Movement Skill Acquisition using Imitation and Reinforcement Learning

AFFAN PERVEZ

KTH Computer Science
and Communication

Master of Science Thesis
Stockholm, Sweden 2013

Movement Skill Acquisition using Imitation and Reinforcement Learning

A F F A N P E R V E Z

DD221X, Master's Thesis in Computer Science (30 ECTS credits)
Master Programme in Machine Learning 120 credits
Royal Institute of Technology year 2013
Supervisor at CSC was John Folkesson
Examiner was Danica Kragic

TRITA-CSC-E 2013:012
ISRN-KTH/CSC/E--13/012--SE
ISSN-1653-5715

Royal Institute of Technology
School of Computer Science and Communication

KTH CSC
SE-100 44 Stockholm, Sweden

URL: www.kth.se/csc

Abstract

This thesis focuses on having a robot learn to play the game of darts. Playing darts involves multiple tasks. For e.g. how to throw a dart and where to target on the board. It has been shown that with a few demonstrations by the user, the robot can learn how to produce trajectories for hitting a given point on the board; with improvement in accuracy along with the experience. On the other hand we showed how a robot can discover regions for hitting on the board so that it can maximize its expected score.

Contents

List of Figures

Chapter 1

Introduction

It is quoted by Calinon "While accuracy and speed have for a long time been top of the agenda for robot design and control, the development of new actuators and control architectures is now bringing a new focus on compliant robots that are safe for the user to interact with, bringing new perspectives in human-robot collaboration" [25].

As is further stated in [26] "The machine learning tools that have been developed for precise reproduction of reference trajectories need to be re-thought and adapted to these new challenges. For planning, storing, controlling, predicting or re-using motion data, the encoding of a robot skill goes beyond its representation as a single reference trajectory that needs to be tracked or set of points that needs to be reached. Instead, other sources of information need to be considered, such as the local variation and correlation in the movement. Most of the machine learning tools developed so far are decomposed into an offline model estimation phase and a reproduction phase. Instead, learning in compliant robots should view demonstration and reproduction as an interlaced process that can combine both imitation and reinforcement learning strategies to incrementally refine the task".

This project studies the link between learning from demonstration and learning by exploration. The skill (of throwing dart at some particular point on the board) is represented by Gaussian Mixture Model (GMM). GMM are being combined with dynamical systems (the robot is moved as if it was pulled by a combination of virtual spring-damper systems) for encoding attractor point to generate movement trajectories. GMM are used for learning joint probability of input and output variables using Expectation Maximization. Gaussian Mixture Regression (GMR) is then used for retrieving output distribution for a given input variable.

This idea can then be extended by viewing a demonstrated trajectory with respect to multiple frames of references. The role of the robot is to autonomously figure out which frame of reference matters along the task.

This behavior is studied in the context of a dart playing game, with a dart-board having unordered scores ranging from 1 to 20 and areas to double and triple the score (official dartboard). It has previously been shown that the target that one

should aim at differs depending on the skill of the player [5]. The goal of this work is to discover this target point by using samples from the board. The challenge lies in the fact that this target point can change along with the accuracy of the player as a player is less accurate when he/she starts playing the game for the first time but the skill level progressively increases with the experience. Also there may be multiple options (target points). A method for multi optima search has been proposed where agent/player starts with a simple solution and the solution space gradually changes with the accuracy of the player.

Two experiments have been conducted. One is simulation based in which samples drawn from the dart board are used to discover multiple solutions. Second experiment is performed on Barret WAM (7 DOF robotic arm) where it learned to throw ball in a basket after few initial demonstrations from the teacher.

1.1 Report outline

Chapter 2 presents literature reviewed and current state of the art.

Chapter 3 mentions different strategies for robot Programming by Demonstration (PbD).

Chapter 4 overviews a brief survey of Reinforcement Learningn (RL).

Chapter 5 presents the Multi Optima search strategy using adaptive Gaussian mixture models.

Chapter 6 discusses a simulation based result of using the Multi Optima search in the game of Darts.

Chapter 7 shows different approaches of movement skill acquisition using dynamical systems.

Chapter 8 discusses results of simulation and experiments using statistical dynamical systems.

Chapter 9 concludes the report and mention recommendations for future work.

Chapter 2

Background

The literature reviewed during this work is for addressing two problems. Firstly how a player can maximize his/her score by discovering/tracking multiple regions during the game of dart. Secondly how to generate motion trajectories for a robotic arm to hit targets using Imitation and Reinforcement learning.

For maximizing a player score Tibshirani [5] showed how players can obtain personalized heat maps by modeling throw precision with normal distribution. He provided an analytical way by using convolution for calculating target point in static scenarios (maximum in heat map) while earlier work focused on using Monte Carlo methods [7, 8]. This work used these heat maps as ground truth for proposing MultiOptima search. Although darts is considered as potential application, Multi-Optima search is not limited to this domain and can be applied to similar scenarios where we have a reward/score to maximize. MultiOptima search can discover multiple optimal regions using sample from heatmap (reward landscape) and provides continuous exploration/exploitation strategy. In darts the throwing skill of a player changes continuously, making policy land-scape dynamic and raising need of continuous learning.

Different approaches for searching a solution landscape are studied which include Covariance Matrix Adaptation Evolution Strategy (CMA-ES) (Evolutionary search) [1], Cross Entropy (CE) method (optimization strategy) [2] and Policy learning by Weighting Exploration with the Returns (PoWER) (Reinforcemet learning) [3]. For MultiOptima search PoWER is used along with the GMM components adaptation strategy proposed by Zhang [4]. It is shown that MultiOptima approach has a great advantage in dynamic environment where optimal solution can switch discontinuously, making it undetectable by tracking single optima. Data Spectroscopy (DaSpec) [9] which is a non-parametric spectral clustering procedure for initializing a GMM using Eigenspaces of convolution operator is also reviewed and can be used instead of [4].

For movement skill acquisition two different approaches using dynamical systems and Gaussian Mixture Models have been studied. Ijspeert [10] first proposed the idea of using dynamical systems for representing control policies. Most of the

initial work using dynamical systems was concentrated on imitation learning without subsequent improvement [11, 12, 13] with exception from [14, 15]. Kober [3] introduced PoWER which is an Expectation Maximization (EM) [6] based reinforcement learning algorithm and showed that both single-stroke movement and rhythmic movements can be learned using Dynamical Systems.

In [16] Billard and Calinon showed the use of GMM for encoding motion data and provided a way for retrieving it using conditional properties of normal distribution for Gaussian Mixture Regression (GMR). By using GMM we not only get the mean trajectory but we also get the information about the variability allowed during execution of the task [17]. The dynamical systems approach was later combined with GMM, resulting in a DS-GMR model [18] for imitating movements. Motion is modeled as if it is generated by an attractor point pulling object/end-effector using spring damper system. GMM is then used to encode motion of attractor point instead of modeling motion of the object directly. Motion data is recorded from different frame of references and combined in an optimal way to get the resultant GMM. We have used the methodology proposed by [18] for learning the skill of throwing a ball in a basket. It is further shown that the robot can acquire self improvement ability like reinforcement learning while using this approach.

Chapter 3

Robot Programming by Demonstration

Robot *Programming by Demonstration (PbD)* is a method in which a robot learns new skills through human guidance. It is also known as learning by imitation or apprenticeship. It is worth noting that PbD is not just copying of the demonstrated task but involves generalization of the learned skill. There are several methods developed for PbD tasks but we have mainly focused on two of them i.e. by using dynamical systems and using Gaussian Mixture models.

3.1 PbD by using dynamical systems

Use of dynamical systems was first proposed by Ijspeert [10]. Their model consists of two dynamical systems with one way parameterized connection such that one system drives the other (acting as a clock). As a result of this pre-structuring the system is guaranteed to be stable. Dynamical system can either form point attractor or limit cycles which make them suitable for imitating single-stroke movements or rhythmic tasks and provide robustness against perturbation. The approach relies on reshaping the attractor landscape by using non-linear regression for imitating demonstrated movements [19]. After learning the parameters from demonstrations [14] and [15] showed that the learned parameters can be further improved by using reinforcement learning. Kober [3] proposed PoWER (discussed later) for learning complex motor skill game ball in a cup.

3.2 PbD by using Gaussian Mixture Model (GMM)

The use of GMM provides a probabilistic approach for PbD tasks. The GMM does not only encode the learned trajectory but also the variability contained in the demonstrations [17]. A GMM with k components is parameterized by $\{\pi_k, \mu_k, \Sigma_k\}_{k=1}^{K}$; representing priors, centers and covariance respectively. The probability density value of a data point ξ_j to belong to the GMM is determined by $\mathcal{P}(\xi_j) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\xi_j; \mu_k, \Sigma_k)$. Expectation Maximization can be used for fitting a GMM to a dataset. The GMM can be used to recognize gestures by estimating the log-

likelihood of the observed data having been generated by the model. The log-likelihood of a dataset consisting of N data points is defined as

$$\mathcal{L}(\xi) = \sum_{j=1}^{N} log(\mathcal{P}(\xi_j)) \tag{3.1}$$

3.2.1 Expectation Maximization (EM) for GMM

Expectation Maximization was originally proposed by Dempster [6] as a way for GMM to increase the likelihood function (for given data) with respect to its parameters π (mixing coefficients), μ (means) and, Σ (covariance). The parameters are usually initialized using k-means clustering. Then, the model parameters are iteratively updated by cycling through Expectation step (E-step) and Maximization step (M-Step)

E-step:

$$p_{k,j}^{(t+1)} = \frac{\pi_k^t \mathcal{N}(\xi_j; \mu_k^{(t)}, \Sigma_k^t)}{\sum_{i=1}^{K} \pi_i^t \mathcal{N}(\xi_j; \mu_i^{(t)}, \Sigma_i^t)},$$
$$E_k^{t+1} = \sum_{j=1}^{N} p_{k,j^{t+1}},$$

M-step:

$$\pi_k^{t+1} = \frac{E_k^{t+1}}{N},$$
$$\mu_{k^{t+1}} = \frac{\sum_{j=1}^{N} p_{k,j}(t+1)\xi_j}{E_k^{t+1}},$$
$$\Sigma_k^{t+1} = \frac{\sum_{j=1}^{N} p_{k,j}(t+1)(\xi_j - \mu_{k^{t+1}})(\xi_j - \mu_{k^{t+1}})^\top}{E_k^{t+1}},$$

Where $p_{k,j}$ corresponds to the responsibility or contribution of the component k, for generating the data point j while the E_k corresponds to the effective number of points assigned to cluster k. During the E-step, the posterior probabilities are calculated while in the M-step those posterior probabilities are used to re-estimate the parameters of a GMM. The iteration stops when the increase in log-likelihood becomes very small i.e. $\frac{\mathcal{L}^{t+1}}{\mathcal{L}^t} < Threshold$; where log-likelihood is calculated as defined in equation (3.1) .

3.2.2 Gaussian Mixture Regression (GMR)

Gaussian Mixture Regression is based on linear combination of Gaussian densities and conditional Gaussian densities. Calinon et al [16] have extensively used GMR for Programming by Demonstration (PbD) tasks. Once the dataset has been encoded in a GMM; GMR provides a fast and analytic way to retrieve smooth output trajectories along with covariance matrices describing the variability and correlation among different variables. Some useful properties of Gaussian Distribution taken

from [16] are highlighted in Table 3.1 . It is worth noting that in all of the three properties of combining two Gaussian Distributions; the resulting distribution is again Gaussian. Regression is a way of modeling the relationship between a given set of *predictor* variables $\xi^{\mathcal{I}} \in \mathbb{R}^p$ and *response* variables $\xi^{\mathcal{O}} \in \mathbb{R}^q$. While fitting a GMM there is no distinction between input and output variables. For the particular case of trajectory learning time can be input variable $\xi^{\mathcal{I}}$ while position can be output variables $\xi^{\mathcal{O}}$. The aim of regression is to estimate the conditional expectation of $\xi^{\mathcal{O}}$ for a given time step $\xi^{\mathcal{I}}$. Thus we can compute expected value of position at each time step to extract smooth generalized trajectories. Like any other regression procedure GMR can also suffer from under-fitting or over-fitting. If the number of components in the GMM is lower than the required amount to model the behavior of variable, it can result in under-fitting, while having too many components can result in over-fitting. Both of these scenarios can effect the generalization capability of GMR. Cross validation can be used to determine the optimal number of GMM components.

During GMR the data set of N data points and D dimensions is encoded in a GMM of K Gaussians. The probability density function value of a data point $\xi = [\xi^I, \xi^O]$ belonging to the GMM is defined by

$$\mathcal{P}(\xi) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\xi; \mu_k, \Sigma_k) = \sum_{k=1}^{K} \pi_k \frac{1}{\sqrt{(2\pi)^D |\Sigma_k|}} \exp^{\frac{-1}{2}(\xi-\mu_k)^\top \Sigma_k^{-1}(\xi-\mu_k)} \tag{3.2}$$

where π_k are prior probabilities and $\mathcal{N}(\mu_k, \Sigma_k)$ represent Gaussian distributions defined by centers μ_k and covariance matrices Σ_k. Input and Output componenets are represented separately as

$$\mu_k = \begin{bmatrix} \mu_k^I \\ \mu_k^O \end{bmatrix}, \Sigma_k = \begin{bmatrix} \Sigma_k^I & \Sigma_k^{IO} \\ \Sigma_k^{OI} & \Sigma_k^O \end{bmatrix} \tag{3.3}$$

For a given input variable ξ^I and a given Gaussian distribution k, the expected distribution of ξ^O is defined by

$$\begin{aligned} \mathcal{P}(\xi^O|\xi^I, k) &\sim \mathcal{P}(\hat{\xi_k}, \hat{\Sigma_k}) \\ \text{where} \qquad \hat{\xi_k} &= \mu_k^O + \Sigma_k^{OI}(\Sigma_k^I)^{-1}(\xi^I - \mu_k^I) \\ \hat{\Sigma_k} &= \Sigma_k^O - \Sigma_k^{OI}(\Sigma_k^I)^{-1}\Sigma_k^{IO)} \end{aligned} \tag{3.4}$$

By considering the complete GMM, the expected distribution of ξ^O, when ξ^I is known can be estimated as

$$\begin{aligned} \mathcal{P}(\xi^O|\xi^I) &\sim \sum_{k=1}^{K} h_k \mathcal{N}(\hat{\xi_k}, \hat{\Sigma_k}) \\ \text{with} \qquad h_k &= \frac{\mathcal{P}(k)\mathcal{P}(\xi^I|k)}{\sum_{i=1}^{K} \mathcal{P}(i)\mathcal{P}(\xi^I|i)} = \frac{\pi_k \mathcal{N}(\xi^I; \mu_k^I, \Sigma_k^I)}{\sum_{i=1}^{K} \pi_i \mathcal{N}(\xi^I; \mu_i^I, \Sigma_i^I)} \end{aligned} \tag{3.5}$$

An illustrative example of GMR can be seen in Figure 3.1.

Table 3.1 Properties of Normal Distributions

Linear combination of Gaussian Densities

If $x_1 \sim \mathcal{N}(\mu_1, \Sigma_1)$ and $x_2 \sim \mathcal{N}(\mu_2, \Sigma_2)$; there linear combination$A_1x_1 + A_2x_2 + c$, where A_1 and A_2 define transformation/rotation matrices and c defines constant offset vector, follows the distribution

$$A_1x_1 + A_2x_2 + c \sim \mathcal{N}(\mu, \Sigma) \tag{3.6}$$

with

$\mu = A_1\mu_1 + A_2\mu_2 + c$
$\Sigma = A_1\Sigma_1A_1^T + A_2\Sigma_1A_2^T$

Conditional Gaussian Densities

Let $x \sim \mathcal{N}(\mu, \Sigma)$ be defined by

$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$, $\mu = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}$, $\Sigma = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix}$
The conditional probability density function is then defined by

$$p(x_1|x_2 = a) \sim \mathcal{N}(\hat{\mu}, \hat{\Sigma}) \tag{3.7}$$

with

$\hat{\mu} = \mu_1 + \Sigma_{12}(\Sigma_{22})^{-1}(a - \mu_2)$
$\hat{\Sigma} = \Sigma_{11} - \Sigma_{12}(\Sigma_{22})^{-1}\Sigma_{21}$

Product of Gaussian Densities

If $x_1 \sim \mathcal{N}(\mu_1, \Sigma_1)$ and $x_2 \sim \mathcal{N}(\mu_2, \Sigma_2)$; there product $x_1.x_2$, follows the distribution

$$C.\mathcal{N}(\mu, \Sigma) = \mathcal{N}(\mu_1, \Sigma_1).\mathcal{N}(\mu_2, \Sigma_2) \tag{3.8}$$

with

$C = \mathcal{N}(\mu_1; \mu_2, \Sigma_1 + \Sigma_2)$
$\mu = (\Sigma_1^{-1} + \Sigma_2^{-1})^{-1}(\Sigma_1^{-1}\mu_1 + \Sigma_2^{-1}\mu_2)$
$\Sigma = (\Sigma_1^{-1} + \Sigma_2^{-1})^{-1}$

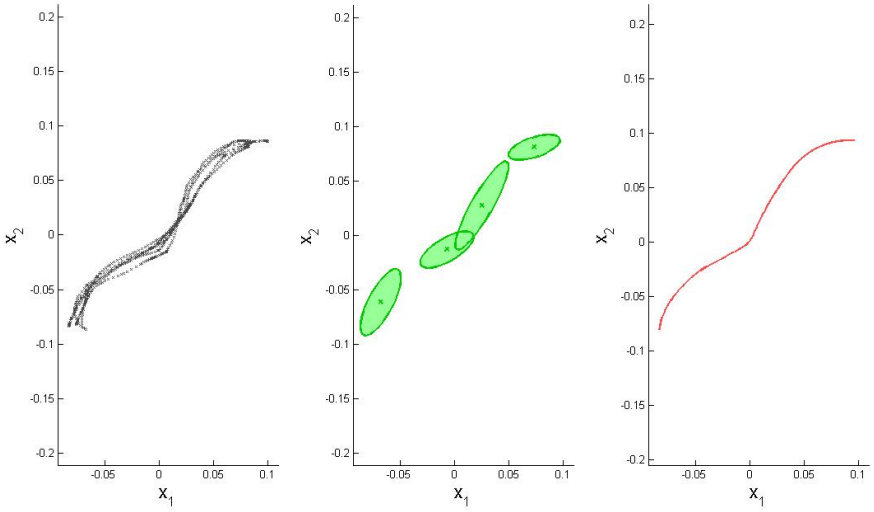

Figure 3.1. Left: Two dimensional data showing different trajectories Center: GMM fitted to the data on left Right: GMR used to retrieve smooth mean trajectory from GMM with x_1 as input and x_2 as output

Chapter 4

A brief survey/overview of reinforcement learning

Reinforcement Learning (**RL**) is the branch of machine learning in computer science where the aim is to determine how an agent ought to take actions in an environment to maximize cumulative reward. It can also be termed as trial and error learning.

The early RL methods were formulated for Markov Decision Processes (MDPs) whose basic elements are

1. Set of states
2. Set of actions
3. State transition probabilities
4. Immediate reward
5. Observations (Deterministic in MDPs while stochastic in POMDPs (Partially Observable Markov Decision Process))

Apart from that there are two more possibilities

1. Model of the environment is known or a simulation based environment is available but analytic solution is unknown.
2. Model is not known but we can get information about the environment by interacting with it.

In **RL**, a policy is any function mapping from states to action i.e.

$$\pi : S \to A \tag{4.1}$$

While a value function for a given policy π can be defined as

$$V^{\pi}(s) = E[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + ...|s_0 = s, \pi] \tag{4.2}$$

Where γ is discount factor such that γ belongs to $[0, 1)$. Given a fixed policy π its value function satisfies

$$V^{\pi}(s) = R(s) + \gamma \sum_{s' \epsilon S} P_{s\pi(s)}(s') V^{\pi}(s') \tag{4.3}$$

So the optimal policy will be the one which maximize value function.

$$V^{*}(s) = max_{\pi} V^{\pi}(s) \tag{4.4}$$

4.1 Discrete domain RL

For discrete states and known environment one can either do value function approximation (value iteration) or policy approximation based on current value function (policy iteration) to come up with an optimal policy.

If the environment is unknown; we can use Model free methods like Monte Carlo method which update the value function after reaching the goal state (slow convergence). To speed up the process one can use TD (Temporal Difference) learning, Q-learning or SARSA where learning is performed on each step instead of waiting for the agent to reach a goal/terminal state.

4.2 Continuous domain RL

All of the methods discussed above are useful for discrete states and actions and fails to run in continues environment or high dimensional environments. An alternative is to devise a parameterized policy $\pi(\theta)$ and search directly in parameter space instead of policy space. Policy gradient methods have been devised to cope with continuous states and action. Episodic REINFORCE [28] or episodic Natural Actor-Critic (NAC) [27] are the few examples which have been demonstrated to work for high dimensional continuous problems but their main shortcoming is the dependence on learning rate which is often difficult to set. This learning rate is essential for good performance but it is often difficult to tune for real world control problems.

4.2.1 Expectation Maximization (EM) based RL

Recently alternate approaches for avoiding gradient computation have been developed which turn RL problems into probabilistic estimation problems. The core idea is to consider strictly positive rewards such that the rewards can be interpreted as probabilities. By doing so it turns out that expectation maximization procedures can be used to come up with a policy. The only learning parameter which we need to set is exploration noise which is relatively easy to set and visualize, thus avoiding gradient computations while yielding faster convergence [3].

Dayan and Hinton [20] argued that apart from random search, simulated annealing and genetic algorithms, almost all methods rely on taking small steps in the direction of the gradient, which make their effect local. However, they quoted

that "it is possible to make large, well-founded changes to the parameters without explicitly estimating the curvature of the space of expected payoffs, by a mapping onto a maximum likelihood probability density estimation problem which in effect, maximize reward by solving a sequence of probability matching problems, where μ is chosen at each step to match as best it can a fictitious distribution determined by the average rewards experienced on the previous step. Although there can be large changes in μ from one step to the next, we are guaranteed that the average reward is monotonically increasing. The guarantee comes with the same reason as in EM [6], and as with EM, there can be local optima". Due to its simplicity and ease of implementation EM based reinforcement learning has attracted many researches in robotics community ([3],[21]).

Chapter 5

Multi Optima search with adaptive Gaussian Mixture Model

This chapter mention the advantages of tracking multiple solutions and presents two methods for Multi Optima search i.e. adaptive Gaussian Mixture Model and Data Spectroscopy. The former is simulated for a Dart game experiment where it successfully discovered and tracked the multiple solutions while the later is also simulated on a synthetic dataset to prove its viability.

5.1 Policy learning by Weighting Exploration with Returns

Kobers and Peters proposed *Policy learning by Weighting Exploration with Returns* **(PoWER)** [3] which is an EM based RL algorithm and they successfully demonstrated its application in learning complex task of catching ball in a cup. PoWER is an EM algorithm where the action is treated similarly to an unobserved variable and rewards are considered as improper probability distributions. It relies on the idea of reward weighted imitation and on the intuition that the safe way to generate new policies is to search in convex combination of sampled policies.

$\theta^{(n-1)}$ represents the current best estimate of the policy. Exploration can be performed by sampling from $\mathcal{N}(\theta^{(n-1)}, \Sigma^{(n-1)})$ where $\Sigma^{(n-1)}$ is a predetermined diagonal covariance matrix. Then a scalar reward r for sampled policies is calculated. PoWER uses importance sampling to select M best policies with high return to speed up the learning process $\{\theta_k\}_{k=1}^{M}$ with $r(\theta_1) \geq r(\theta_2) \geq ... \geq r(\theta_M)$. By doing so the solution remains close to the policies with high return and away from the policies with low returns. $[\theta^{(m)} - \theta^{(n-1)}]$ represents the relative exploration between the policy parameters, weighted by the corresponding return. A new policy $\theta^{(n)}$ is estimated and used for generating new samples for next iteration

$$\theta^{(n)} = \theta^{(n-1)} + \frac{\sum_m^M r(\theta_m)[\theta^{(m)} - \theta^{(n-1)}]}{\sum_m^M r(\theta_m)}. \tag{5.1}$$

5.2 Exploiting covariance information in RL

The RL algorithm proposed earlier used state independent and zero mean Gaussian noise for exploration but such unstructured exploration at every step has many disadvantages namely

1. It causes a large variance during the parameters update which grows with time.
2. It perturbs the actions too frequently, as the system acts as a low pass filter the perturbations average out and thus their effects are washed out.
3. Can damage the system executing the trajectory by bringing it into unsafe regions.
4. Small exploration noise can be accurate but slow while large exploration noise has the disadvantage of being unsafe.

R*ü*ckstieβ [22] put forward the idea of state dependent exploration. As suggested by [23] the exploration noise can also be optimized along with learned policy parameters.

$$\Sigma^{(n)} = \frac{\sum_m^M r(\theta_m)[\theta^{(m)} - \theta^{(n-1)}][\theta^{(m)} - \theta^{(n-1)}]'}{\sum_m^M r(\theta_m)} + \Sigma_0, \tag{5.2}$$

where Σ_0 is the predetermined minimum exploration noise (diagonal covariance matrix). It is worth noting that the policy is now in the form of multivariate Gaussian distribution with a mean and covariance. This property can be exploited in different manners. This representation conveys useful information about neighborhood e.g. shape, spread and principal directions. For example spread can be used to terminate learning (in case of static environment/model), principle directions can be used to search for robust solutions.

In case of multimodal policy space for exploration PoWER can have multiple start ups with different means so that it can converge to different solutions and later we can select the best solution found for exploitation. An illustrative diagram of mean and covariance update can be seen in Figure 5.1.

5.2.1 Different ways of modeling exploration noise

One way of modeling exploration noise is mentioned in Equation 5.2 where we add a diagonal covariance matrix to let the exploration happen equally in both principal axis. This approach has few disadvantages like it forces the covariance matrix to become circular and due to equal exploration in two principal axis it minimizes the advantage of exploration. An alternate approach is to do more exploration in the larger principal axis of covariance matrix. This can be achieved by decomposing the covariance matrix in the form of eigenvalues and eigenvectors and then adding exploration term in eigenvalues proportional to their magnitude which means higher

Figure 5.1. Left: Red circle showing mean while green ellipse representing exploration term (covariance matrix). Black circles represent samples while their size represent the weights Middle: Elite samples picked using their weights Right: Updated mean and covariance using PoWER

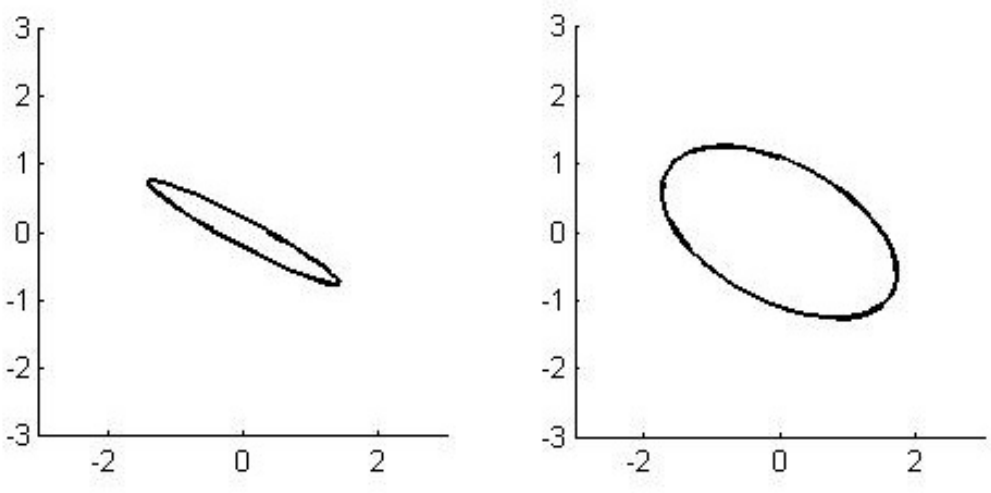

Figure 5.2. Initial covariance ellipse on left and resultant ellipse by adding constant diagonal matrix on right

exploration for major principal axis and lower exploration for minor principal axis and then reconstructing the covariance matrix by using new eigenvalues. By adopting this approach we do not only preserve the shape of covariance matrix but also make exploration more efficient as demonstrated in Figures 5.2 and 5.3.

5.3 Multi Optima search using GMM

The proposed algorithm works well for unimodal distributions where the solution landscape has single optimum. The above algorithm can be extended to find multiple peaks by using a GMM [31], where a set of normal distributions will be iteratively

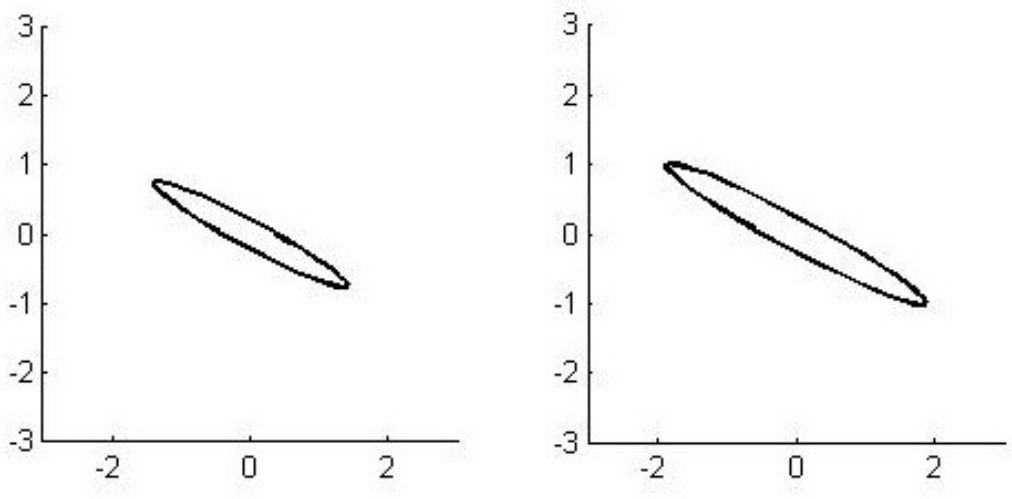

Figure 5.3. Initial covariance ellipse on left and resultant ellipse by incrementing eigenvalues proportional to their magnitude on right

optimized using EM algorithm. Each μ_i represent the best local guess while π_i represents the importance of that solution and Σ_i represent the variability that we can exploit in that solution. The complete algorithm at iteration number n for each $i \in \{1, ..., K\}$ and with M samples for each component is given as,

$$\mu_i^{(n)} = \mu_i^{(n-1)} + \frac{\sum_m^M r_i(\theta_m) h_i(\theta_m)[\theta^{(m)} - \mu_i^{(n-1)}]}{\sum_m^M r_i(\theta_m) h_i(\theta_m)} \tag{5.3}$$

$$\Sigma_i^{(n)} = \frac{\sum_m^M r_i(\theta_m) h_i(\theta_m)[\theta^{(m)} - \mu_i^{(n-1)}][\theta^{(m)} - \mu_i^{(n-1)}]^\top}{\sum_m^M r(\theta_m) h_i(\theta_m)} + \Sigma_0 \tag{5.4}$$

$$\pi_i^{(n)} = \frac{\sum_m^M r_i(\theta_m) h_i(\theta_m)}{\sum_k^K \sum_m^M r_k(\theta_m) h_k(\theta_m)} \tag{5.5}$$

where

$$h_i(\theta_m) = \frac{\pi_i \mathcal{N}(\theta_m | \mu_i^{(n-1)}, \Sigma_i^{(n-1)})}{\sum_k^K \pi_k \mathcal{N}(\theta_m | \mu_k^{(n-1)}, \Sigma_k^{(n-1)})} \tag{5.6}$$

$h_i(\theta_m)$ represents the responsibility of i_{th} GMM component for m_{th} sample. The above formulation is useful for two scenarios:

1. When the solution landscape has multiple peaks which perform equally well in terms of reward.

2. When there are multiple peaks in the solution landscape which cannot be tracked and represented by single Gaussian.

5.3.1 Adapting the number of components

The main bottle neck of the above algorithm lies in the fact that the number of components in the GMM is unknown beforehand. Also in real world scenarios the policy landscape may not be static and can change during policy search (as will be shown later). This can occur due to the continuous wear of the robots actuator (where policy landscape is continuously changing) or a sudden break down of the robots motor (sudden change in policy landscape). Such dynamic policy landscapes make the RL problem more challenging and methods which rely on exploration phase followed by an exploitation phase may not work well for these conditions.

Zhang [4] proposed Split and Merge algorithm for determining number of components in data clustering problems. The key idea behind their approach was that:

1. If merging two components into a single component yields a slight decrease in the overall likelihood of the data then we should merge them.

2. Similarly if splitting a single component into two components yields a significant increase in overall likelihood of the data then we should split those components.

The reasoning behind splitting approach is very simple which follow the Occam's principle stating that among the competing hypothesis, the one with fewest assumptions should be selected. By merging two components, we are decreasing the GMM ability to fit the data. So merging will yield lower likelihood value than before. Thats why we only merge when we see only a slight decrease in the likelihood value. Which means that the simpler model is good enough in explaining the data compared to a more complex model and it should be selected. But if we keep on merging even with a large decrease in likelihood then it can result in under-fitting. Under-fitting means that the model is unable to describe the underlying relationship because its too simple and occurs when the model has fewer parameters than required.

In the same way by splitting a component, we are increasing the GMM ability to fit the data well. So splitting will yield higher likelihood value than before. Which means that if we keep on splitting, it can result in over-fitting. Over-fitting means that the model is describing random errors and noises instead of the underlying relationship. It occurs when the model is excessively complex because of having too many parameters. Thats why we only split when we see a significant increase in likelihood value.

The complete Algorithm is mentioned below:

1: **for** $n \leftarrow 1$ to N **do**
2: $\quad \boldsymbol{\Theta}_n \leftarrow$ randomSampling(currentGMM)
3: $\quad$ currentGMM $\leftarrow$ EM(currentGMM,$\boldsymbol{\Theta}$)
4: $\quad$ **for** $i \leftarrow 1$ to K **do**
5: $\quad\quad$ candidateGMM$_i \leftarrow$ splitGauss(currentGMM, i)
6: $\quad\quad$ $L_i^s \leftarrow$ evaluateLikelihood(candidateGMM$_i$)

end for
$\hat{L} \leftarrow$ evaluateLikelihood(currentGMM)
if $(\max(\boldsymbol{L}^{\mathcal{S}}) - \hat{L}) > T^{\mathcal{S}}$ **then**
    currentGMM $\leftarrow$ candidateGMM$_{\arg\max(\boldsymbol{L}^{\mathcal{S}})}$
end if
for $i \leftarrow 1$ to $K-1$ **do**
    for $j \leftarrow i+1$ to K **do**
        candidateGMM$_{ij}\leftarrow$mergeGauss(currentGMM, i, j)
        $L_{ij}^{\mathcal{M}} \leftarrow$ evaluateLikelihood(candidateGMM$_{ij}$)
    end for
end for
$\hat{L} \leftarrow$ evaluateLikelihood(currentGMM)
if $(\hat{L} - \max(\boldsymbol{L}^{\mathcal{M}})) < T^{\mathcal{M}}$ **then**
    currentGMM $\leftarrow$ candidateGMM$_{\arg\max(\boldsymbol{L}^{\mathcal{M}})}$
end if
end for

For a Multi Optima $\boldsymbol{\Theta}$ search, the gradual resolution decrease or increase of solution space is achieved by merging and splitting the Gaussian distributions and associated priors in the current GMM. The merging of two components into a single component is a well posed problem and a closed form solution exist. The merging of two Gaussians j and k into a Gaussian i us characterized by $\pi_i = \pi_j + \pi_k$ and $\pi_i\mathcal{P}(\boldsymbol{\Theta}|i) = \pi_j\mathcal{P}(\boldsymbol{\Theta}|j) + \pi_k\mathcal{P}(\boldsymbol{\Theta}|k)$. It can be shown that the closed form solution satisfies the relation $\pi_i\mu_i = \pi_j\mu_j + \pi_k\mu_k$ and $\pi_i(\Sigma_i + \mu_i\mu_i^T) = \pi_j(\Sigma_j + \mu_j\mu_j^T) + \pi_k(\Sigma_k + \mu_k\mu_k^T)$ [4].

While for splitting there is no closed form solution. They used the eigenvectors of the covariance matrix for determining the direction for the split operation. By defining the set of parameters $l \in [1, 2, ..., N,]\alpha \in [0, 1], \beta \in [0, 1]$, and $u \in [0, 1]$, a Gaussian $\mathcal{N}(\mu_i, \Sigma_i)$ and $\mathcal{N}(\mu_k, \Sigma_k)$ with parameters

$$\pi_j = \pi_i\alpha \tag{5.7}$$

$$\pi_k = \pi_i(1-\alpha) \tag{5.8}$$

$$\mu_j = \mu_i - \sqrt{\frac{\pi_k}{\pi_j}}ua_{i,l} \tag{5.9}$$

$$\mu_k = \mu_i - \sqrt{\frac{\pi_j}{\pi_k}}ua_{i,l} \tag{5.10}$$

$$\Sigma_j = \frac{\pi_k}{\pi_j}\Sigma_i + (\beta - \beta u^2 - 1)\frac{\pi_i}{\pi_j}a_{i,l}a_{i,l}^\top + a_{i,l}a_{i,l}^\top \tag{5.11}$$

$$\Sigma_k = \frac{\pi_j}{\pi_k}\Sigma_i + (\beta u^2 - \beta - u^2)\frac{\pi_i}{\pi_k}a_{i,l}a_{i,l}^\top + a_{i,l}a_{i,l}^\top \tag{5.12}$$

In the above equations, D eigenvectors with largest eigenvalues are selected for calculating Σ_i. $\Sigma_i = A_iA_i^\top$ represents the ordered (arranged in the descending order of their eigenvalues) eigencomponents decomposition of Σ_i with $A_i = [a_{i,1}, a_{i,2}, ..., a_{i,D}]$. The parameters $\alpha = \beta = \frac{1}{2}, u = \sqrt{\frac{3}{4}}, l = 1$ and $D = 1$ are used in

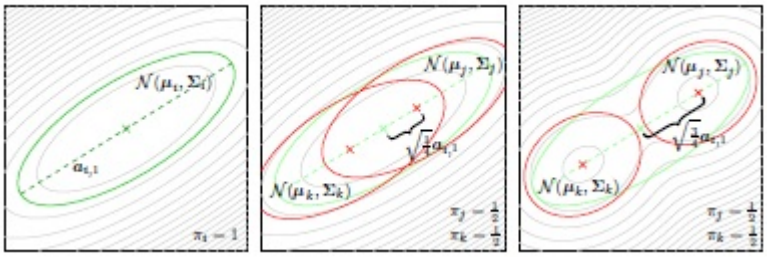

Figure 5.4. Demonstration of splitting procedure. The image has been taken from [23]

my experiments. Here $\alpha = \frac{1}{2}$ is for equally splitting weight , $\beta = \frac{1}{2}$ sets same shape for two covariance ellipses, u sets the distance between the new components and l and D are for following the major principal direction of the Gaussians. Fig. 5.4 presents an illustration of the split operation. The variable *currentGMM* in the pseudocode refers to the current GMM parameterized by $\{\pi_i, \mu_i, \Sigma_i\}_{i=1}^{K}$(initialized with a single Gaussian).

5.3.2 Alternate approach for Multi Optima search

Alternatively Data Spectroscopy clustering (DaSpec) defined by [9] can be used. It is a non-parametric spectral clustering method for initializing a GMM which does not require to define the number of components and the initialization of the model's parameters (as is required in k-means clustering which can also be used for initializing GMM). It can detect the number of components by building a proximity or affinity matrix of the data and then using eigenvalues and eigenvectors of this matrix for segmentation. Its only input parameter is ω (Gaussian bandwidth) which can also be omitted as automatic procedures exist for estimating ω for a given dataset [9]. The only requirement for the algorithm to work is that the GMM components should be well separated from each other. The algorithm was originally designed for data clustering but can be used for our problem by re-sampling data with Stochastic Universal Sampling [24] using their weights or alternatively can be modified to use weights directly as mentioned below:

1. Normalize weights to give them probabilistic interpretation by dividing them with $\max\{r(x_1), r(x_2), ..., r(x_n)\}$ so that they lie in between 0 and 1.

2. Compute the Gaussian kernel matrix K_n where $\omega > 0$ is the Gaussian kernel bandwidth and n is the sample size:

$$(K_n)_{ij} = r(x_i)r(x_j)e^{-\frac{||x_i - x_j||^2}{2\omega^2}}, \ i, j = 1, ..., n \tag{5.13}$$

3. Estimate the number of clusters:

Identify all eigenvectors ν_j of K_n that have no sign change up to precision ϵ_j [We say that a vector $e = (e_1, ..., e_n)'$ has no sign change up to ϵ if either $\forall\ i$ $e_i > -\epsilon$ or $\forall\ i$ $e_i < \epsilon$]. With $\epsilon_j = max_i(|\nu_j(x_i)|)/n$.

The number of groups $\hat{G}$ are estimated to be number of such eigenvectors.

Denote these eigenvectors and the corresponding eigenvalues by $v_0^1, v_0^2, ..., v_0^{\hat{G}}$ and $\lambda_0^1, \lambda_0^2, ..., \lambda_0^{\hat{G}}$ respectively.

4. Assign a cluster label to each data point x_i as: $\hat{L}(x_i) = argmax_g\{|v_{0i}^g| : g = 1, 2, ..., \hat{G}\}$.

Results of DaSpec on synthetic data

A dataset of 1000 points is generated by sampling from $\mathcal{N}(0, \Sigma)$ with $\Sigma = 9.I$. Weights are assigned by using probability density function of a GMM as mentioned in Equation 5.14 with $\{\pi_i = 0.25\}_{i=1}^4$ and $\mu_1 = [3\ 3], \mu_2 = [-3\ 3], \mu_3 = [3\ -3], \mu_4 = [-3\ -3]$ and $\{\Sigma_i = 2.I\}_{i=1}^4$. Although the sampling distribution was different than weighting distribution, DaSpec correctly figured out 4 labels for the data by using their weights as shown in Figure 5.5.

$$r(x_i) = \sum_{k=1}^{4} \pi_k \mathcal{N}(x_i; \mu_k, \Sigma_k) = \sum_{k=1}^{4} \pi_k \frac{1}{\sqrt{(2\pi)^D |\Sigma_k|}} \exp^{\frac{-1}{2}(x_i - \mu_k)^\top \Sigma_k^{-1}(x_i - \mu_k)} \quad (5.14)$$

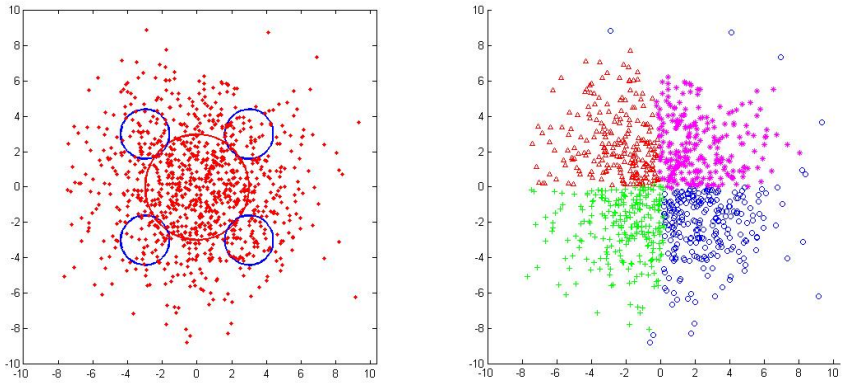

Figure 5.5. Left: Data generated using normal distribution (displayed as red ellipse) and weights are assigned using probability density function of a GMM (displayed as blue ellipses) Right: Four components correctly identified by DaSpec by using weights of the data

Chapter 6

Dart game experiment

This chapter present the experiments conducted for validating the Multi Optima search strategy presented in section 5.3. It has been shown that how an agent can maximize its expected score using the adaptive GMM. The results show the advantage of using continuous exploration/exploitation strategy in case of dynamic reward landscape. The proposed strategy also has an advantage of detecting discontinuous switch of the global optima.

6.1 Darts

Darts is form of throwing game in which metallic missiles (darts) are thrown on the circular target (dartboard). Hitting double bull (DB) (inner most ring) gives 50 points while hitting single bull (outer circle to DB eye) gives 25 points. The remaining boards is divided into 20 pie-sliced sections of scores varying from 1 to 20 with score getting double or tripled if it hits double ring or triple ring respectively.

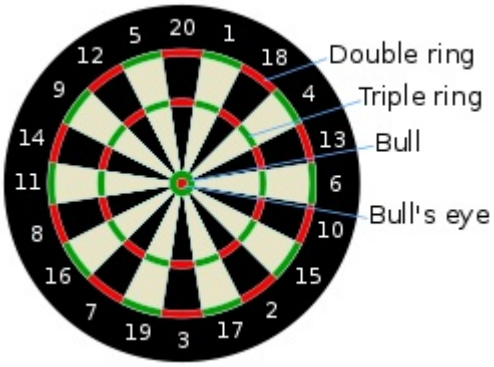

Figure 6.1. Dart board

6.2 Generating heat map from dart board

The most common game of darts focuses on maximizing a player's score. It may be surprising to know that if an amateur player tries to hit T19 (Triple 19) or T20 (Triple 20) he will get a lower expected score than aiming at another region on the board [5]. This problem has been addressed in [1] and it has been shown that a player can select an aiming point for maximizing his/her score based on their accuracy. The throw uncertainty of a player can be modeled as multivariate normal distribution. It has been shown in [5] that for estimating the throw distribution of a player it is sufficient to record 50 scores of a player and then using EM for its calculation. Using EM helps to avoid recording the precise position of dart and to simplify the task.

Once we have this normal distribution we can calculate the expected value of score associated with any point on the board by running the naive Monte Carlo method [7],[8]. But the draw back with the Monte Carlo method is that accuracy is dependent on the number of samples. It is instead propsed in [5] to take convolution of dart scores with normal distribution which can be rapidly computed using two Fast Fourier Transforms (FFTs) and one inverse FFT. This allow us to generate heat map of unstructured scores which would not be possible using Monte Carlo procedures.

$$f_{\sigma^2} * s = \mathcal{F}^{-1}[\mathcal{F}(f_{\sigma^2}).\mathcal{F}(s)] \tag{6.1}$$

where $*$ represents a convolution while $\mathcal{F}$ and $\mathcal{F}^{-1}$ denote the FFT and the inverse FFT respectivley. f_{σ^2} and s are the 2-dimensional arrays of the normal density and the scores respectively, evaluated on a millimeter scale.

6.3 Finding targeting points in darts

An agent throwing darts at target and refining its throwing skills has been simulated. The agent searches for the regions of the dartboard where it should aim at to obtain high scores. The agent refines its throwing skill linearly in time and homogeneously in space, which can be described with a Gaussian distribution $\mathcal{N}(0, \sigma^2 I)$ with standard deviation σ linearly decreasing from 75 to 16 for 3000 iterations. The dart scores have been estimated from heat maps at that particular time. The parameter of the experiment were empirically set to $S = 150$ (latest trials), $M = 15$ (importance sampling parameter of PoWER), $T^S = 0.16$ (Split threshold for average log-likelihood), $T^M = 0.01$ (Merge threshold for average log-likelihood). The experiment was repeated 20 times and Figure 6.2 represents the result for most common run.

In many reinforcement learning algorithms there is an exploration phase followed by an exploitation phase. They can be used when we have a static reward landscape because after reaching the optimal solution, we don't need to learn any more. But in case of the dynamic reward landscape, the same strategy can not be followed

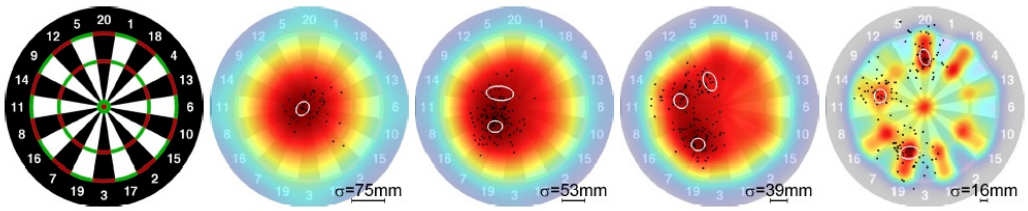

Figure 6.2. Evolution of the search process. The colored heat maps correspond to the reward distribution for the current throw accuracy of the agent. The snapshots represents states at iteration 1,1060,1757 and 3000 from left to right. Split occurred at step 1060 and 1757. The black dots show the last S=150 trials and the white ellipses represent promising solutions discovered so far by using adaptive GMM.

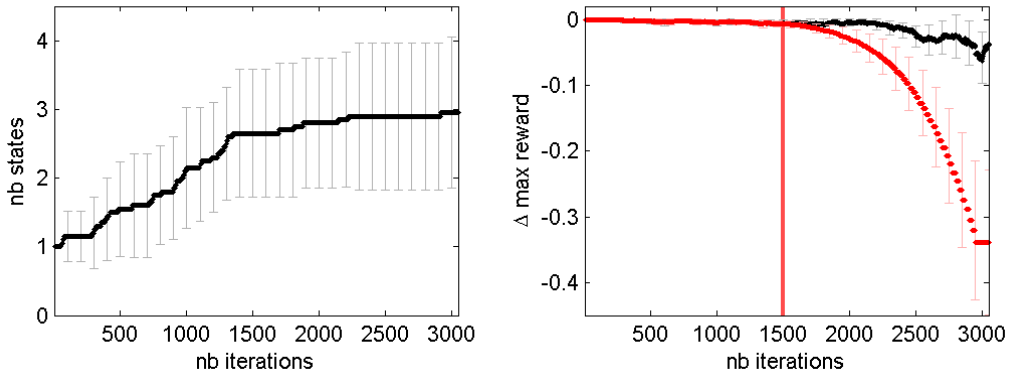

Figure 6.3. Left: Average number of Gaussians in the GMM (over 20 runs) at each iteration with standard deviation depicted as vertical bars. It can be seen that on average 3 GMM components are discovered. Right: Difference in theoretical maximum reward and discovered reward. The red curve shows what happens if we stop learning after 1500 iterations while the black curve shows the effect of continuous learning. The curves have been generated by averaging over 20 experiments with standard deviation shown as vertical bars.

because with the the passage of time the discovered solution can degrade in case of halting exploration. This behavior can be observed in the game of darts where an agent refining its precision can perform poorly if it stops learning.

Figure 6.3-left shows the result of average number of states after running simulation for 20 times. In Figure 6.3-right we can see the comparison between *continue learning until end (black curve) Vs stopping to learn after 1500 Iterations (Red Curve)*. As we can see the black curve outperforms the red curve (red curve goes down exponentially) showing the benefit of continuous learning and adaptation strategy.

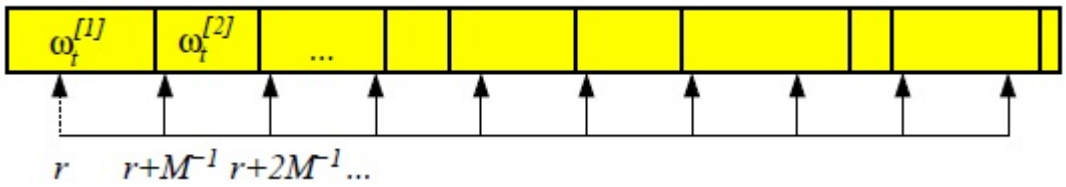

Figure 6.4. An illustrative diagram of how stochastic universal sampling works. The image has been taken from [29]

6.4 Average log-likelihood of weighted data

For splitting and merging operations we need to calculate the likelihood of the data. The likelihood of data with N points and K GMM components is calculated as

$$\text{Likelihood of Data for a given GMM} = \prod_{n=1}^{N} [\sum_{k=1}^{K} \pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)] \tag{6.2}$$

But there does not exist any method for calculating the likelihood of data with weights. One possible solution is to resample date using stochastic universal sampling [24] as shown in Figure 6.4 and then using Equation 6.2 for calculating likelihood of re-sampled data by discarding weights. The reason behind this approach is that we can consider all data points to have same weight after re-sampling, which means that every data point will have same importance. But this method is not accurate and to increase the accuracy of the solution we can decrease bin size for getting finer resolution. By doing this we will get more data points after re-sampling which will change the likelihood value (as it is a product and changing the size of a dataset will change it). To solve this problem we can calculate the average log likelihood of the re-sampled data by using Equation (6.3) because it is an average which is independent of the size of a dataset.

$$\text{Average-log likelihood of data for a given GMM} = \frac{1}{N}\Big[\sum_{n=1}^{N} \log\Big(\sum_{k=1}^{K}[\pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)]\Big)\Big] \tag{6.3}$$

It can be seen in Figure 6.5 that the average log likelihood converges as resolution is increased (decreasing bin size). To avoid re-sampling a closed form solution is devised and is mentioned in Equation (6.4). When it is applied to the same data and the GMM used for generating Figure 6.5 the solution comes out to be 1.8621.

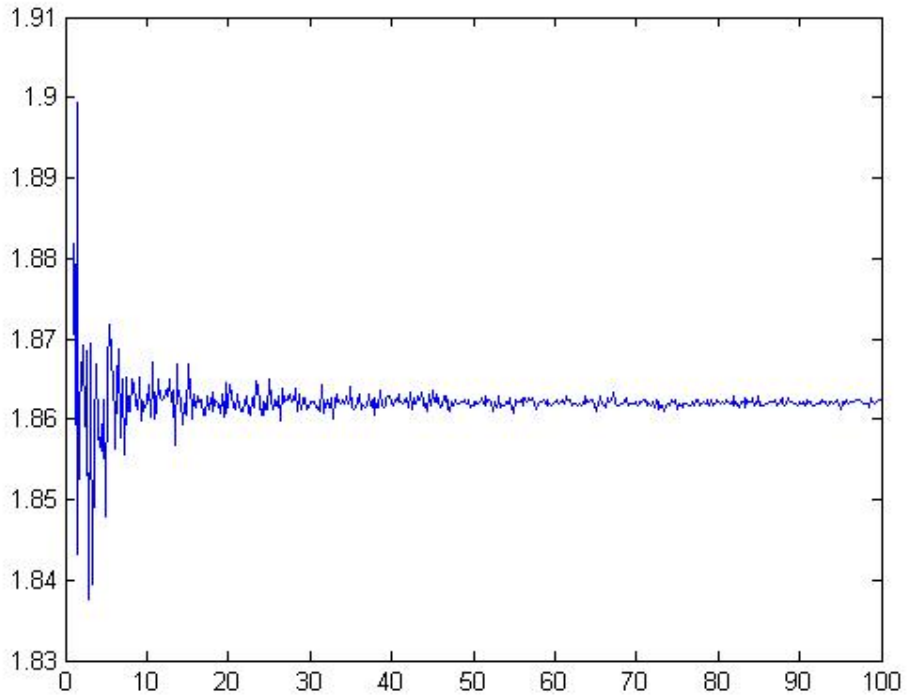

Figure 6.5. Average log-likelihood converges as we increase resolution in re-sampling; x-axis represents factor by which data is increased after re-sampling and y-axis represent estimated value of average log-likelihood

$$\begin{aligned}\text{Average-log likelihood of Weighted data for a given GMM} &= \sum_{n=1}^{N} \log \sum_{k=1}^{K} [\pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)]^{\widetilde{w}_n} \\ &= \sum_{n=1}^{N} \widetilde{w}_n \log \sum_{k=1}^{K} [\pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)] \end{aligned} \tag{6.4}$$

where $\widetilde{w}$ are normalized weights having sum equal to one $\sum_{n=1}^{N} \widetilde{w}_n = 1$

6.5 Advantage of discovering/tracking multiple optimas

There are many advantages of discovering/tracking multiple optimas rather than single solution. First of all there is higher probability of discovering global optima or in case all the tracked solutions are local optima we can select the best among them. Secondly if we have some constraints that prevent us from executing one

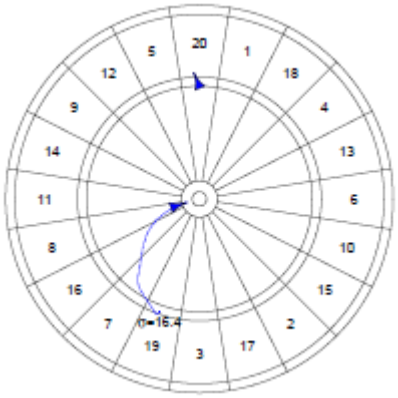

Figure 6.6. Movement of global optima parameterized by σ. At $\sigma = 0$ the global optima is at the center of T20, then it jumps at T19 at $\sigma = 16.4$ and finally stops at the lower left of bullseye at $\sigma = 100$. The image has been taken from [5].

solution, having multiple optimas provides alternate options. For example if we cannot hit certain part of the board due to an obstacle, it still provides us a way to maximize our score by going for an alternate region. Thirdly if the global optima discontinuously switches (instantaneously jumps) to another point, it can not be detected by tracking a single solution. But in case of multiple solutions, if the global optima switches to one of the previous local optima already marked by the GMM, we can detect this switch and ultimately can keep track of global optima. This phenomenon can be observed in the Darts, where global optima discontinuously switch at $\sigma = 16.4$ from T19 to T20 (as shown in Figure 6.6) where it is detected by a Gaussian already present there.

The proposed approach can also be used for modeling a solution landscape if we consider multi optima search algorithm with all data points (without importance sampling) which make the procedure similar to the well known EM algorithm for data clustering but with adaptive number of Gaussian components.

Chapter 7

Movement skill acquisition

This chapter states the different approaches (especially dynamical systems) for motor primitives learning. A robot can not only learn a skill using imitation learning strategies but can also improve its performance using reinforcement learning. It also discuss a recently proposed approach of statistical dynamical system which combine statistical machine learning techniques with dynamical systems.

7.1 State space control

The modern control theory (which is not modern anymore) evolved in early 1960s and dealt control problems using state space analysis. The systems were directly analyzed in time domain which was due to the development of computers and advances in numerical analysis [30]. Equation (7.1) mentions that how the state variable equations are represented.

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned} \tag{7.1}$$

This representation is a mathematical model of a physical system. It uses a first order differential equation to describe the relationship between output, input and state variables. Here x is the state vector, y is the output vector, u is the input vector and A B C and D are the state matrix, input matrix, output matrix and feed-forward matrix respectively. The properties of a system for e.g. observability, controllability and stability were analyzed using state variable.

7.2 Early work in Motor Primitives (MPs)

It is discussed in [3] that now a days most of the robots in industries are programmed by humans, who use their knowledge and understanding for accomplishing a desired task. This has become main bottleneck in manufacturing of cheap merchandise in low numbers. This problem can be solved if the robots can learn new skills and improve their performance automatically. This has motivated many researchers to

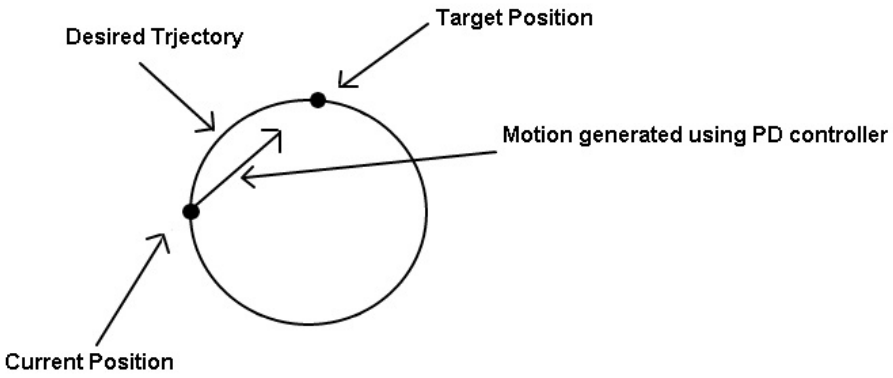

Figure 7.1. An illustrative figure showing the problems in tracking caused by using a PD controller

develop programming by demonstration (PbD) techniques where a skill is transfered to a robot by demonstrating it, instead of programming. Most of the motor skills rely on a small set of MPs [11] which raises the importance of MPs learning.

As described by Schaal [19], most of the control literature around 1950s and 1960s concentrated on the task of learning a control policy:

$$u = \pi(x, t, \theta) \tag{7.2}$$

where π is the policy that maps state vector x, time t and θ the parameters to the control input u. The policy can be generated by a neural network or a function approximation. Optimization theory, dynamic programming and reinforcement learning offer promising solutions for learning policy π. By using these approaches one can approximate a desired trajectory $[x_d(t), \dot{x}_d(t)]$ and then can use an error driven controller like a PD controller to track this trajectory

$$u = K_x(x_d(t) - x) + K_{\dot{x}}(\dot{x}_d(t) - \dot{x}) \tag{7.3}$$

This formulation can be used in fully static and predictable environment but suppose that the desired trajectory is circular in shape. An illustrative example can be of a robot trying to track the surface of a ball with its fingertip. If someone holds the finger for a long interval during the execution of the task, it can cause two problems upon release. Firstly it will try to move fast for approaching the target point (due to a large error) which can bring it to unsafe regions of control input. Secondly instead of tracking the desired trajectory the finger will take the shortest possible path by moving from inside the ball and eventually colliding with it [19] as shown in Figure 7.1. Due to the problems associated with time indexed desired trajectories other ways of generating control policies have been studied. One such approach is to encode the system in the form of a dynamical system

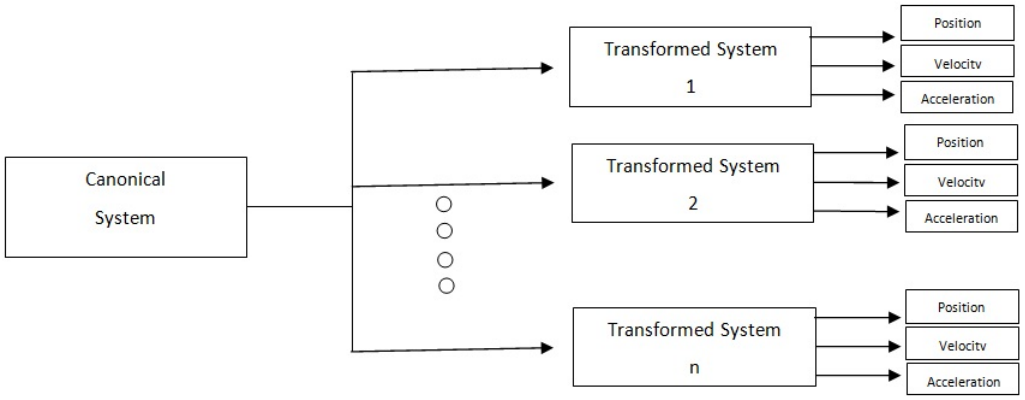

Figure 7.2. Schematic of modeling dynamical system with two subsystems

$$\dot{x} = f(x, \theta, t) \tag{7.4}$$

The use of dynamical systems has two advantages. First of all it can be used to model discrete movement like reaching tasks (point attractor) or can be used to form rhythmic movements like swimming and walking (limit cycles). The type of policy described above is termed as non-autonomous policy due to its explicit dependence on time. Most of the dynamical system approaches emphasize learning autonomous policy by removing time as a driving variable.

$$\dot{x} = f(x, \theta) + C \tag{7.5}$$

where C defines all the other sensory feedbacks and perception about the environment.

7.3 Imitation learning of MPs

The idea of using dynamical system for encoding motor primitives has been widely accepted in robotics community. Learning the complete dynamical system can be complicated. [10] suggested the use of two dynamical systems with one way parameterized connection such that one system drives the other as shown in Fig. 7.2. Due to this prestructuring, the system is guaranteed to be stable; avoiding unstable and chaotic behavior that can result in attempting to learn the complete system.

There is a hidden canonical system

$$\dot{z} = u(z) \tag{7.6}$$

which act as adjustable clock and derives the other system by determining the phase of the movement with respect to z. For discrete MPs a canonical system is

defined as $\dot{z} = -\tau\alpha_z z$ while for rhythmic MPs it is defined as $\dot{z} = \tau\omega$. The second system is defined as

$$\dot{x} = v(x, z, g, \theta) \tag{7.7}$$

for all considered degree of freedoms (DOFs) i. Where θ are the parameters that we want to learn and g is the goal state. For reshaping the above system to produce desired trajectories we can add amplitude modifier terms as suggested by [3] and can express v in the following form.

$$\dot{x_2} = \tau\alpha_x(\beta_x(g - x_1) - x_2) + \tau A f(z) \tag{7.8}$$

$$\dot{x_1} = \tau x_2 \tag{7.9}$$

where τ is the time constant and parameters α_x and β_x are set such that system is critically damped when $A = 0$. Diagonal matrix A acts as an amplitude modifier and f is the transformation function defined as

$$f(z) = \sum_{n=1}^{N} \psi_n(z)\theta_n \tag{7.10}$$

ψ_n are weights defined as normalized Gaussian kernels with centers 'c' and widths 'h', placed equally spaced over the duration of MP execution. The number of kernels N can be estimated through cross-validation. These functions localize the interaction in phase space.

$$\psi_n = \frac{exp(-h_n^{-2}(z - c_n)^2)}{\sum_{m-1}^{N} exp(-h_m^{-2}(z - c_m)^2)} \text{ (for discrete MPs)} \tag{7.11}$$

Now from the parameters and the reference trajectory q_t, $\dot{q}_t$ and $\ddot{q}_t$, calculate the reference transformation function f_t^{ref} by rearranging Equation (7.8).

$$f_t^{ref} = \ddot{q}_t/(\tau^2[A]_{ii}) - \alpha_x(\beta_x(g - q_t) - \dot{q}_t/\tau)/[A]_{ii} \text{(for discrete MPs)} \tag{7.12}$$

where $q = x_1$, $\dot{q} = \dot{x}_1$ and $\ddot{q} = \tau\dot{x}_2$. Now we have a linear system of equation whose parameters can be learned by weighted regression and the solution for θ as described by [3] come out to be

$$\theta^n = (\mathbf{Z}^\top \Psi \mathbf{Z})^{-1}(\mathbf{Z}^\top \Psi \mathbf{f}^{ref}) \tag{7.13}$$

with $[\mathbf{Z}]_t = z_t$ obtained by integrating $\dot{z}$ (Equation (7.6)), $\Psi = diag(\psi_1^n, ..., \psi_t^n, ..., \psi_T^n)$ and $[\mathbf{f}^{ref}]_t = f_t^{ref}$ to learn the n_{th} shape parameter.

7.4 Reinforcement learning of MPs

Most of the early work in MP learning was concentrated on imitation learning ([11],[12],[13]) without subsequent improvement of the learned task. The problem with this approach is that the demonstration may not be optimal and may not generalize well for different situations. As described in section 5.1, Kober [3] used PoWER for improving the capability of the learned task and demonstrated it for complex motor skill such as in a 'ball in a cup' game. They performed several rollouts with white Gaussian state dependent exploration

$$a = (\theta + \varepsilon_t)^\top \phi(s,t) \tag{7.14}$$

$$where \qquad \varepsilon_t \sim \mathcal{N}(0, \sigma^2) \tag{7.15}$$

for all time steps t and used an unbiased estimate of value function

$$Q^\pi(s,a,t) = \sum_{\hat{t}=t}^{T} r(s_{\hat{t}}, a_{\hat{t}}, s_{\hat{t}+1}, \hat{t}). \tag{7.16}$$

Then, after discarding low rewarded roll outs, the policy is updated as

$$\theta_{k+1} = \theta_k + \frac{\langle \sum_{t=1}^{T} \varepsilon_t Q^\pi(s,a,t) \rangle}{\langle \sum_{t=1}^{T} Q^\pi(s,a,t) \rangle} \tag{7.17}$$

until convergence $\theta_{k+1} \approx \theta_k$

7.5 Imitation using statistical dynamical systems

The idea of using statistical dynamical systems was first introduced by [18]. They used the statistical machine learning techniques (GMM) for encoding correlations along with Dynamical Systems (DS) that can react in controlled manner to perturbations, which they called DS-GMR. The system is modeled as it is being pulled by a virtual spring-damper system. Their formulation relies on modeling trajectory of attractor point instead of modeling movement variables directly. An analogy would be modeling the motion of a boat instead of modeling motion of a water skier directly such that the water skier follows the desired path as shown in Figure 7.3.

7.5.1 Transforming movement variables into attractor points

In [18], it is proposed to control a robots end effector by considering it as a virtual mass driven by weighted superposition of spring -damper systems.

$$\ddot{x} = \sum_{i=1}^{K} h_i [K_i^p [\mu_i^x - x] - \kappa^\nu \dot{x}] \tag{7.18}$$

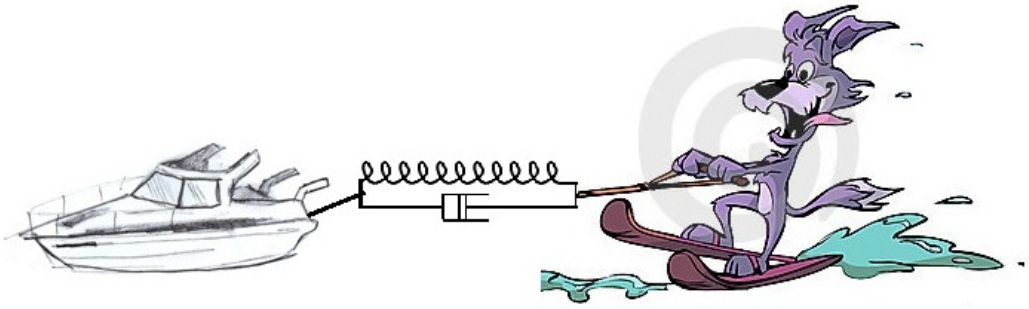

Figure 7.3. An illustrative cartoon for virtual attractor point

where x, $\dot{x}$ and $\ddot{x}$ are the positions, velocities and accelerations of the end-effector. K_i^p, κ^ν, h_i and μ_i^x are the stiffness term, damping term, mixing coefficient (with $\sum_{i=1}^{K} h_i = 1$) and the attractor point of the i-th virtual spring damper system. The system in equation (7.18) has static attractors but varying stiffness terms due to changing h_i. Alternatively it can be modeled by a moving attractor point with a constant stiffness term κ^p. Now Equation (7.18) can be equivalently written as

$$\ddot{x} = \kappa^p[y - x] - \kappa^\nu \dot{x}] \tag{7.19}$$

It is further assumed that the movement is generated by Equation (7.19). By rearranging equation (7.19) the collected trajectory in the form of x, $\dot{x}$ and $\ddot{x}$ is transformed into

$$y = \ddot{x}\frac{1}{\kappa^p} + \dot{x}\frac{\kappa^\nu}{\kappa^p} + x \tag{7.20}$$

where y corresponds to the estimate of a single virtual attractor point for each data instance x, $\dot{x}$ and $\ddot{x}$ with the constant κ^p and κ^ν. The values of κ^p and κ^ν have to be defined manually.

This approach has several advantages. It first reduces the dimension of the data by a factor of three (as now we need only position of attractor point instead of position, velocity and acceleration of the data points) which make it relatively easier to learn. Also, modeling the attractor point is helpful for generating smooth trajectories even with slight perturbations of attractor point. Once we have our data in the form of virtual attractor, we can use GMM to encode it, by learning the parameters of Equation 7.18 using EM [6]. The learned trajectories can then be retrieved using GMR as mentioned in Section 3.2 .

7.5.2 Extension to task-parameterized movement

It is shown in [18] that the DS-GMR approach can further be extended to the case where the spring-damper system can act in various candidate frames of reference.

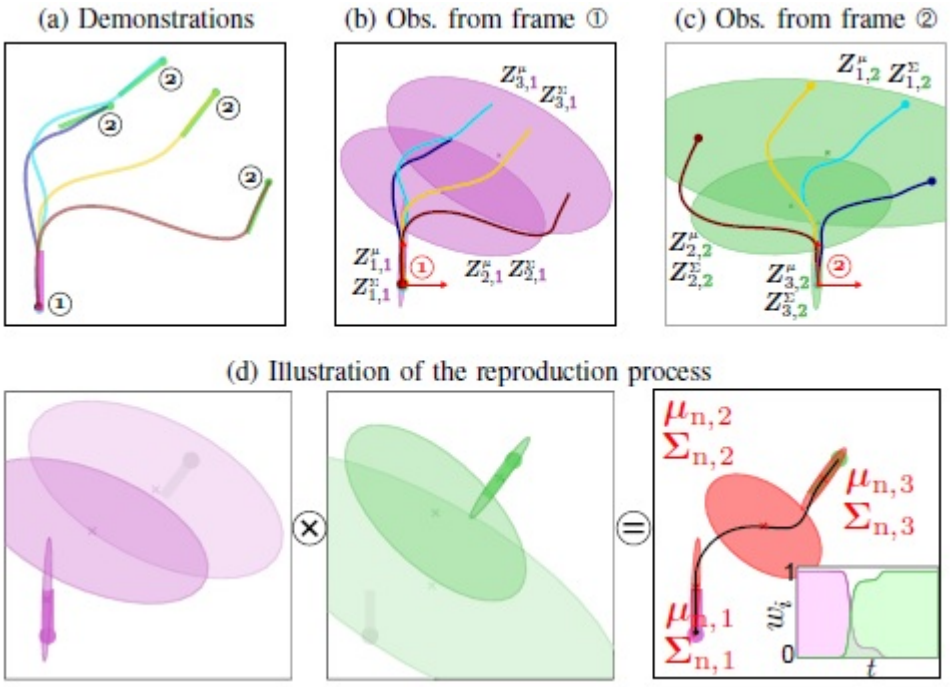

Figure 7.4. Illustration of task parameterized movement learning based on DS-GMR. The task consists of reaching frame ② from ① with a certain angle, which may represent certain skill such as inserting a peg in a hole. The image has been taken from [18].

The data is recorded in multiple frames such that it is viewed relatively to that frame of reference. A frame of reference can be defined by an offset position vector b and a linear transformation matrix A.

Now let us assume that each demonstration contains T_m data points forming a dataset $\{\xi_n\}_{n=1}^{\sum_m^M T_m}$, where each datapoint $\xi_n = [t_n, y_n]^\top \in \mathbb{R}^{D+1}$(e.g. $D = 3$ in cartesian space) is associated with task parameters $\{A_{n.j}, b_{n,j}\}_{j=1}^{P}$ representing respectively P candidate frame of reference, with offset position vectors $b_{n,j}$ and linear transformation matrices $A_{n,j}$. By denoting the model's parameters $Z_{i,j}^{\mu}$ and $Z_{i,j}^{\Sigma}$, the center $\mu_{n,i}$ and covariance matrix $\Sigma_{n,i}$ of the i-th Gaussian in the the GMM at iteration n are computed as products of linearly transformed Gaussians.

$$\prod_{j=1}^{P} \mathcal{N}(A_{n,j} Z_{i,j}^{\mu} + b_{n,j}, A_{n,j} Z_{i,j}^{\Sigma} A_{n,j}^{\top}) \tag{7.21}$$

where products of Gaussian distribution is computed as defined in Table 3.1

Figure 7.4 shows in (a) several demonstrations starting from frame ① and approaching frame ② with certain angle. In (b) and (c) it is shown how the demonstrations look when viewed from the two frames of references. In (d) we can see

the reproduction process where the GMM obtained by fitting the demonstrations were first projected to new positions of frame ① and ② and then multiplied to get a new GMM after which GMR was used to obtain a trajectory of virtual attractors. The covariance information provides a way to produce several variations of the task (trajectory of attractor points) while still maintaining the essential characteristics of the movement as shown by [17]. The parameters of the model $\{Z_{i,j}^{\mu}, Z_{i,j}^{\Sigma}\}$ in Equation (7.21) are iteratively estimated by the EM procedure as mentioned in section 3.2.1 .

Chapter 8

Simulation and experiments with statistical dynamical systems

This chapter mentions the use of statistical dynamical system and its application in learning to throw an object. A simulation is done to show that an agent can learn to throw darts for targeting any point on the board. It is further shown that an agent can improve its performance from its experience. Lastly an experiment of throwing a ball in a basket using a 7-DOF robotic arm is presented but with an extension of learning the orientation of the gripper.

8.1 Simulation of hitting on dart board

A basic simulator in Matlab was provided to test the statistical dynamical system approach in the game of darts. Demonstrations are generated by selecting some random points from a specified region. These random points are then used as attractors for generating motion trajectories as shown by the red dashed lines in Figure 8.1. Upon release, the later path of the motion is generated by simulating it as a projectile motion. Time is used as input variable (first variable in A and b matrix) while x,y and z position are the outputs variables (second, third and forth variable in A and b matrix respectively)

For observing movement this is how frames are placed to generate the model as mentioned in Equation (7.21).

$$b_{①} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad A_{①} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad b_{②} = \begin{bmatrix} 0 \\ Target_x \\ Target_y \\ Target_z \end{bmatrix} \tag{8.1}$$

The first row and column of $A_{②}$ are similar to $A_{①}$ while the remaining part ($A_{②}(2:4, 2:4)$) is defined as one vector pointing from origin towards target point while two perpendicular vectors are selected from the plane perpendicular to this

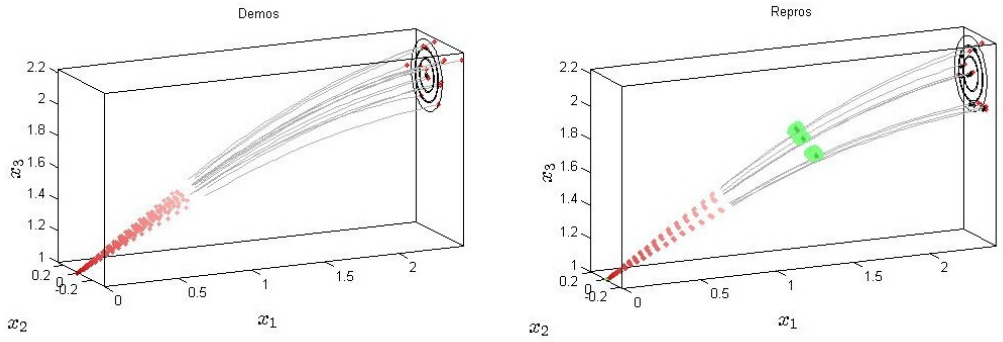

Figure 8.1. Left: Demonstrations of throwing dart. Red part indicate movement generated while gray lines indicate projectile motion Right: Reproduced trajectories with black dots on board indicating target points while red dots indicate impact of dart.

vector. Using the imitation learning approach described in section 7.5, the system learned to target points on the board as shown in Figure 8.1.

The green ellipses in Figure 8.1 indicate the current GMM positions. GMM with two components is used to model target point. It is worth noting that GMM is concentrated at a point because we actually generated movements using static point attractors (the demonstrations were just straight line motions).

8.2 Improvement strategy for statistical dynamical systems

The approach described in section 7.5 is good for imitation learning but does not have self improvement capability like reinforcement learning. To tackle this a bootstrap method (as it can generate demonstrations for itself) is adopted in which after failing to hit a point exactly on the board, we can record the true position of the dart and use this along with the reproduced trajectory to add this trial to the training set as a new demonstration. By doing so the system incrementally improves its performance. The GMM should have an appropriate number of components to avoid under-fitting or over-fitting. Otherwise the reproduced trajectory can look very different from the demonstrations and as a result can not be added to training set.

Figure 8.2 shows how the system recomputes model parameters after every reproduction attempt. Figure 8.3 shows that the mean error of reproduction attempts decreases with experience.

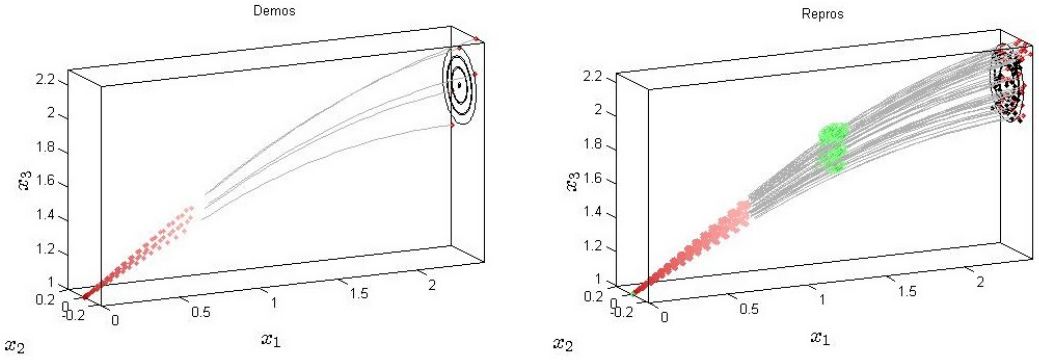

Figure 8.2. Left: Demonstrations of throwing dart. The red part indicates movement generated while gray lines indicate projectile motion Right: Reproduced trajectories with black dots on board indicating target points and red dots indicating impact points of dart. The system recomputes model parameters after every iteration to include new experience for improvement in accuracy.

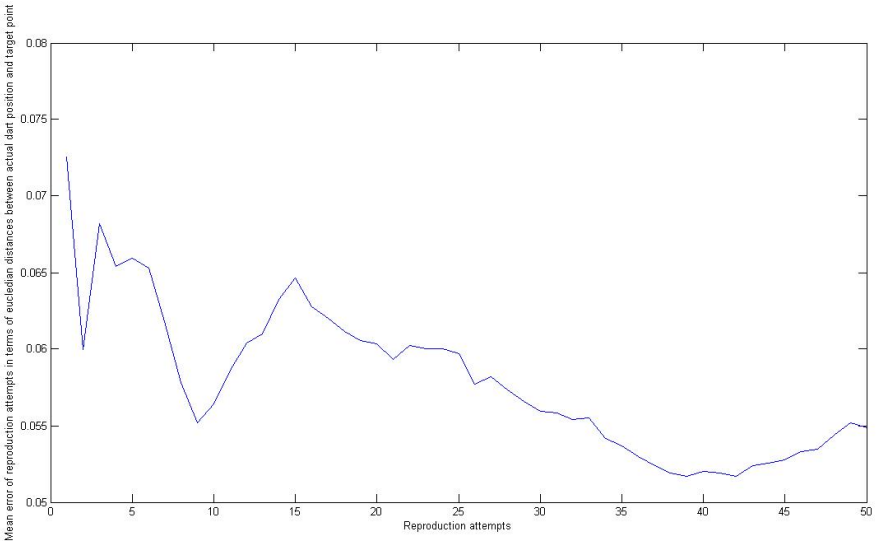

Figure 8.3. The error of statistical dynamical system decreasing with experience.

Figure 8.4. Barret WAM 7 DOF robotic arm.

8.3 Experimental setup

8.3.1 Hardware used

Barret WAM robotic arm

A Barret WAM robotic arm of 7 DOFs is used for the experiments of throwing a ball in a basket. The BH8-series BarrettHand is a programmable multi-fingered grasper with the ability to grasp target objects of different sizes, shapes, and orientations. An initial pose is selected for the experiments and all of the demonstrations/reproductions started from that pose. The fingers were set to a constant position, based on the size of the ball to make it naturally fit in the robot's hand.

OptiTrack system

An OptiTrack system composed of 12 cameras is used for tracking the position of the ball relatively to the arm. The OptiTrack camera is an integrated image capture and processing unit. A B&W CMOS imager can process 100 frames per second. The marker data is transfered by the image processing module to the computer for display and post processing. The markers are covered with special material to make them highly reflective and once a marker is visible in multiple cameras it can be easily localized.

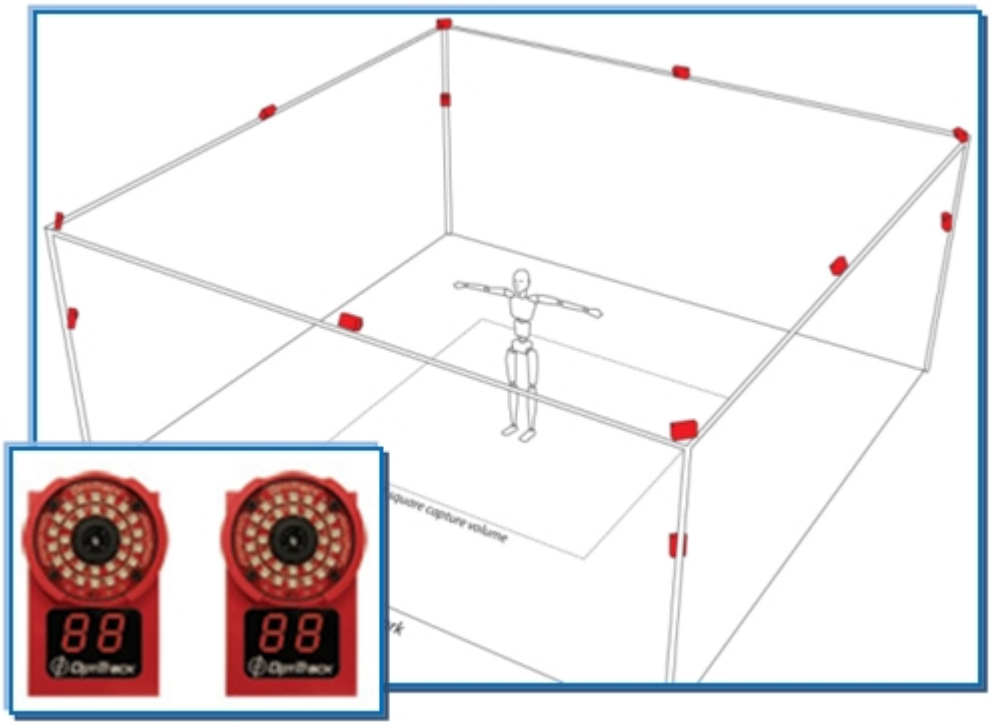

Figure 8.5. An llustrative diagram of OptiTrack system.

8.3.2 Software used

MATLAB

MATLAB is used for the multi-optima search experiments and for dart throwing simulation. The Barret WAM data is also processed in MATLAB for reproducing trajectories.

C++ and OpenFrameworks

C++ along with OpenFrameWorks is used when interacting with the Barret WAM. OpenFrameworks (OF) is a collection of many software libraries integrated together. For example, openGL is a library for graphics, freeImage is an image library, and fmod is a sound library. Many of the OF functions and classes access these libraries functionality and wrap the functions from the library so that they are easier to use and work in a similar way.

OF is used for the Barret WAM simulator, which when connected with the robot, display its movement along with the motion of attractor point.

8.4 Experiments of throwing a ball in a basket

The proposed strategy was tested with the experiment of throwing tennis balls in a basket, by using a 7 DOF Barrett WAM robotic arm. The task of throwing a ball demonstrates the same principles of learning, as that of throwing a dart but without some of the technical challenges of gripping and releasing a dart. The robot is controlled in task space by providing a force command to move the end effector while the robot is gravity compensated.. There are two sets of attractor that need to be defined along with their stiffness and damping constants. One is position attractor and the other is orientation attractor. We thus not only need to learn the position of the gripper but also the orientation of the gripper.

8.4.1 Learning orientation of the gripper

The orientation of the end-effector is described in quaternion (4 dimensional).To reduce the number of variables and learn the orientation of the gripper it is first converted into Angle of attack (α) and side slip angle (β) (often used in aerodynamics). With this approach our feature vector has six dimensions; one input (time) and five output variables namely t, x, y, z, α and β. Now the A and b matrix look like this

$$b_{①} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad A_{①} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad b_{②} = \begin{bmatrix} 0 \\ Target_x \\ Target_y \\ Target_z \\ Target_\alpha \\ Target_\beta \end{bmatrix} \tag{8.2}$$

where $Target_\alpha$ and $Target_\beta$ are defined by converting orientation vector pointing from current position towards target position into $\alpha\beta$ form. $A_{②}$ is similar to $A_{①}$ except $A_{②}(2:4, 2:4)$ are defined as described in (8.1). Alternatively $b_{①}$ can also be defined as mentioned in Equation (8.3) where $StartPos_x, StartPos_y$ and $StartPos_z$ are the starting positions x, y and z respectively which make it more robust to change in initial starting position of the end effector.

$$b_{①} = \begin{bmatrix} 0 \\ StartPos_x \\ StartPos_y \\ StartPos_z \\ 0 \\ 0 \end{bmatrix} \tag{8.3}$$

Using 13 initial demonstrations and 3 GMM components; trajectory was generated along with the orientation of the gripper as shown in Figure 8.6.

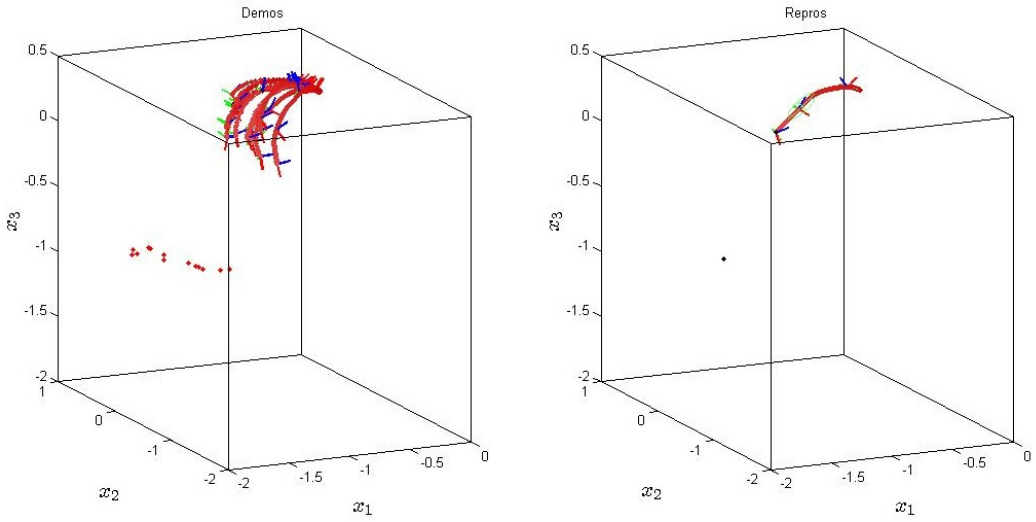

Figure 8.6. Left: All demonstrations along with orientation of the gripper in the form of frame (three perpendicular vectors). The target points are marked with red dots Right: Reproduced trajectory for a novel point along with orientation of the gripper with target point marked in black.

8.4.2 Two approaches for task-parameterized learning

Global task-parameterized approach

Global task-parameterized approach uses all of the demonstrations for generating a model. This approach is interesting at a theoretical level as it provides a better generalization capability. Figure 8.6 shows the reproduced trajectory using this approach, for a new position of the basket. It can be seen that the reproduced trajectory has the smoothness properties similar to the demonstrations.

Local regression approach

Instead of building a global model, an alternative strategy consists of looking at the trajectories which threw the ball around the target point and do local regression for reproduction attempts. This approach has a benefit of producing accurate results in the close vicinity of the selected points but the learned model will not have good generalization capability. Figure 8.8 shows how closest trajectories are selected based on the target point. Figure 8.9 shows the generated trajectory while Figure 8.10 shows that the orientation data is consistent in selected trajectories for fitting a GMM.

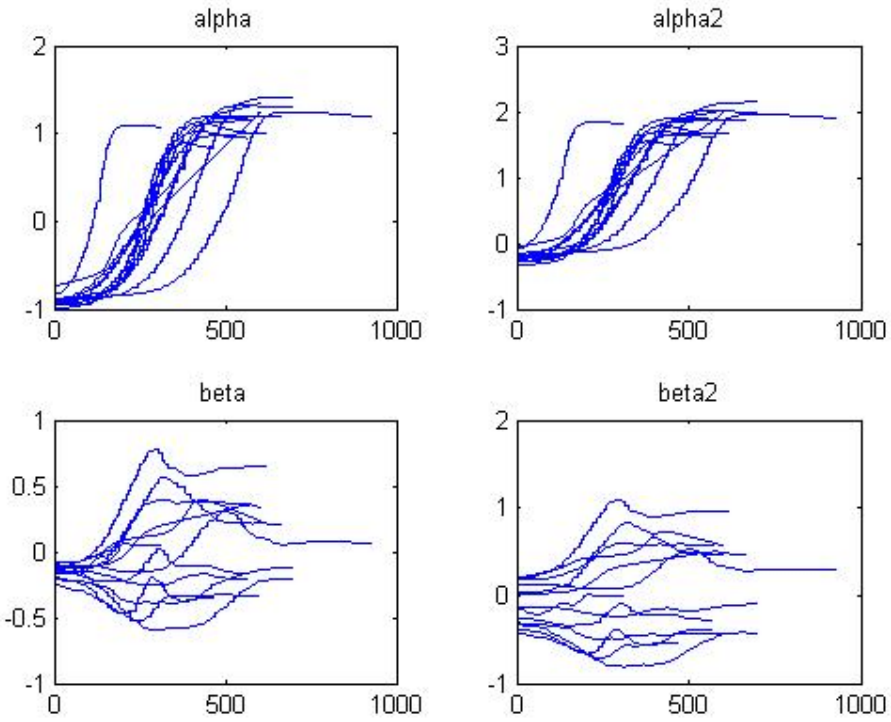

Figure 8.7. Left: α β of demonstrations in Fame 1 Right: α β of demonstrations in Fame 2.

8.4.3 Time synchronization of demonstrations data

The data for demonstrations was recorded manually on the Barret WAM by first starting recording and then moving the arm with the hand. By doing so the data may not be consistent in time. We can for example start the first demonstration after a pause of 0.1 sec while the second is started after a pause of 0.5 sec, which can be problematic when time is the driving mechanism. To compensate for that, we can define thresholds on magnitude of velocity vectors such that we record the data once the velocity vector exceeds certain threshold and stops recording when the velocity vectors goes below a certain threshold, as shown in Figure 8.11.

8.4.4 Results of using task-parameterized learning

By using the global task-parameterized approach, the robot could only throw ball in a narrow range of angles (up to 10-15 degree). The reason is that the data of β was not consistent to fit a GMM in different frame of reference as can be seen in Figure 8.7 which made it difficult for the model to converge at a good solution. On the other hand by using local regression approach, the robot was able to throw ball

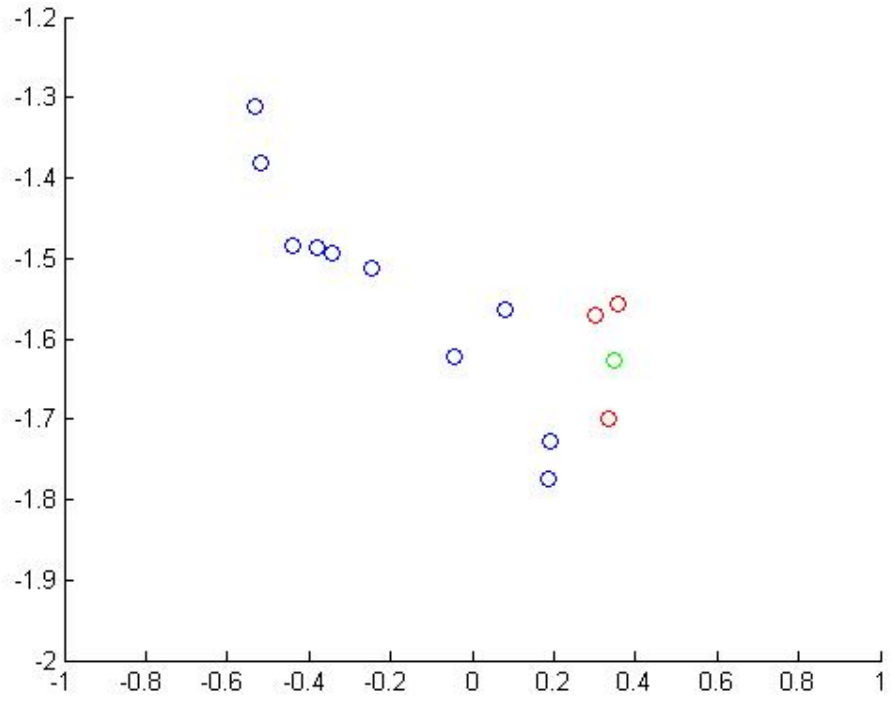

Figure 8.8. The blue circles show the basket positions for demonstrations, the green circle show the new positions of the basket and the red circles show the 3 closest basket positions for selecting the relevant demonstrations.

at wider angles (up to 45-50 degree). With local regression approach the learned model can only throw ball in the close vicinity of the point for which it is generated and cannot generalize for the basket positions away from that point. A video of the experiments can be found at *https://vimeo.com/55363862* and it can be seen in the video that the robot learned to throw the tennis balls in the basket with acceptable accuracy.

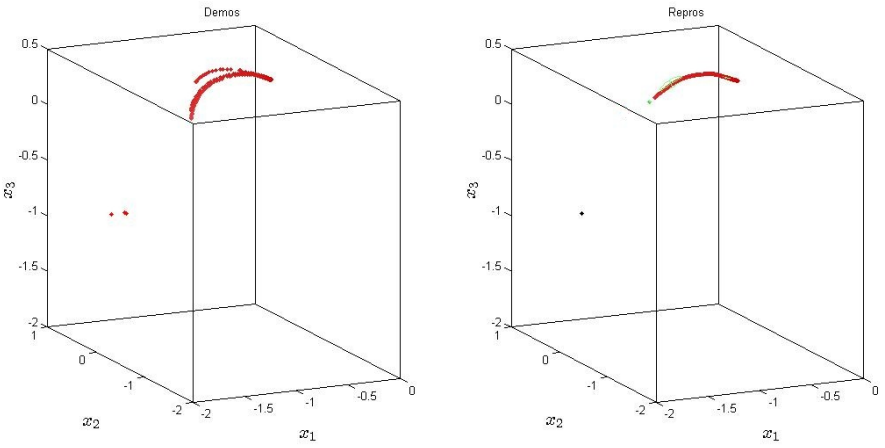

Figure 8.9. Left: Chosen trajectories for Local Regression Right: Generated trajectory for a novel point.

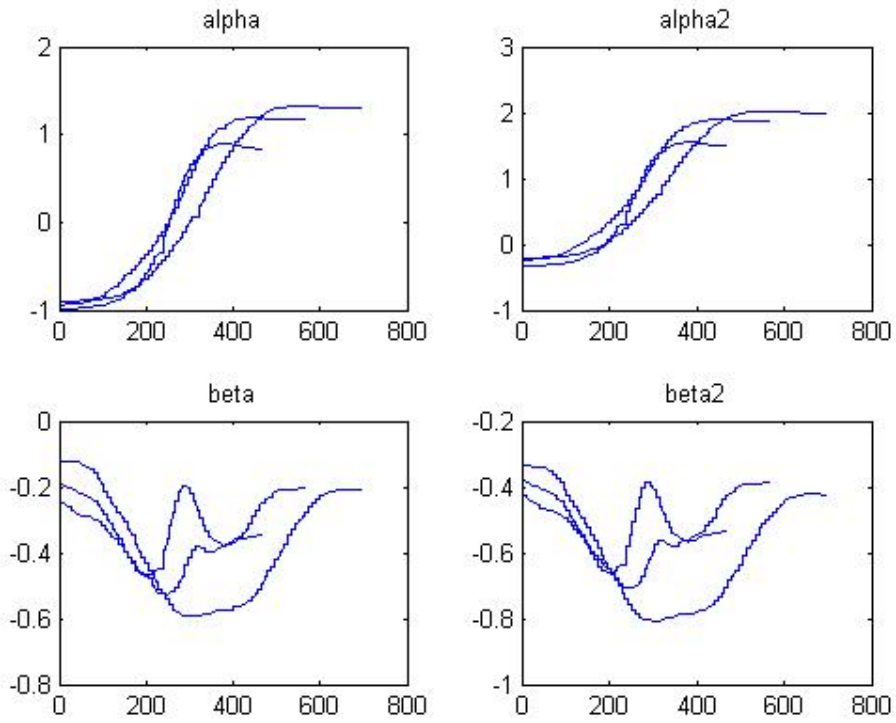

Figure 8.10. Left: $\alpha\beta$ of chosen trajectories for local regression in Frame 1 Right: $\alpha\beta$ of chosen trajectories for local regression in Frame 2.

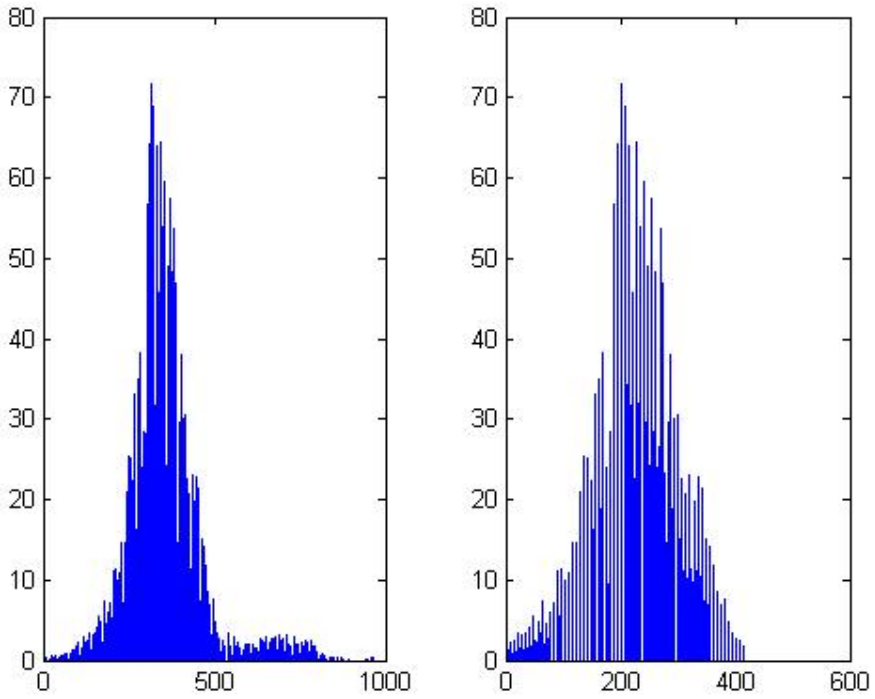

Figure 8.11. Left: Magnitude of velocity vector of a demonstration before synchronization Right: Same demonstration as left after velocity synchronization.

Chapter 9

Conclusion and future work

The report explored the different ways of letting a robot to improve its skill by trial-and-error, by using the game of dart as an example.. Split and Merge procedure mentioned by [4] was thoroughly studied. The performance of Multi Optima search is mainly dependent on exploration noise which is set manually. Automatic ways for adjusting this parameter on the fly need further consideration. An alternate approach proposed in the form of DaSpec also need further investigation as it does not have the systematic increment/decrement of components like Split and Merge but can still be useful when starting with a complex reward landscape as it can model reward landscape on the fly.

When using DS-GMR; we used time as the driving mechanism because its formulation and analysis are simpler. However, this also makes the policy non-autonomous. A further possible improvement would be to use the current state of the robot as input instead of time to generate autonomous policy.

During local regression, the 'n' closest demonstrations were used for generating GMM model and then GMR was used for generating motion trajectories. A possibly better approach could be to use all demonstrations or 'n' demonstrations but weighting them with an exponential decaying function like Gaussian weighting function and then using weighted EM for generating GMM. The local regression approach suffers from the problem that it can only interpolate data, and requires demonstration data that are sufficiently close to the desired new point. By using all of the data, the resulting approach could have better generalization ability.

Bibliography

[1] Hansen, Nikolaus. "The CMA evolution strategy: A tutorial." Vu le 29 (2005).

[2] De Boer, Pieter-Tjerk, et al. "A tutorial on the cross-entropy method." Annals of operations research 134.1 (2005): 19-67.

[3] Kober, Jens, and Jan Peters. "Imitation and reinforcement learning." Robotics & Automation Magazine, IEEE 17.2 (2010): 55-62.

[4] Zhang, Zhihua, et al. "EM algorithms for Gaussian mixtures with split-and-merge operation." Pattern Recognition 36.9 (2003): 1973-1983.

[5] Tibshirani, Ryan J., Andrew Price, and Jonathan Taylor. "A statistician plays darts." Journal of the Royal Statistical Society: Series A (Statistics in Society) 174.1 (2010): 213-226.

[6] Dempster, Arthur P., Nan M. Laird, and Donald B. Rubin. "Maximum likelihood from incomplete data via the EM algorithm." Journal of the Royal Statistical Society. Series B (Methodological) (1977): 1-38.

[7] D. Percy. Winning darts. Mathematics Today, 35(2):54-57, 1999.

[8] Column, Scott M. Berry. "A Statistician Reads the Sports Pages." CHANCE 15.3 (2002): 48-55.

[9] Shi, Tao, Mikhail Belkin, and Bin Yu. "Data spectroscopy: Eigenspaces of convolution operators and clustering." The Annals of Statistics 37.6B (2009): 3960-3984.

[10] Ijspeert, Auke Jan, Jun Nakanishi, and Stefan Schaal. "Movement imitation with nonlinear dynamical systems in humanoid robots." Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on. Vol. 2. IEEE, 2002.

[11] Flash, Tamar, and Binyamin Hochner. "Motor primitives in vertebrates and invertebrates." Current opinion in neurobiology 15.6 (2005): 660-666.

[12] Pongas, Dimitris, Aude Billard, and Stefan Schaal. "Rapid synchronization and accurate phase-locking of rhythmic motor primitives." Intelligent Robots

and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on. IEEE, 2005.

[13] Nakanishi, Jun, et al. "Learning from demonstration and adaptation of biped locomotion." Robotics and Autonomous Systems 47.2 (2004): 79-91.

[14] Peters, Jan, and Stefan Schaal. "Reinforcement learning of motor skills with policy gradients." Neural Networks 21.4 (2008): 682-697.

[15] Guenter, Florent, et al. "Reinforcement learning for imitating constrained reaching movements." Advanced Robotics 21.13 (2007): 1521-1544.

[16] Billard, Aude, et al. "Robot programming by demonstration." Handbook of robotics 1 (2008).

[17] Calinon, Sylvain, Irene Sardellitti, and Darwin G. Caldwell. "Learning-based control strategy for safe human-robot interaction exploiting task and robot redundancies." Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS), Taipei, Taiwan. 2010.

[18] Calinon, S., Li, Z., Alizadeh, T., Tsagarakis, N.G. and Caldwell, D.G. (2012). Teaching of bimanual skills in a compliant humanoid robot. Intl Workshop on Human-Friendly Robotics (HFR).

[19] Schaal, Stefan, Peyman Mohajerian, and Auke Ijspeert. "Dynamics systems vs. optimal control-a unifying view." Progress in brain research 165 (2007): 425-445.

[20] Dayan, Peter, and Geoffrey E. Hinton. "Using expectation-maximization for reinforcement learning." Neural Computation 9.2 (1997): 271-278.

[21] Kormushev, Petar, Sylvain Calinon, and Darwin G. Caldwell. "Robot motor skill coordination with EM-based reinforcement learning." Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS). 2010.

[22] R*ü*ckstieβ, Thomas, Martin Felder, and J*ü*rgen Schmidhuber. "State-dependent exploration for policy gradient methods." Machine Learning and Knowledge Discovery in Databases (2008): 234-249.

[23] Calinon, Sylvain, Petar Kormushev, and Darwin G. Caldwell. "Compliant skills acquisition and multi-optima policy search with EM-based reinforcement learning." Robotics and Autonomous Systems (2012).

[24] Baker, James E. (1987). "Reducing Bias and Inefficiency in the Selection Algorithm". Proceedings of the Second International Conference on Genetic Algorithms and their Application (Hillsdale, New Jersey: L. Erlbaum Associates): 14-21

[25] Calinon, Sylvain. "Challenges in adapting imitation and reinforcement learning to compliant robots." The International Conference SKILLS. Vol. 1. 2011.

[26] http://www.iit.it/en/advr-labs/learning-and-interaction.html

[27] Peters, Jan, and Stefan Schaal. "Natural actor-critic." Neurocomputing 71.7 (2008): 1180-1190.

[28] Williams, Ronald J. "Simple statistical gradient-following algorithms for connectionist reinforcement learning." Machine learning 8.3 (1992): 229-256.

[29] Thrun, Sebastian, Wolfram Burgard, and Dieter Fox. Probabilistic robotics. Vol. 1. Cambridge, MA: MIT press, 2005.

[30] Stefani, Raymond T. Design of feedback control systems. Oxford University Press, Inc., 1993.

[31] Calinon, S., A. Pervez, and D. G. Caldwell. "Multi-optima exploration with adaptive Gaussian mixture model." Proc. Intl Conf. on Development and Learning (ICDL-EpiRob).

TRITA-CSC-E 2013:012
ISRN-KTH/CSC/E--13/012-SE
ISSN-1653-5715

www.kth.se