summarize, this thesis assumes that an action is a short-duration closed-loop motion which allows goal setting.

This definition is not enough to implement actions on robots; a mathematical representation is needed. As an example, assume that there is a robotic arm with a gripper which needs to reach and grasp a cup and the arm also has a sensor that measures the distance between the cup and the gripper. An obvious strategy is to minimize the distance between the gripper and the cup for reaching and closing the gripper when this distance is below a certain value. A mathematical model is needed to map the current and the desired distance between the gripper and the cup to the actuator commands of the arm and specify the speed of reaching.

Formally, an action is represented by a mathematical model which defines input-output relationships between its variables with parameters affecting this relationship. For robots, sensory values, goal and time could be inputs and actuator commands could be outputs. By employing this definition on the previous example, current and desired (goal) distances correspond to the inputs, actuator commands correspond to the outputs and speed specification is done by a parameter.

Action generation refers to calculating the outputs of an action model given inputs and applying the resulting outputs to the robot. In online action generation, outputs are calculated while the action is generated. In offline action generation, the outputs are calculated before the action.

1.2 Preliminary Definitions

Some definitions are given here as preliminaries to ease upcoming descriptions. These definitions are not exhaustive and will use discrete time notation for simplicity.

An action takes a certain amount of time to be completed, during which its variables change. The history of this change i.e., the time series of a variable is called its *trajectory*.

The output variable of an action is sometimes called the controlled variable. There are different control variables that could be used. Some of these are joint variables (angles, velocities and accelerations), image coordinates and cartesian variables. The space of the controlled variable is called the *task space*.

All robots have actuators which control their joints. The space of joint positions is called the *joint space*. Note that the task space and the joint space can coincide.

All or a subset of variables of an action in a particular point in time define its *state*. Assuming an ideal world, if the complete state of an action is known at a point in time, then the rest the of the action can be calculated ad infinitum using its mathematical model. The state of the system is usually defined in its task space.

If an action model's output is the state then its called a *trajectory generator*. Take the action model of the previous example of the robot and the cup. If it generates a change in gripper-cup distance, then it is called a trajectory generator. Formally the form of a trajectory generator is;

$$\hat{x}(t+1) = \Omega(x(t), g, \alpha, \omega, t), \tag{1.1}$$

where Ω is the trajectory generating policy, $\hat{x}(t+1)$ is the calculated next state, $x(t)$ is the current state, g is the goal state, α depicts constant parameters, ω depicts adaptive parameters (e.g. parameters of a function approximator) and t is time. Bear in mind that x could also be a vector. The adaptive parameters could be used for optimization or for action learning, an issue addressed later on in this thesis. A separate model is needed to realize the calculated next state i.e., convert the state into actuator commands.

If the action model calculates the actuator commands necessary to reach a desired state directly, then it is called an *inverse model*. If the action model in the previous example is used to calculate the actuator commands to reach the cup e.g., joint speeds that would close the distance, then the action model is called an inverse model. Its general form is;

$$u(t) = \Pi(g, x(t), \alpha, \omega, t), \tag{1.2}$$

where Π is the inverse model and $u(t)$ is the generated motor command. The rest of the parameters are as they appear in equation 1.1. In robotics, the terms inverse model and controller are used interchangeably.

An inverse model is usually accompanied by a *forward model* that calculates the next state of a system given the current state and the motor command. Its general form is;

$$\hat{x}(t+1) = \Phi(x(t), u(t), \alpha, \omega, t), \tag{1.3}$$

where Φ is the forward model, $\hat{x}(t+1)$ is the calculated next state, $u(t)$ is the motor command. The rest of the parameters are as they appear in equation 1.1. In robotics, the terms forward

model and predictor are used interchangeably. They are used to predict the outcomes (e.g., state) of actions (e.g., motor commands).

1.3 What is Action Recognition?

Consider an agent, called the actor, performing an action being observed by another agent, called the observer, who tries to understand the actor's action. This problem is called *action recognition*.

As mentioned before, the meaning of action and action recognition varies. In vision studies, action recognition is typically considered as a pattern recognition problem with an extra time dimension [8]. Some authors from the robotics field, approach recognition as understanding the effect of an action, such as in [9]. These approaches are considered as *passive action recognition*.

The approach presented in this thesis utilizes action generation mechanisms for recognition. Both the actor and the observer have action generation mechanisms available to them. The actor uses these to generate actions and the observer uses these to simulate actions. Generation and simulation are similar apart from the fact that no commands are sent to actuators during the latter. For recognition, the observer compares its simulated action trajectories to the observed trajectory and makes a decision accordingly. This approach can be considered as *active action recognition*.

The action recognition approaches also vary according to their timing. If recognition is done before the action is completed, it is called *online* action recognition, if it is done after the action is completed, it is called *offline* action recognition.

There are many challenges towards action recognition. The choice of matching space is one of them. Action generation could be in joint space but observations could be in cartesian coordinates. In this case, observations should be mapped onto the space of generation. This is not a trivial task, due to the differences in the actor's and observer's body structure. Moreover, differences between the viewpoints of the actor and the observer, called the correspondence problem requires the agent to map the observed space to its internal representation [10].

If the space of matching problem is solved, then one faces with the problem of finding a

suitable metric to evaluate the similarity of an observed action to a simulated action.

In the light of the previous discussion, some general requirements for action recognition using action generation mechanisms can be stated as:

- The actor and the observer should have similar action generation models.
- Observations should be mapped onto a space that is appropriate for action generation models.
- A similarity metric should be defined.
- An online methodology should be present (Not a must for action recognition but is an aim of this thesis).
- Recognition should be generalized to different goals (Not a must for action recognition but is an of this thesis).

1.4 What is Learning by Demonstration?

In order for the recognition approach developed in this thesis to work, the actor and the observer need to have similar action generation mechanisms. If the actor is human, then the observer robot should have the ability to generate actions similar to the ones that humans can.

Learning action models from demonstrations is a viable option to solve this problem. These demonstrations are usually provided by the actor to ensure similar generation mechanisms. Learning by demonstration is also called programming by demonstration.

Mathematically, learning by demonstration can be formulated as estimating the adaptive parameters (ω) of equations 1.1 and 1.2 from demonstrations ($g, x(t)$).

1.5 Organization

The organization of the thesis is as follows: The literature survey is presented in the next chapter. In Chapter 3, the action recognition approach and the used metrics are detailed. Chapter 4 explains the equipment and the experimental setup used for collecting data and

testing the approach. The experiments, their results and analysis are given in Chapter 5. Chapter 6 discusses the results and addresses future research directions.

CHAPTER 2

LITERATURE SURVEY

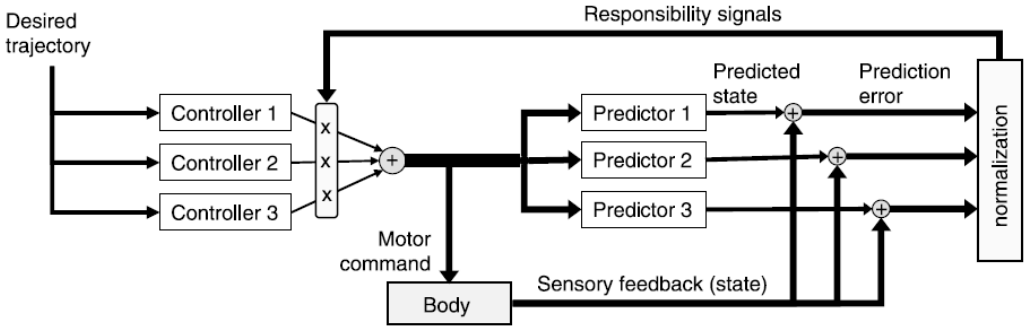
There have been various modelling efforts and studies related to integrated action generation-recognition approaches after the discovery of mirror neurons. Several conceptual and computational models have been proposed.

Conceptual models usually attribute many functions to mirror neurons such as action recognition and imitation ability, see e.g. [11, 12]. There are also theories that relate mirror neurons to language [13], citing the similarity between mirror neurons in the macaque monkey and the Broca's region, the language understanding part of the human brain. The problem with conceptual models is that they do not state how to implement attributed functions mathematically and are usually unrealistic from a computational point of view, thus such conceptual models are out of scope of this thesis.

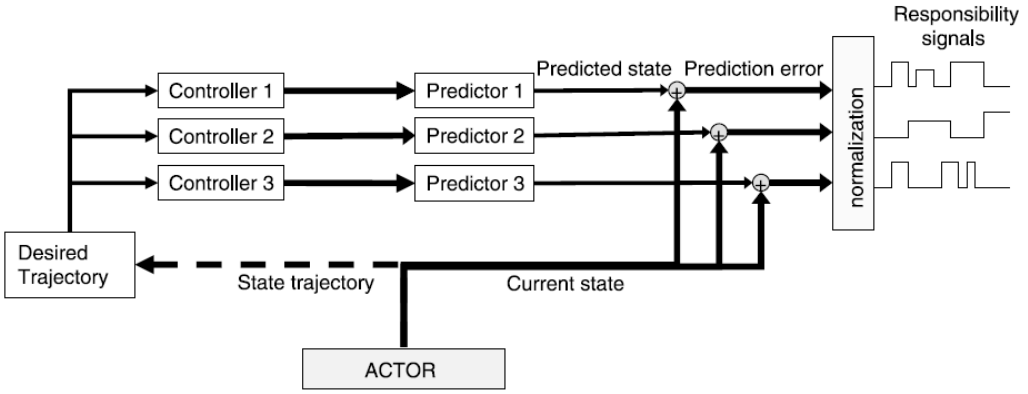
Some computational models are developed to explain and/or mimic the mirror neuron system and some other models to be used directly in robotics (action generation etc.). From the robotics side, these models mostly deal with action generation, action learning and imitation, mentioning recognition as a side study. A review regarding the computational models of imitation learning is given in [14]. In addition, [1] presents a review about imitation methods and their relations to mirror neurons. This literature survey will present computational models that could be used as integrated action generation-recognition mechanisms.

2.1 MOSAIC

The Modular Selection and Identification for Control (MOSAIC) architecture [15, 16] is a decentralized and modular control architecture for adaptive control. The architecture consists of



(a) Action Generation with MOSAIC. For simplicity, feedback to the controllers are not shown.



(b) Action Recognition with MOSAIC

Figure 2.1: Different uses of the MOSAIC architecture. (Figures taken from [1] and reproduced with kind permission of the corresponding author Erhan Öztop)

multiple controller-predictor pairs which compete and cooperate to achieve a desired control task. The pairs contribute to overall control according to their prediction performance.

Action generation using MOSAIC is depicted in figure 2.1(a). The desired trajectory of the controlled variables is given to the system. Then the controllers compute motor commands which are multiplied by corresponding *responsibility signals* and summed together. Motor commands are sent both to the robot and to the predictors. The difference between the predicted and the actual motion is used to calculate responsibility signals.

The responsibility signal of a pair indicates how well the corresponding controller is suited for the given control task. First the likelihood of a controller to generate the current state is calculated as

$$l_i(t) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-||x(t)-\hat{x}^i(t)||/2\sigma^2}, \tag{2.1}$$

where t is time, l is the likelihood, x is the measured state (sensory feedback) and $\hat{x}^i$ is the i^{th} predicted state and σ is the standard deviation of the forward model dynamics. Then the

responsibility signal is defined as

$$\lambda_i(t) = \frac{l_i(t)}{\sum_{j=1}^{n} l_j(t)}, \tag{2.2}$$

where λ_i is the responsibility signal of the i^{th} pair and n is the total number of pairs.

The MOSAIC architecture can be used for action recognition, which is shown in figure 2.1(b). The actor's trajectory is fed to the system as the desired trajectory, after mapping it to a state appropriate for MOSAIC. Motor commands are generated and provided as input to the predictors. Then these predictions are compared with the next observations to calculate responsibility signals. The computed responsibility signals represent the observed action. The responsibility signals of known actions are compared with the observed action's signals for recognition. Responsibility signals can be treated as parameters.

The pairs of the MOSAIC architecture can be acquired by learning. The predictors learn the next state of the system given the current state and the motor command, and the controllers learn to generate necessary motor commands to reach a desired state given current state. For controller learning, a target motor command is needed, which is approximated by a feedback controller. Responsibility signals are also used during learning as multipliers of the learning rate. This way, a pair which has a better performance of generating the desired trajectory, is trained more.

The MOSAIC architecture can also be used for imitation learning. It is important to note here that learning of the pairs is not imitation learning, unless the desired trajectory given to the system is the observed trajectory which is mapped to a suitable space to be used by the architecture. However, there is no study that has implemented this yet. Another approach is taken for imitation learning in [17]. The responsibility signals calculated during observations are stored and then given to the system during action generation to achieve imitation.

2.2 HAMMER

Hierarchical Attentive Multiple Models for Execution and Recognition (HAMMER) is a family of architectures, first used for imitation in [2, 18] and then extended for recognition in [19, 20]. HAMMER architecture consist of multiple forward and inverse model pairs, as in MOSAIC. However these models do not interact and only a single pair contributes to overall

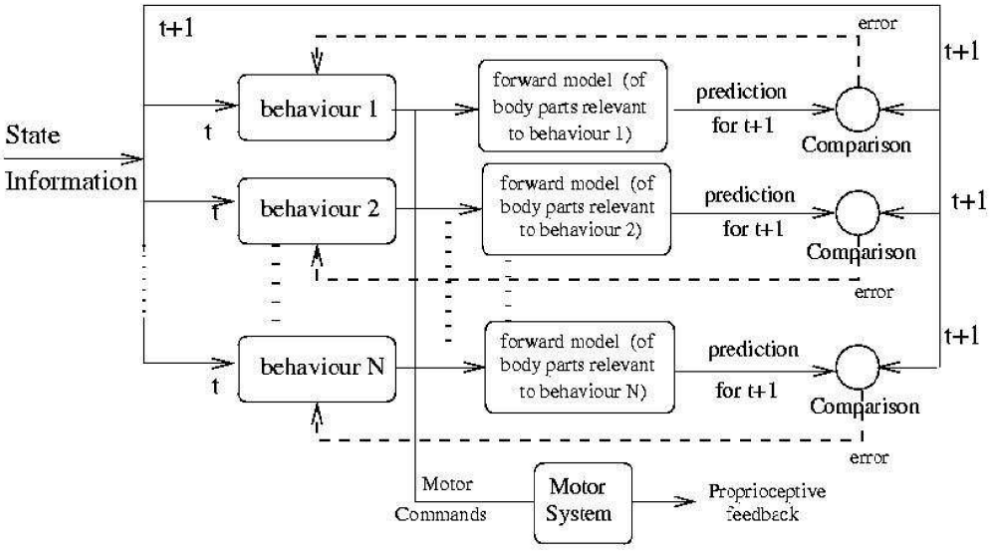


Figure 2.2: General depiction of the HAMMER architecture. Figure taken from [2] and reproduced with kind permission of the corresponding author Yiannis Demiris.

control task which is determined by its prediction performance.

General depiction of the HAMMER architecture can be seen in figure 2.2. Behavior boxes in this figure represent the actions which are modelled as inverse models. HAMMER is not directly used for generating actions but rather for imitation. Imitation in this context is defined as performing the same action as the actor, not to be confused with imitation learning mentioned before.

Imitation procedure (considered as the action generation part of this model) is done as follows: The actor's state information (e.g. cartesian positions or joint angles) is given to the behavior model which calculates necessary motor commands to reach it. These commands are then given to the forward models to predict the next state. Predicted state and the actual state is compared to calculate *confidence values*. The motor commands of the action that has the highest confidence is used for action generation.

The aforementioned imitation scheme can be directly converted to recognition; by taking the action with the highest confidence value as the recognized action.

Authors have defined different ways to calculate confidence values in different studies. In [19], simple actions of a mobile robot such as moving to an object, picking up an object etc. is used as the behaviors of the HAMMER architecture which is utilized to recognize the action

of a human. These simple actions are accompanied by forward models. If the prediction of a forward model is in the same direction as the observed action, corresponding confidence value is increased, otherwise it is decreased. Note that only the direction matters and prediction and observed values are ignored. Confidence values are calculated per action as

$$C_i(t) = \begin{cases} C_i(t-1) + 1 + N & \text{if prediction and observation is in the same direction} \\ C_i(t-1) - 1 - N & \text{if prediction and observation is in opposite directions} \end{cases}, \tag{2.3}$$

where C_i is the i^{th} action's confidence value, t depicts time and N represents the previous number of correct (if prediction is in the same direction) or incorrect (otherwise) prediction steps of that action.

The action which has the highest confidence value is treated as the recognized action. This is a very qualitative measure of action similarity, but authors have shown that it can be used for recognition. The action models of the robot and the human are quite different and looking at the direction is the only option for recognition in this case.

In [20], reach-to-grasp action of humans is modelled by a biologically plausible minimum variance model [21] on a simulated two-DOF planar robot arm with a gripper. Thumb and index finger are modelled by the gripper tips. Human reach-to-grasp behavior is observed and authors show that recognition is possible through generation of large confidence values for congruent actions. Note that in order to recognize the action, the action generation part was carefully modelled. In this setting, there are more than one feature to track (position of the wrist and the finger tips of the human) and confidence value has to reflect this. A change in confidence value due to a single feature is calculated as,

$$\Delta C_i(t) = sgn(x^i(t) - x^i(t-1)) \times sgn(\hat{x}^i(t) - \hat{x}^i(t-1)) \times (\hat{x}^i(t) - \hat{x}^i(t-1)) \times \omega_i, \tag{2.4}$$

where t denotes time, i denotes the feature, x^i is the observed feature, $\hat{x}^i$ is the predicted feature and ω_i is a constant that determines the relative importance of the feature in confidence calculation. Then the confidence value is defined as,

$$C(t) = C(t-1) + \sum_{i}^{n} \Delta C_i(t), \tag{2.5}$$

where $C(t)$ is the confidence value and n is the number of features. Action recognition is done as the same way as described before; the action which has the highest confidence value is treated as the recognized action.

2.3 Mental State Inference

Mental State Inference (MSI) [22] model is a computational model of the macaque monkey sensory feedback and action generation system which is extended to action recognition. The model includes predictors to compensate for sensory processing delays. MSI model has two working modes, namely the actor mode and the observer mode.

In the actor mode, shown in upper part of figure 2.3, MSI model generates actions. Intention module, selects the action to be performed which in turn determines the variable to be controlled (denoted as X). This module also sets the goal (denoted as X_{des}) of the action. The goal is fed to the movement planning block to generate necessary change in motor variables which are then sent to the movement execution block where the actual control is handled. The resulting motion is observed and necessary variables are calculated and sent to the movement planning block to close the loop. MSI model assumes that the sensory processing (observation) is slow and a forward model should be used to compensate for the resulting delay. This forward model takes desired change in motor variables as input and outputs predicted next state, which is then fed back to movement planning block to close the loop.

MSI model argues that the mirror neurons act as these forward models (predictors) which compensate the sensory delay. It is necessary to deal with this delay if a fast or accurate motion is desired.

In the observer mode, shown in the lower part of 2.3, MSI model tries to infer the intention of the actor i.e., tries to recognize the action being performed and the goal of the action. It exploits the forward model to recognize observed actions. In this mode, recognition starts with an initial intention (action and goal) estimate. Motor planning block generates motor commands, then forward model predicts next state of the control variable. This is called the *mental simulation*. At the same time, related control variables are extracted from observations. The intention estimate is updated according to the difference between predicted and measured control variables. Loop continues on until the predictions converge to the observations.

It is important to note that the control variables are calculated with respect to the object being acted on so that same variables could be used both during the actor mode and the observer mode.

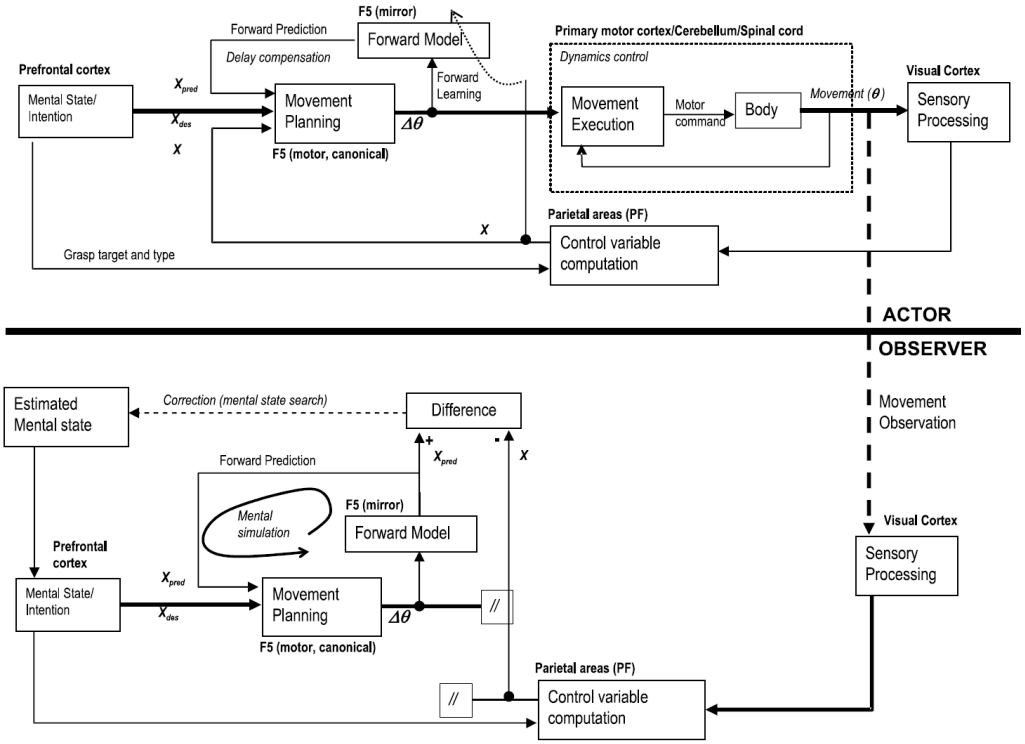


Figure 2.3: Upper part: Actor mode. Lower part: Observer mode. (Figure taken from [1] and reproduced with kind permission of the corresponding author Erhan Öztop)

2.4 RNNBP Approach

Recurrent Neural Network with Parametric Biases (RNNPB) is a modified Jordan-Type recurrent neural network with some special neurons in its input layer [23, 24]. These neurons constitute the *parametric bias* (PB) vector, which can both be input and output, according to the context. RNNPBs are used to learn, generate and recognize actions [25].

Recurrent neural networks simulate dynamic systems and the PB vector of an RNNPB is used as bifurcation parameters. Roughly, bifurcation means a qualitative change of behavior of a dynamical system. In theory, by changing PB vectors, infinitely many trajectories can be generated by an RNNPB. In this context, PB vectors represent actions.

In the learning mode, depicted in figure 2.4(a), of the RNNPB approach, the network is presented with sensory motor data of actions as both input and output so that it becomes a sensorimotor predictor after training. The actions have individual PB vectors but share the network weights.

The PB vectors and the network weights are calculated during learning. Weights are estimated using back-propagation through time (BPTT) algorithm. RNNPBs employ an iterative scheme to generate PB vectors during learning. PB vectors are calculated to reduce prediction error. These iterations and iterations of BPTT are done in an interleaved manner.

In the generation mode, shown in figure 2.4(b), PB vector corresponding to desired behavior is given as input to the network. In addition to being fed back as input, motor component of the outputs are sent to necessary actuators to generate the action, which is not shown in the figure. In the figure, sensory component of the output is directly fed to back to network. This could be replaced with actual sensory data for a closed-loop operation. If the outputs are not sent to the actuators, then RNNPBs just generate sensorimotor trajectories.

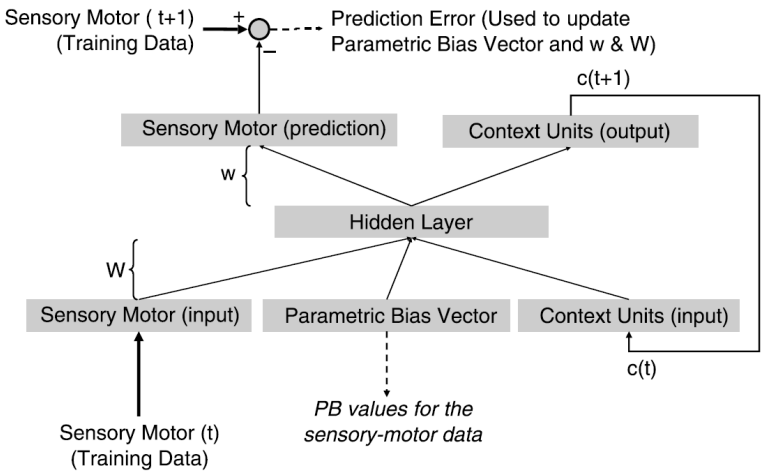
In the recognition mode, depicted in figure 2.4(c), observations are used as sensory data and motor output is connected to motor input. Difference between the predicted and observed sensory data i.e. prediction error is used to compute PB vectors as in learning. When PB vector converges, it is compared with the vectors acquired during learning for recognition. Thus recognition is done in the parameter space over the PB vector.

An interesting property of this type of neural network is that the PB vectors can be both inputs and outputs according to the mode of operation. In the learning and recognition modes PB vectors are outputs and in generation mode they are inputs.

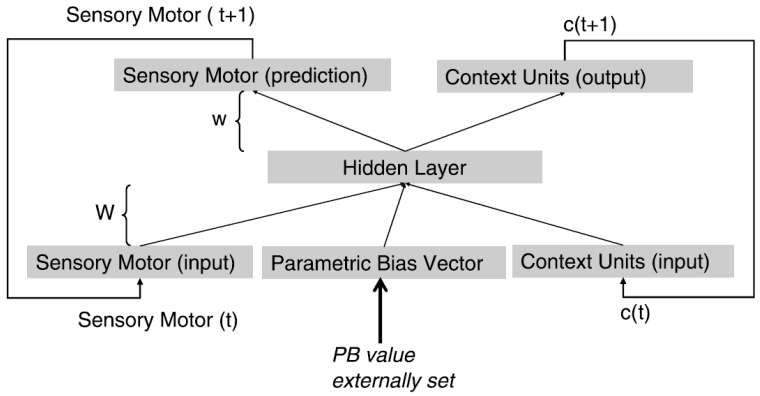
2.5 Mirror Neuron System (MNS) 1&2

Mirror Neuron System (MNS) models are computational models of the macaque mirror neuron system [26, 27]. These models assume that the mirror neurons are not innate (i.e., present at birth) but develop during infancy through observing self-executed actions to provide sensory feedback for future actions. This property is extended for action recognition, where the feedback is calculated for the observed action.

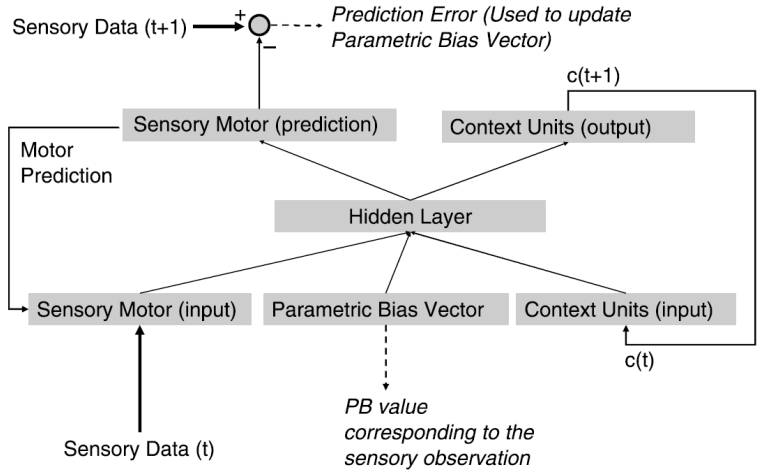
MNS1 implements a feed-forward neural network and MNS2 implements a Jordan-type recurrent neural network for modeling. Predefined actions are generated in a simulated environment and some features are extracted from these. Time course of these features are taken as inputs and action codes are taken as targets to the neural networks. Action codes represent



(a) Network in Learning Mode



(b) Network in Generation Mode



(c) Network in Recognition Mode

Figure 2.4: Modes of RNNPB Network. (Figures taken from [1] and reproduced with kind permission of the corresponding author Erhan Öztop)

the action being performed.

The features are defined with respect to the object (e.g. distance between hand and object) being acted upon, similar to the MSI model. This eliminates the *agent* concept, making inputs to the system during acting and observing the same. This way MNS models work the same way both during generation and observation of actions.

These models are used to describe the sensory feedback, not the action, thus they do not provide any means of action generation and learning. However, their use of agent independent features is a key point.

2.6 Dynamic Movement Primitives

Dynamic Movement Primitives (DMP) are trajectory generators formulated as nonlinear dynamic systems [28, 29]. They are inspired by mass-spring-damper (MSD) equations. DMPs are used to learn and generate actions. Their mathematical formulation is

$$\ddot{\vec{x}} = K(\vec{g} - \vec{x}) - D\dot{\vec{x}} + K\ \vec{f}(s, \vec{w}). \tag{2.6}$$

In equation 2.6, $\vec{x}$ represents the controlled degrees -of-freedom (DOF). The controlled DOFs could be joint positions or cartesian coordinates. K is the spring constant and D is the damping coefficient of the system. The goal point is specified by $\vec{g}$, set as the desired value of $\vec{x}$. $\vec{f}(\cdot)$ is a nonlinear function that augments the MSD system to generate different trajectories. $\vec{w}$ and s, represent the parameters and the variable of the nonlinear function respectively. s is also called the phase variable and shared among the DOFs (i.e. elements of $\vec{x}$).

Equation 2.6 is used to create necessary accelerations so that the controlled DOF reaches the goal point. For trajectory generation, these accelerations can be integrated to get the position and velocity. For action generation, these accelerations are converted to necessary motor commands and sent to the actuators. For example inverse kinematics can be used for this conversion if the controlled DOFs are the cartesian coordinates of a robotic arm's end-effector. If the controlled DOFs are directly joint variables, then the conversion is trivial.

A DMP formulation has two parts, namely the canonical part and the nonlinear part. The canonical part of equation 2.6 is,

$$\ddot{\vec{x}}_c = K(\vec{g} - \vec{x}) - D\dot{\vec{x}}, \tag{2.7}$$

where $\ddot{\vec{x}}_c$ is called the canonical acceleration. Equation 2.7, is the force equation of a MSD system with unit mass. MSD is a robust second order system with point attractor dynamics, i.e., the system converges to $\vec{g}$ with zero velocity, even in the presence of perturbations. Equation 2.7 also corresponds to PD-Control policy with K being the proportional gain and D being the derivative gain.

An MSD system has a characteristic measure called the damping ratio which describes the energy dissipation rate of the system. For DMPs, damping ratio of the MSD system is chosen as 1 to ensure that the system reaches its goal as fast as possible without oscillating. This damping ratio leads to $D = 2\sqrt{K}$.

The canonical part can only generate straight line trajectories in the controlled space. The term $\vec{f}(\cdot)$ is included in the DMP formulation to generate more complex trajectories. The nonlinear part of the equation 2.6 is

$$\ddot{\vec{x}}_n = K\, f(s, \vec{w}), \tag{2.8}$$

where $\ddot{\vec{x}}_n$ is called the nonlinear acceleration. The nonlinear function can have many different forms. The main reason to include such a function in the formulation is to have an imitation learning ability. Hence, $\vec{f}(\cdot)$ is mostly formulated as a function approximator.

A linear basis function model for the nonlinear function is used in [30]. Let $f(\cdot)$ denote an arbitrary element of the function $\vec{f}(\cdot)$. Then,

$$f(s, \vec{w}) = \frac{\sum_{i=1}^{N} \psi_i(s)\, w_i}{\sum_{i=1}^{N} \psi_i(s)} s, \tag{2.9}$$

where $\Psi_i(s)$ is the i^{th} kernel(basis) function, $\vec{w}$ is the adaptive parameters of $f(\cdot)$, w_i is the i^{th} element of $\vec{w}$ and N is the total number of basis functions. There are different forms of kernel functions but most common one is the radial basis function formulated as,

$$\psi_i(s) = e^{-h_i(s-c_i)^2}, \tag{2.10}$$

where c_i is the i^{th} kernel center and h_i is the i^{th} kernel's width. The kernel centers and the kernel widths need to be adjusted according to the phase variable formulation. The phase variable may decrease exponentially, thus to capture the end of the motion, the kernel centers

need to be closer together and they need to be narrower towards the end. If phase variable decreases uniformly, centers should be equispaced and widths should be the same.

The phase variable s is used to bind individual DOFs together in order to synchronize their motions. The phase variable is constrained to monotonically decrease or increase between [0 1]. This makes the phase variable act as an indicator of the amount of completion of an action or in other sense normalized time which is not necessarily flowing linearly. A common formulation of s is as follows,

$$\dot{s} = -\alpha s, \tag{2.11}$$

where α is a positive constant, defining the decay rate of s. With an initial value of 1, the phase variable decays from 1 to 0 in a monotonic manner as the action progresses. The time evolution of the phase variable can be acquired by solving the equation 2.11 to get,

$$s = e^{-\alpha(t-t_0)}, \tag{2.12}$$

where t_0 is the initial time.

Imitation learning corresponds to estimating $\vec{w}$ parameter in equation 2.8 from observed trajectories i.e., demonstrations. In order to accomplish this, $f(\cdot)$ is left alone in equation 2.6 to get,

$$f(t) = \frac{\ddot{x}(t) - K(g - x(t)) + D\dot{x}(t)}{K}. \tag{2.13}$$

For learning, it is assumed that parameters of the canonical part are known and the motion is observed i.e., K and D are known, $x(t)$, $\dot{x}(t)$, $\ddot{x}(t)$, and g are available and α in equation 2.11 is estimated from the duration of motion. Then, $f(t)$ is calculated using equation 2.13 and $s(t)$ is calculated using equation 2.12. From these, $f(s)$ is obtained and the *regularized least squares* method is used to calculate $\vec{w}$.

A different weight vector, $\vec{w}$, is learned per action. To generate a learned action, corresponding weight vector is used in equation 2.6 to generate that action. The set of all learned parameter vectors are called the *action repertoire* of an agent.

In [31, 32], the aforementioned imitation learning approach is extended for action recognition using DMPs. The observed action is treated as a new action to be learned and $\vec{w}$ is estimated. This is done after the whole action is observed. These parameters of the observed action are then compared with previously learned parameters. Comparison is done with a correlation

metric, defined as,

$$C = \arg\max_i \left(\frac{\vec{w}_i^T \vec{w}_0}{|\vec{w}_i||\vec{w}_0|}\right) \quad \text{for i} = 1 \ldots \text{n}, \tag{2.14}$$

where C is the recognition decision, $\vec{w}_o$ is the learned parameter vector of the observed action, $\vec{w}_i$ is the parameter vector of the i^{th} action in the observer's repertoire and n is the total number of actions in the observer's repertoire.

2.7 General Approaches

Two general approaches for action recognition could be derived from the studies that have been reviewed. These are namely the parameter space approach and the trajectory space approach.

In the parameter space approach the observed action is treated as a new action to be learned and parameters are calculated from observations, as in the DMP and the RNNPB approaches. These parameters are then compared with previously learned parameters. An immediate way of comparison is the correlation metric, described by the equation 2.14. For this approach to be applicable, there needs to be an imitation learning mechanism available. This approach has a major limitation in the sense that the action has to be fully observed for recognition which is not suitable for online recognition and the purpose of this thesis.

In the trajectory space approach, observer simulates trajectories using his action generation mechanisms and compares these trajectories to the observed one. A suitable matching criterion needs to be defined which is not as trivial as in the parameter space case. Recognition approach of MSI and HAMMER models could be given as examples.

At first, it seems that any action generation mechanism could be facilitated for action recognition using the trajectory space approach, provided that the actor and the observer have similar action generation mechanisms. However, issues about the matching criterion, space of matching and the related correspondence problem mentioned in Chapter 1 need to be carefully addressed. The approach of this thesis can be classified as a trajectory space approach.

2.8 Discussion

In this section, the aforementioned approaches are compared against the action recognition requirements presented in section 1.3. Table 2.8.6 is used to ease and to summarize the discussion.

2.8.1 Action Models

Action recognition using action generation requires that the actor and the observer have similar action generation mechanisms. There are two main approaches to accomplish this. The first one is to have an imitation learning mechanism (see discussion in section 1.4) and the second one is to model the actor's actions. Another option is to assume that the actor and the observer has the same action models. Refer to the *Action Models* column of the table 2.8.6 for a summary. The approaches that the methods take can be stated as,

- The DMP and the RNNPB approaches provide full imitation learning as described in the relevant parts of the literature survey.
- The MOSAIC approach can be considered to have a way for imitation learning to some extent if controller learning is devised accordingly.
- The MSI and the HAMMER approaches do not address any imitation learning and assume that the actor and the observer have similar actions. HAMMER approach uses modelling in some implementations to achieve this.

The addition of new actions is also another concern in the context of developmental robotics. Adding new actions is easy in the DMP, the MSI and the HAMMER approaches since actions are modelled with separate controllers. Adding new actions to the RNNPB and the MOSAIC approaches requires retraining of all the actions since these model the actions in a distributed architecture.

2.8.2 Variable Selection

The variables in the MSI and the MNS models are calculated with respect to the object. This allows the same variables to be used during action generation and action recognition. Space

of matching becomes the space of these variables.

Other approaches require the mapping of observed variables to a suitable space. The most common example is to map the observations to a joint space which is not so easy to achieve. The variables that are used in each approach is given in the *Variables* column of the table 2.8.6.

2.8.3 Action Similarity Metric

Defining action similarity is the problem of finding a suitable matching criterion i.e., a similarity metric. This metric should be defined to compare two actions. To summarize the related metrics which are described in the previous sections of this chapter are as follows;

- The RNNPB and the DMP approaches use correlation of parameters as their similarity metric, since they employ the parameter space approach.
- Confidence values are defined as the similarity metric in the HAMMER architecture. Definition of confidence values differ across implementations.
- The MSI approach uses the instantaneous error between its predictions and the observed motion.
- In the MOSAIC model, the responsibility signals are used to compare the actions which are calculated from instantaneous error.

Similarity metrics should be robust to reduce the effect of noise on decision. However, both the MSI and MOSAIC approaches use instantaneous error to calculate their similarity metrics, which is prone to noise. HAMMER approach doesn't suffer from this. This discussion is only for online recognition so it is not applicable to the other two approaches.

2.8.4 Online Recognition

Online action recognition is at the heart of this thesis. Thus it is necessary to point out what the methods that have been surveyed offer in this sense. The nature of the recognition methods are as follows;

- The DMP and the RNNPB models use parameter space approach for recognition, thus they are limited to offline recognition.
- Both of the recognition schemes of the MSI and the HAMMER models are online.
- In the MOSAIC approach, the nature of recognition depends on the implementation. Recall that responsibility signals calculated from the observation are compared with previously generated responsibility signals for recognition and that the responsibility signals have a time evolution. If a partial comparison of these signals suffices, then recognition approach is online, if not, then it is offline.

For a summary, refer to the *Action Recognition* column of table 2.8.6.

2.8.5 Goal Setting

It is important to note that some methods allow for goal setting and some do not. The importance of this is that the method could explicitly be generalized to many objects (i.e., goals) if it allows this. To summarize;

- The RNNPB approach does not provide any means for goal-setting which is a major drawback, since there is no way to input the goal to the neural network.
- Goal setting in MOSAIC and HAMMER are not directly addressed.
- Both the MSI model and the DMP approach provide means for explicitly setting the goal of an action.

2.8.6 Verification

In robotics, it is necessary to demonstrate that the theoretical methods work on a practical problem. All the work that has been reviewed can theoretically be used for action recognition but not all of them have been tested in this context. To give a brief summary;

- The recognition approaches of the RNNPB and DMP have been tested on real world data. Refer to the relevant references for more detail.

- The MOSAIC model has also been tested but not for action recognition. In addition, more experiments should be done to fully test the theoretical claims.
- The confidence values are calculated from real world data in the HAMMER approach, but these are not explicitly used for recognition.
- The MSI model has been tested on simulated data.
- MNS models are trained and tested in a simulated environment.

From this summary, it can be seen that there has not been any real world experiments focused solely on the online recognition of actions using action generation mechanisms. This thesis aims to fill this gap.

Table 2.1: Action recognition related properties of the approaches that have been reviewed.

	Action Models	**Variables**	**Similarity Metric**	**Action Recognition**	**Goal Setting**
MOSAIC	Controller Learning	Joints	Responsibility Signals	Depends on implementation	Not directly addressed
HAMMER	Modelling or Assuming the Same	Depends on Implementation	Confidence values	Online	Not directly addressed
MSI	Assuming the Same	Object-centered	Instantaneous Error	Online	Possible
RNNPB	Learning by Demonstration	Sensor space (Joints, image)	Correlation	Offline	Not Possible
DMP	Learning by Demonstration	Joint or Cartesian Space	Correlation	Offline	Possible

CHAPTER 3

RECOGNITION APPROACH

The action recognition approach developed in this thesis uses an action generation mechanism at its core. Instead of developing a new action generation mechanism, DMPs were chosen to be used. The reasoning behind this choice is that DMPs either satisfy or are easily modified to satisfy the action recognition requirements presented in section 1.3.

3.1 Detailed DMP Analysis

3.1.1 Strong Points

- DMPs have good generalization properties. They can be formulated to be invariant under affine transformations [33], which allows them to produce trajectories for any starting and end point.
- DMPs are robust and stable. The linear part of a DMP converges to a given goal even in the presence of noise. The nonlinear part decays to zero as it contains a multiplication by the phase variable (equation 2.9) which is defined to decay to zero (equation 2.12).
- Imitation learning is fast. Estimating $\vec{w}$ is not an iterative process if regularized least square method is used. This method involves a matrix inversion as its most costly operation which is fast for modern computers given practical problem sizes.
- Action generation is online i.e., equation 2.6 can be computed in real time and its output applied to the actuators.
- Goal setting and online changing of the goal point is trivial through a single variable, $\vec{g}$, in equation 2.6.

- DMPs are desirable from a practical point of view. They are fairly easy to understand and implement. The linear part (MSD system) is described in many text-books. The function approximation algorithm used in DMPs is common and there are many computer libraries that implement it.

3.1.2 Shortcomings

DMPs are designed for imitation learning from a single demonstration and generalization of learned actions to different start and end points. However they have a number of shortcomings, mostly from an online action recognition point of view. These shortcomings are;

- The recognition scheme is not suitable for online action recognition, that is the action needs to be fully observed to estimate $\vec{w}$ for comparison. The function approximation method dictates this necessity.
- The phase of the system, s, is independent from $(x, \dot{x})$, which determines the state of the system. This means that the nonlinear part of the system runs in an open-loop fashion and s and $(x, \dot{x})$ are only coupled in time. If somehow the time evolution of the state is disturbed (e.g. robot arm gets stuck to an obstacle for a duration), s will not be able to adapt itself and it will continue to decay.
- The action has to be fully observed to estimate α of equation 2.11. This parameter is estimated according to the duration of the action.
- The starting point of the action, t_0, needs to be observed to calculate s values, either through equation 2.11 or equation 2.12. Some DMP formulations incorporate the initial position x_0. These are the internal parameters of the system and their correct observations cannot be guaranteed.
- Imitation learning is done through a one-dimensional parameter, s, which is not suitable for learning from multiple demonstrations. Instead of capturing interesting points of the demonstrated trajectories, it would tend to average them out.

Another drawback of DMPs is that they are designed for learning from a single demonstration. Even though DMPs are invariant under affine transformations, generalizations of an action

for different starting points may not be just the scaled and rotated versions (affine transformation) of that demonstrated action. This combined with lack of proper learning from multiple demonstrations can be problematic in some scenarios.

In short, there are two major problems with DMPs for action recognition. First is the lack of an online recognition scheme, second one is the phase variable formulation. The following section is about the modifications done on the DMPs to overcome their shortcomings and the other section addresses the online recognition issues.

3.2 Modifications to DMPs

3.2.1 Phase Variable

The phase variable formulation in 2.11 depends on only itself and the initial time (internal parameter) and is independent of the state of the system which makes online recognition difficult. There could be different phase variable formulations. For example ($\frac{x-x_0}{g-x_0}$) in [32] overcomes some of the aforementioned shortcomings. However, this is still insufficient, since the system could be at the same x with different $\dot{x}$. Then, these would have the same phase value when they should not. In addition, this formulation will not generate any motion on a DOF where $x_0 = g$. This is the case for example where the robot needs to pick up a glass on a table and put it to another position on the same table. In this case, the initial (x_0) and desired (g) height of the object are the same and assuming zero initial velocity, the equation 2.6 will produce zero acceleration.

Instead of mapping the state onto an explicit phase variable and learn $\vec{f}(\cdot)$ with respect to it, this study chose to learn the function directly with respect to the state by replacing, $\vec{f}(s, \vec{w})$ with $\vec{f}(\vec{z}, \vec{w})$ in equation 2.6, where state vector $\vec{z}$ is constructed by concatenating $\vec{x}$ and $\dot{\vec{x}}$ vectors. Such a change of formulation makes the nonlinear part dependent on the state, allowing it to run in a closed-loop manner, like the canonical part. Moreover, it removes the dependence on internal parameters (x_0 and t_0), making recognition easier.

Note that the $\vec{z}$ vector could possibly be constructed from different variables and as long as they can be measured online, the arguments about the system would not change.

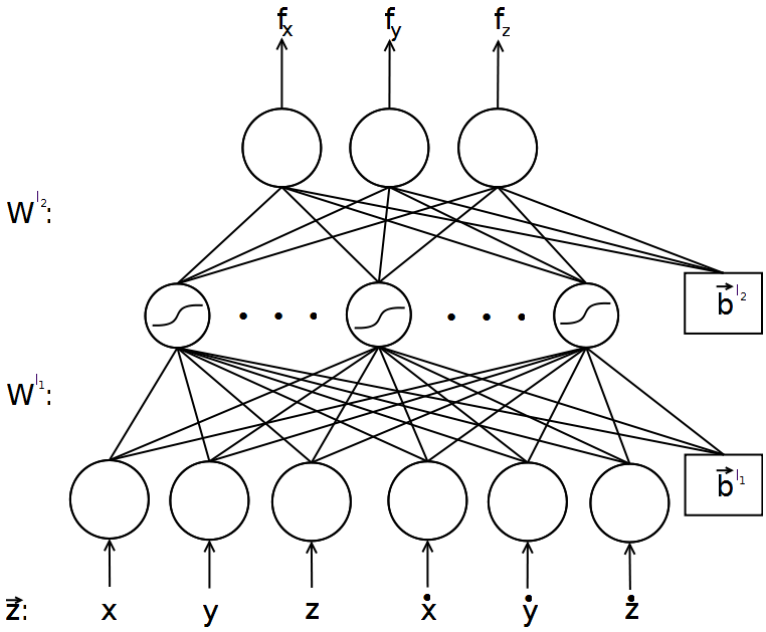


Figure 3.1: Neural network used as the $f(\vec{\cdot})$.

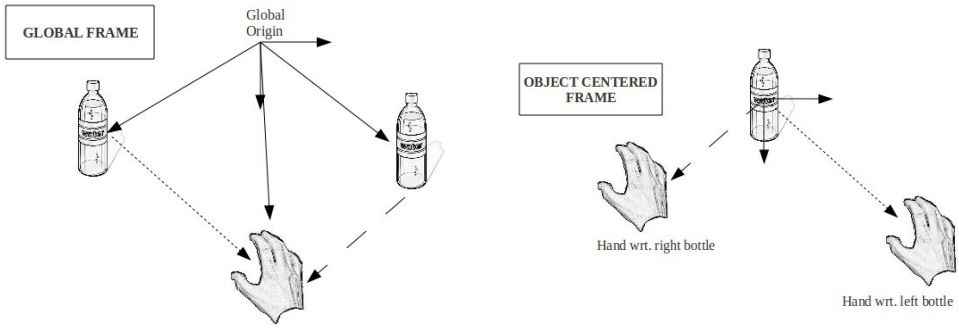
3.2.2 Nonlinear Part

The nonlinear part should be adapted to incorporate its new input, $\vec{z}$. A linear basis function model was used for $f(\cdot)$ with the previous phase variable formulation which was one-dimensional. However, $\vec{z}$ is multi-dimensional and a linear basis function model wouldn't scale as before. Suppose that there are 20 basis functions in a model with a single degree of freedom. If this is extended to an n-DOF system, then the model will have 20^n number of basis functions which is not practical. Instead a multi-layer feed-forward neural network is used to learn $f(\vec{\cdot})$ which makes the $\vec{w}$ in equation 2.6 the weights of the neural network.

Levenberg-Marquardt (LM) backpropagation [34] is used to train the neural network as it has better convergence efficiency than gradient descent methods. The implementation of LM backpropagation algorithm in the Neural Network Toolbox™ of Matlab® is used.

For a given set of demonstrations, multiple neural networks with different number of hidden layer neurons are trained. These networks are then used in equation 3.1 to create trajectories given the initial state. These trajectories are compared with the actual trajectories and the network which creates the most similar ones is chosen as the corresponding action's network.

Neural network simulation is implemented in C++ because of the performance concerns. This



(a) Global Frame (b) Object Centered Frame

Figure 3.2: Different frames that can be used to measure Cartesian coordinates.

implementation is two orders of magnitude faster than Matlab®'s.

3.2.3 Controlled Variables

The variables, $\vec{x}$ in equation 3.1, are defined in cartesian space with respect to the object and are called *hand-object relations*. This choice, inspired by [22, 26], permits the same variables to be used both during observation and generation of actions, greatly reducing actor-observer differences, allowing seamless use of generation mechanisms in recognition.

The hand-object relations are defined as the relative position and velocity of the end-effector with respect to the object. This is achieved by placing the origin of the coordinate frame used to define the coordinates on the object as shown in figure 3.2. This makes the goal parameter g of equation 2.6 disappear.

Cartesian space are chosen over the joint space because it is easier to observe the end-effector rather then individual joints during recognition.

3.2.4 Resulting DMP Formulation

Considering the previous modifications and the choice of variables, the trajectory generator equation for a multi-dimensional system becomes,

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

The trajectory generation block diagram can be seen in figure 3.3.

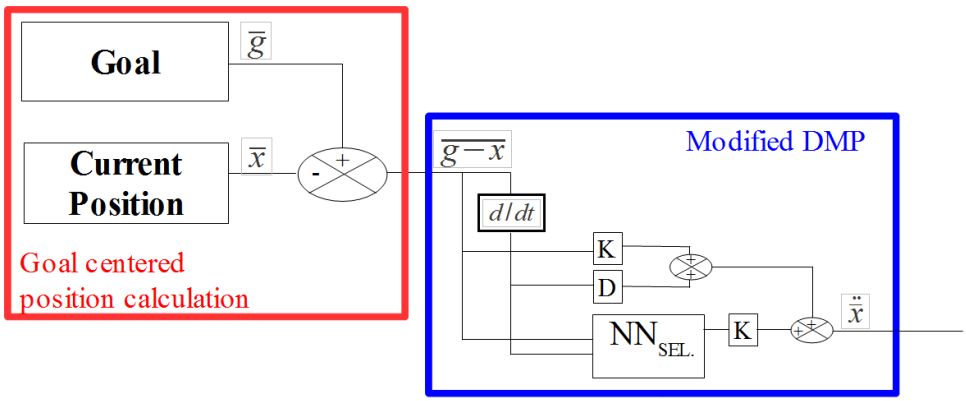


Figure 3.3: Action generation diagram. End-effector position of the robot is calculated from joint angles using forward dynamics (not shown). Motion capture system tracks the object being acted on. From these, hand-object relations are calculated and fed to the DMP. The DMP part then calculates the necessary accelerations to realize the action.

3.2.5 Imitation Learning

Imitation learning for an action is done by calculating the $f(t)$ values given observed the trajectories, $(x(t), \dot{x}(t), \ddot{x}(t))$, through the equation 2.13 and feeding these as targets and feeding $(x(t), \dot{x}(t))$ as inputs to the neural network.

Modifications to $f(\cdot)$ allow proper learning from multiple demonstrations. A single neural network is trained for different starting and end positions. Moreover, demonstrated actions with different starting and end positions do not have to be just the scaled and rotated versions of each other. Considering that human action data differ for different points and are noisy, training from multiple demonstrations is advantageous.

3.3 Recognition

The aim of this study is to achieve online action recognition using an action generation mechanism. Modified DMPs detailed before are the action generation mechanism that is at the core of the system. They will be used to simulate actions (i.e., generate trajectories) which are then compared to the observed action.

The block diagram of the recognition scheme is presented in figure 3.4. Recognition starts when the actor starts his action which is determined by tracking the actor's end-effector ve-

locity. If the velocity reaches above a certain threshold, actor is assumed to have started his action. At the start of recognition, initial state observations are given to action generation systems as initial conditions and future trajectories are simulated. These are then compared with the observed trajectory by calculating the cumulative error;

$$err_i(t_c) = \sum_{t=t_0}^{t_c} \left\| \vec{x}_o(t) - \vec{x}_i(t) \right\|, \tag{3.2}$$

where err_i is the i^{th} behavior's cumulative error, t_0 is the observed initial time, t_c is the current time, $\vec{x}_o$ is the observed position, $\vec{x}_i$ is the i^{th} behavior's simulated position.

Then, to have a quantitative measure of similarity between actions, *recognition signals* are defined as:

$$rs_i(t_c) = \frac{e^{-err_i(t_c)}}{\sum_j e^{-err_j(t_c)}}. \tag{3.3}$$

Recognition signals could be interpreted as the likelihood of the observed motion to be the corresponding simulated action. These are similar to the responsibility signals in [16]. However, responsibility signal calculation is based on instantaneous error which is prone to noise, whereas recognition signal calculation is based on cumulative error.

During the recognition phase, if the ratio of the highest recognition signal and the second highest gets above a certain threshold (Υ), the recognition decision is made and the action corresponding to the highest signal is chosen.

In case there are multiple objects in the environment, behaviors are simulated per object, by calculating the hand-object relations accordingly. If there are M behaviors and L objects in the scene, then $M \times L$ trajectories are simulated and compared with the observed one. This allows to recognize which object is being acted upon in addition to action type.

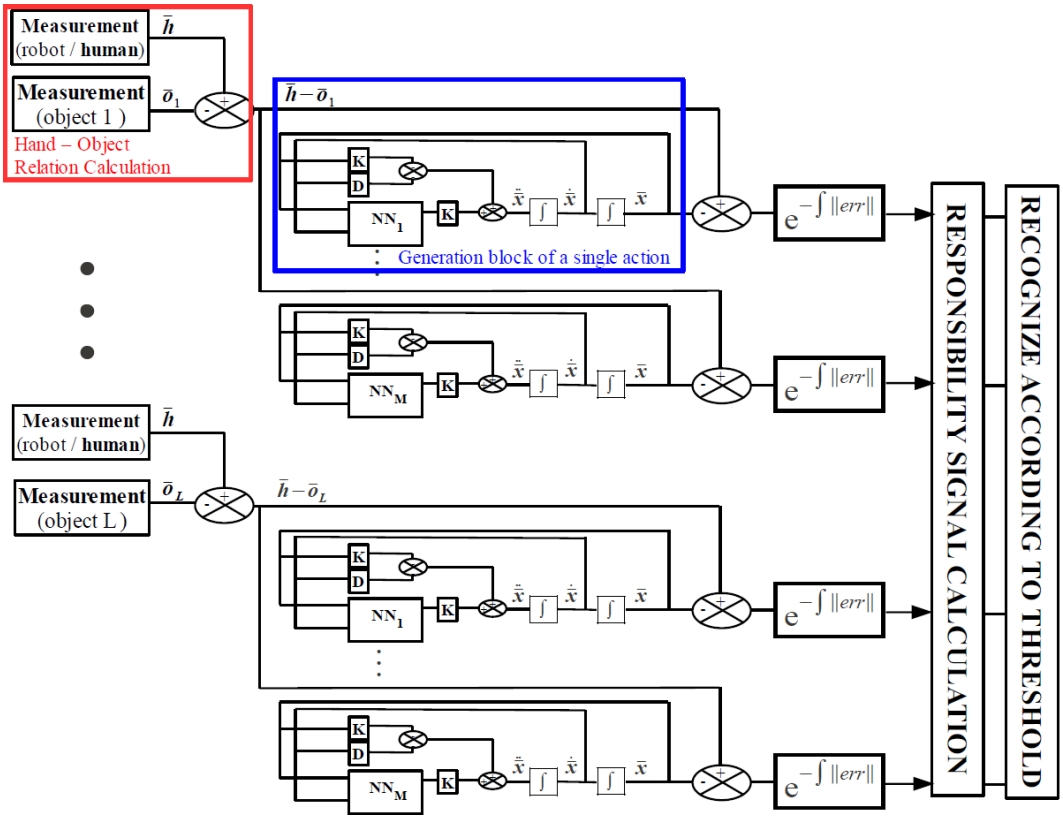


Figure 3.4: Recognition architecture flow diagram. Motion capture system tracks possible objects and the end-effector from which state variables are calculated. When recognition starts, the observed state is used as initial values to the learned behaviors (not shown). The learned behaviors (i.e., generation systems) then simulate future trajectories. As the action unfolds, observed and simulated trajectories are compared and responsibility signals are calculated. From these signals, recognition decision is made according to threshold.

CHAPTER 4

EXPERIMENTAL EQUIPMENT

4.1 The iCub Humanoid Robot

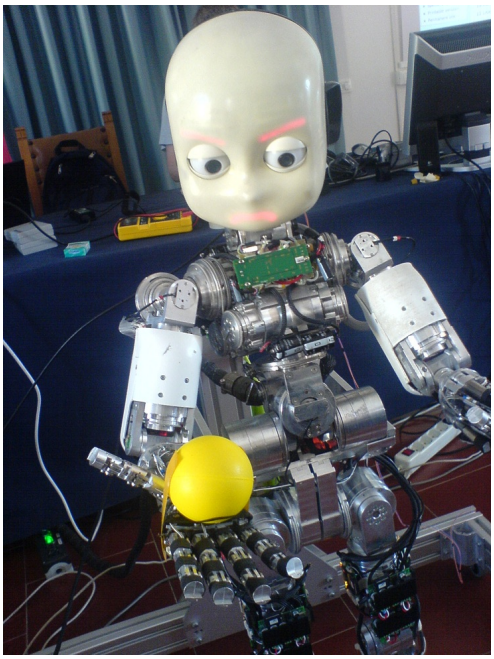
The iCub humanoid platform, shown in figure 4.1, is developed to study embodied cognition and developmental robotics [35] as part of the European Union Frame Program 7 Project *RobotCub* [36].

iCub is designed to resemble a three and a half year old child, in appearance and in physical and sensory capabilities. It is 104 cm tall and weighs approximately 22 kgs. It has 53 degrees-of-freedom (dof), with 7 on each arm, 9 on each hand (highly dexterous), 6 on each leg, 6 on its head and 3 on its torso. Its joints are actuated by brushed (head and hands) and brushless (rest) DC motors. Hands, shoulders, waist and ankles are driven by tendons. iCub can not stand on its own so it is fixed from its hip to a base, as seen in figure 4.1(b).

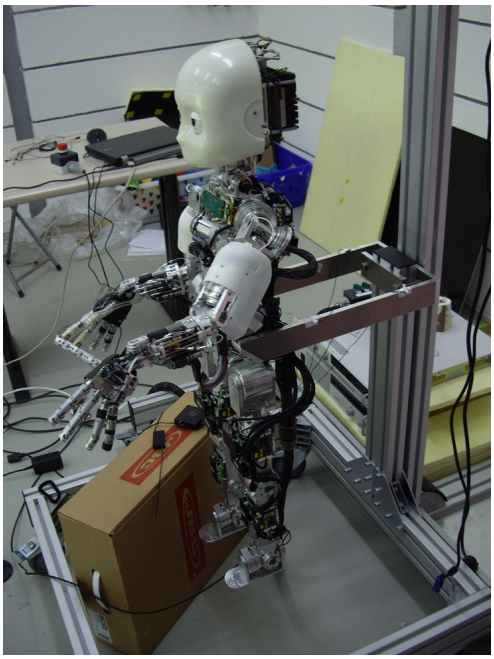
iCub can also display face expressions by moving its eyelids, controlled with a servo motor, and by using LEDS underneath the head cover to emulate eyebrows and a mouth as seen in figure 4.1(a).

The main processor of the robot is an Intel based dual-core PC104, situated in its head along with other electronics that is used for communication and control. The actuators of the robot are controlled with custom designed electronics and have separate circuits for power and control.

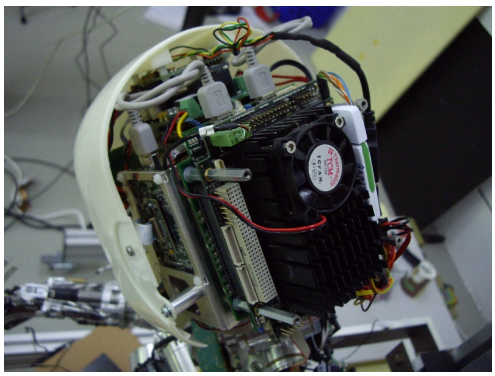
The middleware that is used to communicate with the robot is *Yet Another Robotics Platform* (YARP) [37, 38]. YARP is easy to use and configurable and allows distributed communication. It is aimed to support reusable software. It provides interfaces to control sensors and



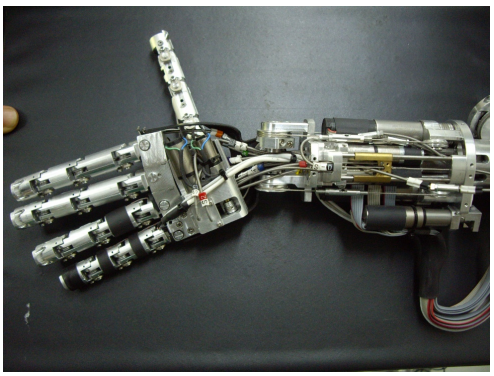
(a) Front view of iCub with face expressions.



(b) Side view of iCub. Also showing the base.

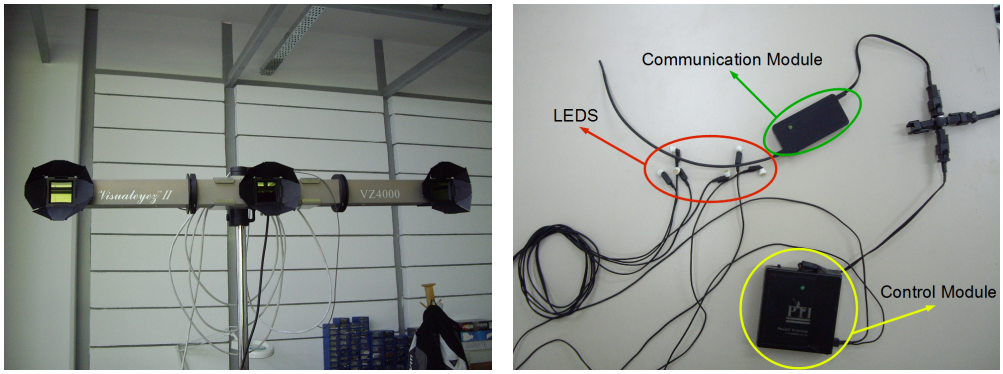


(c) Inside of iCub's head.



(d) iCub's arm.

Figure 4.1: The iCub Humanoid Robot



(a) Motion Capture Sensor (b) Accessories

Figure 4.2: Motion Capture Equipment

actuators. It implements many communication protocols used in robotics.

One significant property of iCub is that its full design, mechanic, electronic and software, is open source. For full specifications and latest developments please refer to [39].

4.2 Motion Capture System

The Visualeyez^{TM}VZ 4000 [40] (Phoenix Technologies Incorporated USA) is used to track the positions of the actor's end-effector and objects. The motion capture system can be seen in figure 4.2.

The motion capture system uses active markers (LEDS) to measure the 3D positions of the points of interest. It is calibrated by placing 3 markers to the desired $X-Y$ plane, after which the system measures the positions of the markers and calculates the necessary transformations.

The resolution of the system is 0.015 mm at 1.2 m with a volume accuracy of 0.5 mm. Volume accuracy is defined as the average accuracy achieved within a specified volume. The specified volume in this case is a constrained spherical space with radius between 0.6-2.2 m, azimuth angle between ±40 deg and zenith angle between ±30 deg.

CHAPTER 5

EXPERIMENTS AND RESULTS

5.1 Setup

In the experimental setup, the actor and the observer are seated facing each other as shown in figure 5.1. There are two objects hanging in between them. A motion capture system is used to track the actor's actions and the objects. This setup is used to gather training and testing data and to demonstrate the feasibility of using the action recognition scheme with a robot.

It can be seen from figure 5.1 that there are markers on top of the objects and wrist of the actor. Wrist of the actor is chosen to minimize the risk of occlusion.

5.2 Actions

For the experiments, three different reaching actions towards two objects are considered, as depicted in figure 5.2(a). The actions are defined as:

- R(X): Reaching object X from right
- D(X): Reaching object X directly
- L(X): Reaching object X from left

The objects are designated as:

- LO: Left object (wrt. the actor)
- RO: Right object (wrt. the actor)

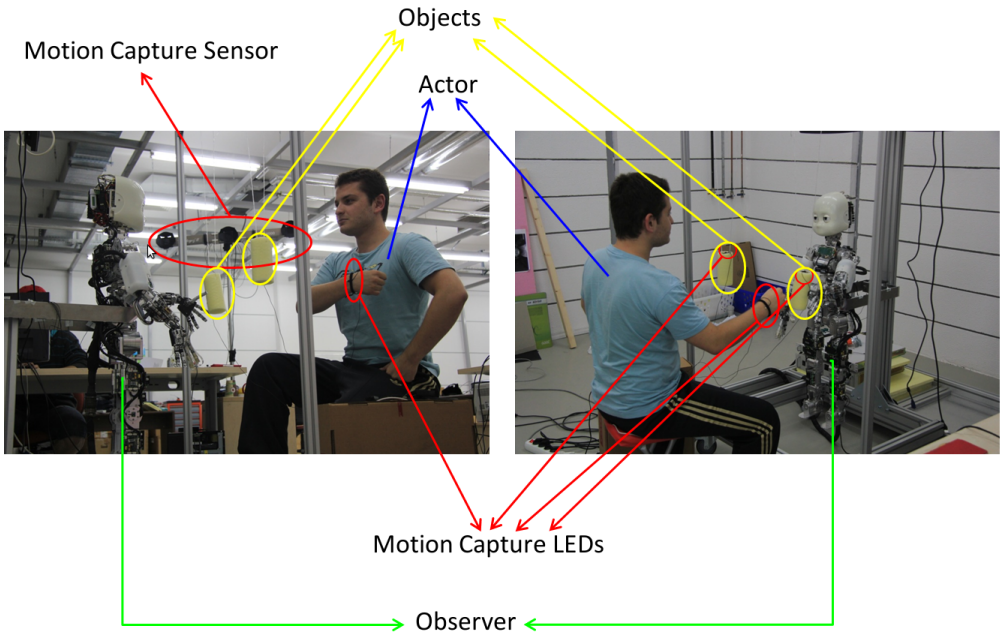


Figure 5.1: The experimental setup.

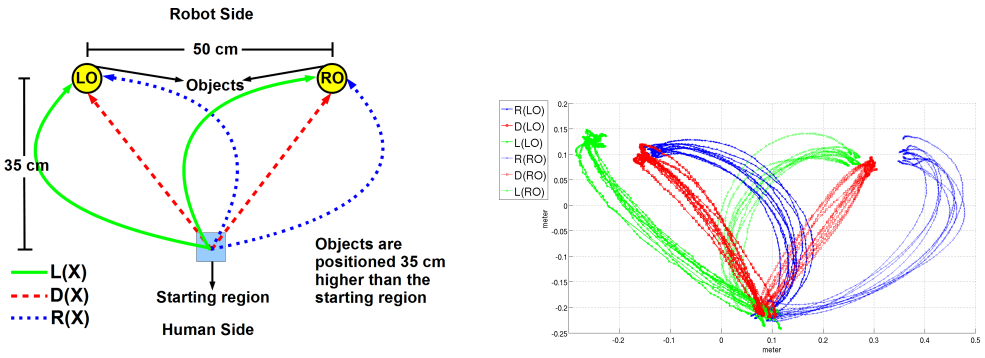
Multiple objects and actions were chosen to show that the recognition approach can recognize both the action and the object. In addition, multiple actions and objects result in a variety of options to make a recognition decision from. Having more options decreases the chance of making a correct decision randomly.

The actor(a human) performed 50 repetitions of each action on each object, totalling 300 repetitions which were recorded. Both object centers and the trajectory of the actor's wrist were tracked. Some recorded trajectory samples can be seen in figure 5.2(b). 60% of the trajectories are used to train the neural networks of three actions and the rest are used for testing the recognition approach.

Humans cannot start from the same position every time and follow these trajectories perfectly; i.e., real behaviors have noise and variance, as seen in figure 5.2(b).

The beginning and end of the recordings are truncated automatically to eliminate the parts before and after the action where there is no motion. This truncation is done according to a velocity threshold which is determined empirically to be $0.05 m/s$.

The motion capture system only records positions. However, recognition phase requires velocities and training phase requires both velocities and accelerations. Velocities and acceler-



(a) Depiction of behaviors and approximate geometric information from top view of the setup

(b) A subset of recorded human trajectories

Figure 5.2: Defined and applied actions on the setup

ations are obtained through simple finite difference approximations i.e., $\dot{x}(t) = (x(t + \Delta t) - x(t))/\Delta t$.

The noise in the recorded data is amplified due to differentiation. This effect is severe for acceleration calculation. Moreover, calculated $f(\cdot)$ from equation 2.13 becomes highly non-smooth which is undesirable for neural network training. To remedy, calculated velocities are filtered with an ideal low pass filter with a bandwidth of $10Hz$ to allow smoother differentiation. Position and acceleration are recalculated from the filtered velocity which are used to calculate $f(\cdot)$ to be given to the neural network as target. However, to increase generalization, non-filtered (i.e., noisy) positions and velocities are given to neural network as input.

The DMP formulation in equation 3.1 has K and D parameters which are determined empirically as $K = 10$ and $D = 2\sqrt{10} \approx 6.32$.

It is important to note again that number of neural networks trained is equal to the number of actions. Since there are three actions, only three neural networks are trained. Inputs of the neural networks change according to the object being acted upon.

5.3 Recognition Experiments

The emphasis of this thesis is on online recognition. This corresponds to making a correct recognition decision before the action completes i.e., decision times should be low and the success of the decisions should be high.

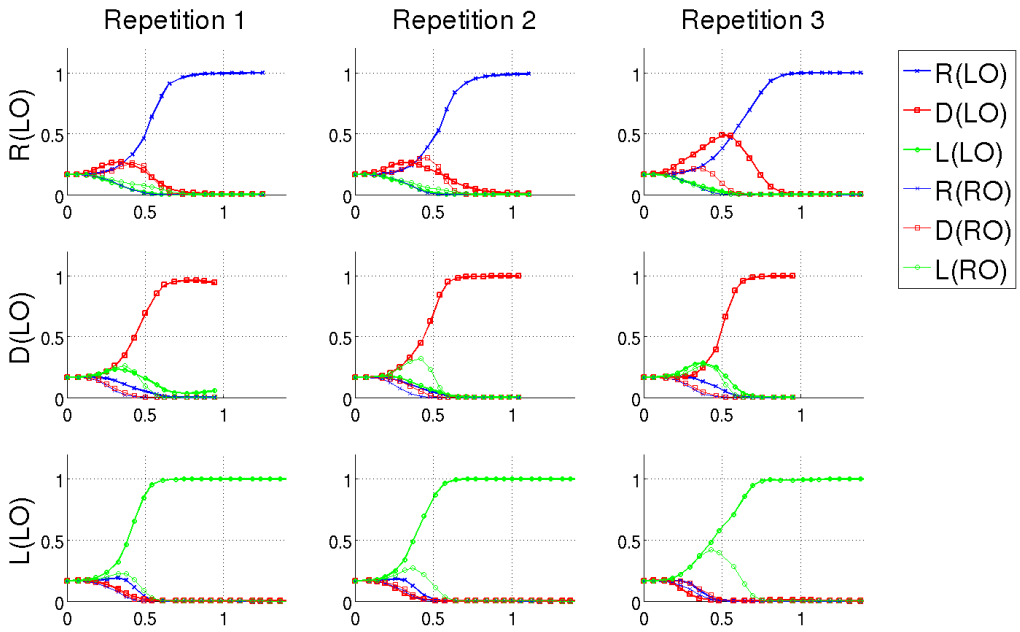


Figure 5.3: The time evolution of recognition signals. X-axis is time in seconds and Y-axis is recognition signal magnitude.

In order to show that the recognition system works, 20 repetitions of each action on each object, which result in a total of 120 recordings (40% of all recordings), are tested with the recognition approach detailed in section 3.3. Specifically, trajectory of the recognition signals of these recordings are calculated using the equation 3.3 and the decisions are made with different Υ values. The success rate of decisions and the mean and variance of the decision times are calculated. Moreover, the confusion matrix is constructed to show the mistakes that the system makes.

The time evolution of recognition signals for nine different recordings are plotted in figure 5.3. In all plots, recognition signals start from the same initial value ($\frac{1}{6}$) and as the action unfolds recognition signal of the corresponding action goes to one while suppressing others. Although there may be a confusion in the beginning parts of an action (e.g. top right plot), the system recognizes the action correctly as it unfolds.

There is a trade-off between the decision time and the success rate and both are directly affected by the threshold value, Υ. Success rates and decision times for correct decisions are calculated for different Υ values and are plotted in figure 5.4. Actions have varying durations thus the decision times are not represented in time units but by the percentage of completion of the corresponding action. The figure 5.4 shows the trade-off. As Υ increases both the

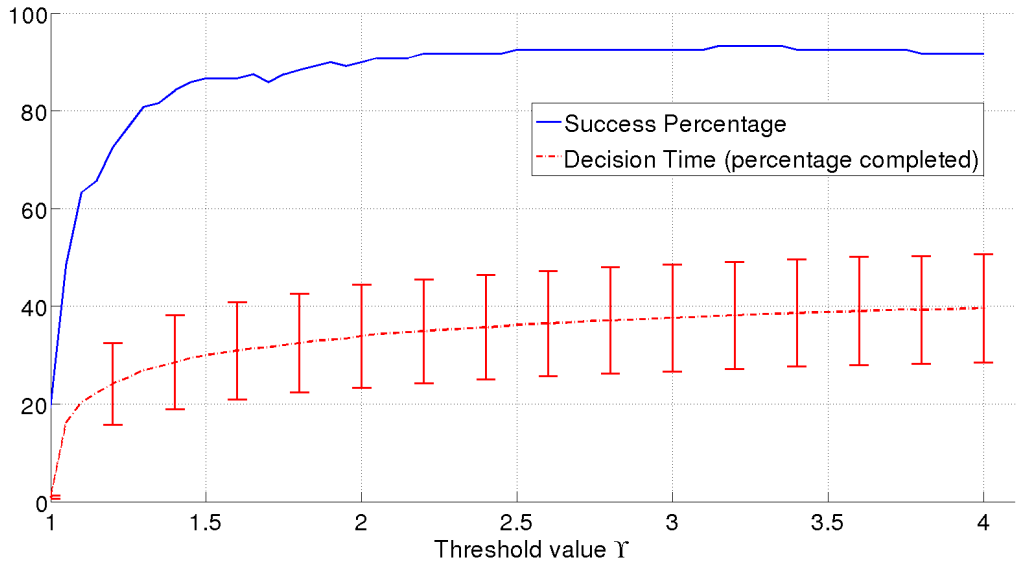


Figure 5.4: Recognition rate and decision time (percentage completed) vs. threshold value ϒ. Percentage completed is defined as decision time divided by action completion time.

success rate and mean decision time increase. At this point, the choice of ϒ depends on the application. For this thesis, ϒ is chosen as 1.9 to obtain 90% recognition rate. On the average, the system makes a decision when 33% of the observed action is completed with this ϒ value.

Histogram of the decision times of all recordings for ϒ = 1.9 are plotted in figure 5.5. This shows the distribution of the correct and the wrong decisions in time. Note that most of the decisions are made before the half of the action is completed.

Table 5.1 shows the confusion matrix for ϒ = 1.9. Cases where the object acted upon is not correctly decided is low: 25% of wrong decisions and 2.5% of all decisions. Since we want to make recognition before the action is completed, there are confusions between reaching right of the left object and reaching left of the right object (See figure 5.2(a)). Also, there is a confusion between reaching directly and reaching from left for the left object. These should be expected since human motion has noise and variance between repetitions and a demonstrator may not give the initial curvature expected from the action every time (see figure 5.2(b)).

5.4 Interactive Game

Recognition system was tested with an interactive game on the aforementioned setup to demonstrate its online capabilities. The interactive game is as follows: Actor applies one

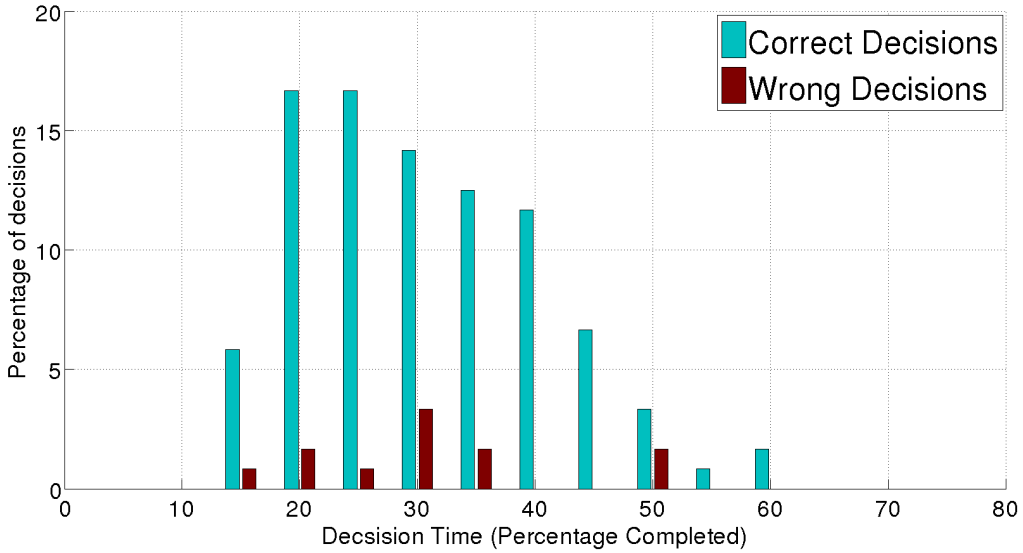


Figure 5.5: Distribution of decision times (percentage completed) for $\Upsilon = 1.9$

Table 5.1: Confusion matrix for $\Upsilon = 1.9$ (Recognition rate = 90%)

	Object	LO			RO		
Object	Behavior	R	D	L	R	D	L
LO	R	17	1	0	0	0	2
	D	0	15	4	0	0	1
	L	0	2	18	0	0	0
RO	R	0	0	0	18	1	1
	D	0	0	0	0	20	0
	A	0	0	0	0	0	20

of the actions to one of the objects. The robot raises its eyebrows and blinks when it recognizes the action and reacts by turning his head to the predicted object. The robot then makes a hand-coded counter action which is defined as bringing its hand on the opposite side of the object. The reason for using hand-coded actions is that action generation is out of the scope of this thesis. DMPs have already been shown to have good generation characteristics.

Snapshots from a captured video of the game can be seen in figure 5.6. This figure shows that the action recognition method can be used for online interaction with robots.

In the interactive game, velocities are calculated in real time. Moreover, position and velocity of the actor's end-effector and positions of the objects are used to understand the beginning and ending of the action as well as to determine whether the actor is ready to start his next action.

For the generation mechanism $K = 10$, $D = 2\sqrt{10} \approx 6.32$ and for the recognition system $\Upsilon = 1.9$ is used.

Figure 5.6: Demonstrations with the robot: Each row shows a different demonstration. The first column shows the starting point of the actions. The second column shows the point where the system recognizes the action (indicated by the eye-blink). The third column is the point where the demonstrator finishes his action and the last column is the point where the robot finishes his action.

CHAPTER 6

CONCLUSION

In this thesis, an online recognition approach is demonstrated which can recognize an action before it is completed. The feasibility of the approach is shown with a real robot in an interactive setting.

The approach uses an action generation mechanism at its core. The generation mechanism is based on DMPs which are modified to overcome some of their shortcomings. They were made entirely closed-loop. One drawback of the modifications is that the stability of the generation mechanism is not proven mathematically.

The architecture allows for imitation learning which is necessary to ensure that the actor and the observer have similar action generation mechanisms. If they do not have similar mechanisms, then the system would not work.

The choice of variables, i.e., the hand-object relations, allow seamless integration of the generation system into the recognition architecture. This solves space of matching problem and the correspondence problem to some extent.

Recognition signals are defined to have a quantitative way to measure similarity of actions. These signals are robust against noise since cumulative error is used in their calculation. They can be interpreted as likelihoods. Recognition signals are similar to mirror neuron system responses, although more experiments need to be done to have a more rigorous claim. The recognition signals are the solution to the matching criterion problem mentioned in the introduction chapter.

One of the crucial but under-emphasized part of this thesis is that any action generation mech-

anism could be used as long as;

- It can take the defined hand-object relations as input.
- It guarantees that the actor and the observer have similar action generation models.
- It is not dependent on internal variables.

The performance and generalization capability of the action generation mechanism has a direct impact on the recognition performance. Improving the action generation part would increase the recognition performance. There are a few things that could be done. Using more training data from a wider work space is a viable option. Different learning and function approximation algorithms for the learning part of DMPs could be tried. An immediate idea is to use a recurrent neural network architecture instead of a feed-forward architecture. Although DMPs have many appealing properties, any other online action generation mechanism could be used. Trying different action generation mechanisms could be a future research direction.

A specialized motion capture system is used to track the objects and the actor's end-effector. This is not very suitable for mobile robotics both because of the limited space on a mobile robot and cost of the system. Implementation of a vision based tracking system is more beneficial for mobile robots.

Current definition of hand-object relations proved to be very useful for reaching actions. However, there are other possibilities that could be tried. In addition, other actions might require other variables to be tracked. For example, grasping would also require to track individual fingertips. Finding better hand-object relations is a possible future research direction. Moreover, discovering relations from training data would be a good approach for developmental robotics.

REFERENCES

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

[2] J. Demiris and G. Hayes, "Imitation as a dual-route process featuring predictive and learning components; a biologically-plausible computational model," in *Dautenhahn, K. and Nehaniv, C., Imitation in animals and artifacts*, MIT Press, 2002.

[3] G. Rizzolatti, L. Fadiga, V. Gallese, and L. Fogassi, "Premotor cortex and the recognition of motor actions," *Cognitive brain research*, vol. 3, pp. 131–141, 1996.

[4] V. Gallese, L. Fadiga, L. Fogassi, and G. Rizzolatti, "Action recognition in the premotor cortex," *Brain*, vol. 119, no. 2, pp. 593–609, 1996.

[5] 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*, vol. 13, pp. 400–4, 2001.

[6] G. Buccino, F. Binkofski, and L. Riggio, "The mirror neuron system and action recognition," *Brain and Language*, vol. 89, pp. 370–376, 2004.

[7] *EU FP7 Project ROSSI*, http://www.rossiproject.eu. Last Visited 22.07.2010.

[8] V. Krüger, 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.

[9] L. Montesano, M. Lopes, A. Bernardino, and J. Santos-Victor, "Learning object affordances: From sensory–motor coordination to imitation," *Robotics, IEEE Transactions on [see also Robotics and Automation, IEEE Transactions on]*, vol. 24, no. 1, pp. 15–26, 2008.

[10] C. Nehaniv and K. Dautenhahn, "The Correspondence Problem," in *Dautenhahn, K. and Nehaniv, C., Imitation in animals and artifacts*, p. 41, The MIT Press, 2002.

[11] M. Iacoboni, R. Woods, M. Brass, and H. Bekkering, "Cortical mechanisms of human imitation," *Science*, vol. 286, no. 5449, pp. 2526–2528, 1999.

[12] G. Buccino, S. Vogt, A. Ritzl, G. Fink, and K. Zilles, "Neural Circuits Underlying Imitation Learning of Hand Actions An Event-Related fMRI Study," *Neuron*, vol. 42, pp. 323–334, 2004.

[13] M. A. Arbib, "From monkey-like action recognition to human language: An evolutionary framework for neurolinguistics," *Behavioral and Brain Sciences*, vol. 28, pp. 105–124, April 2005. discussion pages: 125-167.

[14] S. Schaal, A. Ijspeert, and A. Billard, "Computational approaches to motor learning by imitation," *Philosophical Transactions: Biological Sciences*, vol. 358, pp. 537–547, 2003.

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

[16] 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.

[17] K. Doya, K. Katagiri, D. M. Wolpert, and M. Kawato, "Recognition and imitation of movement patterns by a multiple predictor–controller architecture," *Technical Report IEICE*, pp. 33–40, 2000.

[18] Y. Demiris and M. Johnson, "Distributed, predictive perception of actions: a biologically inspired robotics architecture for imitation and learning," *Connection Science*, vol. 15, pp. 231–243, December 2003.

[19] Y. Demiris and B. Khadhouri, "Hierarchical attentive multiple models for execution and recognition of actions," *Robotics and Autonomous Systems*, vol. 54, pp. 361–369, 2006.

[20] Y. Demiris and G. Simmons, "Perceiving the unusual: Temporal properties of hierarchical motor representations for action perception.," *Neural Networks*, vol. 19, pp. 272–284, 2006.

[21] M. C. Harris and D. M. Wolpert, "Signal-dependent Noise Determines Motor Planning," *Nature*, vol. 394, no. August, pp. 780–784, 1998.

[22] E. Oztop, D. Wolpert, and M. Kawato, "Mental state inference using visual control parameters," *Cognitive Brain Research*, vol. 22, pp. 129–151, 2005.

[23] 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.

[24] 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.

[25] J. Tani, M. Ito, and Y. Sugita, "Self-organization of distributedly represented multiple behavior schemata in a mirror system : reviews of robot experiments using RNNPB," *Neural Networks*, vol. 17, pp. 1273–1289, 2004.

[26] E. Oztop and M. A. Arbib, "Schema design and implementation of the grasp-related mirror neuron system," *Biological Cybernetics*, vol. 87, pp. 116–140, 2002.

[27] J. Bonaiuto, E. Rosta, and M. Arbib, "Extending the mirror neuron system model, I. Audible actions and invisible grasps.," *Biological cybernetics*, vol. 96, pp. 9–38, 2007.

[28] A. Ijspeert, J. Nakanishi, and S. Schaal, "Trajectory formation for imitation with nonlinear dynamical systems," *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180)*, pp. 752–757, 2001.

[29] S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert, "Learning movement primitives," in *International Symposium on Robotics Research*, Springer, 2003.

[30] H. Hoffman, P. Pastor, D.-H. Park, and S. Schaal, "Biologically-inspired dynamical systems for movement generation: Automatic real-time goal adaptation and obstacle avoidance," in *2009 IEEE International Conference on Robotics and Automation*, pp. 2587–2592, May. 2009.

[31] A. Ijspeert, J. Nakanishi, and S. Schaal, "Movement imitation with nonlinear dynamical systems in humanoid robots," in *Proceedings 2002 IEEE International Conference on Robotics and Automation*, pp. 1398–1403, IEEE, 2002.

[32] 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.

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

[34] 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.

[35] 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.

[36] *EU FP7 Project RobotCub*, http://www.robotcub.org. Last visited 22.07.2010.

[37] G. Metta, P. Fitzpatrick, and L. Natale, "Yarp: Yet another robot platform," *International Journal on Advanced on Advanced Robotics Systems 3*, pp. 43–48, 2006.

[38] *YARP*, http://eris.liralab.it/yarpdoc/what_is_yarp.html. Last Visited 22.07.2010.

[39] *Official iCub Portal*, http://www.icub.org. Last Visited 22.07.2010.

[40] *VisualEyez^{TM} Specifications*, http://www.ptiphoenix.com/VZ- models.php. Last Visited 22.07.2010.