BACHELOR'S THESIS

Degree in Mathematics

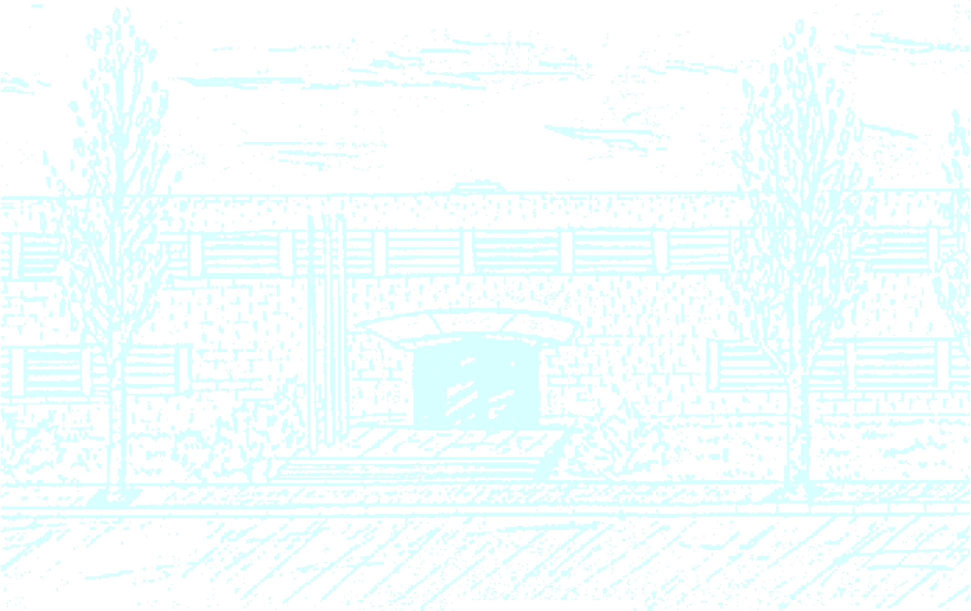
Title: Learning tasks with bimanual robots using motion symmetries

Author: Enric Cosp Arqué

Advisor: Adrià Colomé and Carme Torras

Department: Institut de Robòtica i Informàtica Industrial

Academic year: 2016/17



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Matemàtiques i Estadística

Universitat Politècnica de Catalunya
Facultat de Matemàtiques i Estadística

Degree in Mathematics
Bachelor's Degree Thesis

Learning tasks with bimanual robots using motion symmetries

Enric Cosp Arqué

Supervised by Adrià Colomé and Carme Torras

June, 2017

Firstly, I wish to express my sincere thanks to my tutors, Adrià Colomé and Carme Torras. I am extremely thankful and indebted to them for sharing expertise, and sincere and valuable guidance and encouragement extended to me.

I take this opportunity to express gratitude to my parents and brother for the incessant encouragement, support and attention. I am also grateful to my partner who supported me throughout this venture.

Finally, place on record, my sense of gratitude to one and all, who directly or indirectly, have lent their hand.

Abstract

The main aim of this work is to present a motion learning framework in order to make a *Whole Arm Manipulator* (WAM) robot learn a task. In addition, we will improve the motion learning algorithm by reducing the dimensionality of the problem in both general one arm cases, and two arm motions which include symmetries. In the last case, we will use the symmetry feature in order to upgrade the learning process.

Keywords

Motion learning, Robotics, Symmetries, Bimanual Robots.

Contents

1 Introduction

1.1 Motivations and project picture

Which tasks, problems, concepts and challenges does a researcher face day by day? This question has been boiling in the bowels since the beginning of the last year and the time to take it off the pot has arrived. From the curiosity of the applications of Machine Learning and the chance to collaborate with the Institut de Robòtica i Informàtica Industrial, CSIC-UPC, has flourished the idea to solve the latent question in an ideal environment.

In greedy fields as they are, nowadays, robotics and machine learning, it was not difficult to find a goal that both satisfied my inquietudes and was useful to the community. The main aim of this work is to present part of the process developed in order to make a *Whole Arm Manipulator* (WAM) robot learn a task. The subject that we are focusing in is the robot motion learning. Moreover, we want to implement this model in order to make the robot learn movements which involve symmetries, and use this feature to optimize the process. This task jumbles a variety of scientific areas, including kinematics, dynamics, probability, statistics and more.

This multi-disciplinary feature is also appealing as, lately, more problems are being approached in this way, and the trend seems to continue this way.

In order to carry out the motion learning process, we will characterize trajectories with Dynamic Movement Primitives (DMPs) and afterwards, we will use the latest advances in Reinforcement Learning, more specifically Policy Search, for the learning itself. To reduce the time required to implement this process, we will use a well known algorithm: Principal Component Analysis. Moreover, when applying this framework to bimanual tasks which include symmetries between arms, we will propose a method to use these symmetries to refine the algorithm.

In the introduction, we are going to contextualize this work regarding which fields it is placed in. Moreover, we will give light over how this work is placed in a more general and ambitious work inside Artificial Intelligence and Robotics. In the second chapter, we will describe Reinforcement Learning (RL) in terms of which problems it tries to solve. In addition, we will explain the general approach of RL algorithms. Chapter three will go over several algorithms and concepts that we will need in our motion learning process and upgrading.

The learning model and its implementation will be described in Chapter four. Different methods will be compared at some stages of the learning process. Being the goal of this work the understanding of a motion learning framework, approach and algorithm, rather that the application itself, we will devote to simple tasks.

Chapter five will be devoted to the improvement of the motion learning model, in terms of time and resources needed to achieve our goal. We will approach this challenge by reducing the dimensionality of our problem. Dimensionality reduction is gaining interest within robotics, as for many problems, some stages of the learning process require human interaction. The amount of required human presence is proportional to the number of dimensions. Thus, reducing the dimensionality implies needing less human interaction, which leads to a great improvement concerning the amount of time and resources needed for the robot to learn its tasks.

1.2 Global context

In this section we are going to travel through the recent years' tendencies and define the main field where this project is placed in. In particular, we will briefly discuss machine learning, artificial intelligence and robotics. The goal of this explanation is to make clear what kind of problems are solved in each of this subjects and how they are present in today's world.

1.2.1 Artificial Intelligence and Machine Learning

Artificial Intelligence

Artificial intelligence (AI) [5] is defined in computer science as the study of *intelligent agents*: any device that perceives its environment and takes actions that maximize its chance of success at some goal. As machines become increasingly capable, mental abilities once thought to require intelligence are removed from the definition. For example, optical character recognition is no longer perceived as an example of artificial intelligence, due to the fact that it has become a routine technology. Typical illustration of AI capabilities are understanding human speech, competing at a high level in strategic game systems (such as chess), self-driving cars, military simulations, and interpreting complex data.

AI research is divided into subfields that focus on specific problems and approaches. The central goals of AI research include reasoning, knowledge, planning, learning, natural language processing, perception and the ability to move and manipulate objects. Approaches include statistical methods, computational intelligence, and traditional symbolic AI, using tools such as versions of mathematical optimization, logic and methods based on probability. The AI field draws upon computer science, mathematics, psychology, linguistics, philosophy, neuroscience and so on.

Machine Learning

Pattern recognition and computational learning theory evolution gave birth to the study of algorithms that can learn from data. In contrast to static program instructions, those new algorithms are to build a model from sample inputs. Machine learning is a subfield of AI employed in a range of computing tasks where designing and programming explicit algorithms with good performance is difficult or infeasible; example applications include detection of network intruders , optical character recognition and computer vision.

Machine learning is closely related to computational statistics and mathematical optimization, as it also focuses on prediction-making and often uses optimization theory and methods. Machine learning is sometimes both confused and combined with data mining, where the latter subfield is more focused on exploratory data analysis and performed as unsupervised learning, although machine learning can also be unsupervised.

Machine learning tasks are typically classified into three broad categories, depending on the nature of the learning ”signal” or ”feedback” available to a learning system. These are:

- Supervised learning: the computer is presented with example inputs and their desired outputs, given by a programmer, and the goal is to learn a general rule that maps inputs to outputs.
- Unsupervised learning: no labels are given to the learning algorithm, leaving it on its own to find structured insights in its input.
- Reinforcement learning: a computer program interacts with a dynamic environment in which it must perform a certain goal. The program is provided feedback in terms of rewards and punishments as it navigates the problem itself. This project is inside this branch of machine learning. We will deep dive on it in later chapters.

1.2.2 Robotics

Robotics [10] is the study of machines that can replace human beings in the execution of a task, regarding both physical activity and decision making. That includes mechanical engineering, electrical engineering, computer science, and others. Robotics deals with the design, construction, operation, and use of robots, as well as computer systems for their control, sensory feedback, and information processing.

Being robotics one of the most important applications of Machine Learning, and being also such an ornate taste of the advance of human knowledge and technology, there is always an eye on it. However, not everything said about it is true. This realm is still crawling in his early days. Although the progress is significant, we are a far cry from achieving anything close to true Artificial Intelligence at a human level in many terms. On the one hand, there are some problems which computers can solve far more effectively and efficiently than people. On the other hand, there is a great amount of abilities that machines struggle to learn.

Robotics is a wide subject. In order to give context of where in the whole robot creation process is this work, we will take a quick glance at IRI's activity. At IRI there are four lines of research:

- Automatic control
- Kinematics and robot design
- Mobile robotics and intelligent systems
- Perception and manipulation

The automatic control line develops basic and applied research in automatic control, with special emphasis on modeling, control and supervision of nonlinear, complex and/or large-scale systems.

The kinematics and robot design group carries out fundamental research on design, construction, and motion analysis of complex mechanisms and structures, such as parallel manipulators, multi-fingered hands or cooperating robots.

The mobile robotics team's goal is to gift mobile robots the necessary skills to aid humans in everyday life activities. These skills range from pure perceptual activities such as tracking, recognition or situation awareness, to motion skills, such as localization, mapping, autonomous navigation, path planning or exploration.

The research of perception and manipulation group focuses on enhancing the perception, learning, and planning capabilities of robots to achieve higher degrees of autonomy and user-friendliness during everyday manipulation tasks. This work has been developed in this department with the priceless help of Adrià Colomé.

Although creating a robot involves several assignments, we will be focusing in the motion learning process, meaning that we will not go into information gathering problems nor building or other tasks.

2 Reinforcement learning

2.1 Introduction

This work is placed in Reinforcement Learning [5], a branch of Machine Learning inspired by behaviorist psychology, concerned with how software agents ought to take actions in an environment so as to maximize (or minimize) some notion of cumulative reward (or cost). Reinforcement Learning has applications in many areas and businesses, such as Industrial control, Production control, Automotive control, Autonomous vehicles control, Logistics, Telecommunication networks, Sensor networks, Ambient intelligence, Robotics, Finance; and plays an important role in both renovating lifelong industries and opening the door to the upcoming ones.

Due to the fact that many Reinforcement learning algorithms use dynamic programming techniques, the environment is usually formulated as a Markov Decision Process (MDP). The main difference between the classical techniques and Reinforcement Learning algorithms is that the latter do not need knowledge about the MDP and they target large MDP's where exact methods become infeasible.

2.2 Method

2.2.1 Markov Decision Problems

Markov Decision Processes[5] provide a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker. More precisely, a Markov Decision Process is a discrete time stochastic control process.

The basic reinforcement learning model consists of a Markov Decision Process (MDP):

- A set S of environment and agent states
- A set A of actions of the agent
- Policies: deciding the actions to take at every state

- Rules that determine the scalar immediate reward of a transition (based on what we want to achieve)

- Rules that describe what the agent observes

A reinforcement learning works in discrete time steps. At each time t, the reinforcement learning(RL) agent is at a state s_t , which typically includes the reward R_t (the reward is a measure of goodness). It then chooses an action a_t from the set of actions available, which is sent to the environment. Due to that action, the environment moves to a new state s_{t+1} and the reward R_{t+1} associated with the transition (s_t, a_t, s_{t+1}) is determined. The goal of a RL agent is to collect as much reward as possible (or as little, depending on the reward definition). This reward is used by a policy search algorithm in order to learn from the past experience.

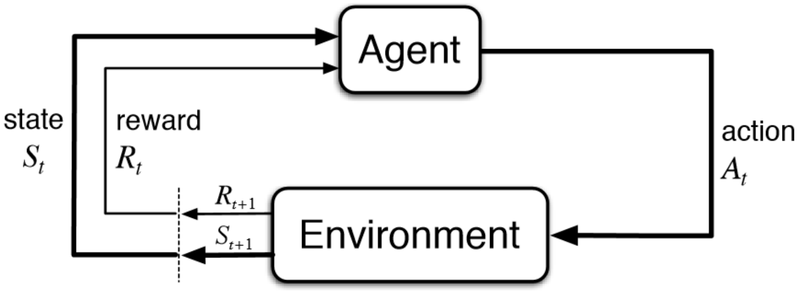
When the agent's performance is compared to that of an agent which acts optimally from the beginning, the difference in performance gives rise to the notion of regret. Note that in order to act near optimally, the agent must reason about the long term consequences of its actions: in order to maximize my future income I would better go to school now, although the immediate monetary reward associated with this might be negative. Thus, reinforcement learning is particularly well-suited to problems which include a long-term versus short-term reward trade-off.

Note: The theory of Markov decision processes does not state that S or A are finite, but basic algorithms assume that they are finite.

2.2.2 Policy Search

In this work, we will use stochastic Policy Search (PS) [1], denoted by $\pi(a|s)$. After setting an MDP, we build a reward/cost function, which depends on the trajectory $y = (s_0, a_0, s_1, a_1, \ldots)$, being a_t the action that alters the state s_t of the robot and its environment to state s_{t+1}, according to the probabilistic transition function $p(s_{t+1}|s_t, a_t)$. In robotics, s_i is usually the position or speed in the i-th timestep, and a_i the acceleration assignment at timestep i. This y is often called path or rollout. This reward/cost function is a measure of goodness of a trajectory.

In order to perform our RL algorithm, we initialize the parameters μ and Σ that define the initial policy: $\pi(a|s) \sim N(\mu, \Sigma)$. We use this policy to create N_k (fixed number, usually greater than $d \cdot N_f$) trajectories, which are perturbations of an initial demonstrated trajectory. Policy Search aims to update the policy taking into account the reward/cost function, i.e. finding good parameters for a given policy parametrization. PS is well suited for robotics as it can cope with high-dimensional state and action spaces, one of the main challenges in robot learning.



Model-free policy search is a general approach to learn policies based on sampled trajectories. We classify model-free methods based on their policy evaluation strategy, policy update strategy, and exploration strategy and present a unified view on existing algorithms. Learning a policy is often easier than learning an accurate forward model, and, hence, model-free methods are more frequently used in practice. However, for each sampled trajectory, it is necessary to interact with the robot, which can be time consuming and challenging in practice. Model-based policy search addresses this problem by first learning a simulator of the robots dynamics from data. Subsequently, the simulator generates trajectories that are used for policy learning.

We have used two different policy searches:

1. Weighted Maximum Likelihood Policy Search (WMLPS)
2. Relative Entropy Policy Search (REPS)

These methods use the information given by the MDP in order to find a policy that generates better trajectories according to the reward function. Essentially, they weight the trajectories, giving more importance to the ones with better reward.

Weighted Maximum Likelihood Policy Search

Let w_k, R_k, $\forall k \in |N_k|$ be the samples obtained with the old parameters and the rewards of the trajectories. WLPS uses the following update of the normal distribution parameters:

$$\mu^{new} = \frac{\sum_{k=1}^{N_k} w_k e^{R_k}}{\sum_{k=1}^{N_k} e^{R_k}} \tag{1}$$

$$\Sigma^{new} = \frac{\sum_{k=1}^{N_k} (w_k - \mu^{new})(w_k - \mu^{new})^T e^{R_k}}{\sum_{k=1}^{N_k} e^{R_k}} \tag{2}$$

In WLPS, the policy parameters are updated by iteratively maximizing the weighted log-likelihood for the obtained sample sequence. WMLE-based policy search methods use the reward R_k to compute a weight l_k for each sample, such that $\sum_{k=1}^{N} l_k = 1$ and, subsequently, the mean and covariance matrix of the upper-level policy $\pi(\theta)$ is updated by a weighted MLE. In this case, the weights are computed as an exponential transformation of the rewards $l_k = e^{R_k}$. The rewards are always negative and the closer the rewards are to zero, the better the trajectories are (if it is a cost function, it is always positive, the closer to zero, the better the trajectories are).

Relative Entropy Policy Search

This is also a Weighted Maximum Likelihood based policy search. In general, this policies are updated like in the previous method, with the weights computed as $l_k \propto e^{R_k/\eta}$, being η the so called temperature parameter. In the particular case of REPS, the policy update can be formulated as constrained optimization problem where we want to maximize the expected return of the new policy under the Kullback-Leibler constraint [3]. This problem consists in maximizing the expectation of the reward

$$\pi = argmax \mathbb{E}_\theta[R|\theta] \tag{3}$$

subject to:

$$\int \pi(w)dw = 1 \tag{4}$$

$$KL(\pi||q) \leq \epsilon \approx 0.5 \tag{5}$$

being π the new policy, $q \sim N(\mu^{old}, \Sigma^{old})$ the old one and KL the Kullback-Leibler divergence: $KL(\pi||q) = \sum_i \pi(i) ln \frac{\pi(i)}{q(i)}$ in its discrete form. KL divergence is a measure of how one probability distribution diverges from a second expected probability distribution [6].

The main intuition behind this bound is that we can directly control the exploration-exploitation trade-off with the parameter. On the one hand, for a large ϵ (more greedy), the variance of the new upper level policy will decrease quickly such that, it will give much higher importance to high-reward samples, ignoring lower-performing ones.
always choose the sample with highest reward. On the other hand, for a small ϵ (less greedy), the new search policy and the old search policy would be almost identical.

This optimization is solved for our samples, due to the fact that it cannot be solved analytically, as the reward of the trajectory generated by W, R_W

is unknown. As a result, we get a transformation $l_k \propto e^{R_k/\eta}$, where the parameter η is found by minimizing the dual function of the optimization problem

$$g(\eta) = \eta\epsilon + \eta \ log(\sum_{k=1}^{N} \frac{1}{N} e^{R_k/\eta}) \tag{6}$$

with $\eta > 0$. This dual function comes from solving the optimization problem by means of Lagrange multipliers.

3 Required multi-disciplinary knowledge

3.1 Robot geometrical structure

The key feature of a robot is its geometrical structure [10] bodies(links) interconnected by articulations (joints). A manipulator is composed by an arm that gifts mobility, a wrist that confers dexterity, and an end-effector that performs the task required of the robot.

Another fundamental feature of a manipulator is the kinematic chain. From a geometrical viewpoint, a kinematic chain is termed open when there is only one sequence of links connecting the two ends of the chain. Alternatively, a manipulator contains a closed kinematic chain when a sequence of links forms a loop. We work with WAM robots, inspired in human arms. These have an open chain, born in the base and culminate in the end-effector, which would simulate a hand in a human arm.

Mobility is ensured by joints. These are classified in prismatic and revolute joints. In an open kinematic chain, each prismatic or revolute joint provides the structure with a single degree of freedom (DOF). A prismatic joint creates relative translation between the two links, whereas a revolute joint creates a relative rotation between the two links.

The degrees of freedom should be properly distributed in the mechanical structure, in order to execute the given task. For the most general task, consisting of positioning and orienting and object in a 3D-space, 6 DOFs are required: 3 for positioning the object and 3 for orienting it. A manipulator is said to be redundant from a kinematic point of view in the number of DOFs available exceeds the number of task variables. On the case of the WAM, there are 7 DOFs: 3 in the first link (shoulder), 1 on the second(elbow) and 3 on the third (wrist), with human-like kinematics.

3.2 Dynamic Movement Primitives

3.2.1 What are DMPs?

Dynamic movement primitives (DMPs) are a method of trajectory control and planning. The first aim of this work was to find a way to represent complex motor actions that can be flexibly adjusted without manual parameter tuning or having to worry about instability. They where first presented by Auke Ijspeert in 2002 and then updated in 2013 [4].

Complex movements have long been thought to be composed of sets of primitive action. DMPs are a proposed mathematical formalization of these primitives. In robotics, among all Movement Primitives (MPs), they are the most used ones. The basic idea is that given dynamical system with well specified, stable behaviour, one may add another term that makes it follow some interesting trajectory as it goes about its business. There are two kinds of DMPs: discrete and rhythmic. We will only discuss discrete DMPs, as they are the ones that we will use.

We may have two systems: an imaginary system where we plan trajectories, and a real system where we carry them out. When we use a DMP what we are doing is planning a trajectory that will be followed by the real system. A DMP has its own set of dynamics, and by setting up your DMP properly we can get the control signal for our actual system to follow. We are not going to talk about the real system, but it is important to keep the perspective that the DMP framework is for generating a trajectory to guide the real system.

3.2.2 Building DMPs

In this section we will be using some of the work developed by Ijspeert [9].

Let y be our system state, g the goal, τ a time constant, and α and β gain terms. The following system describes our trajectory y:

$$\ddot{y}/\tau = \alpha_y(\beta_y(g - y) - \dot{y}/\tau) \tag{7}$$

This is system of second order differential equations. It is well known how to solve this kind of equations both analytically and numerically. Therefore, it is a suitable way to represent our trajectories. We usually choose $\beta_z = \alpha_z/4$ in order to make the system critically damped, so that y monotonically converges towards g. Now, we add a forcing term f to modify this trajectory:

$$\ddot{y}/\tau = \alpha_y(\beta_y(g-y) - \dot{y}/\tau) + f \tag{8}$$

In order to build the nonlinear function f to get the desired behaviour, we will use an additional nonlinear system, called canonical dynamical system:

$$\dot{x} = -\alpha_x x \tag{9}$$

The use of this system allows us to define f over time, giving the problem giving the problem a well defined structure that can be solved in a straight-forward way and easily generalizes.

In order to construct the forcing function, we need to define some more parameters. Let y_0 be the initial position of the system. Let

$$\phi_i = exp\left(-0.5\left(x-c_i\right)^2/d_i\right), \forall i \in [N_f] = \{1, \dots N_f\} \tag{10}$$

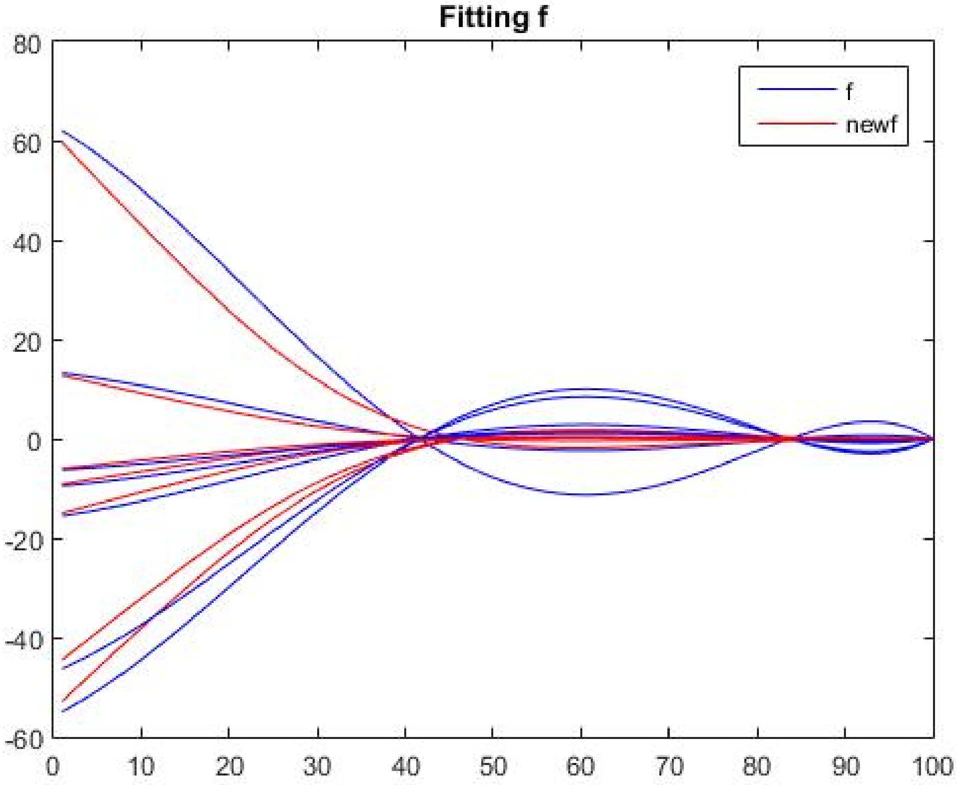
be a set of Gaussians with center at c_i and width d_i. N_f is the chosen number of Gaussians per degree of freedom. Let w_i be a set of weightings for the given basis functions ϕ_i, $\forall i \in N_f$. Now, the forcing function is:

$$f^{new}(x,g) = \frac{\sum_{i=1}^{N_f} \phi_i w_i}{\sum_{i=1}^{N_f} \phi_i} x(g-y_0) \tag{11}$$

a set of Gaussians that are activated as the canonical system x converges to its target. Their weighted summation is normalized, and then multiplied by the $x(g-y_0)$ term, which acts as both a diminishing and spatial scaling term.

In order to compute f^{new}, we have used $\psi_t^T = I_d \otimes h(x_t)^T$, being $h_i(x) = \frac{\phi_i}{\sum_{j=1}^{N_f}\phi_j}x, \forall i \in [N_f]$, and $\otimes$ the Kronecker product. Then,

$$f^{new} = \Psi_t^T w. \tag{12}$$



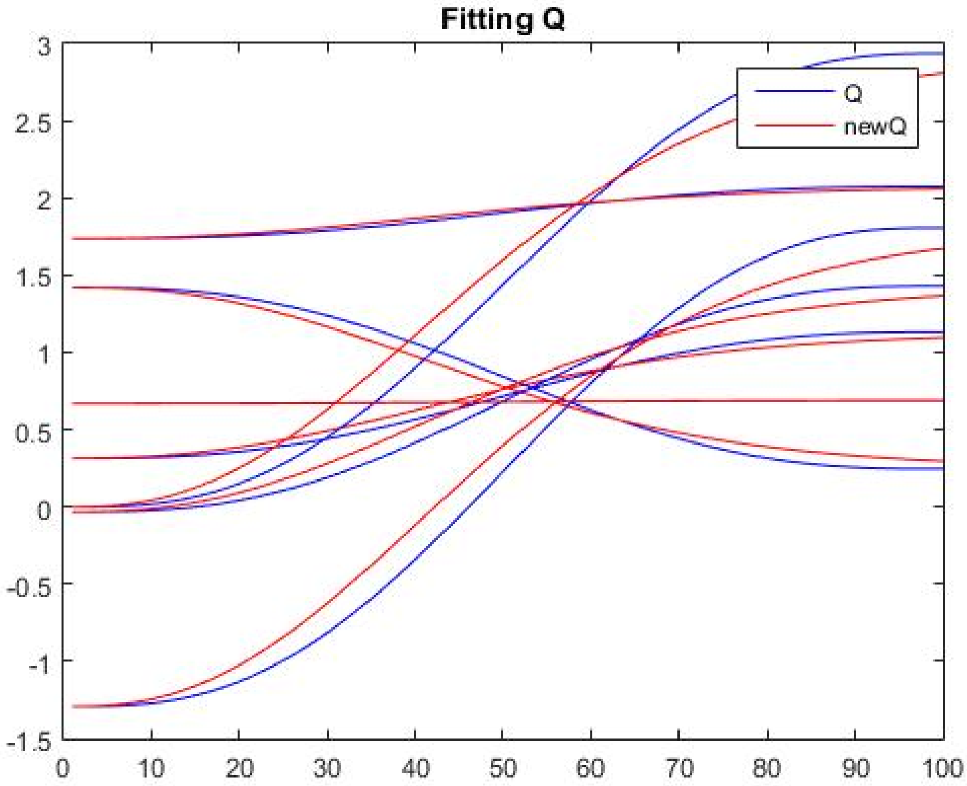
If we assumed that the canonical system starts at $x_0 = 1$ and decays to 0 (exponentially, due to the definition of x) as time goes to ∞. On the one hand, the basis functions ψ_i are activated as a function of x. As the value of x decreases from 1 to 0, each of the Gaussians are centered around different x values. On the other hand, these basis functions are also assigned a weight, w_i.

Incorporating the x term into the forcing function guarantees that the contribution of the forcing term goes to zero over time, as the canonical system does. This means that the system will eventually return to its simpler point attractor dynamics and converge to the target.

The $(g - y_0)$ term of the forcing function handles the spacial scaling. It does so by activating the basis functions ψ_i to be relative to the distance to the goal, causing the system to cover more or less distance. DMPs are key to our work, as they will be used in order to characterize the trajectories of the end-effectors of our WAMs (robot arms) by means of second order dynamical-systems.

Finally, to build the trajectories using the fitting of f^{new}, we solve the ode (8), using the new forcing excitation function:

$$\ddot{y}^{new}/\tau = \alpha_y(\beta_y(g - y^{new}) - \dot{y}^{new}/\tau) + f^{new} \tag{13}$$



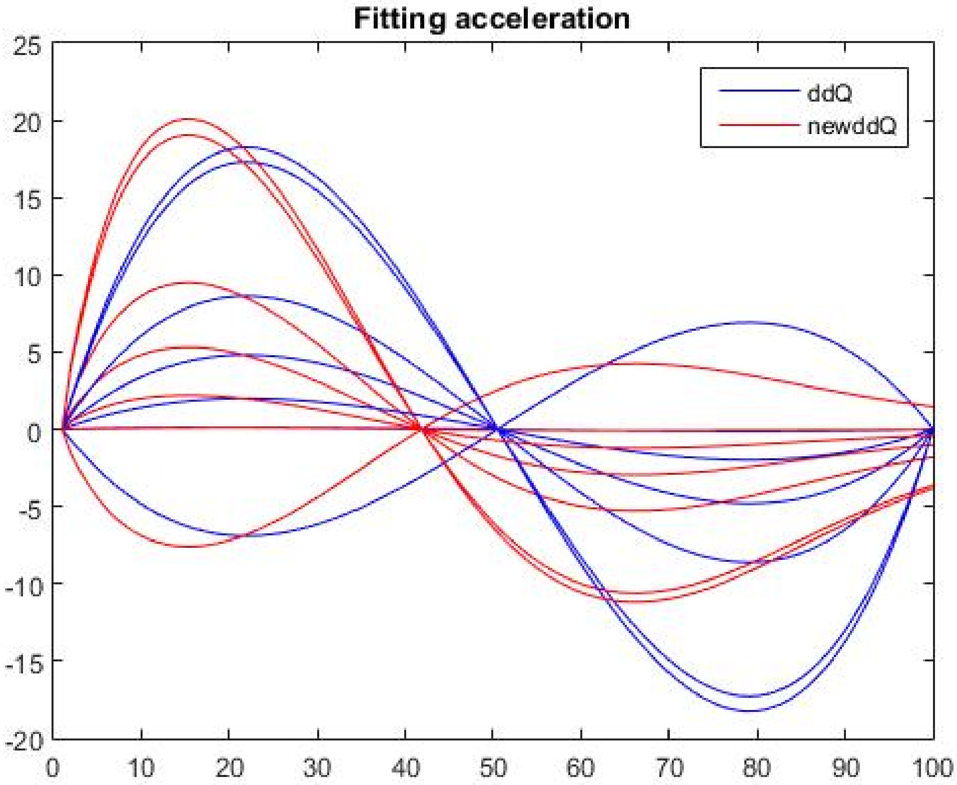
When computing this step numerically, we set $y_1^{new} = y_1$, $\dot{y}_1^{new} = y_1$ and $\ddot{y}_1^{new} = \ddot{y}_1$ and we get along $\forall i \in [N_t]$:

$$\ddot{y}_i^{new}/\tau = \alpha_y(\beta_y(g - y_{i-1}^{new}) - \dot{y}_{i-1}^{new}/\tau) + f_{i-1}^{new} \tag{14}$$

$$\dot{y}_i^{new} = \dot{y}_{i-1}^{new} + dt\ddot{y}_i^{new} \tag{15}$$

$$y_i^{new} = y_{i-1}^{new} + dt\dot{y}_i^{new} \tag{16}$$

being dt the timestep.



3.3 Principal Component Analysis

3.3.1 Goal and intuitive view

Principal Component Analysis (PCA) is a statistical procedure used to reduce the dimension of a matrix, loosing as little information as possible. It consists of an orthogonal transformation that converts a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. Let d be the initial number of variables (or observations) and r the number of principal components. Then $r \leq d$, i.e. the number of principal components is less than or equal to the smaller of the number of original variables (or the number of observations). This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. The resulting vectors are an uncorrelated orthogonal basis set. PCA is sensitive to the relative scaling of the original variables.

Although PCA is mostly used as a tool in exploratory data analysis and for making predictive models, we will use it with a different objective. Our goal will be to reduce the dimensionality of our learning process in order to diminish the number of reproductions of the motion (rollouts) needed for a sufficiently good performance.

PCA can be done by eigenvalue decomposition of a data covariance (or correlation) matrix or singular value decomposition of a data matrix, usually after mean centering the data matrix for each attribute. The results of a PCA are usually discussed in terms of component scores (the transformed variable values corresponding to a particular data point), and loadings (the weight by which each standardized original variable should be multiplied to get the component score).

This algorithm allows us to visualize the insides of the data in a way that best explains the variance in the data. We think of this in the following way. If a multivariate dataset is visualized as a set of coordinates in a high-dimensional space, PCA can supply the user with a lower-dimensional picture, that is to say, a projection of this object when viewed from its most informative viewpoint. This is done by using only the first few principal components so that the dimensionality of the transformed data is reduced.

3.3.2 Method

Let X be a $n \times p$ matrix. We think about this matrix in the following way: each row is an individual and each column is an active variable. This means that we place n individuals in a vectorial space of dimension p (we will suppose $\mathbb{R}^p$. Let G be the center of gravity of the p points. Let m be the mass of that individual and d_i the distance between the point i and G. Then, we define the inertia of the cloud of points (individuals) with respect to G as:

$$I = \sum_{i=1}^{n} md^2. \tag{17}$$

If every point has the same mass, we can see the inertia as the variance $\frac{1}{n}\sum_{i=1}^{n} d^2$, and the center of gravity G as the mean.

We want to project the cloud of points over a subspace. Our goal is to find the subspace over which the projection has maximum inertia. In other words, we want to minimize the loss of inertia in the projection. We can visualize it, for example, thinking about the shadow of a pencil. The pencil is a cloud of points in $\mathbb{R}^3$ and its shadow is its projection in $\mathbb{R}^2$. If the light is coming from the tip, the shadow will be very small. This means that we loose a lot of information regarding the shape of the pencil. Nevertheless, if the light comes from the side, the silhouette will give us much more clue of the shape of the pencil.

To build this subspace, we first find a vector u with $\|u\|= 1 = u \cdot u'$ such that when we project the cloud over the straight line defined by $G + \{u\}$, the inertia is maximum. Now, we project using the scalar product:

$$\Psi = \begin{bmatrix} \psi_1 \\ \vdots \\ \psi_n \end{bmatrix} = X \cdot u \tag{18}$$

ψ_i are the principal components. They are *artificial* variables.
The inertia of the projection is $\frac{1}{n}\Psi\Psi \sum_{i=1}^{n} \psi_i^2 = \Psi' N \Psi$, being N a matrix of weights w_i, which verify $\sum_{i=1}^{n} w_i = 1$.

$$N = \begin{bmatrix} w_1 & & \\ & \ddots & \\ & & w_n \end{bmatrix} \tag{19}$$

These weights are usually $w_i = \frac{1}{n}, \forall i \in [n]$. In this case, the inertia is $\frac{1}{n}\Psi\Psi'$

In terms of X, the inertia associated to the projection using the vector u is:

$$I_u = u'X'NXu. \tag{20}$$

Now we want to maximize this expression. To do so, we calculate the Lagrangian $L = u'X'NXu - \lambda(u'u - 1)$ and solve the equation $\frac{\delta L}{\delta u} = 2X'NXu - 2\lambda u = 0 \longrightarrow X'NXu = \lambda u$.

We want $u = u_1$ to be an eigenvector of $X'NX$ with associated eigenvalue $\lambda = \lambda_1$, being λ_1 the highest eigenvalue of $\Psi'N\Psi$. Taking u_2 the eigenvector which is associated to the second greatest eigenvalue λ_2. The plane $\{u_1, u_2\}$ is a projection plane where there is maximum inertia $I = \lambda_1 + \lambda_2$.

The total inertia is:

$$\begin{aligned} I_T &= \sum_{i=1}^{n} \frac{1}{n} d^2(x_i, G) \\ &= \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{p} (x_{ij} - \bar{x_j})^2 \end{aligned} \tag{21}$$

$$= \frac{1}{n} \sum_{j=1}^{p} \sum_{i=1}^{n} (x_{ij} - \bar{x_j})^2 = \sum_{j=1}^{p} s_j^2$$

being s_j^2 is the $j - th$ variance.

We know that the covariance matrix $X'NX$ is symmetrical. Thus, its eigenvectors are orthogonal. Taking eigenvectors $u_1, \ldots, u_p$ with associated eigenvalues $\lambda_1 > \ldots > \lambda_p$ as the orthogonal base of a the projection subspace, we will have total inertia:

$$I_T = \sum_{i=1}^{p} \lambda_i, \tag{22}$$

as $diag(X'NX) = \frac{1}{n} \sum_{i=1}^{p} (x_{ij} - \bar{x_j})^2 = s_j^2$. The inertia associated to an eigenvector is the eigenvalue associated to it.

In order to know which principal components are significant, we use the ratio $\tau_j = \frac{var(x_j)}{I_T} = \frac{s_j^2}{\sum_{i=1}^{p} s_i}$. There are several criteria to split significant and non significant components, such as the *Kaiser rule* or the *Last elbow rule.* To give a common measure to each variable, we normally normalize our data before performing PCA, i.e. we apply the transformation $x_{ij} = \frac{x_i j - \bar{x_j}}{s_j}$, $var(x_j) = 1, \forall j \in [p]$, $I_T = p$, $\bar{\lambda_i} = 1$. With this normalization, $X'NX$ is the correlation matrix.

3.4 Moore-Penrose Pseudo-inverse

Let A be a $m \times n$ matrix $(m>n)$ and b a column vector of length m. The method of least squares is a way of solving an overdetermined system of linear equations $Ax = b$. The goal of the least squares method is to minimize the sum of the squares of the errors. In general, for an overdetermined mn system Ax = b, there are solutions x minimizing $\|Ax - b\|^2$. These solutions are given by the square $n \times n$ system $A^T Ax = A^T b$[8].

The minimum norm least squares solution x^+ can be found in terms of the pseudo-inverse A^+ of A. Let

$$A = VDU^T, \tag{23}$$

where $D = diag(\lambda_1, ..., \lambda_r, 0, ..., 0)$ is an mxn matrix $(\lambda_i>0, \forall i)$. Let $D^+ = diag(1/\lambda_1, ..., 1/\lambda_r, 0, ..., 0)$ an nxm matrix. The pseudo-inverse of A is defined as

$$A^+ = UD^+V. \tag{24}$$

The following results is the main reason for us to use the Moore-Penrose pseudo-inverse:

The least-squares solution of smallest norm of the linear system $Ax = b$, where A is an $m \times n$ matrix, is given by

$$x^+ = A^+b = UD^+V^Tb$$

4 Implementation

In this section we will explain how we have coded the learning framework. Some of the concepts, approaches and algorithms explained in the previous chapters are key items of the success of this motion learning model. We will present a framework thought for the motion learning of one WAM. The same procedure could be implemented in parallel for two WAM arms in order to learn the motion of each arm. In the next chapter, we will introduce some notions concerning the improvements of the two WAM motion learning.

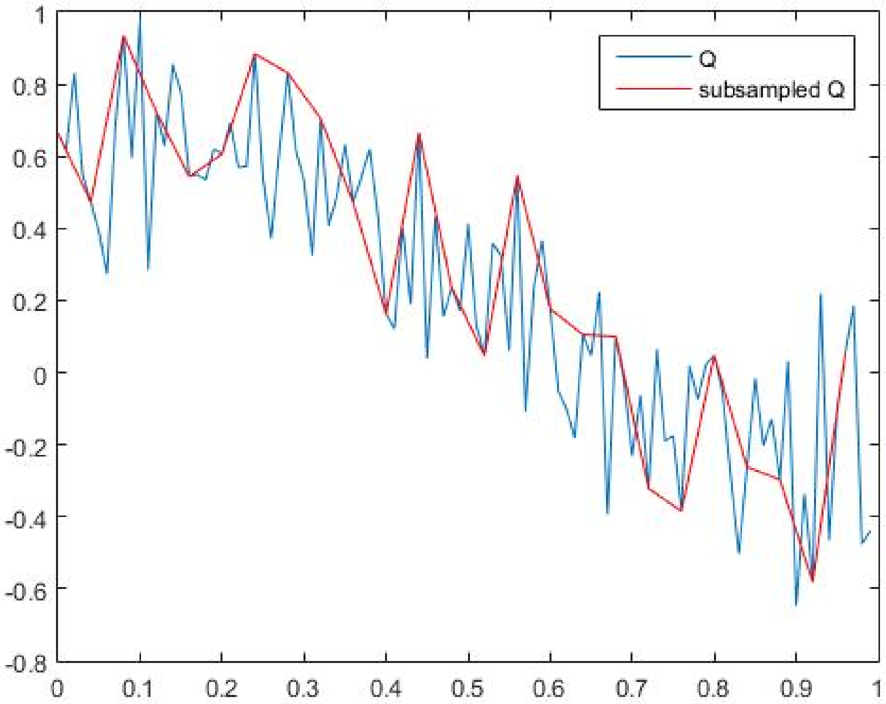
4.1 Process and code

In order to make it more understandable for the reader, we will accompany the explanation with a naive 1-dimensional example.

Firstly, we put the robot in gravity compensation mode and kinesthetically teach the desired trajectory.
we manually make the robot do the desired trajectory. The motion is saved in a computer, so that we know at every time-step (0.06 seconds) the position, speed and acceleration of every join. Let Q be a matrix that stores the time in the first column, the position of the d joints in the $2, \dots, d+1$ columns, the speed of every joint in the $d+2, \dots, 2d+1$ and the accelerations in the columns $2d+2 \dots 3d+1$. Each row represents a time-step.

We may have noisy data. If that is the case, in order to smooth it, so that we can work with it, we take a subsample of the initial trajectory. For example, we might take one of every four time-steps. We must adjust the subsampling so that we have enough information, but we can also work with it in a proper way.

Another method to smooth the noise in our data is to filter the acceleration. Being K a constant and a_t the acceleration, we compute $a_t = a_tK + a_{t-1}(1-K), \forall t \in [N_t]$, where N_t is the number of timesteps. With



the trajectory matrix notation, this would be: $Q(i,:) = Q(i, 2d+2 : 3d+1)K + Q(i-1, 2d+2 : 3d+1)(1-K)$. In order to simplify the formulation, we will assume the following equalities:

$$y_i = Q(i, 2, d+1) \tag{25}$$

$$\dot{y}_i = Q(i, d+2, 2d+1) \tag{26}$$

$$\ddot{y}_i = Q(i, 2d+2, 3d+1) \tag{27}$$

Once we can start working with the data, we choose gain terms α_y, β_y (usually $\beta_y = \alpha_y/4$, to make the ode critically damped) and the goal g in order to start computing DMPs. Here is a reminder on how we build DMPs, with a more numerical approach.
We compute the excitation function:

$$f_i = \ddot{y}/\tau - \alpha_y(\beta_y(g-y) - \dot{y}/\tau) \quad , \quad \forall i \in [N_t]$$

and solve the canonical dynamical system:

$$x_i = exp(-\alpha_x t_i) \quad , \quad \forall i \in [N_t]. \tag{28}$$

being t_i the time until the $i-th$ timestep. Now, we choose the centers C and widths D of the Gaussians, so that they are balanced:

$$c = [\frac{\tau}{N_f+1}, \dots, \frac{i\tau}{N_f+1}, \dots, \frac{N_f\tau}{N_f+1}] \tag{29}$$

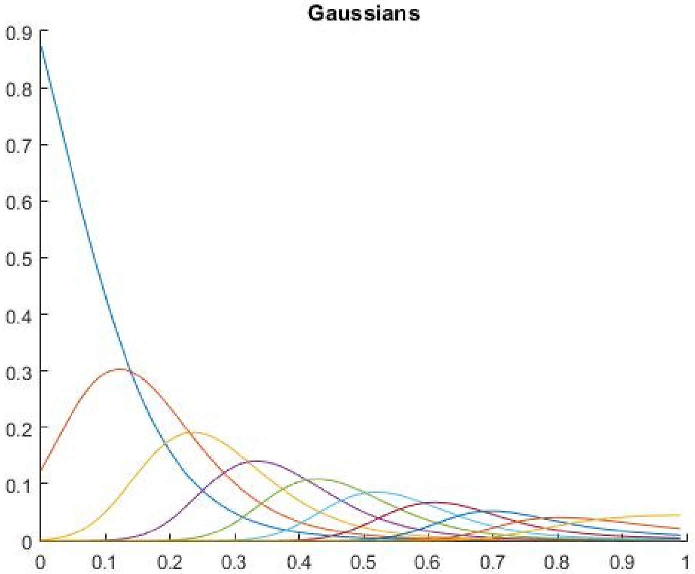
$$C = exp(\frac{-\alpha_x c}{\tau}) \tag{30}$$

$$D = (\frac{diff(C)}{0.55})^2, \tag{31}$$

where $diff(C)$ is a vector that stores $C_i - C_{i-1}, \forall i = 1 \dots N_f$ and the parameter 0.55 is found empirically.

Then, we calculate the set of Gaussians:

$$\phi_{i,j} = exp(\frac{(-x_i - c_j)^2}{D_j}) \ , \ \forall i \in [N_t] \ , \ \forall j \in [N_f] \tag{32}$$



and the function:

$$h_{i,j} = \frac{x_i \phi_{i,j}}{\sum_{j=1}^{N_f} \phi_{i,j}} \ , \ \forall i \in [N_t] \ , \ \forall j \in [N_f]. \tag{33}$$

Let A be the pseudo-inverse of h. To store the weights, we define $w = Af$.

$$W_{(j-1)N_f+i} = w_{i,j} \ , \ \forall i \in [N_f] \ , \ \forall j \in [d], \tag{34}$$

which means

$$W = \begin{bmatrix} w_1 \\ \vdots \\ w_{N_f} \end{bmatrix}. \tag{35}$$

Thus, the fitting of f is:

$$f_i^{new} = (\,(I_d \otimes h_i, :)\ W\,)^T \ , \ \forall i \in [N_f] \ , \ \forall j \in [d] \tag{36}$$

and the fitting of the trajectory, initializing with $Q^{new}(:,1) = Q(:,1)$, $y_1^{new} = y_1$, $\dot{y_1}^{new} = \dot{y}_1$, $\ddot{y_1}^{new} = \ddot{y}_1$ and updating $\forall i = 2 \ldots N_t$:

- Position:

$$\ddot{y_i}^{new} = \tau(\alpha_z(\beta_z(g - y_i^{new}) - \dot{y_i}^{new}/\tau) + f_{i-1,:}^{new})$$

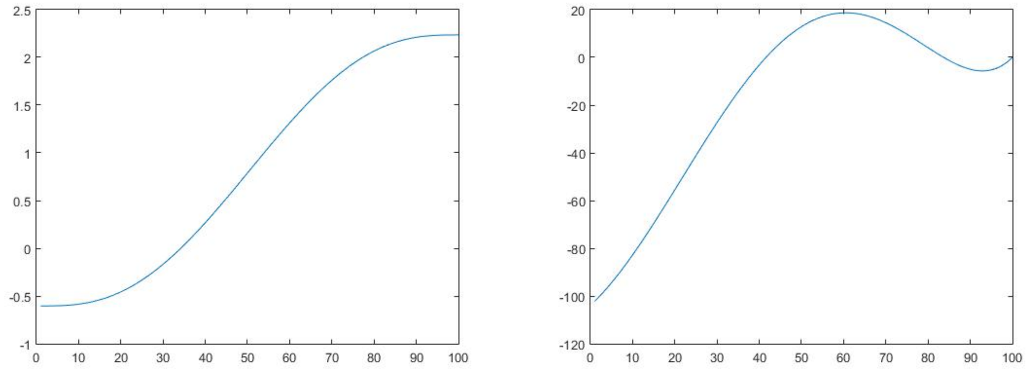
- Speed:

$$\dot{y_i}^{new} = \dot{y_i}^{new} + dt\ddot{y_i}^{new} \tag{37}$$

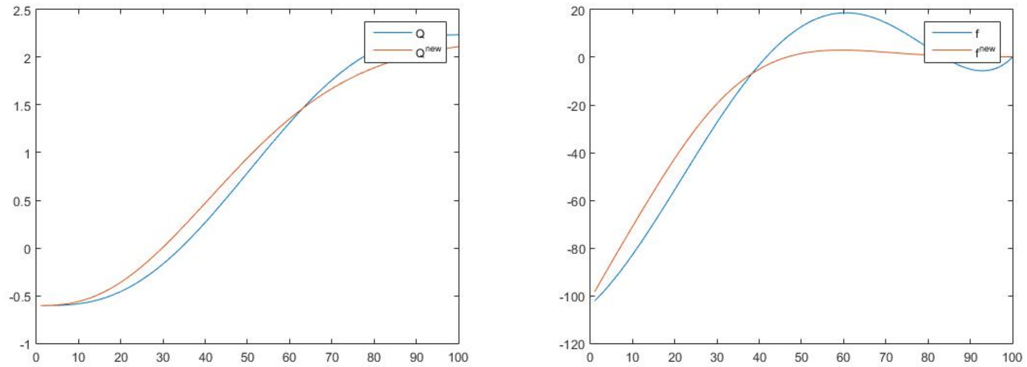
- Acceleration:

$$y_i^{new} = y_i^{new} + dt\dot{y_i}^{new} \tag{38}$$

For our naive example we will be working with the following trajectory:



After computing the DMPs, we compare Q^{new} and f^{new} with Q and f. We have used $\alpha_z = 12$, $\beta_z = \frac{\alpha_z}{4}$, $N_f = 10$ and $\alpha_x = 2.5$.



Once performed the DMPs, we start running a Policy Search algorithm. To do so, we must specify a reward function, according to the aim. In our example, we want the trajectory to be the closer the better to a point in a certain time-step. Let $p_Q = 0.4$ be the point and $p_t = 50$ the time point where we want p_Q to be reached. In the example, the reward function is:

$$R = mean(-|y - p_Q| - 0.000015 \sum \ddot{y}^2), \tag{39}$$

which takes into account how close the trajectory is to p_Q at p_t. It also considers high accelerations to be high, as it may be dangerous.
We set $\lambda = 50$ to initialize our covariance matrix $\Sigma = diag(\lambda)$. The mean starts being $\mu = W$. Let N_{upd} be the number of updates of μ and σ. Let

N_k be the number of samples created from each update of μ and σ.

For each update, we create N_k samples $W_k, k = 1 \dots N_k$. These are created according to a $(\mu, \Sigma)-$normal distribution, i.e. $w_k \sim N(\mu, \Sigma)$ and are $(N_f d)-$weight vectors. Let W be a matrix that stores the $k - th$ vector of motion parameters in the $k - th$ column. For each one of these samples, we create a new excitation function

$$f^{new}(x) = \Psi^T w_k \tag{40}$$

being $\Psi^T = I_d \otimes h(x)$ and $W = [w_1, \dots, w_{N_k}]$. Expressed in a more coded way:

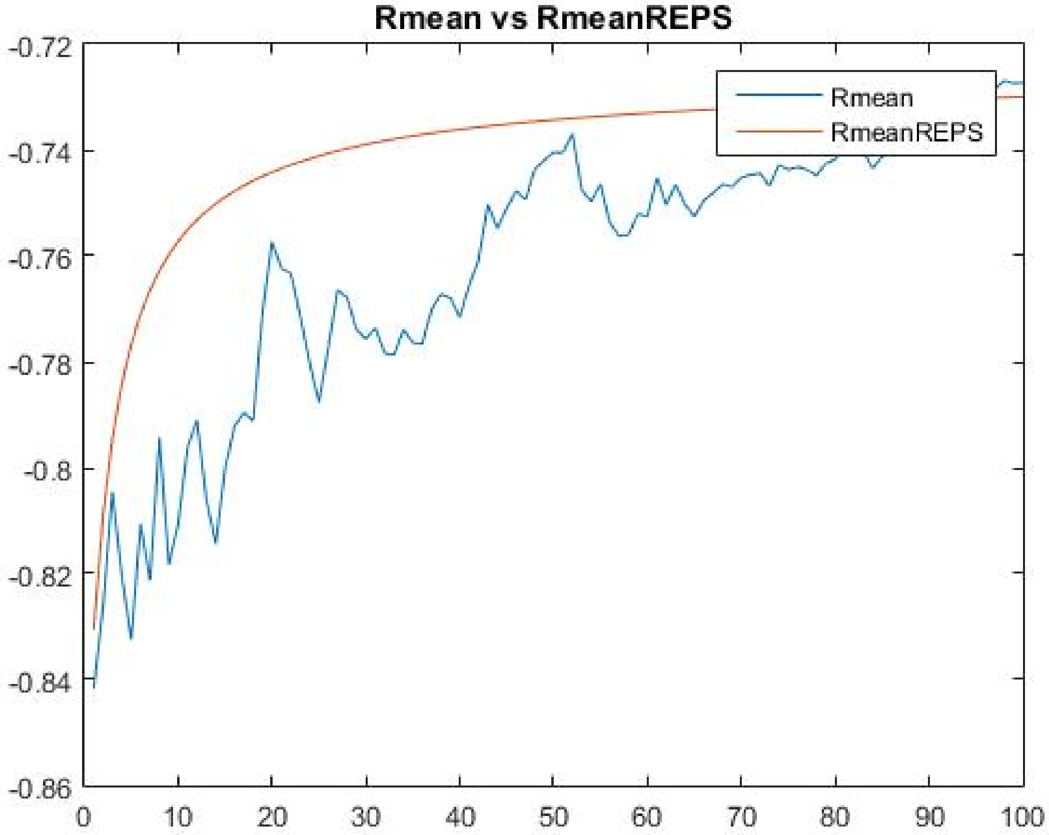
$$f_i^{new} = ((I_d \otimes h(+_{i,:})\ W_{:,k})^T\ ,\ \forall i \in 1 \dots N_f\ ,\ \forall j \in 1 \dots d. \tag{41}$$

and create a new trajectory Q^{new}, in the same way that we did the first time. We evaluate the function reward, inputing Q^{new} and store this reward, as well as the excitation function f^{new}.

Having evaluated the reward of the trajectories created with every sample, we update the parameters of the normal distribution: μ and Σ. To do so, we have used both Weighted Maximum Likelihood Policy Search (WMLPS) and Relative Entropy Policy Search (REPS). As explained in previous chapters, REPS is usually used to palliate the greediness and instability of Weighted Maximum Likelihood.

Once we have updated μ^{new} and Σ^{new}, we use $f^{new} = \mu^{new}$ in order to create a new trajectory Q^{new}. The policy update involves the reward function and the samples. This is where the algorithm is actually learning, by giving more importance (weight) to the trajectories with less reward (in absolute value). We also evaluate the reward function with this new trajectory.

In the following figures, we can observe the evolution of the reward function obtained with both methods, considering only the reward of the trajectory created using μ^{new} updated with the policy search method: As we can see,



REPS is performing better at every time-step. As mentioned earlier in this section, it is a more stable method. Therefore, its curve is smoother than the one provided by WMLPS. Focusing on the REPS curve, we can see that it is not monotonous. Instead, it decreases as expected and then grows a little bit. This might be due to numerical errors in the calculus.

This process leaves as an output, a trajectory that has learned from every policy update, and every trajectory with its associated reward.

5 DOF reduction

We have presented a framework for motion learning. This model requires several rollouts in order to find a proper policy update. Moreover, many parameters are needed to achieve a good fitting of the initial trajectory. When applying DMPs in learning models, we must bear in mind several considerations:

- Reinforcement learning can be performed through simulation or with a real robot. If we have a good simulator of the robot and its environment, we will go through simulation. Nevertheless, when accurate models are not available, we will have to use a real robot. In this case, reducing the number of parameters and iterations (rollouts) is vital.
- Moreover, certain tasks might not depend on all the Degrees of Freedom (DoFs) of the robot. In this case, the algorithm may be exploring motions that are meaningless to the task. Also some exploration values could generate strong oscillations, rapid accelerations and other dangerous motions.
- In addition, complex robots usually require many parameters for a proper trajectory representation. The number of parameters is strongly dependent on the trajectory length. In a 7-DoF robot like the WAM that we are using, we use over 20 Gaussian kernels per joint for a 20-second trajectory, meaning that we might have more than 140 parameters in total. The more parameters we use, the better the fitting of the initial trajectory will be, generally speaking. However, this a large number of parameters may result in a slower learning. In this sense, there is a tradeoff between better exploitation (many parameters) and efficient exploration (fewer parameters).

For these reasons, performing Dimensionality Reduction (DR) on the DMPs DoF is an effective way of dealing with the setoff between exploration and exploitation in order to help the algorithm converge faster to a likely better solution.

5.1 General case

In this section, we will describe how to reduce the dimensionality of the problem with a coordination framework for DMPs [3].

To achieve our goal, we will perform a different fitting of the excitation function:

$$f(x_i) = \Omega \Psi_i^T W \ , \ \ \forall i \in |N_t| \tag{42}$$

being Ω a $(d \times r)$ matrix ($r < d$ is the dimension that we want to reduce to), $\Psi_i^T = I_r \otimes h(i,:)$ (I_r is the identity matrix of dimension $r \times r$) and W the matrix of samples of motion parameters, as in the previous chapter. This representation is equivalent to having r movement primitives codifying the d-dimensional vector $f^{new}(x)$. Intuitively, the columns of Ω represent the couplings between the robots DoF.

In order to learn the coordination matrix Ω, we need an initialization, an updating method and an algorithm to eliminate needless DoFs from the DMP, according to the reward. We can assume that the probability of having certain excitation function values $f_i = f(x_i)$ at a timestep, given the sample of weights W, is $p(f_i|W) \sim N(\Omega\Psi_i^T W, \Sigma_f)$, with Σ_f being the system covariance (noise). Thus, if $W \sim N(\mu_W, \Sigma_W)$, the probability of f_i is:

$$P(f_i) = N(\Omega\Psi_i^t \mu_W, \Sigma_f + \Omega\Psi_i^T \Sigma_W \Psi_i \Omega^T) \tag{43}$$

the coordination matrix Ω can be initialized with a Principal Component Analysis(PCA) [3]. Particularly, we will perform it over the values of $f = \Psi^T W$, with Ψ calculated with the identity matrix of $d \times d$ dimensions (this is the first fitting of the excitation function). To do so, we build the $(d \times N_t)$ matrix:

$$\bar{F} = \begin{bmatrix} f_{de}^{(1)}(x_0) - \bar{f}_{de}^{(1)} & \dots & f_{de}^{(1)}(x_{N_t}) - \bar{f}_{de}^{(1)} \\ \dots & & \dots \\ f_{de}^{(d)}(x_0) - \bar{f}_{de}^{(d)} & \dots & f_{de}^{(d)}(x_{N_t}) - \bar{f}_{de}^{(d)} \end{bmatrix} \tag{44}$$

being $\bar{f}_{de}$ the joint average of the DMP excitation function, for the demonstrated motion (f_{de}). Note that $\bar{F}$ is the matrix of all the N_t timesteps $f_i^{(j)}$ and d joins. Afterwards, we Perform Singular Value Decomposition (SVD) of $\bar{F}$, so we have $\bar{F} = U_{pca}\Sigma_{pca}V_{pca}^T$.

The next step in this method is to take the r eigenvectors with the highest singular values as the initialization of coordination matrix Ω. These eigenvectors are the first r columns of U_{pca}, thus $\Omega = [u_1, \dots, u_r]$ and their associated singular values $\sigma_1 > \dots > \sigma_r$, being $U_{pca} = [u_1, \dots, u_r, \dots u_d]$ and $\Sigma_{pca} = diag(\sigma_1, \dots, \sigma_r, \dots, \sigma_d)$. This way, we minimize the error in the reprojection $e = \|\bar{F} - \Omega\bar{\Sigma}V^T\|^2_{Frob}$ (with $\bar{\Sigma} = diag(\sigma_1, \dots, \sigma_r)$), while reducing the set of DoF of dimension r, which activate the robot joints (dimension d).

Now, we are going to describe how to update the coordination matrix in every rollout. We assume that we have performed N_k reproductions of the motion, namely rollouts. Thus, we obtain the excitation function $f_i^{(j),k}$, for every time-step $i = 1 \dots N_t$, rollout $k = 1 \dots N_k$ and DoF $j = 1 \dots d$. Now, we evaluate the trajectories attached to every excitation function with the reward function. Then, we can associate a relative weight P_i^k to each rollout and timestep, regarding the reward values. These weights are given by the policy search algorithms that we are using (WMLPS and REPS). Thus, we obtain a new matrix

$$F_{co}^{new} = \begin{bmatrix} \sum_{k=1}^{N_k} f_1^{(1),k} P_1^k & \dots & \sum_{k=1}^{N_k} f_{N_t}^{(1),k} P_{N_t}^k \\ \dots & & \dots \\ \sum_{k=1}^{N_k} f_1^{(d),k} P_1^k & \dots & \sum_{k=1}^{N_k} f_{N_t}^{(d),k} P_{N_t}^k \end{bmatrix} \tag{45}$$

This $d \times N_t$ matrix contains the excitation functions weighted by their importance according to the rollout result. Then, the coordination matrix can be updated by PCA. Nevertheless, we have to refit the parameters μ_W, Σ_W to make the trajectory representation fit the same trajectory. Let $\hat{\mu}$ and $\hat{\Sigma}$ be the old distribution parameters.

We want to minimize the loss of information. To do so, we will minimize the Kulbach-Leibler divergence between $\hat{p} \sim N(\hat{\mu}_W, \hat{\Sigma}_W)$ and $p \sim N(M\mu_W, M\Sigma_W M^T)$, being $M = (\hat{\Omega}\hat{\Psi}_i^T)^+ \Omega \Psi_i^T$ (+ is the Moore-Penrose pseudo-inverse operator).

Derivating $KL(\hat{p}||p)$ with respect to μ_W and Σ_W and equating the derivative to zero, we obtain the following updates of the policy parameters:

$$\mu_W = M^+ \hat{\mu}_W \tag{46}$$

$$\Sigma_W = M^+(\hat{\Sigma}_W + (M\mu_W - \hat{\mu}_W)(M\mu_W - \hat{\mu}_W)^T)M^{T,+} \tag{47}$$

Being this the parameter update that minimizes the KL divergence, it results in the update with the least loss of information, in terms of probability distribution on the excitation function, i.e. we minimize the loss of inertia of the excitation functions created with these parameters with respect to the old parameters.

Currently, in reinforcement learning, not all the DoF affect the task the the robot tries to learn. However, these DoF are still considered through all the learning process, causing a slowdown in the learning process or result in motions in which a part of the trajectory may not be necessary. Hence, the use of the coordination matrix build as we described outfits the framework, removing unnecessary DoF. With the described framework we achieve to reduce the dimensionality from d to r.

5.2 Symmetric tasks

Once described a method to reduce the number of DoF in a generic motion, we will now focus in bimanual tasks which in a part of the trajectory, include motion symmetries between end-effectors. The goal of this section is to pose a methodology that uses motion symmetries is bimanual tasks in order to reduce the number of parameters.

We will operate in $\mathbb{R}^3 \times t$, which is the euclidean space that is usually used to describe 3-dimensional trajectories (we will consider the trajectory points as 3-d points $p = (x, y, z)$). We have chosen to work in the Cartesian coordinates instead of working in the joint space for two reasons:

- It is a more intuitive space to work in and it is easier to interpret.
- Sometimes, symmetries might be occurring in the trajectories (between end-effectors), but the joints of both WAMs may be working in a different way in order to make the end-effector execute that motion. Therefore, we simplify our problem, as we do not have to solve the inverse kinematic problem (given the end-effector's trajectory, determine the motion of the joints), which is more complex than the direct kinematic problem (given the motion of the joints, determine the end-effector's trajectory).

However, we will give a general methodology, so that it can be applied over any finite-dimension space, such as the joint space.

5.2.1 Method

Essentially, we propose to build linear varieties (which are easy to characterize) at every rollout, such that minimize the distance between the symmetric trajectory of a WAM (with respect to the variety) and the trajectory of the other WAM

$$Min_{\mathrm{V}} \sum_{i=1}^{N_t} \|\mathrm{mirror}(yc_i^{(1)}, V) - yc_i^{(2)}\| \tag{48}$$

being $yc_i^{(j)}$ the position of the trajectory of the WAM j at the timestep i in the Cartesian space (in the same reference) and the norm is the euclidean norm. The function *mirror* is the function that given a linear variety V as a system of equations, and a point p, returns as an output the symmetric point of p with respect to V.

In order to obtain the trajectories in the Cartesian space, we have solved the direct kinematics problem with a `Matlab` function from `robot` library (by Peter Corke), called `fkine`. We have also performed a change of reference of the second WAM reference to the first.

$$\begin{tikzcd}
JointSpace2 \arrow[d, "\texttt{fkine}"'] & JointSpace1 \arrow[d, "\texttt{fkine}"] \\
CartestianSpace2 \arrow[r, "T"] & CartesianSpace1
\end{tikzcd} \tag{49}$$

being T the change of reference matrix.

Once we have the trajectories in the same Cartesian space, we compute the curve of middle points $y_i^{(mp)} = \frac{y_i^{(1)}+y_i^{(2)}}{2}$. Let n be the dimension of the space that we are working in (in our case, $\mathbb{R}^3$, $n = 3$). Let $n - m$ be the dimension of the varieties with respect to which we want to make symmetries (m is the number of equations that define a variety). Now, from the curve $y^{(mp)}$, we take $m + 1$ linearly independent points $p_0, \ldots, p_m$. Let X be the matrix that stores these points:

$$X = \begin{bmatrix} p_{0,1} & \cdots & p_{m,1} \\ \vdots & & \vdots \\ p_{0,n} & \cdots & p_{m,n} \\ 1 & \ldots & 1 \end{bmatrix} \tag{50}$$

By solving the system $X^T A^T = 0^T$, we obtain the equations (whose coefficients are stored in A) that define the m-dimensional linear variety V that contains $p_0, \ldots, p_m$. This variety will be the starting point from which we

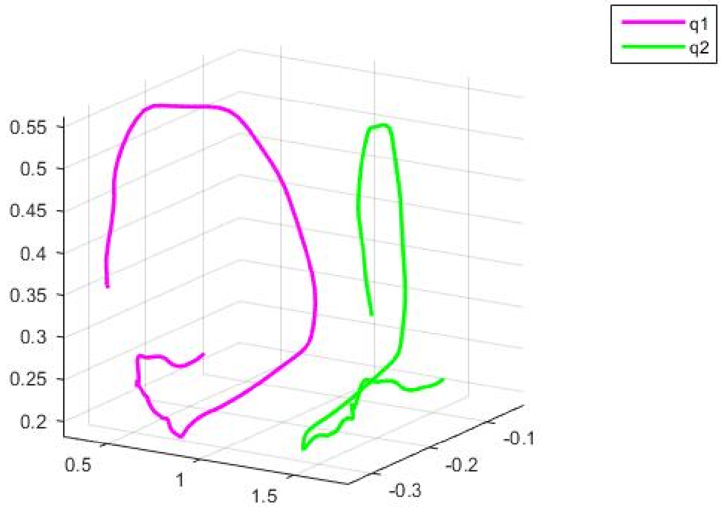


Figure 1: Symmetric trajectories

will start to search the minimum.

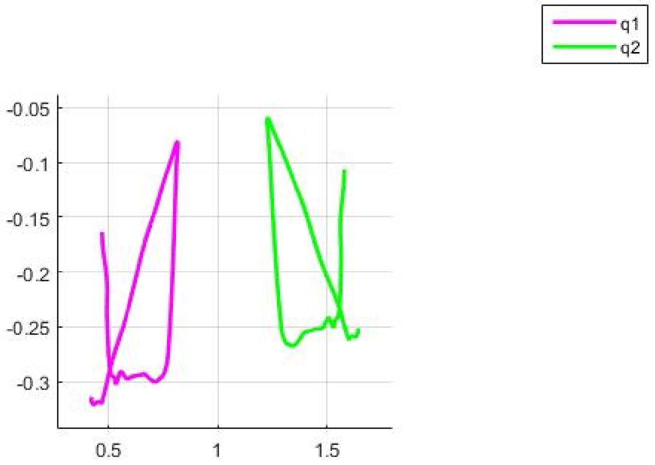


Figure 2: Symmetric trajectories

We choose samples from the middle points curve in order to bound the search in a logical way, given the aim of the problem. Moreover, to im-

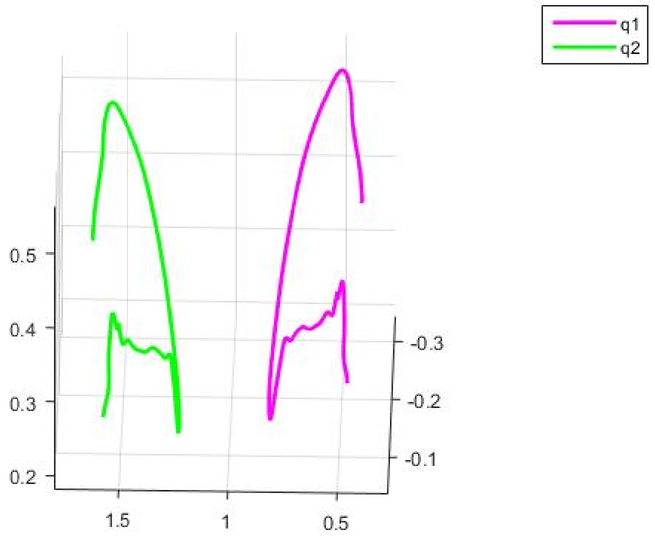


Figure 3: Symmetric trajectories

prove this choice, we split the curve in $m+1$ uniform parts and we take one point from each part.

Having an initial guess and a function, we can implement a minimization algorithm. We have implemented Gradient Descent, which is a first-order iterative optimization algorithm. This algorithm takes steps proportional to the negative of the gradient of the function. Let θ_0 be the initial guess and F the function that we want to minimize. Gradient Descent iterates:

$$\theta_{k+1} = \theta_k - \alpha \nabla F(\theta_k) \tag{51}$$

being α a constant that determines how much we advance in the gradient direction. We have applied a criteria that makes the algorithm take bigger steps when we are decreasing and smaller when increasing:

$$F(\theta_{k+1}) > F(\theta_k) \Longrightarrow \alpha = 1.2\alpha \tag{52}$$

$$F(\theta_{k+1}) < F(\theta_k) \Longrightarrow \alpha = 0.5\alpha \tag{53}$$

This is a simple algorithm that finds local minimums. However, it converges fast and it is good enough for us. In these kind of problems, quick algorithms like this are commonly used, as well as heuristics such as random search or genetic algorithms. As we are finding local minimums, we create diverse sets of points, i.e. initial varieties, and we perform various Gradient Descents in order to find a *good enough* local minimum.

Once we have a sufficiently good solution to our problem, i.e. a linear variety that minimizes (or is close to minimize) the distance between the mirror trajectory of WAM_1 motion and WAM_2 motion; we create this symmetric trajectory $y^{(m)}$ and evaluate the reward function considering two trajectories: the one created for WAM_1 with the algorithm described in chapter 4 (implementation) with the DoF reduction that we presented for general cases, and the $y^{(m)}$. This process is computed for every trajectory generated in order to make WAM_1 learn its task. Thus, the robot is learning, at each rollout, from both the trajectory of the WAM_1 and its mirror motion, reducing the DoF number from d to $r/2$.

In $\mathbb{R}^3$, we consider 3 kinds of variety: points, lines and planes. Each type is useful for a specific type of symmetry, depending on the number of dimensions over which there is symmetry and if they are direct or inverse symmetries.

6 Conclusions

We have presented a framework for motion learning and implemented it with a WAM robot with success. In addition, we have reduced the dimensionality of the problem from dimension d to $r<d$, exploring just in the most significant directions. Finally, we have developed a method which uses symmetries between trajectories in bimanual tasks (two WAM robots) in order to reduce the number of DoFs. We have been able to reduce such number from d to $r/2$ in this particular case. This symmetry model is thought to work with any kind of symmetry and dimension.

Our next step will be to test the symmetry model with several tasks, which may involve different kinds of symmetries and measure its performance in terms of reward per update. Moreover, we will develop a methodology to split trajectories based on the relationship between the trajectories of both arms (kind of symmetry by segment of trajectory).

References

[1] Deisenroth, M. P., Neumann, G., & Peters, J. (2013). Introduction(pp. 4). *A survey on policy search for robotics.* Foundations and Trends in Robotics, 2(12), 1-142.

[2] Colomé, Adrià and Torras, Carme. *Dimensionality Reduction for Dynamic Movement Primitives and Application to Bimanual Manipulation of Clothes*, submitted 2016.

[3] A. Colomé and C. Torras. *Dimensionality reduction and motion coordination in learning trajectories with dynamic movement primitives*, 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2014, Chicago, pp. 1414-1420

[4] DeWolf, Travis. Dynamic Movement Primitives Part 1: the basics <*https://studywolf.wordpress.com/2013/11/16/dynamic-movement-primitives-part-1-the-basics/*>, 2013

[5] Siciliano, B. et al (2007). Part A. AI Reasoning Methods for Robotics (pp. 336). *Handbook of robotics.*

[6] Kullback, S. and Leibler, On information and sufficiency (pp. 79-80), *Annals of Mathematical Statistics*, 1951.

[7] Gradient Descent. <*https://en.wikipedia.org/wiki/Gradient_descent*>.

[8] Gerig G. *Least Squares, Pseudo-Inverses, PCA & SVD.* <*http://www.sci.utah.edu/~gerig/CS6640-F2012/Materials/pseudoinverse-cis61009sl10.pdf*>. University of Utah.

[9] Ijspeert, A. J., Nakanishi, J., Hoffmann, H., Pastor, P., & Schaal, S. (2013). *Dynamical movement primitives: learning attractor models for motor behaviors.* Neural computation, 25(2), 328-373.

[10] Siciliano, B. (2009) Chapters 1-3 (pp. 1-159) *Robotics modelling, planning and control.*

Appendix: Code

```
% main min_jerk

close all
clear variables

% generem trajectria
d=7;
min_jerk_trajectory;

% Notaci i guardar parmetres donats
aux = size(T);
Q = zeros(aux(1),3*d+1);
Q(:,1) = T;
Q(:,2:d+1) = y;
Q(:,d+2:2*d+1) = yd;
Q(:,2*d+2:3*d+1) = ydd;

plot(Q(:,2))

Nt = length(Q(:,1));
Nf = 10;
alphaz = 12;
alphax = 2.5;
betaz = alphaz/4;
%tau = 20.1;
K = 0.8;
g = Q(Nt, 2:d+1);
dt = Q(2,1)-Q(1,1);

% carreguem les dades de la trajectoria, trobem els parmetres i la f i
% filtrem l'acceleraci
[ddy_filt,f,x,Q,Nt,g] = ini_filt_param(Q,tau,d,Nf,alphax,alphaz,betaz,K,g);

% calculem les gaussianes i fem el plot
% per on volem que passi
Q1 = [0.4, 0.7, 0.5, 0.45, 0.3, 0.6, 0.55];
t1 = 50;
lambda = 1;
Nk = Nf*d + 5;

% calculem gaussianes i la matriu P
[Ct,C,D,W,p,P] = gaussianes(Q,Nf,tau,alphax,x,d);

% troba els pesos, calcula la nova f i fa plots
[new_f,w,A,new_Q,Id,w_aux] = pesos(P,Nf,d,x,f,Q,Nt,alphaz,betaz,g,dt,tau,lambda)
    ;

% guardem alguns parmetres per a poguer fer policy (actualitzant mu i sigma amb
     Maximum Likelihood)
%i policyREPS (acutalitzant amb REPS) per separat
lambda = 5000;
```

```
Q1ini = Q1;
t1ini = t1;
lambdaini = lambda;
Qini = Q;
new_Qini = new_Q;
wini = w;
%f_newini = f_new;
new_fini = new_f;
Pini = P;

Nupdates=100;
% genera mostres a partir de la normal i actualitza la mu i la sigma usant
% Maximum Likelyhood
[R,Rmean,new_mu,new_S,W,fs,new_f] = policy(Q1,t1,lambda,Nk,Q,new_Q,dt,Id,alphax,
    alphaz,betaz,Nf,d,w,new_f,g,Nt,P,tau,Nupdates,t1);

% policy search amb REPS
r = 1;
[RREPS,RmeanREPS,new_mu,new_S,W,fs,dws,new_f,M,S] = policyREPS(Q1ini,t1ini,
    lambdaini,Nk,Qini,new_Qini,dt,Id,alphax,alphaz,betaz,Nf,d,wini,new_fini,g,Nt
    ,Pini,tau,Nupdates,r,t1);

figure;
plot(Rmean(2:end, 1))
hold on;
plot(RmeanREPS(2:end, 1))
title('Rmean vs RmeanREPS')
legend('Rmean', 'RmeanREPS')
```

```
% main min_jerk

close all
clear all
addpath('/Users/acolome/Desktop/MATLAB/robot')
% read data
Qtot = load('FullData.txt');
Q1 = Qtot(:,[1:8,16:22,30:36]);
Q2 = Qtot(:,[1,9:15,23:29,37:43]);

time = Q1(end,1);

d = 7;

% for i = 2:(d+1)
%     figure
%     plot(Q1(:,i))
% end
% no movement in the firt 15-20 timesteps
Q1 = Q1(15:end, :);
Q2 = Q2(15:end, :);
%subsampling
ss=3;
Q1 = Q1(1:ss:end, :);
%Q2 = Q2(1:3:end, :);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%555

% for i = 2:(d+1)
%     x = 1:length(Q1(:,1));
%     y = Q1(x,i);
%     xx = 0:.25:length(Q1(:,1));
%     yy = spline(x,y,xx);
%     figure
%     plot(x,y,'o',xx,yy)
%     hold on
%     plot(Q1(:,i), 'LineWidth',2);
% end

dt1 = Q1(2,1)-Q1(1,1); %=0.0597
timelapse = Q1(end,1)-Q1(1,1); %=8.9545

dt1 = 1/length(Q1(:,1));
T1 = 0: dt1 :1-dt1;
Q1(:,1) = T1;
dt2 = 1/length(Q2(:,1));

T2 = 0: dt2 :1-dt2;
Q2(:,1) = T2;
tau = 1;

dt = dt2*time;
T = T2;

yy = zeros( size(Q2) );
yy(1,:) = Q1(1,:);
yy(:,1) = Q2(:,1);
x = 1:length(Q1(:,1));
xx = 1:(1/ss):length(Q1(:,1));
for i = 2:(d+1)
    y = Q1(x,i);
    yy(:,i) = spline(x,y,xx);

    for j = 2:length(yy(:,1))
        yy(2:end,d+i) = diff(yy(:,i)) / dt;
        yy(2:end,2*d+i) = diff(yy(:,d+i)) / dt;
%           yy(:,d+i) = gradient(yy(:,i));
%           yy(:,2*d+i) = gradient(yy(:,d+i));
    end

%     figure
%     plot(x,y,'o',xx,yy);
%     hold on
%     plot(Q1(:,i), 'LineWidth',2);
end

%for i = (2*d+2):(3*d+1)
for i = 1:22
    figure
    plot(yy(:,1), yy(:,i));
    hold on
    plot(Q1(:,1), Q1(:,i), 'LineWidth',2);
```

```
end

Q1 = yy;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%5


%dt = Q1(2,1)-Q1(1,1); %=0.0597
timelapse = Q1(end,1)-Q1(1,1); %=8.9545

%dt = 1/length(Q1(:,1));
T = 0: dt2 :1-dt2;
Q1(:,1) = T;
Q2(:,1) = T2;
tau = 1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Nf = 10;
alphaz = 10;
alphax = 3.5;
betaz = alphaz/4;
tau = 1;
K = 0.8;
Nt = length(Q1(:,1));
g = Q1(Nt, 2:d+1);

% translation coordinates
l1 = 0.985;
l2 = 0;
l3 = 0;
% rotation angle
theta = 0;
WAM;
[Qc1, Qc2] = change_ref(Q1,Q2,l1,l2,l3,theta,Nt,d,wam);

% carreguem les dades de la trajectoria, trobem els parmetres i la f i
% filtrem l'acceleraci
[ddy_filt,f,x,Q1,Nt,g] = ini_filt_param(Q1,tau,d,Nf,alphax,alphaz,betaz,K,g);

Qg = [1, -0.22, -0.25;
    1.2, -0.3, -0.2;
    1.05, -0.22, 0.5];
tg = [10, 25, 40]';
%tgv = Q1(tg,1);
lambda_reg = 0.1;
Nk = Nf*d + 2;
% Nk = 10;

% compute Gaussians and weights, and make the fitting of f and Q
[Ct,C,D,W,p,P] = gaussianes(Q1,Nf,tau,alphax,x,d);

[new_f,w,A,new_Q1,Id,w_aux] = pesos(P,Nf,d,x,f,Q1,Nt,alphaz,betaz,g,dt,tau,
    lambda_reg);
```

```
% safe parameters to compare WMLPS and REPS
lambda = 5000;

Qgini = Qg;
tgini = tg;
lambdaini = lambda;
Q1ini = Q1;
new_Q1ini = new_Q1;
wini = w;
new_fini = new_f;
Pini = P;

Nupdates = 10;
% WMLPS
new_Q = new_Q1;
[R,Rmean,new_mu,new_S,W,fs,new_f] = policy2(Qg,tg,lambda,Nk,Q1,Q2,new_Q1,dt,Id,
    alphax,alphaz,betaz,Nf,d,w,new_f,g,Nt,P,tau,Nupdates,Qc2,l1,l2,l3,theta)


% REPS with coordination matrix DoF reduction
r = 4;
[RREPS,RmeanREPS,new_mu,new_S,W,fs,dws,new_f,M,S] = policyREPS2(Qgini,tgini,
    lambdaini,Nk,Q1ini,new_Q1ini,Q2,dt,Id,alphax,alphaz,betaz,Nf,d,wini,new_fini
    ,g,Nt,Pini,tau,Nupdates,r,Qc2,l1,l2,l3,theta);


figure;
plot(Rmean(2:end,1))
hold on;
plot(RmeanREPS(2:end,1))
title('Rmean vs RmeanREPS')
legend('Rmean', 'RmeanREPS')

figure;
hold on;
plot(RREPS)
hold on;
title('R vs RREPS')
legend('R','RREPS','Rnormal')

% initial variety
dist_min = Inf;
eix_min = [1 0 0 0];

% middle point curve
Qpm(:,1) = Qc1(:,1);
for i = 1:Nt
    Qpm(i,2:4) = ( Qc1(i,2:4) + Qc2(i,2:4) ) / 2;
end


figure
plot(Qc2(:,2))
hold on
plot(Qc1(:,2))
```

```
hold on
plot(Qm(:,2))
hold on
plot(Qpm(:,2))
legend(’Q2’,’Q1’,’Qm’,’Qpm’)


figure
plot3(Qc1(:,2),Qc1(:,3),Qc1(:,4),’m’,’LineWidth’,2)
hold on
plot3(Qc2(:,2),Qc2(:,3),Qc2(:,4),’k’,’LineWidth’,2)
plot3(Qm(:,2),Qm(:,3),Qm(:,4),’g’,’LineWidth’,2)
grid on
% axis equal
legend(’q1’,’q2’,’qm’)

x=-2:.1:2;
[X,Y] = meshgrid(x);
aa=eix_min(1); bb=eix_min(2); cc=eix_min(3); dd=eix_min(4);

Z=(dd- aa * X - bb * Y)/cc;
h=surf(X,Y,Z)
shading flat
h.EdgeColor=’none’
h.FaceColor=’b’






% middle point curve
Qpm(:,1) = Qc1(:,1);
for i = 1:Nt
    Qpm(i,2:4) = ( Qc1(i,2:4) + Qc2(i,2:4) ) / 2;
end
```

```
function [x,xd,xdd] = min_jerk_step(x,xd,xdd,goal, tau, dt)

% function [x,xd,xdd] = min_jerk_step(x,xd,xdd,goal,tau, dt) computes
% the update of x,xd,xdd for the next time step dt given that we are
% currently at x,xd,xdd, and that we have tau until we want to reach
% the goal

if tau<dt,
    return;
end;

dist = goal - x;
a1 = 0;
a0 = xdd * tau^2;
v1 = 0;
v0 = xd * tau;

t1=dt;
t2=dt^2;
```

```
t3=dt^3;
t4=dt^4;
t5=dt^5;

c1 = (6.*dist + (a1 - a0)/2. - 3.*(v0 + v1))/tau^5;
c2 = (-15.*dist + (3.*a0 - 2.*a1)/2. + 8.*v0 + 7.*v1)/tau^4;
c3 = (10.*dist+ (a1 - 3.*a0)/2. - 6.*v0 - 4.*v1)/tau^3;
c4 = xdd/2.;
c5 = xd;
c6 = x;



x   = c1*t5 + c2*t4 + c3*t3 + c4*t2 + c5*t1 + c6;
xd  = 5.*c1*t4 + 4*c2*t3 + 3*c3*t2 + 2*c4*t1 + c5;
xdd = 20.*c1*t3 + 12.*c2*t2 + 6.*c3*t1 + 2.*c4;
```

```
% min_jerk d joints
tau = 1; %temps final
dt = 0.01; %discretization step
% d = 7; %dimensio es diu al main
y = zeros(tau/dt,d);
yd = zeros(tau/dt,d);
ydd = zeros(tau/dt,d);
dof = size(y,2);
y(1,:) = zeros(1,d); %starting position
g = ones(1,d)'; %Goal

for i=1:tau/dt-1
    for j=1:d
        [y(i+1,j),yd(i+1,j),ydd(i+1,j)] = min_jerk_step(y(i,j),yd(i,j),ydd(i,j),
           g(j), tau-i*dt, dt);
        % function [x,xd,xdd] = min_jerk_step(x,xd,xdd,goal,tau, dt) computes
        % the update of x,xd,xdd for the next time step dt given that we are
        % currently at x,xd,xdd, and that we have tau until we want to reach
        % the goal
    end
end

 T =(0:dt:tau-dt)';
```

```


tau = 1; %temps final
dt = 0.01; %discretization step
%d = 1; %dimensio
y = zeros(tau/dt,d);
yd = zeros(tau/dt,d);
ydd = zeros(tau/dt,d);
dof = size(y,2);
y(1,:) = zeros(1,d)+randn(1,d); %starting position
g = ones(1,d)'+randn(1,d)'; %Goal

for i=1:tau/dt-1
    for j=1:dof
```

```
        [y(i+1,j),yd(i+1,j),ydd(i+1,j)] = min_jerk_step(y(i,j),yd(i,j),ydd(i,j),
           g(j), tau-i*dt, dt);
        % function [x,xd,xdd] = min_jerk_step(x,xd,xdd,goal,tau, dt) computes
        % the update of x,xd,xdd for the next time step dt given that we are
        % currently at x,xd,xdd, and that we have tau until we want to reach
        % the goal
    end
end

 T =(0:dt:tau-dt)';
```

```


function [Qc1, Qc2] = change_ref(Q1,Q2,l1,l2,l3,theta,Nt,d,wam)

%       WAM;

    % change of reference transformation
    T = [cos(theta) -sin(theta) 0 l1;
         sin(theta) cos(theta)  0 l2;
         0          0           1 l3;
         0          0           0  1];

    Qc1 = Q1(:,1);
    Qc2 = Q2(:,1);
    for t = 1:Nt
        % Q2 is the trajectory of WAM2 in the Joint Space 2
        q = (Q2(t,2:d+1))';
        T2 = fkine(wam, q);
        Qc2(t,2:4) = T2(1:3,4);  % Qc2 is Q2 in the Cartesian Space 2
        aux = (T*[Qc2(t,2:4)' ; 1])';
        Qc2(t,2:end) = aux(1,1:3);  % transform from Cartesian Space 2 to 1
        % Q1 is the trajectory of WAM1 in the Joint Space 1
        q = (Q1(t,2:d+1))';
        T1 = fkine(wam, q);
        Qc1(t,2:4) = T1(1:3,4);       % Qc1 is Q1 in the Cartesian Space 1

    end

end
```

```


clear L
L{1} = link([-pi/2 0 0 0 0],'standard');
L{2} = link([pi/2 0 0 0 0],'standard');
L{3} = link([-pi/2 0.045 0 0.55 0],'standard');
L{4} = link([pi/2 -0.045 0 0 0],'standard');
L{5} = link([-pi/2 0 0 0.3 0],'standard');
L{6} = link([pi/2 0 0 0 0],'standard');
L{7}=link([0 0 0 0.06 0],'standard');

L{1}.m = 8.3936;
L{2}.m = 4.8487;
L{3}.m = 1.7251;
L{4}.m = 1.0912;
```

```
L{5}.m = 0.3067;
L{6}.m = .4278;
%L{7}.m=0.1;
L{7}.m=1.3;

L{1}.r = [ 0.0003506    0.1326795          0.0006286 ];
L{2}.r = [ -.000223  -.02139    .01337];
L{3}.r = [ -.0387565  .21791 0.0000252];
L{4}.r = [ 0.01175    -.0001    0.1359];
L{5}.r = [ 0.000058    0.02838     0.00019];
L{6}.r = [ -0.00003    -0.01486    .0256];
L{7}.r = [ -0.000    -0.0001823    -.2847];

L{1}.I = [  0.095157 0.092032 0.059291 0.000246 -0.000963 -0.000095 ];
L{2}.I = [29327e-6 20781e-6 22807e-6 -43e-6 1349e-6 -129e-6];
L{3}.I = [56662e-6 3158e-6 56806e-6 -2321e-6 -17e-6 8e-6];
L{4}.I = [18891e-6 19341e-6 2027e-6 -1e-6 -1721e-6 18e-6];
L{5}.I = [321e-6 172e-6 351e-6 0 0 0];
L{6}.I = [604e-6 269e-6 507e-6 0 -62e-6 0];
L{7}.I = [21e-4 22e-4 42e-4 -1e-5 -1721e-5 18e-5];

L{1}.Jm =  20e-12;
L{2}.Jm =  20e-12;
L{3}.Jm =  20e-12;
L{4}.Jm =  33e-12;
L{5}.Jm =  33e-12;
L{6}.Jm =  33e-12;
L{7}.Jm =  33e-12;

L{1}.G =  -20.6111;
L{2}.G =  107.815;
L{3}.G =  -53.7063;
L{4}.G =  76.0364;
L{5}.G =  71.923;
L{6}.G =  76.686;
L{7}.G =  76.686;
% viscous friction (motor referenced)
L{1}.B =   1.48e-3;
L{2}.B =   .817e-3;
L{3}.B =    1.38e-3;
L{4}.B =   71.2e-6;
L{5}.B =   82.6e-6;
L{6}.B =   36.7e-6;

% Coulomb friction (motor referenced)
L{1}.Tc = [ .395        -.435];
L{2}.Tc = [ .126        -.071];
L{3}.Tc = [ .132        -.105];
L{4}.Tc = [ 11.2e-3 -16.9e-3];
L{5}.Tc = [ 9.26e-3 -14.5e-3];
L{6}.Tc = [ 3.96e-3 -10.5e-3];



%
% some useful poses
%
```

```
%qz = [0 0 0 0 0 0]; % zero angles, L shaped pose
%qr = [0 pi/2 -pi/2 0 0 0]; % ready pose, arm up
%qs = [0 0 -pi/2 0 0 0];
%qn=[0 pi/4 pi 0 pi/4  0];


wam = robot(L, 'Wam arm', 'Barrett', 'dh params');
clear L
wam.name = 'Wam arm';
wam.manuf = 'Barrett';
```

```
function [ddy_filt,f,x,Q,Nt,g] = ini_filt_param(Q,tau,d,Nf,alphax,alphaz,betaz,K
    ,g,dt)

    ddy_filt=[];
    Nt=size(Q,1);

%       acceleration filtering
    ddy_filt = zeros(Nt, 7);
    ddy_filt(1,:) = Q(1,2*d+2:3*d+1);
    for i = 2:Nt
        ddy_filt(i,:) = Q(i,2*d+2:3*d+1)*K + Q(i-1,2*d+2:3*d+1)*(1-K);
    end
    %Q(:,2*d+2:3*d+1) = ddy_filt;


    % compute excitation function f
    for i = 1:Nt
        f(i,:) = Q(i, 2*d+2:3*d+1)/tau - alphaz*(betaz*(g - Q(i, 2:d+1)) - Q(i,
            d+2:2*d+1)/tau);
    end

    for i = 1:Nt
        x(i) = exp(-alphax*Q(i,1));
    end

end
```

```
function [Ct,C,D,W,p,P] = gaussianes(Q,Nf,tau,alphax,x,d)

    Ct = [];
    for i = 1:Nf
        Ct = [Ct;(i*tau)/(Nf+1)];
    end
    W = zeros(d,Nf);
    C = exp(-alphax/tau*Ct);    % Gaussian centers
    D = abs((diff(C)/0.55).^2);      % Gaussian widths
    D = [D;D(end)];

    figure
    hold on
    for i=1:Nf
        norm = normpdf(x(i), C(i), D(i));
        %plot(Q(:,1), norm);
    end

    for i = 1:length(x)
        for j = 1:length(C)
            p(i,j) = exp( (-(x(i)-C(j))^2) / (D(j)) );
        end
    end

    for i = 1:length(x)
        for j = 1:length(C)
            P(i,j) = x(i)*p(i,j)/sum(p(i,:));
        end
    end

    plot(Q(:,1),P)
    title('Gaussians')


end
```

```
% troba els pesos, calcula la nova f i fa plots
function [new_f,w,A,new_Q,Id,w_aux] = pesos(P,Nf,d,x,f,Q,Nt,alphaz,betaz,g,dt,
    tau,lambda)

    % Moore-Penrose pseudo-inverse
    [u,s,v]=svd(P);
    A=zeros(size(P))';
    for j=1:min(size(s))
        A=A+s(j,j)/(s(j,j)^2+lambda^2)*v(:,j)*u(:,j)';
    end

    % weights
    w_aux = A*f;
    for i = 1:Nf
        for j = 1:d
            w( (j-1)*Nf + i , 1) = w_aux(i,j);
        end
    end

    % f fitting
    Id = eye(d,d);
    new_f = zeros(size(f));
    for i = 1:length(Q(:,1))
        new_f(i,:) = (kron(Id, P(i,:))*w)';
    end

    figure
    plot(f, 'b')
    hold on
    plot(new_f,'r') %, 'LineWidth',2)
    title('Fitting f')
    legend('f', 'newf')

    figure
    plot(Q(:,1),x)
    title('x curve')

    %clear new_f;
    %clear new_Q;

    % trajectory fitting
    new_Q(:,1) = Q(:,1);
    new_Q(1,1:3*d+1) = Q(1,1:3*d+1);
    for i = 2:Nt
        new_Q(i,2*d+2:3*d+1) = tau*alphaz* (betaz*(g - new_Q(i-1,2:d+1) ) -
            new_Q(i-1,d+2:2*d+1) / tau) + new_f(i-1,:);
        new_Q(i,d+2:2*d+1) = new_Q(i-1,d+2:2*d+1) + dt*new_Q(i,2*d+2:3*d+1);
        new_Q(i,2:d+1) =  new_Q(i-1,2:d+1) + dt*new_Q(i,d+2:2*d+1);
    end

    figure
    for i = 2:8
        plot(Q(:,i),'b')
        hold on
        plot(new_Q(:,i), 'r') %, 'LineWidth',2)%, 'r')
    end
    %legend('Q', 'newQ')
    title('Fitting Q')
    legend('Q', 'newQ')
%
    figure
    plot(Q(:,16:22),'b')
    hold on
    plot(new_Q(:,16:22), 'r') %,'LineWidth',2)
    title('Fitting acceleration')
    legend('ddQ','newddQ')
%
%
%

end
```

```
%
function [R,Rmean,new_mu,new_S,W,fs,new_f] = policy(Q1,t1,lambda,Nk,Q,new_Q,dt,
    Id,alphax,alphaz,betaz,Nf,d,w,new_f,g,Nt,P,tau,Nupdates,tg)

% per on volem que passi
% Q1 = [0.4, 0.7, 0.15, 1.5, -0.1, -0.25, 0.1];
% t1 = Q(50,1);

fs = zeros(Nt,d,Nk);

new_mu = w;
new_S = lambda*eye(d*Nf);
R = zeros(1,Nk);
R(1) = mean(-abs(Q(tg,2:d+1) - Q1) - 0.000015*sum( (Q(:,2*d+2:3*d+1)).^2 ));

W = zeros(d*Nf, Nk);

hold on
iter = 1;
Rmean=zeros(Nupdates,1);
for iter=1:Nupdates
    [iter Rmean(iter)]
```

```
% perform Nk rollouts
for k = 1:Nk
    W(:,k) = mvnrnd(new_mu, new_S)' ;    % samples
    clear new_f;
    clear new_Q;

    % new trajectory
    new_Q(:,1) = Q(:,1);
    new_Q(1,2:d+1) = Q(1,2:d+1);
    new_Q(1,d+2:2*d+1) = Q(1,d+2:2*d+1);
    for i = 2:Nt
        new_f(i,:) = (kron(Id, P(i,:))*W(:,k))';
        new_Q(i,2*d+2:3*d+1) = tau*alphaz* (betaz*(g - new_Q(i-1,2:d+1) ) -
            new_Q(i-1,d+2:2*d+1) / tau) + new_f(i,:);
        new_Q(i,d+2:2*d+1) = new_Q(i-1,d+2:2*d+1) + dt*new_Q(i,2*d+2:22);
        new_Q(i,2:d+1) =  new_Q(i-1,2:d+1) + dt*new_Q(i,d+2:2*d+1);

    end

    %plot(new_Q(:,2:8),'Color', Colorset(1,:) );

    % Evaluate Reward
    R(k)= mean(-abs(new_Q(tg,2:d+1) - Q1) - 0.000015*sum( (new_Q(:,2*d+2:3*d
        +1)).^2 ));

    fs(:,:,k) = new_f;

end
%         plot(new_Q(:,2:8),'Color', Colorset(1,:) );
%         title('Q vs newQ')
%         %legend('Q')

new_mu=zeros(Nf*d,1);
for i = 1:Nk
    new_mu = new_mu +  W(:,i)*exp(R(i)) / sum( exp(R) );
end
new_S=zeros(Nf*d,Nf*d);
for i=1:Nk
    new_S = new_S + (W(:,i) - new_mu)*(W(:,i) - new_mu)'*exp(R(i))/sum(exp(R
        ));
end

% new mean trajectory
clear new_f;
clear new_Q;
new_Q(:,1) = Q(:,1);
new_Q(1,2:d+1)=Q(1,2:d+1);
new_Q(1,d+2:2*d+1)=Q(1,d+2:2*d+1);
%new_f=f_new;
for i = 2:Nt
    new_f(i,:) = (kron(Id, P(i,:))*new_mu)';
    new_Q(i,2*d+2:3*d+1) = tau*alphaz* (betaz*(g - new_Q(i-1,2:d+1) ) -
        new_Q(i-1,d+2:2*d+1) / tau) + new_f(i-1,:);
    new_Q(i,d+2:2*d+1) = new_Q(i-1,d+2:2*d+1) + dt*new_Q(i,2*d+2:3*d+1);
    new_Q(i,2:d+1) =  new_Q(i-1,2:d+1) + dt*new_Q(i,d+2:2*d+1);
    end

    % Reward of the mean trajectory
    Rmean(iter+1) = mean(-abs(new_Q(tg,2:d+1) - Q1) - 0.000015*sum( (new_Q(:,2*d
        +2:3*d+1)).^2 ));


end

%      figure
%      plot(R)
%      hold on
%      plot(Rmean)
%      title('Reward')
%      legend('R', 'Rmean')
%
end
```

```
function [REPS_output,RmeanREPS,new_mu,new_S,W,fs,dws,new_f,MU,SS] = policyREPS(
    Q1,t1,lambda,Nk,Q,new_Q,dt,Id,alphax,alphaz,betaz,Nf,d,w,new_f,g,Nt,P,tau,
    Nupdates,r,tg)
plot_flag=0;

dws=1;
% store parameters
SS=[];
MU=[];

% Compute Psi
Psi = zeros(r*Nf,Nt*r);
for t = 1:Nt
    Psi( : , ((t-1)*r + 1):((t-1)*r + r)) = (kron( eye(r), P(t,:) ))' ;
end

fs = zeros(Nt,d,Nk);

new_mu = w;
new_S = lambda*eye(d*Nf);
RREPS = zeros(1,Nk);
RREPS(1) = mean(-abs(Q(tg,2:d+1) - Q1) - 0.000015*sum( (Q(:,2*d+2:3*d+1)).^2 ));
     %0.15*sum( (y).^2 );

REPS_output=RREPS(1);
W = zeros(r*Nf, Nk);
%
%      figure
%      Colorset = varycolor(50);
%      plot(Q(:,2:8),'b','LineWidth',2)

%      hold on
iter = 1;

% inicialize Omega
[U,S,V] = svd(new_f');
Om_old = U(:,1:r);
```

```
%inicialize mu and Sigma
f_aux = new_f';
z = zeros(r*Nt,1);
for t = 1:Nt
    z( (t-1)*r+1 : t*r ,1) = pinv(Om_old)*f_aux(:,t);
end
Psi_aux = zeros(r*Nt,Nf*r);
for t = 1:Nt
    Psi_aux(((t-1)*r + 1):((t-1)*r + r) , :) = Psi( : , ((t-1)*r + 1):((t-1)*r +
         r))' ;
end

old_mu = pinv(Psi_aux) * z;
old_S = 1000*eye(r*Nf);

%inicialize de new_mu i new_S
new_mu = old_mu;
new_S = old_S;

%      % nova trajectria
%      new_Q(:,1) = Q(:,1);
%      new_Q(:,2:5) = Q(:,2:5);
%      new_Q(:,6:9) = Q(:,9:12);
%      new_Q(:,10:13) = Q(:,16:19);


RmeanREPS=zeros(Nupdates+1,2);
for iter = 1:Nupdates   %while(abs(Rmean(iter)) > 0.05)

    [iter RmeanREPS(iter)]
    RREPS = zeros(1,Nk);


    % mu and Sigma
    for k = 1:Nk
        W(:,k) = mvnrnd(new_mu, new_S)' ;   % Sample
        clear new_f;
        clear new_Q;

        % new trajectory
        new_Q(:,1) = Q(:,1);
        new_Q(:,2:d+1) = Q(:,2:d+1);
        new_Q(:,d+2:2*d+1) = Q(:,d+2:2*d+1);

        % fitting f
        for t = 1:Nt
            new_f(t,:) = (Om_old *(kron(eye(r), P(t,:))* W(:,k)))';
        end
        % trajectory
        for i = 2:Nt
            new_Q(i,2*d+2:3*d+1) = tau*alphaz* (betaz*(g - new_Q(i-1,2:d+1) ) -
                new_Q(i-1,d+2:2*d+1) / tau) + new_f(i,:);
            new_Q(i,d+2:2*d+1) = new_Q(i-1,d+2:2*d+1) + dt*new_Q(i,2*d+2:3*d+1);
            new_Q(i,2:d+1) =  new_Q(i-1,2:d+1) + dt*new_Q(i,d+2:2*d+1);
        end
        ALLQ{k}=new_Q(:,2:d+1);
        if plot_flag==1

            plot(new_Q(:,2),'b')
            hold on
        end
        NEW_F{k} = new_f';
        RREPS(k)= mean(-abs(new_Q(tg,2:d+1) - Q1) - 0.000015*sum( (new_Q(:,2*d
            +2:3*d+1)).^2 ));

    end

%             figure
%             plot(-abs(new_Q(50,2:8) - Q1))
%             title(iter)
%             figure
%             plot(0.000015*sum(new_Q(:,16:22)).^2 )
%             title(iter)

    REPS; %-> new_mu and new_S
    Fweighted=zeros(d,Nt);
    for i = 1:Nk
        Fweighted = Fweighted+dw(i)*NEW_F{i}/sum(dw);
    end

    % build coordination matrix
    if iter ~= 1
        Om_old = Om ;
    end
%
    [U,S,V] = svd(Fweighted);
%     if iter==1
        Om = U(:,1:r);
%     end

    O_old = kron(eye(Nt),Om_old);
    O = kron(eye(Nt),Om);
    M = pinv(O_old*Psi')*O*Psi';

    if iter ~= 1
        old_mu = new_mu;
        old_S = new_S;
    end

    % update policy
    new_mu = pinv(M)*old_mu;
    new_S = pinv(M) * ( old_S + (M*new_mu - old_mu) * (M*new_mu - old_mu)' ) *
        pinv(M)';

    % new trajectory
    clear new_f;
    clear new_Q;
    new_Q(:,1) = Q(:,1);
    new_Q(1,2:d+1)=Q(1,2:d+1);
    new_Q(1,d+2:2*d+1)=Q(1,d+2:2*d+1);
    Id = eye(r);
    for t = 1:Nt
        new_f(t,:) = (  Om *(kron(eye(r), P(t,:))* new_mu)  )';
    end
    for i = 2:Nt
        new_Q(i,2*d+2:3*d+1) = tau*alphaz* (betaz*(g - new_Q(i-1,2:d+1) ) -
            new_Q(i-1,d+2:2*d+1) / tau) + new_f(i-1,:);
        new_Q(i,d+2:2*d+1) = new_Q(i-1,d+2:2*d+1) + dt*new_Q(i,2*d+2:3*d+1);
        new_Q(i,2:d+1) =  new_Q(i-1,2:d+1) + dt*new_Q(i,d+2:2*d+1);
    end

    % Reward
    REPS_output(iter+1)= mean(-abs(new_Q(tg,2:d+1) - Q1) - 0.000015*sum( (new_Q
        (:,2*d+2:3*d+1)).^2 ));
    RmeanREPS(iter+1,:) = [mean(REPS_output),2*std(REPS_output)];
    MU=[MU;new_mu'];
    SS=[SS;svd(new_S)'];

%             plot(new_Q(:,2:8),'Color', Colorset(1,:) );
%             title('QREPS vs newQREPS');
    if plot_flag==1
        close all
        plot(new_Q(:,2),'k','LineWidth',2)
        hold on
    end

end
figure
plot(RmeanREPS(2:end,1),'b','LineWidth',2)
hold on
plot(RmeanREPS(2:end,1)+RmeanREPS(2:end,2),'b','LineWidth',1)
plot(RmeanREPS(2:end,1)-RmeanREPS(2:end,2),'b','LineWidth',1)

title('RmeanREPS')

%     figure
%     plot(RREPS)
%     hold on
%     plot(RmeanREPS)
%     title('Reward')
%     legend('RREPS', 'RmeanREPS')
%


end
```

```
%
function [R,Rmean,new_mu,new_S,W,fs,new_f] = policy2(Qg,tg,lambda,Nk,Q1,Q2,new_Q
    ,dt,Id,alphax,alphaz,betaz,Nf,d,w,new_f,g,Nt,P,tau,Nupdates,Qc2,l1,l2,l3,
    theta)

    % per on volem que passi
    % Q1 = [0.4, 0.7, 0.15, 1.5, -0.1, -0.25, 0.1];
    % t1 = Q(50,1);

    fs = zeros(Nt,d,Nk);
```

```
new_mu = w;
new_S = lambda*eye(d*Nf);
R = zeros(1,Nk);

WAM;
[Qc1, Qc2] = change_ref(new_Q,Q2,l1,l2,l3,theta,Nt,d,wam);
% middle point curve
Qpm(:,1) = Qc1(:,1);
for i = 1:Nt
    Qpm(i,2:4) = ( Qc1(i,2:4) + Qc2(i,2:4) ) / 2;
end

R(1) = mean(mean(-abs(Qpm(tg,2:4) - Qg)) );
W = zeros(d*Nf, Nk);

hold on
iter = 1;
Rmean=zeros(Nupdates,1);
for iter=1:Nupdates
    [iter Rmean(iter)]

    % perform Nk rollouts
    for k = 1:Nk
        W(:,k) = mvnrnd(new_mu, new_S)' ;   % samples
        clear new_f;
        clear new_Q;

        % new trajectory
        new_Q(:,1) = Q1(:,1);
        new_Q(1,2:d+1) = Q1(1,2:d+1);
        new_Q(1,d+2:2*d+1) = Q1(1,d+2:2*d+1);
        for i = 2:Nt
            new_f(i,:) = (kron(Id, P(i,:))*W(:,k))';
            new_Q(i,2*d+2:3*d+1) = tau*alphaz* (betaz*(g - new_Q(i-1,2:d+1)
                ) - new_Q(i-1,d+2:2*d+1) / tau) + new_f(i,:);
            new_Q(i,d+2:2*d+1) = new_Q(i-1,d+2:2*d+1) + dt*new_Q(i,2*d+2:22)
                ;
            new_Q(i,2:d+1) =  new_Q(i-1,2:d+1) + dt*new_Q(i,d+2:2*d+1);

        end

        %plot(new_Q(:,2:8),'Color', Colorset(1,:) );

        WAM;
        [Qc1, Qc2] = change_ref(new_Q,Q2,l1,l2,l3,theta,Nt,d,wam);
        % middle point curve
        Qpm(:,1) = Qc1(:,1);
        for i = 1:Nt
            Qpm(i,2:4) = ( Qc1(i,2:4) + Qc2(i,2:4) ) / 2;
        end

        n = 3;
        m = 1;

        % break indx into 3 parts because n-m+1 = 3 (points definig the
            variety)
        indx1 = randi([1 floor(Nt/3)], 1, 500);
        indx2 = randi([floor(Nt/3) floor(2*Nt/3)], 1, 500);
        indx3 = randi([floor(2*Nt/3) floor(Nt)], 1, 500);
        [DIST, Qm] = mirr_traj(m,n,indx1,indx2,indx3,Nt,d,Qc1,Qc2,Qpm);



        % Evaluate Reward
        R(k)= mean(mean(-abs(Qpm(tg,2:4) - Qg)) );
        R(k)
        fs(:,:,k) = new_f;

    end
    %          plot(new_Q(:,2:8),'Color', Colorset(1,:) );
    %          title('Q vs newQ')
    %          %legend('Q')

    new_mu=zeros(Nf*d,1);
    for i = 1:Nk
        new_mu = new_mu +  W(:,i)*exp(R(i)) / sum( exp(R) );
    end
    new_S=zeros(Nf*d,Nf*d);
    for i=1:Nk
        new_S = new_S + (W(:,i) - new_mu)*(W(:,i) - new_mu)'*exp(R(i))/sum(
            exp(R));
    end

    % new mean trajectory
    clear new_f;
    clear new_Q;
    new_Q(:,1) = Q1(:,1);
    new_Q(1,2:d+1)=Q1(1,2:d+1);
    new_Q(1,d+2:2*d+1)=Q1(1,d+2:2*d+1);
    %new_f=f_new;
    for i = 2:Nt
        new_f(i,:) = (kron(Id, P(i,:))*new_mu)';
        new_Q(i,2*d+2:3*d+1) = tau*alphaz* (betaz*(g - new_Q(i-1,2:d+1) ) -
            new_Q(i-1,d+2:2*d+1) / tau) + new_f(i-1,:);
        new_Q(i,d+2:2*d+1) = new_Q(i-1,d+2:2*d+1) + dt*new_Q(i,2*d+2:3*d+1);
        new_Q(i,2:d+1) =  new_Q(i-1,2:d+1) + dt*new_Q(i,d+2:2*d+1);
    end

    WAM;
    [Qc1, Qc2] = change_ref(new_Q,Q2,l1,l2,l3,theta,Nt,d,wam);
    % middle point curve
    Qpm(:,1) = Qc1(:,1);
    for i = 1:Nt
        Qpm(i,2:4) = ( Qc1(i,2:4) + Qc2(i,2:4) ) / 2;
    end

    % Reward of the mean trajectory
    Rmean(iter+1) = mean(mean(-abs(Qpm(tg,2:4) - Qg)) );

end

%      figure
%      plot(R)
%      hold on
%      plot(Rmean)
%      title('Reward')
%      legend('R', 'Rmean')
%
end
```

```
function [REPS_output,RmeanREPS,new_mu,new_S,W,fs,dws,new_f,MU,SS] = policyREPS2
    (Qg,tg,lambda,Nk,Q,new_Q,Q2,dt,Id,alphax,alphaz,betaz,Nf,d,w,new_f,g,Nt,P,
    tau,Nupdates,r,Qc2,l1,l2,l3,theta)
plot_flag=0;

dws=1;
% store parameters
SS=[];
MU=[];

% Compute Psi
Psi = zeros(r*Nf,Nt*r);
for t = 1:Nt
    Psi( : , ((t-1)*r + 1):((t-1)*r + r)) = (kron( eye(r), P(t,:) ))' ;
end

fs = zeros(Nt,d,Nk);

new_mu = w;
new_S = lambda*eye(d*Nf);
RREPS = zeros(1,Nk);

WAM;
        [Qc1, Qc2] = change_ref(new_Q,Q2,l1,l2,l3,theta,Nt,d,wam);
        % middle point curve
        Qpm(:,1) = Qc1(:,1);
        for i = 1:Nt
            Qpm(i,2:4) = ( Qc1(i,2:4) + Qc2(i,2:4) ) / 2;
        end

RREPS(1) = mean(mean(-abs(Qpm(tg,2:4) - Qg)) );
REPS_output=RREPS(1);
W = zeros(r*Nf, Nk);
%
%      figure
%      Colorset = varycolor(50);
%      plot(Q(:,2:8),'b','LineWidth',2)

%      hold on
iter = 1;

% inicialize Omega
[U,S,V] = svd(new_f');
Om_old = U(:,1:r);

%inicialize mu and Sigma
f_aux = new_f';
z = zeros(r*Nt,1);
```

```
for t = 1:Nt
    z( (t-1)*r+1 : t*r ,1) = pinv(Om_old)*f_aux(:,t);
end
Psi_aux = zeros(r*Nt,Nf*r);
for t = 1:Nt
    Psi_aux(((t-1)*r + 1):((t-1)*r + r) , :) = Psi( : , ((t-1)*r + 1):((t-1)*r +
         r))' ;
end

old_mu = pinv(Psi_aux) * z;
old_S = 1000*eye(r*Nf);

%inicialize de new_mu i new_S
new_mu = old_mu;
new_S = old_S;

%      % nova trajectria
%      new_Q(:,1) = Q(:,1);
%      new_Q(:,2:5) = Q(:,2:5);
%      new_Q(:,6:9) = Q(:,9:12);
%      new_Q(:,10:13) = Q(:,16:19);


RmeanREPS=zeros(Nupdates+1,2);
for iter = 1:Nupdates    %while(abs(Rmean(iter)) > 0.05)

    [iter RmeanREPS(iter)]
    RREPS = zeros(1,Nk);


    % mu and Sigma
    for k = 1:Nk
        W(:,k) = mvnrnd(new_mu, new_S)' ;    % Sample
        clear new_f;
        clear new_Q;

        % new trajectory
        new_Q(:,1) = Q(:,1);
        new_Q(:,2:d+1) = Q(:,2:d+1);
        new_Q(:,d+2:2*d+1) = Q(:,d+2:2*d+1);

        % fitting f
        for t = 1:Nt
            new_f(t,:) = (Om_old *(kron(eye(r), P(t,:))* W(:,k)))';
        end
        % trajectory
        for i = 2:Nt
            new_Q(i,2*d+2:3*d+1) = tau*alphaz* (betaz*(g - new_Q(i-1,2:d+1) ) -
                new_Q(i-1,d+2:2*d+1) / tau) + new_f(i,:);
            new_Q(i,d+2:2*d+1) = new_Q(i-1,d+2:2*d+1) + dt*new_Q(i,2*d+2:3*d+1);
            new_Q(i,2:d+1) =  new_Q(i-1,2:d+1) + dt*new_Q(i,d+2:2*d+1);
        end

        ALLQ{k}=new_Q(:,2:d+1);
        if plot_flag==1
            plot(new_Q(:,2),'b')
            hold on
        end
        NEW_F{k} = new_f';


        WAM;
        [Qc1, Qc2] = change_ref(new_Q,Q2,l1,l2,l3,theta,Nt,d,wam);
        % middle point curve
        Qpm(:,1) = Qc1(:,1);
        for i = 1:Nt
            Qpm(i,2:4) = ( Qc1(i,2:4) + Qc2(i,2:4) ) / 2;
        end

        n = 3;
        m = 1;

        % break indx into 3 parts because n-m+1 = 3 (points definig the variety)
        indx1 = randi([1 floor(Nt/3)], 1, 500);
        indx2 = randi([floor(Nt/3) floor(2*Nt/3)], 1, 500);
        indx3 = randi([floor(2*Nt/3) Nt], 1, 500);
        [DIST, Qm] = mirr_traj(m,n,indx1,indx2,indx3,Nt,d,Qc1,Qc2,Qpm);

        RREPS(k)= mean(mean(-abs(Qpm(tg,2:4) - Qg)) );
    end

    %             figure
    %             plot(-abs(new_Q(50,2:8) - Q1))
    %             title(iter)
    %             figure
    %             plot(0.000015*sum(new_Q(:,16:22)).^2 )
    %             title(iter)

    REPS; %-> new_mu and new_S
    Fweighted=zeros(d,Nt);
    for i = 1:Nk
        Fweighted = Fweighted+dw(i)*NEW_F{i}/sum(dw);
    end

    % build coordination matrix
    if iter ~= 1
        Om_old = Om ;
    end
    %
    [U,S,V] = svd(Fweighted);
    %     if iter==1
    Om = U(:,1:r);
    %     end

    O_old = kron(eye(Nt),Om_old);
    O = kron(eye(Nt),Om);
    M = pinv(O_old*Psi')*O*Psi';

    if iter ~= 1
        old_mu = new_mu;
        old_S = new_S;
    end

    % update policy
    new_mu = pinv(M)*old_mu;
    new_S = pinv(M) * ( old_S + (M*new_mu - old_mu) * (M*new_mu - old_mu)' ) *
       pinv(M)';

    % new trajectory
    clear new_f;
    clear new_Q;
    new_Q(:,1) = Q(:,1);
    new_Q(1,2:d+1)=Q(1,2:d+1);
    new_Q(1,d+2:2*d+1)=Q(1,d+2:2*d+1);
    Id = eye(r);
    for t = 1:Nt
        new_f(t,:) = (  Om *(kron(eye(r), P(t,:))* new_mu)  )';
    end
    for i = 2:Nt
        new_Q(i,2*d+2:3*d+1) = tau*alphaz* (betaz*(g - new_Q(i-1,2:d+1) ) -
           new_Q(i-1,d+2:2*d+1) / tau) + new_f(i-1,:);
        new_Q(i,d+2:2*d+1) = new_Q(i-1,d+2:2*d+1) + dt*new_Q(i,2*d+2:3*d+1);
        new_Q(i,2:d+1) =  new_Q(i-1,2:d+1) + dt*new_Q(i,d+2:2*d+1);
    end

    WAM;
    [Qc1, Qc2] = change_ref(new_Q,Q2,l1,l2,l3,theta,Nt,d,wam);
    % middle point curve
    Qpm(:,1) = Qc1(:,1);
    for i = 1:Nt
        Qpm(i,2:4) = ( Qc1(i,2:4) + Qc2(i,2:4) ) / 2;
    end

    % Reward
    REPS_output(iter+1)= mean(mean(-abs(Qpm(tg,2:4) - Qg)) );
    RmeanREPS(iter+1,:) = [mean(REPS_output),2*std(REPS_output)];
    MU=[MU;new_mu'];
    SS=[SS;svd(new_S)'];

    %             plot(new_Q(:,2:8),'Color', Colorset(1,:) );
    %             title('QREPS vs newQREPS');
    if plot_flag==1
        close all
        plot(new_Q(:,2),'k','LineWidth',2)
        hold on
    end

end
figure
plot(RmeanREPS(2:end,1),'b','LineWidth',2)
hold on
plot(RmeanREPS(2:end,1)+RmeanREPS(2:end,2),'b','LineWidth',1)
plot(RmeanREPS(2:end,1)-RmeanREPS(2:end,2),'b','LineWidth',1)

title('RmeanREPS')

%     figure
%      plot(RREPS)
%      hold on
%      plot(RmeanREPS)
%      title('Reward')
%      legend('RREPS', 'RmeanREPS')
%


end
```

```
            %% REPS
            %input: Ekl=0.5

            % REWARDS: Vector fila amb els rewards de les trajectories

            % regularization: valor per regularitzar el rang de Sw, prendre
            %0.001, per exemple.

            % SAMPLES: matriu on cada columna  s un vector w de pesos d'una
            % mostra.  s a dir, cada columna  s w-N(mw,Sw)

            %output:
            % new_mu: mitja ponderada
            %new_S: nova covarian a
            %fmincon options
            %dw: pesos

Ekl=0.5;
REWARDS= (RREPS-min(RREPS))/(max(RREPS)-min(RREPS));
SAMPLES=W;
regularization=1;

options = optimset(
    fmincon);options.Display='off';options.Algorithm='active-set';ndatause=size(REWARDS,2);dualFunctionActual
    = (eta_) dualfunction(eta_, REWARDS, Ekl);
     eta2 = fmincon(dualFunctionActual,0.01,[], [], [], [], 0.0005, 100,[],
         options);
     if or(eta2==100,eta2==0.0005)
         warning('Eta in its boundary')
         eta2
     end
     dw=exp((REWARDS-max(REWARDS)*ones(size(REWARDS)))/eta2)';
     Z=(sum(dw)*sum(dw) - sum(dw .^ 2))/sum(dw);

     % parameter distribution update
     new_mu=sum(bsxfun(times, SAMPLES', dw)',2)./sum(dw);summ=0;for
         ak=1:ndatausesumm=summ+dw(ak)*((SAMPLES(:,ak)-new_mu)*(SAMPLES(:,ak)-new_mu)');endnew_S=summ./(Z+0.000
```

```
function [Sc,Sjerk]=rolloutcost(Ye,Ydde,npi,Ppi,Rpi)
r4=WAMarm4;
for i=1:size(Ye,1)
        Taa=fkine(r4,Ye(i,1:4)')*[0;0;0.3;1];
        Qaa=Taa(1:3,1);
        %Xex=[Xex;Qaa'];
        if i<size(Ye,1)
            Ct(i)=circlecost(Qaa,Ppi',npi,Rpi);
        else
            Ct(i)=0;
        end
        if i>50
            %Cs=norm(Ydde(i+1,:)-Ydde(i,:))^2/0.002;
            Ca(i)=norm(Ydde(i,:))^2;
        else
            Ca(i)=0;
        end
end
 for i=1:size(Ye,1)
    Sc(i,1)=-(Ct(i));
    Sjerk(i,1)=-(Ca(i))/5000000;
 end


% r4=WAMarm4;
% length_traj=0;
% Qaa_all=[];
% for i=1:size(Ye,1)
%         Taa=fkine(r4,Ye(i,1:4)')*[0;0;0.3;1];
%         Qaa=Taa(1:3,1);
%         Qaa_all=[Qaa_all;Qaa'];
%         %Xex=[Xex;Qaa'];
%         if i<size(Ye,1)
%             Ct(i)=circlecost(Qaa,Ppi',npi,Rpi);
%         else
%             Ct(i)=0;
%         end
%         if i>1
%             jump=norm(diff(Qaa_all(i-1:i,:)));
%             length_traj=length_traj+jump;
%         end
%
%
%         if i>50
%             %Cs=norm(Ydde(i+1,:)-Ydde(i,:))^2/0.002;
%             Ca(i)=norm(Ydde(i,:))^2;
%         else
%             Ca(i)=0;
%         end
% end
%  for i=1:size(Ye,1)
%     Sc(i,1)=-(Ct(i));
%     Sjerk(i,1)=-(Ca(i))/5000000;
%  end
%  close all
%  plotcircles2
%  Sc(end,1)=Sc(end,1)
%  length_traj-2*pi*Rpi
```

```
function ColorSet=varycolor(NumberOfPlots)
% VARYCOLOR Produces colors with maximum variation on plots with multiple
% lines.
%
%     VARYCOLOR(X) returns a matrix of dimension X by 3.  The matrix may be
%     used in conjunction with the plot command option 'color' to vary the
%     color of lines.
%
%     Yellow and White colors were not used because of their poor
%     translation to presentations.
%
%     Example Usage:
%         NumberOfPlots=50;
%
%         ColorSet=varycolor(NumberOfPlots);
%
%         figure
%         hold on;
%
%         for m=1:NumberOfPlots
%             plot(ones(20,1)*m,'Color',ColorSet(m,:))
%         end

%Created by Daniel Helmick 8/12/2008

error(nargchk(1,1,nargin))%correct number of input arguements??
error(nargoutchk(0, 1, nargout))%correct number of output arguements??

%Take care of the anomolies
if NumberOfPlots<1
    ColorSet=[];
elseif NumberOfPlots==1
    ColorSet=[0 1 0];
elseif NumberOfPlots==2
    ColorSet=[0 1 0; 0 1 1];
elseif NumberOfPlots==3
    ColorSet=[0 1 0; 0 1 1; 0 0 1];
elseif NumberOfPlots==4
    ColorSet=[0 1 0; 0 1 1; 0 0 1; 1 0 1];
elseif NumberOfPlots==5
    ColorSet=[0 1 0; 0 1 1; 0 0 1; 1 0 1; 1 0 0];
elseif NumberOfPlots==6
    ColorSet=[0 1 0; 0 1 1; 0 0 1; 1 0 1; 1 0 0; 0 0 0];

else %default and where this function has an actual advantage

    %we have 5 segments to distribute the plots
    EachSec=floor(NumberOfPlots/5);

    %how many extra lines are there?
    ExtraPlots=mod(NumberOfPlots,5);

    %initialize our vector
    ColorSet=zeros(NumberOfPlots,3);

    %This is to deal with the extra plots that don't fit nicely into the
    %segments
    Adjust=zeros(1,5);
    for m=1:ExtraPlots
        Adjust(m)=1;
    end
    SecOne    =EachSec+Adjust(1);
    SecTwo    =EachSec+Adjust(2);
    SecThree =EachSec+Adjust(3);
    SecFour   =EachSec+Adjust(4);
    SecFive   =EachSec;

    for m=1:SecOne
        ColorSet(m,:)=[0 1 (m-1)/(SecOne-1)];
    end

    for m=1:SecTwo
        ColorSet(m+SecOne,:)=[0 (SecTwo-m)/(SecTwo) 1];
    end

    for m=1:SecThree
        ColorSet(m+SecOne+SecTwo,:)=[(m)/(SecThree) 0 1];
    end

    for m=1:SecFour
        ColorSet(m+SecOne+SecTwo+SecThree,:)=[1 0 (SecFour-m)/(SecFour)];
    end

    for m=1:SecFive
        ColorSet(m+SecOne+SecTwo+SecThree+SecFour,:)=[(SecFive-m)/(SecFive) 0
            0];
    end

end
```

```
function [DIST] = dist_fun(eix, Nt, d, Qc1, Qc2)

    for i = 1:Nt
        dist(i) = norm( mirror(Qc1(i,2:4), eix) - Qc2(i,2:4) )^2;
    end

    DIST = sum(dist)+(norm(eix)-1)/10^2;

end
```

```
function [g] = dualfunction(eta, batch_return, epsilon)

n_batch = length(batch_return);
g=epsilon*eta+eta*(log(sum(exp((batch_return - max(batch_return))./eta))/n_batch
    )) +max(batch_return);
if imag(g)>1e-15
   warning('Dual function with imaginary part')
end
```

```

function [EIX] = desc_grad(n, m, Qc1, Qc2, Nt, d, eix0)
    % n=space dimension
    % m= number of eqs (n-dimension of the variety)

    Nc = (n+1)*m;  % number of components of the matrix eix

    theta = eix0;
    THETA(1,:) = theta;
    pert = 0.00001;  %perturbation

    theta_min = theta;
    dist_ref=dist_fun(theta,Nt, d, Qc1, Qc2);
    alpha = 0.5;

    for k = 2:150
        theta = THETA(k-1,:);
        [k dist_fun(theta, Nt, d, Qc1, Qc2) alpha];
        dist_old=dist_ref;

        for j = 1:Nc
            pert_j = zeros(1,Nc);
            pert_j(j) = pert;
            deriv(1,j) = ( dist_fun(theta+pert_j,Nt, d, Qc1, Qc2) - dist_ref ) /
                 pert;
        end
        THETA(k,:) = theta - alpha*deriv;
        dist_ref=dist_fun(THETA(k,:),Nt, d, Qc1, Qc2);

        if(dist_fun(THETA(k,:), Nt, d, Qc1, Qc2)<dist_fun(theta_min, Nt, d, Qc1,
             Qc2))
            theta_min = THETA(k,:);
        end
        if dist_ref < dist_old
            alpha = 1.2*alpha;
        end
        if dist_ref > dist_old
            alpha = 0.5*alpha;
        end

        if abs(dist_ref-dist_old)<1e-4
            break
        end

    end
    EIX = theta_min;
end
```

```
function [DIST, Qm] = min_traj(m,n,indx1,indx2,indx3,Nt,d,Qc1,Qc2,Qpm)

% initial variety
dist_min = Inf;
eix_min = [1 0 0 0];

% several initial varieties
for i = 1:3
    pbase = [Qpm(indx1(i),2:4),-1; Qpm(indx2(i),2:4),-1; Qpm(indx3(i),2:4),-1]';

    %l.i. check
    M = pbase;
    Z = zeros(n+1,1);
    lam = null(M);

    if (isempty(lam))
        eix0 = (null(pbase'))';  % equations defining eix0

        [EIX] = desc_grad(n, m, Qc1, Qc2, Nt, d, eix0);
        if(dist_fun(EIX, Nt, d, Qc1, Qc2) < dist_min)
            dist_min = dist_fun(EIX, Nt, d, Qc1, Qc2);
            eix_min = EIX;
        end
    end
end
[DIST] = dist_fun(eix_min, Nt, d, Qc1, Qc2)
Qm = Qc2(:,1);
for i = 1:Nt
   Qm(i,2:4) = mirror(Qc1(i,2:4), eix_min);
end
```

```
function [q] = mirror(p, eix)

S = size(eix);
n = S(2)-1;
m = S(1);

A = eix(:, 1:n);
b = eix(:, n+1);

x = zeros(n-m+1,n);
A_aux = [A ; rand(1,n)];
b_aux = [b ; rand(1)];
x(1,:) = pinv(A)*b;

i = 2;
while i <= n-m+1
    %add equation
    A_aux = [A ; rand(1,n)];
    b_aux = [b ; rand(1)];
    y = pinv(A_aux)*b_aux;

    %check l.i.
    M = [x(1:(i-1),:)' y];
    Z = zeros(n,1);
    lam = pinv(M)*Z;

    % if l.i. add to base
    if isequal(lam, zeros(i,1))
        x(i,:) = y';
        i = i+1;
    end
end

% base of vectors vectors (not points)
for i = 2:(n-m+1)
    v(i-1,:) = x(i,:)-x(1,:);
end

% find w orthogonal to all vectors in the base and with p+w in the variety
% i.e. A(p+w) = b;
```

```
M = A;
for i = 1:(n-m)
    M = [M ; v(i,:)];
end
bp = b-A*p';
bp = [bp ; zeros(n-m,1)];
w = pinv(M)*bp;

% q is the symmetric point of p wrt. eix
q = p + 2*w';
```