RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

School of Computer Science

Internship Project Report

Robot Learning by Demonstration

Author:
Kaushik Subramanian
Rutgers University

Supervisor:
Dr. Gerhard Lakemeyer
Head of Knowledge Based Systems
Group

August 31, 2009

Abstract

In this report, two systems have been developed for robot behavior acquisition using kinesthetic demonstrations. The first enables a humanoid robot to imitate constrained reaching gestures directed towards a target using a learning algorithm based on Gaussian Mixture Regression. The imitation trajectory can be reshaped in order to satisfy the constraints of the task and it can adapt to changes in the initial conditions and to target displacements occurring during the movement execution. The second is focused on behavior learning and walk-gait optimization by simulation using Swarm Intelligence. The fitness of each swarm particle is evaluated using a simulator until the expected behavior is reproduced and then tested on the real robot. The potential of these methods is evaluated using experiments involving Aldebaran's Nao humanoid robot and *Fawkes*, an open source robot software by the KBSG at RWTH University.

Contents

Chapter 1

Introduction

1.1 Motivation

To teach the robot new behaviors without extensive programming.

There are a number of issues that persist in Learning by Imitation, where we are searching for a generic approach to the transfer skills across various agents and situations. These issues have been formulated into a set of generic questions, what-to-imitate, how-to-imitate, when-to-imitate and who-to-imitate. A large body of work [1] focuses on sequencing and decomposing complex tasks into known sets of actions, performable by both the demonstrator and the imitator. A recent approach aims at extracting and encoding low-level features, e.g. primitives of motion in joint space, and makes only weak assumptions as to the form of the primitives or kernels used to encode the motion.

In this work different demonstrations of the same task are performed and two methods are used to extract important aspects of that task - a probabilistically based estimation of relevance and an optimization based estimation. The former provides a continuous representation of the constraints, given by a time-dependent covariance matrix, which can be used to decompose, generalize and reconstruct gestures. The latter explores and exploits the search space using intelligent swarms until the gesture is appropriately reconstructed.

As humanoid robots are endowed with a large number of sensors and actuators, the information contained within the dataset collected by the robot is often redundant and correlated. Through the use of optimization and mixture models, our system finds a suitable representation of the data for continuous data. In our work we use a generic framework which allows for the extraction of a time-dependent continuous representation of the constraints. The model extracts a continuous representation of the constraints with local information on variations and correlations across the variables. It thus provides a localized, efficient, and generic description of the important aspects of the task.

Therefore the question we are trying to answer is how to efficiently teach behaviors to robot systems such that they are able to act in new unseen circumstances with minimal user interaction. To this end, we have designed a system where the user manually controls the robot's end effectors in order to demonstrate the required behavior. This process takes place for various new positions until we are able to arrive at generalized dataset that functions well given new circumstances.

1.2 Contents

Chapter 2 discusses Task Space Learning using Gaussian Mixture Regression. It describes each component of the algorithm along with results presented for 2D mouse task demonstrations and 3D robot demonstrations.

Chapter 3 deals with Behavior Learning using Swarm Intelligence. The algorithm used is based on Particle Swarm Optimization. The general algorithm is presented with adaptations for some real world scenarios.

Chapter 4 gives the Conclusion and Acknowledgements. This followed by an Appendix which contains a list and purpose of the computer programs used for this project.

Chapter 2

Task Space Behavior Learning using GMR

Every behavior that we are trying to teach the robot is broken down in a set of reaching tasks. For example, it can be thought of as a behavior where the robot has to go from state A to B and finally to C. The constraints associated with such a task can be 'how to switch across A, B and C', 'how to ensure the correct order', 'what is time frame involved in changing states' and so on. These constraints are extracted for multiple demonstrations of the same task and then generalized. Taking the constraints into account, the generalized version serves to perform the behavior given new states of A, B and C.

This chapter describes how we perform the input demonstrations for the required behavior and use them to extract the task constraints. The task constraints are generalized using Gaussian Mixture Models [4] and the parameters of which are used to reproduce a trajectory given new positions of the various system objects.

2.1 System Inputs

2.1.1 Kinesthetic Demonstrations

The Nao humanoid robot shown in Fig 2.1 was used to validate our experiments. For behavior acquisition [5], the user decides which end effectors of the robot will be used. This can be the head, right/left arm or right/left leg or a combination of them. The extent of control is limited by the users capabilities to control multiple robot chains at the same time. Once decided, their stiffness is set to a value that allows free movement. This is achieved using *Fawkes* [10], an open source robot software.

Fawkes allows the user to control the various motion, vision and commu-

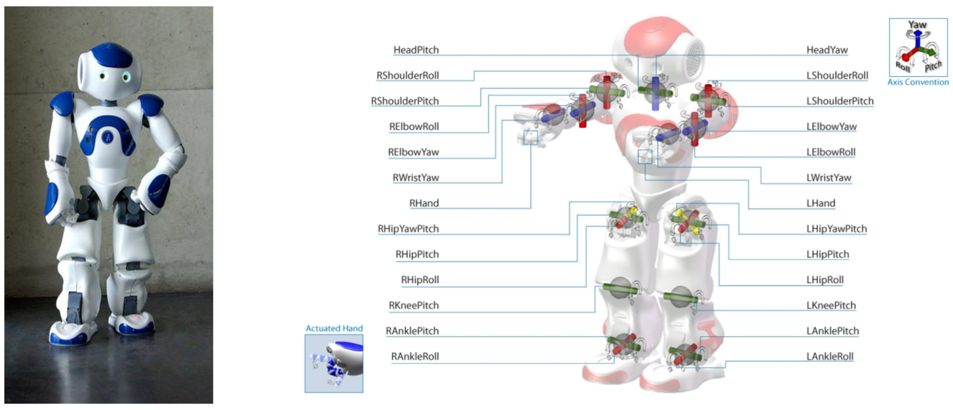
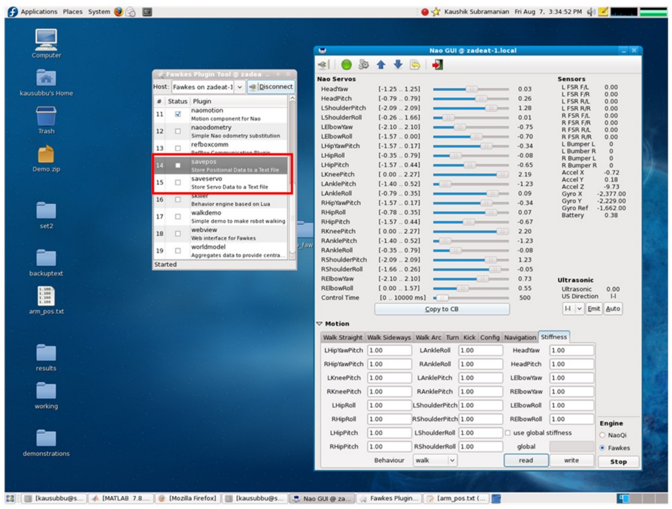


Figure 2.1: Aldebaran's Nao Humanoid Robot with its various Degrees of Freedom

nication components of the robot. It uses plugins to load and unload these components based on our needs. A list and purpose of each program/plugin used has been detailed in the Appendix. With the *naomotion* plugin enabled, we start the *saveservo* plugin and the user manually controls the chain in the way required. For example, the user moves the robotic arm towards a cup and then carries it towards the saucer. During the demonstration, the *saveservo* plugin simply stores the raw servo data of the robot to a text file. In this case it will form a dataset with each row having 22 Dimensions. Having stored the behavior, we run the *savepos* plugin. This re-performs the demonstration and this time it stores a 6D vector that contains the position and orientation of the end effector. A screenshot of the *Fawkes* GUI is shown in Fig 2.2.



Host: Fawkes on zadeat-1 Disconnect

#	Status	Plugin
11	✓	naomotion Motion component for Nao
12	☐	naoodometry Simple Nao odometry substitution
13	☐	refboxcomm RefBox Communication Plugin
14	☐	savepos Store Positional Data to a Text File
15	☐	saveservo Store Servo Data to a Text File
16	☐	skiller Behavior engine based on Lua
17	☐	walkdemo Simple demo to make robot walking
18	☐	webview Web interface for Fawkes
19	☐	worldmodel Aggregates data to provide centra...

Started

Nao Servos

HeadYaw	[-1.25 .. 1.25]	0.03
HeadPitch	[-0.79 .. 0.79]	0.26
LShoulderPitch	[-2.09 .. 2.09]	1.28
LShoulderRoll	[-0.26 .. 1.66]	0.01
LElbowYaw	[-2.10 .. 2.10]	-0.75
LElbowRoll	[-1.57 .. 0.00]	-0.70
LHipYawPitch	[-1.57 .. 0.17]	-0.34
LHipRoll	[-0.35 .. 0.79]	-0.08
LHipPitch	[-1.57 .. 0.44]	-0.65
LKneePitch	[0.00 .. 2.27]	2.19
LAnklePitch	[-1.40 .. 0.52]	-1.23
LAnkleRoll	[-0.79 .. 0.35]	0.09
RHipYawPitch	[-1.57 .. 0.17]	-0.34
RHipRoll	[-0.78 .. 0.35]	0.07
RHipPitch	[-1.57 .. 0.44]	-0.67
RKneePitch	[0.00 .. 2.27]	2.20
RAnklePitch	[-1.40 .. 0.52]	-1.23
RAnkleRoll	[-0.35 .. 0.79]	-0.08
RShoulderPitch	[-2.09 .. 2.09]	1.23
RShoulderRoll	[-1.66 .. 0.26]	-0.05
RElbowYaw	[-2.10 .. 2.10]	0.73
RElbowRoll	[0.00 .. 1.57]	0.55
Control Time	[0 .. 10000 ms]	500

Sensors

L FSR F/L	0.00
L FSR F/R	0.00
L FSR R/L	0.00
L FSR R/R	0.00
R FSR F/L	0.00
R FSR F/R	0.00
R FSR R/L	0.00
R FSR R/R	0.00
L Bumper L	0
L Bumber R	0
R Bumper L	0
R Bumper R	0
Accel X	-0.72
Accel Y	0.18
Accel Z	-9.73
Gyro X	-2,377.00
Gyro Y	-2,229.00
Gyro Ref	-1,662.00
Battery	0.38

Ultrasonic

Ultrasonic	0.00
US Direction	l-l

Motion

Walk Straight | Walk Sideways | Walk Arc | Turn | Kick | Config | Navigation | Stiffness

LHipYawPitch	1.00	LAnkleRoll	1.00	HeadYaw	1.00
RHipYawPitch	1.00	RAnkleRoll	1.00	HeadPitch	1.00
LKneePitch	1.00	LAnklePitch	1.00	LElbowYaw	1.00
RKneePitch	1.00	RAnklePitch	1.00	RElbowYaw	1.00
LHipRoll	1.00	LShoulderPitch	1.00	LElbowRoll	1.00
RHipRoll	1.00	RShoulderPitch	1.00	RElbowRoll	1.00
LHipPitch	1.00	LShoulderRoll	1.00	☐ use global stiffness	
RHipPitch	1.00	RShoulderRoll	1.00	global	

Behaviour: walk

Engine: ○ NaoQi [● Fawkes]

Figure 2.2: Screenshot of *Fawkes* GUI to enable plugins and control stiffness

2.1.2 Task Constraints

Given a behavior, it is necessary for us to describe the constraints [3] associated with it in order to better reproduce the task. Take for example, the mouse task shown in Fig 2.3. The question is how do we define constraints

Figure 2.3: Mouse task - Mouse starting from random position has go to the File and move it to the Bin

for such a task. This can be achieved by calculating the relative positions of the mouse with respect to the file and bin. Consider a set of 4 demonstrations shown in Fig 2.4. These demonstrations were given in an incremental manner [6]. The highlighted portions show the different positions of the

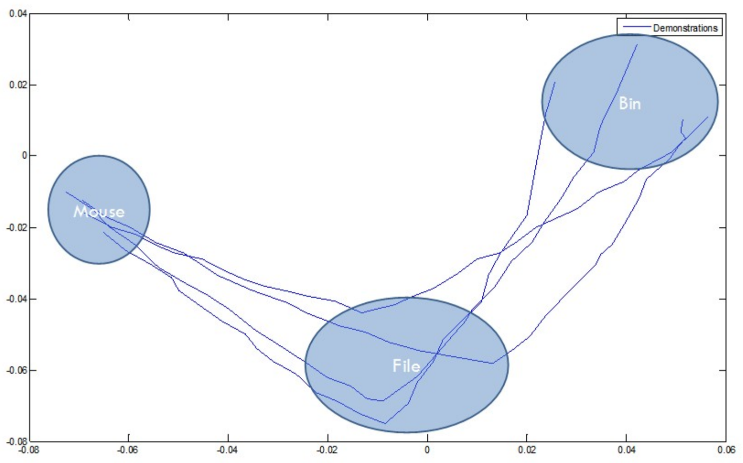
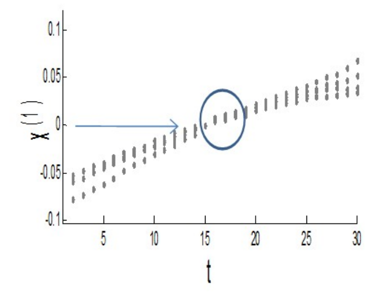
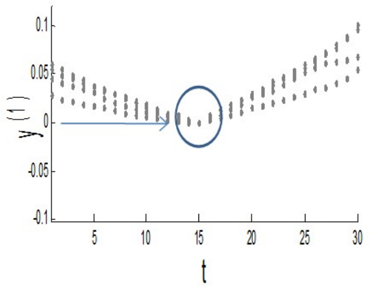


Figure 2.4: A set of 4 demonstrations of the Mouse Task with the positions of the Mouse, File and Bin highlighted.

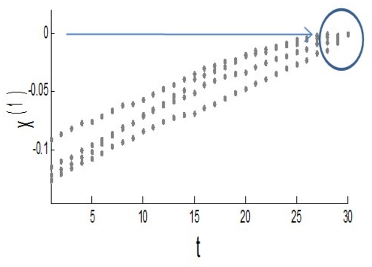
mouse, file and bin. Given this dataset, we calculate the relative distance to extract the constraints. From the demonstrations, we can see that it is necessary for them to be close to each other as the degree of variability is limited. The constraints are shown in Fig 2.5. The highlighted sections show us the sample points where the mouse reaches the file and bin respectively. Therefore at these points the x,y distance coordinates are zero. This Mouse



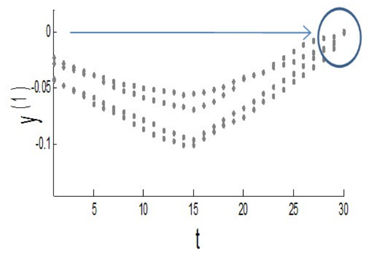
(a) X coordinate position of Mouse relative to the File



(b) Y coordinate position of Mouse relative to the File



(c) X coordinate position of Mouse relative to the Bin



(d) Y coordinate position of Mouse relative to the Bin

Figure 2.5: Task Constraints for the Mouse Task

task can be extended to a number of scenarios and domains, like robot soccer, obstacle avoidance etc. Given these constraints, we sample them and generalize them using Gaussian Mixture Models.

2.2 Gaussian Mixture Regression

A probabilistic representation of the sampled, temporally aligned constraint data is used to estimate the variations and correlations across the variables, allowing a localized characterization of the different parts of the gesture. Mixture modeling is a popular approach for density approximation of continuous data. A mixture model of K components is defined by a probability density function [3]:

$$p(\epsilon_j) = \sum_{k=1}^{K} p(k) p(\epsilon_j|k) \tag{2.1}$$

where ϵ_j is the input dataset, K is the number of Gaussian Components, $p(k)$ the Priors and $p(\epsilon_j|k)$ is the Gaussian function represented by

$$\frac{1}{\sqrt{(2\pi)^D|\Sigma_k|}} e^{-1/2((\epsilon_j-\mu_k)^T\Sigma_k^{-1}(\epsilon_j-\mu_k))}$$

From the above equation, we need to calculate the mean μ_k, covariance matrices Σ_k and priors $p(k)$.

Maximum Likelihood Estimation of the mixture parameters is performed iteratively using the standard Expectation- Maximization (EM) algorithm. EM is a simple local search technique that guarantees monotone increase of the likelihood of the training set during optimization. The algorithm requires an initial estimate, and to avoid getting trapped into a poor local minima a rough k-means clustering technique is first applied to the data. The Gaussian parameters are then derived from the clusters found by k-means. This can be shown in Fig 2.6. Having estimated the Gaussian Parameters, Fig 2.7, the next step is to reproduce new trajectories. We perform Gaussian Mixture Regression for this purpose.

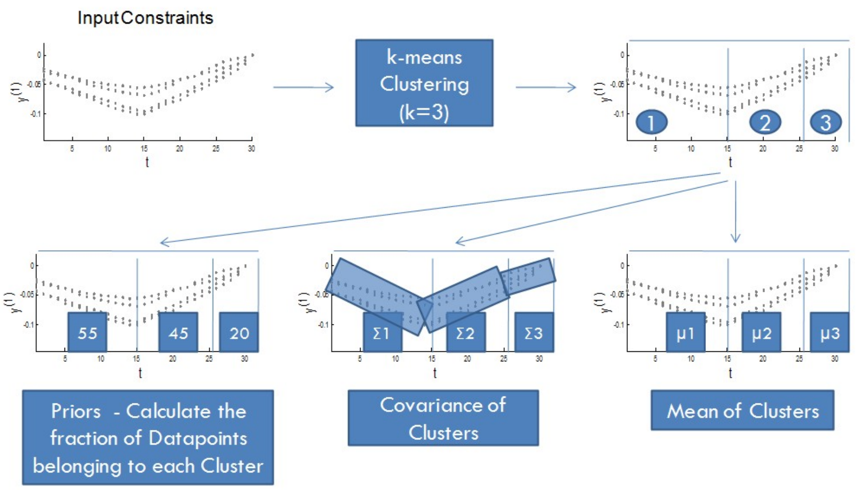


Figure 2.6: Extracting the Mean, Covariance, Priors (GMM Parameters) from the Task Constraints

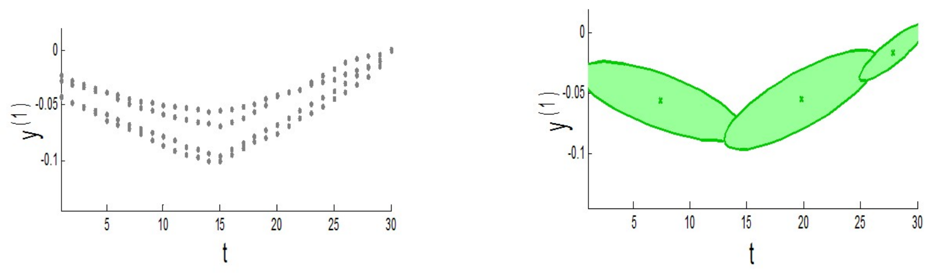
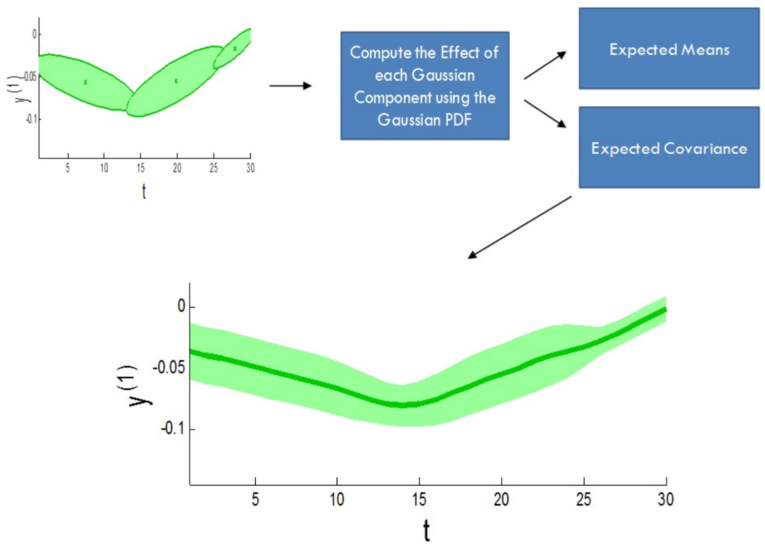


Figure 2.7: Task Constraints on the left have been generalized and represented as Gaussian Models on the right

The basis of Regression is to estimate the conditional expectation of Y given X on the basis of a set of observations (X, Y). Consecutive temporal values are used as query points and the corresponding spatial values are estimated through regression. In our case, X Time sample, Y Cartesian coordinates [2] and [3]. Therefore given time samples as input data, the Regression algorithm outputs a smooth generalized version of the observed trajectories encoded in the GMM shown in Fig 2.8. It should be noted that it is not equivalent to taking the mean and variance of the data at each time step, which would produce jerky trajectories and increase dramatically the amount of parameters (the mean and variance values would be kept in memory for each time step). With a probabilistic model, only the means and covariance matrices of the Gaussians are kept in memory.



Compute the Effect of each Gaussian Component using the Gaussian PDF
Expected Means
Expected Covariance
y(1)
t

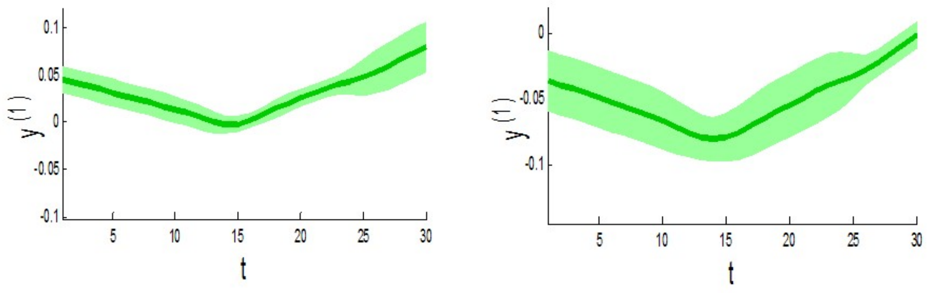
Figure 2.8: Regressed Model of the Generalized data

2.3 Trajectory Reproduction

The Regressed Model that has been obtained is the generalized version of the Relative positions of the File and Bin with respect to the Mouse (Task Constraints). This is shown in Fig 2.9. Given new positions of the Mouse, File and Bin, we must reconstruct the required trajectory. Using the GMM parameters, the reconstruction [7] is performed using the equation below [3]:

$$x_{j+1}^{(n)} = (o^{(n)} + \hat{x}_{j+1}^{(n)}) - x_j \tag{2.2}$$

where j represents the temporal steps, x_j is the position of the robot end effector at step j, $o^{(n)}$ is the position of the object and n is the number of system objects.



(a) GMR in Task Space, relative to File (b) GMR in Task Space, relative to Bin

Figure 2.9: Gaussian Mixture Regressed representations with respect to System Objects

2.4 Experiments and Results

After performing the steps mentioned in the previous sections, we validate our algorithm using 2D Mouse task and 3D robot experiments.

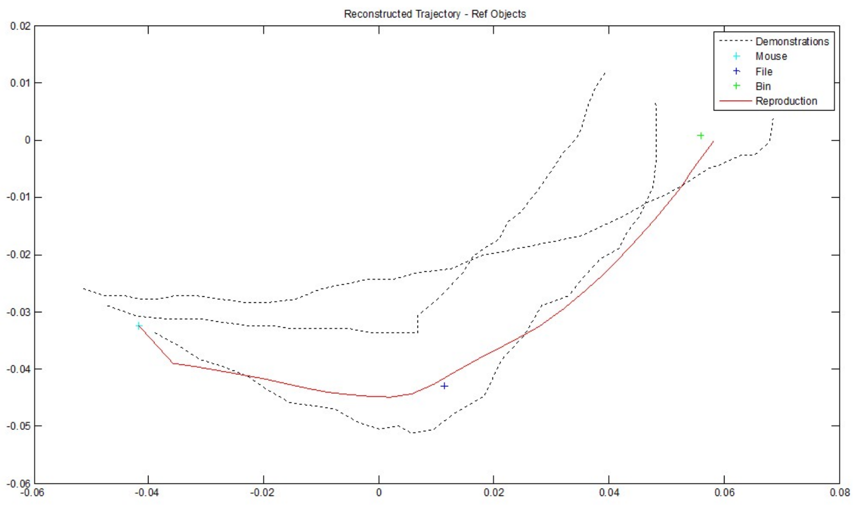


Figure 2.10: This experiment contains a set of 3 demonstrations of the mouse task. The red line indicates the reproduced trajectory for new positions of the mouse, file and bin (indicated by the + signs). We can see the system has produced a acceptable trajectory which passes very close to the new positions.

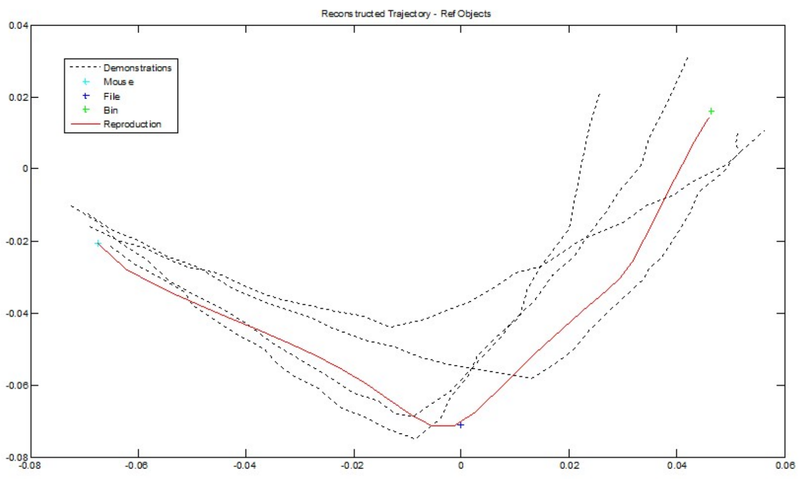


Figure 2.11: Results for a mouse task with 4 demonstrations and new starting positions.

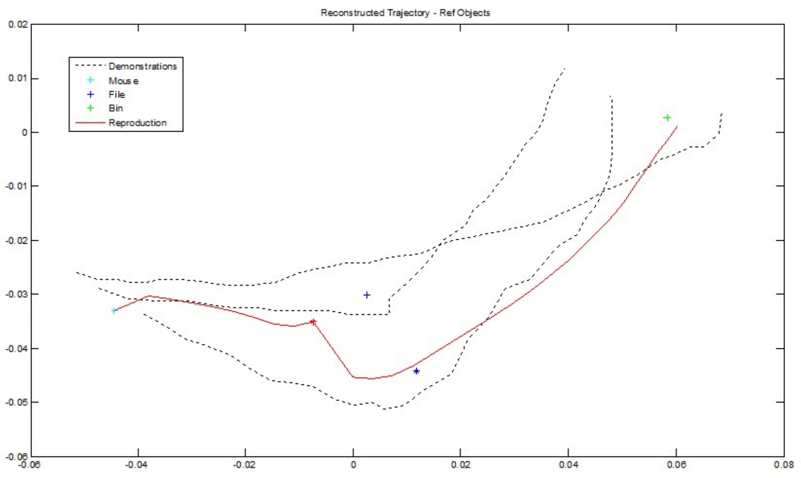


Figure 2.12: Shows the adaptability of the system to changes in the positions during execution of the trajectory. By changing the position of the file, the system is able to account for the new position.

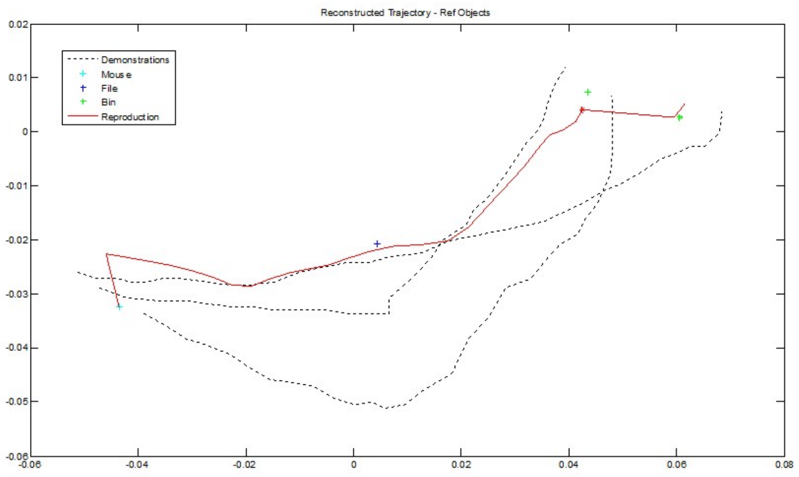


Figure 2.13: By changing the position of the bin in the very last time step, the system is able to account for the new position.

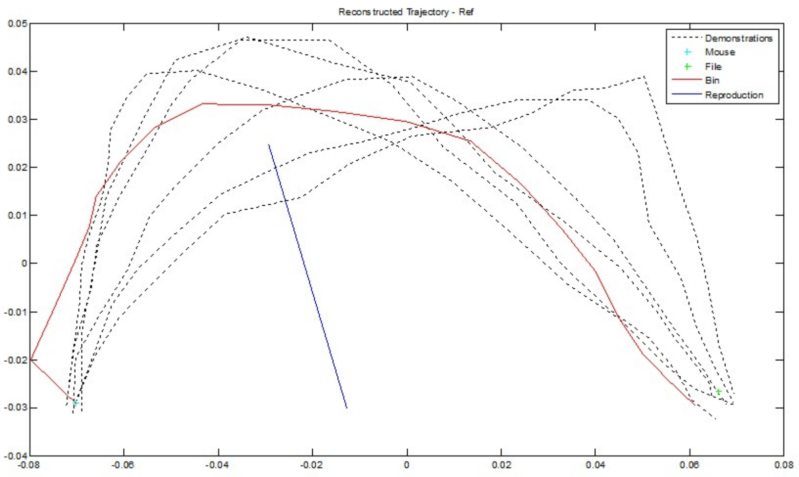


Figure 2.14: A 2D Obstacle Avoidance Task

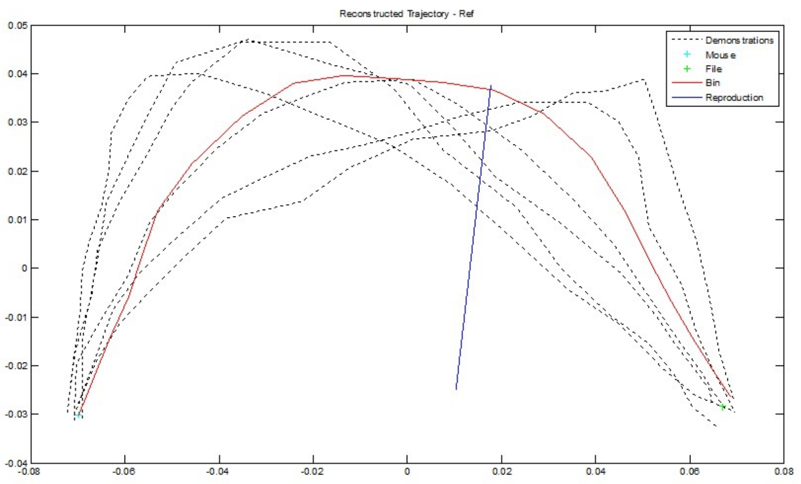


Figure 2.15: A 2D Obstacle Avoidance Task with a new Obstacle height and angle

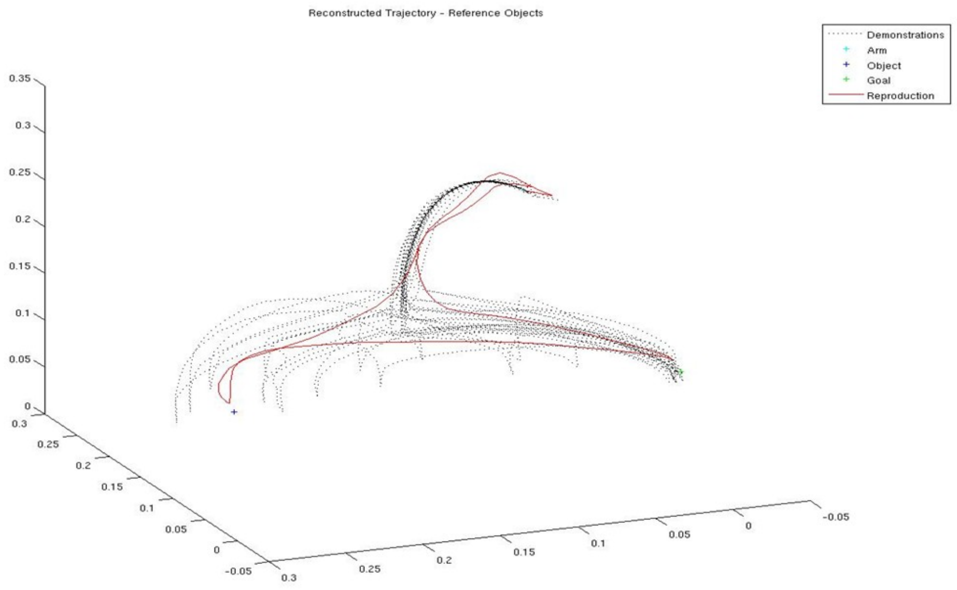


Figure 2.16: A 3D demonstration with the Nao of an arm to object to goal task.

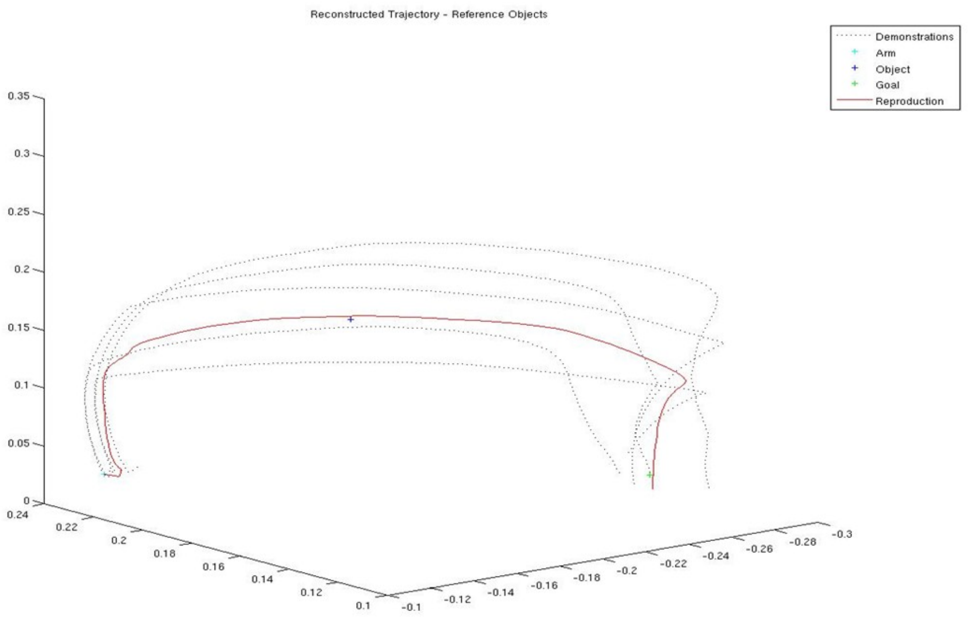


Figure 2.17: Obstacle Avoidance for the Nao after training it for 5,8,12 and 15cm heights. It was tested with 10cm

2.4.1 Observations

Choice of Gaussian Components

From the Fig 2.18 and Fig 2.19, we see that when choosing K=3, the regressed model becomes to narrow and constricted and therefore may not lead to accurate results in all positions. However, changing K=2 produces a regressed model that accounts for those demonstrations also.

Choice of Demonstrations

It is important to keep the demonstrations similar or reasonably close to each other, otherwise the GMM will not be able to generalize the dataset if there is large variation shown in Fig 2.20. If such a gap exists, more demonstrations should be performed focusing on those areas.

2.5 Limitations

There are four main limitations associated with the present implementation of the algorithm -

Task Space Constraints alone do not satisfy the Accuracy required. This

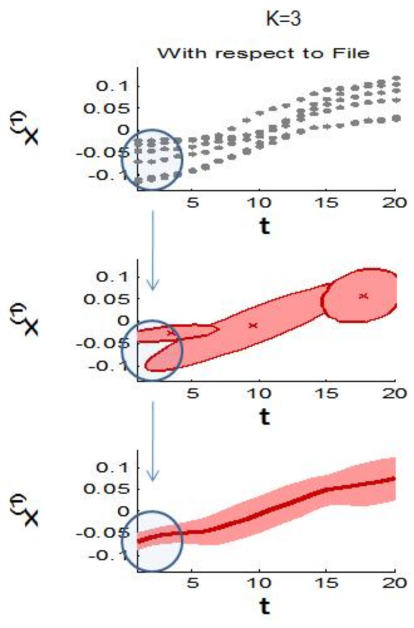


Figure 2.18: GMR in Task Space, relative to File.

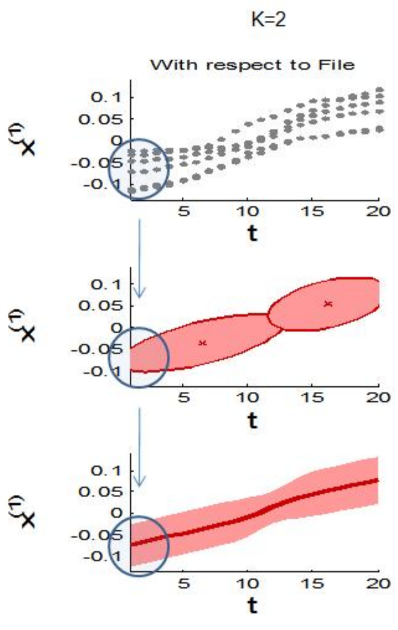


Figure 2.19: GMR in Task Space, relative to Bin.

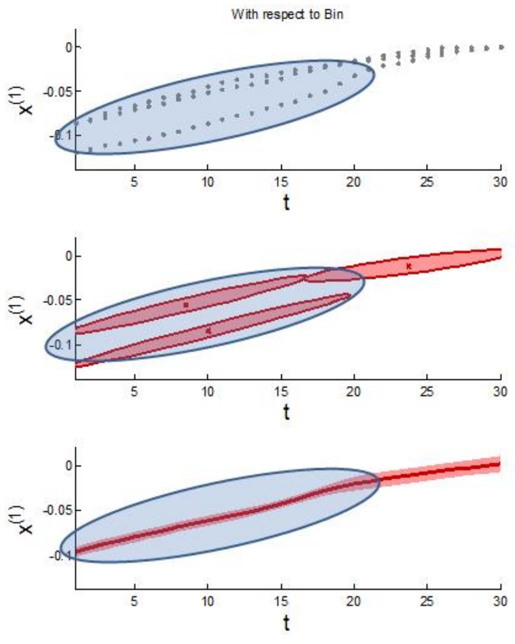


Figure 2.20: 2D Demonstrations with new Positions

means that it is not enough to only generalize the coordinate positions of the objects, better accuracy can be acquired by taking the joint space constraints in account.

Every Object in the System has to be defined by 6 Dimensions. As we are using Task Space constraints and working with the Nao robot, every object needs a 3D position and 3D orientation. This maybe to difficult to calculate and the present solution seems to be to manually take the arm/leg to destination position and register the coordinates and orientation.

A discrete trajectory is reproduced which may contain sharp turns. Less frequently, the algorithm tries to follow the path of the demonstrations already shown, in this attempt it sometimes creates sharp turns and jerky movements. This may harm the servo motors of the robot.

Offline Processing. In the present state, the object positions and other coordinates is acquired offline and not online via the camera module. This is yet to be integrated.

Overcome the Limitations

Use of Forward and Inverse Kinematics Model allows you to employ Joint Constraints and do away with 6D object coordinates.

Use of a Vision System allows positions to be automatically registered.

Chapter 3

Behavior Learning via Simulation using Swarm Intelligence

3.1 Introduction

This chapter briefly describes an imitation learning system based on kinesthetic interactions between a human and a robot and the use of Particle Swarm Optimization as a learning technique. This type of interaction is supported by a simulation engine (Webots). The demonstrations with the robot are used to create a low-dimensional posture space from which control points are acquired. These control points are used to define the behavior and this allows for fast imitation learning.

3.2 Behavioral Control Points

Suppose there is a new behavior to be learnt like kicking at an angle, getting up while lying down on the back, strafing etc It is difficult to manually hard-code such a behavior. To overcome this, the user kinesthetically demonstrates it as shown in Chapter 2, manually control the end effectors as desired and having acquired the raw servo data, we convert it to a low-dimensional posture space using Principal Component Analysis.

3.2.1 Principal Component Analysis

PCA [10] is a multivariate procedure which rotates the data such that maximum variabilities are projected onto the axes. It is used to reduce the dimensionality of a data set while retaining as much information as is possible. Set of correlated variables are transformed into a set of uncorrelated variables which are ordered by reducing variability. These uncorrelated variables are linear combinations of the original variables, and the last of these variables can be removed with minimum loss of real data. Therefore it computes a compact and optimal description of the data set. We use the Eigen-decomposition method to acquire the principal components. After performing the necessary computations, a plot of first and second principal components in shown in Fig 3.1. Having acquired the components, we select

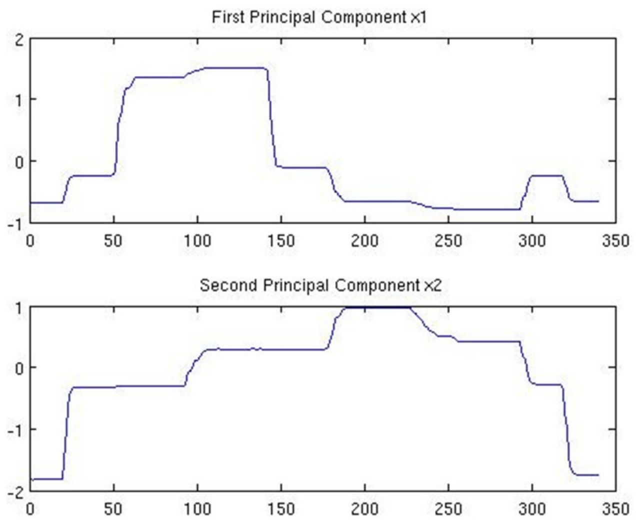


Figure 3.1: Plot of first and second Principal Components

control points [8] from the 3D plot of the components along with the time axis. Specific points are selected from this Trajectory as points which show a change in direction. This is shown in Fig 3.2. Now therefore the entire behavior can be described by using these Control Points alone. It is with these points that the optimization learning algorithm is initiated.

3.3 Particle Swarm Optimization

Particle Swarm Optimization[11] is a stochastic, population-based computer algorithm modeled on swarm intelligence that finds a solution to an optimization problem in a search space in the presence of objectives. A problem is given, and some way to evaluate a proposed solution to it exists in the form of a fitness function. A population of individuals known as the particles is defined randomly and they behave as potential solutions. The particles iteratively evaluate the fitness of the candidate solutions and remember the location where they had their best success. The swarm is typically modeled by particles in multidimensional space that have a position and a veloc-

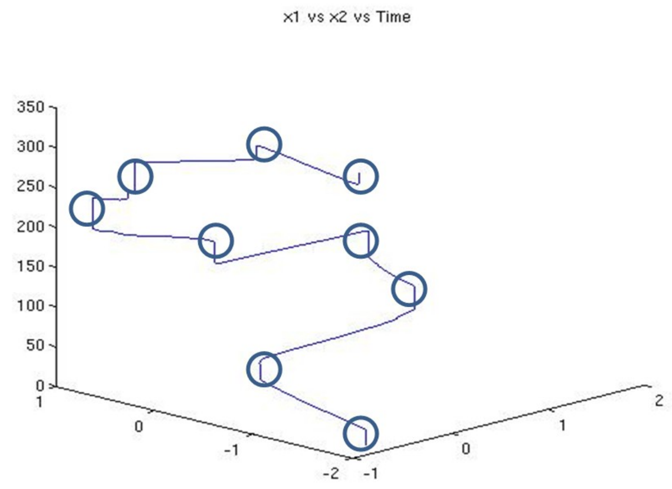


Figure 3.2: 3D plot of components and time axis along with selected control points

ity. These particles fly through hyperspace and have two essential reasoning capabilities: their memory of their own best position and knowledge of the global best. Members of a swarm communicate these positions to each other and adjust their own position and velocity based on these good positions.

So a particle has the following information to make a suitable change in its position and velocity: A global best that is known to all, the local best and an update of the particle position and velocity. A simple representation is shown in Fig 3.3. The blue sphere has a random position and velocity and it is evaluating its fitness to try to get to the red sphere and maintaining its best fitness and the global fitness. In our case, each particle is initiated with random perturbations of the control points and with random velocities. Iteratively these values are updates, their fitness value assigned until they all converge to a global best.

3.4 Fitness Evaluation using Webots Simulator

In order to assess the fitness of a particle, we use the Webots Simulator. The Nao template allows us to provide raw servo data and the simulator robot reacts accordingly. At each iterative step, for every particle, the control points are reversed to obtain the raw servo data. This is done by interpolation followed by inverse PCA to acquire the original data space. This is then fed into the simulator program which then performs the behavior. Depending on how well it performs the behavior, the user assigns a fitness value.

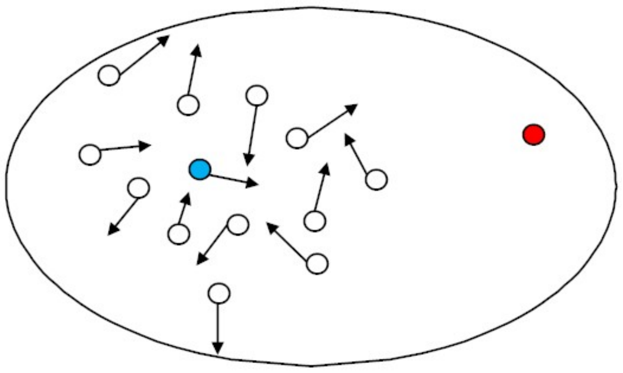


Figure 3.3: Representation of a particle with random positions and velocities exploring and exploiting for the optimized solution

For example, if the behavior is to stand up while lying down, the fitness value can be assigned as the height of the head. Higher the head, greater the fitness, if the Nao falls down a fitness of zero is given. We tested with an experimental roll-over behavior. We demonstrated a task where the Nao

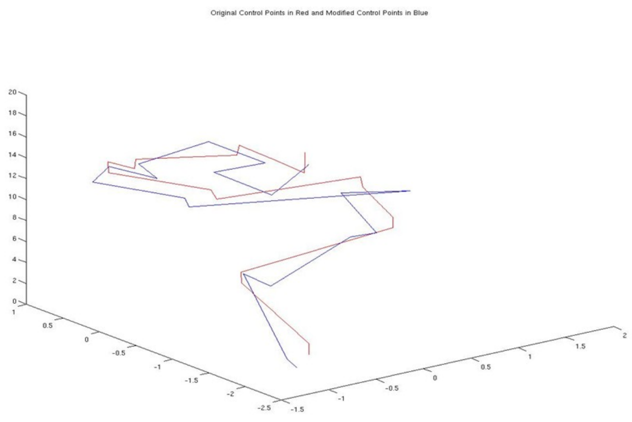


Figure 3.4: Modified Control Points after optimizing

had to roll over from its back to lie on its front. First we played the demonstrated behavior in the simulator and it was clear that the Nao was unable to reproduce the task. We then initialized 3 particles and 10 iterations and thus performed 30 simulations, every step we assigned fitness values. At the end, we acquired the modified control points shown in Fig 3.4. These were then transferred to the real Nao and we find that it was able to learn to roll-over. Overall this method may at first appear to be a random search but given the right number of particles and iterations, it forms a balance between Exploration and Exploitation.

3.5 Extension to Walk Gait Optimization

For all humanoid robots, the most important behavior is the ability to walk accurately and efficiently. After much research, several parameters have been defined that control the basic walk of the Humanoid. These are referred to as the walk parameters [12], which are [Step-Length, Step-Height, Step-Turn, Hip-Compensation, ZMP-Forward...].

Given our algorithm, particles can be initialized with random perturbations of these values and the swarm optimization can be started. For the Fitness Value, each Particles parameters are tested on the real robot and the fitness is given either by the distance covered or the time a set distance has been covered. This experiment was not implemented, but the algorithm indicates it may result in positive results.

Chapter 4

Conclusion

In the project, two methods were presented.

The first is a probabilistic framework that automatically extracts the essential features characterizing a skill by handling constraints in task space. The method was validated in three experiments in which the Nao robot was taught simple behavioral tasks through kinesthetics. This allows the user to embody the robot's body and thus, this way, the correspondence problem can be simplified. It was then demonstrated that the GMM approach could be applied successfully to learn generically new behavioral skills at a trajectory level by generalizing over several demonstrations and by extending the learned skills to new positions of objects. Further work will focus on building a generic forward and inverse kinematics model that would allow Joint Space Constraints to be accounted for during trajectory reproduction. The second improvement is focussed on making use of a Vision System allows positions of system objects to be automatically registered. An additional improvement could involve the use of dimensionality reduction techniques such as Principal Component Analysis to pre-process the data, and to extend the metric and associated optimization paradigm in this latent space of lower dimensionality.

The second method demonstrated an optimization technique which if properly supported by robot simulators allows the robot to learn behaviors. A preprocessing step using PCA was performed which is followed by extraction of control points. The PSO algorithm was initiated with these points and with the help of the Webots simulator, the behavior learning simple experiments showed positive results. The algorithm can be extended to a variety of tasks. Further improvements will focus on speeding up the algorithm over traditional approaches. Although the algorithm takes a lot of time, it depends on the users requirement for exploration and exploitation.

Acknowledgements

Firstly, I would like to thank Prof. Gerhard Lakemeyer for inviting me to work in the KBSG lab at RWTH, Germany, for giving me the opportunity to make use of the advanced resources of his lab. His continuous support and encouragement were great help during my stay. I would also like to thank all my colleagues for the support and friendship that they offered me during my stay, Tim Niemueller for patiently teaching me the basics of *Fawkes*, robotics and software programming. He was very patient and answered all my questions, Masrur Doostdar for taking a keen interest in my work, we would engage in discussions which often led to new ideas for my research. I would like to express my gratitude towards Daniel Beck and Stefan Schiffer for their support and interest in my research. Furthermore I would like to thank Vaishak Belle and Sukanya Krishnamurthy for helping me comfortably settle down and make me feel at home in Germany. Thanks go out to my family and friends for encouraging me and allowing me to pursue my interests. I would also like to thank Prof. Michael Littman and members of the RL3 Lab at Rutgers University. The experience I gained there played a very important role in the progress of my internship.

Bibliography

[1] A. Billard and R. Siegwart, *Robot learning from demonstration*, Robotics and Autonomous Systems, vol. 47, no. 2-3, pp. 65.67, 2004.

[2] S. Calinon, F. Guenter, and A. Billard, *Goal-directed imitation in a humanoid robot*, in Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2005.

[3] Sylvain Calinon and Aude Billard, *A Probabilistic Programming by Demonstration Framework Handling Constraints in Joint Space and Task Space*, in proceedings of IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS), 2008.

[4] S. Calinon, F. Guenter, and A. Billard, *On learning the statistical representation of a task and generalizing it to various contexts*, in Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2006.

[5] S. Calinon and A. Billard, *What is the teacher's role in robot programming by demonstration? - Toward benchmarks for improved learning,* Interaction Studies, vol. 8, no. 3, pp. 441.464, 2007.

[6] S. Calinon and A. Billard, *Incremental learning of gestures by imitation in a humanoid robot*, in Proc. ACM/IEEE Intl Conf. on Human-Robot Interaction (HRI), March 2007, pp. 255.262.

[7] Hersch, M., Guenter, F., Calinon, S., Billard, A, *Dynamical system modulation for robot learning via kinesthetic demonstrations*, IEEE Trans. on Robotics (2008)

[8] Erik Berger, Heni Ben Amor, David Vogt, Bernhard Jung, *Towards a Simulator for Imitation Learning with Kinesthetic Bootstrapping*, Workshop Proceedings of SIMPAR 2008, Venice(Italy) 2008 November,3-4 ISBN 978-88-95872-01-8 pp. 167-173

[9] Tim Niemueller, *Developing A Behavior Engine for the Fawkes Robot-Control Software and its Adaptation to the Humanoid Platform Nao*, diploma thesis submitted to Computer Science Department, RWTH Aachen, Germany, 2009.

[10] Jonathon Shlens, *A Tutorial on Principal Component Analysis*, Center for Neural Science, New York University New York City, NY 10003-6603 and Systems Neurobiology Laboratory, Salk Insitute for Biological Studies La Jolla, CA 92037.

[11] J. Kennedy and R. C. Eberhart, *Particle swarm optimization*, vol. 4, 1995, pp. 19421948 vol.4.

[12] Cord Niehaus, Thomas Rofer, Tim Laue, *Gait Optimization on a Humanoid Robot using Particle Swarm Optimization*, in Proc. of Second Workshop on Humanoid Soccer Robots, IEEE-RAS International Conference on Humanoid Robots, 2007.

Appendix A

Computer programs

saveservo

A *Fawkes* plugin. It is used to store the servo data of the Nao robot to a text file in the *tmp* folder on the robot. When the user is performing the kinesthetic demonstrations, this plugin is switched on and it is turned off after the demonstration is done.

savepos

A *Fawkes* plugin. It is used to store the position and orientation of the end effector. After the user has stored the servo data, this plugin is switched on. It performs the same demonstration again but this time stores a 6D vector to a text file in the *tmp* folder.

walkdemo

A *Fawkes* plugin. It is used to perform the demonstration as described by a text file in the *tmp* folder. The plugin reads the servo data from the text file and converts it to actuator actions on the Nao.

gmm-2d

A *Matlab* program. It implements the GMM algorithm for the 2D mouse task example. It contains sub programs that allow the user to provide demonstrations, generalize the data and reconstructs the trajectory.

gmm-nao

A *Matlab* program. It implements the GMM algorithm for the Nao. It contains sub programs that read the text file containing the position and orientation, generalizes the data and reconstructs the trajectory.

pso-nao

A *Matlab* program. It performs multi-dimensional particle swarm optimization for Nao behavior learning. It contains sub programs that read text files, perform PCA and writes files to be read by the simulator.