Reinforcement Learning to improve 4-Finger-Gripper Manipulation

Marco Ojer de Andrés

LUND UNIVERSITY

Department of Automatic Control

MSc Thesis TFRT-9999
ISSN 0280–5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Abstract

In the framework of robotics, Reinforcement Learning (RL) deals with the learning of a task by the robot itself. This work focuses on a recently developed method, Policy Improvement with Path Integrals (PI^2), for the case of a 4-finger-gripper manipulator to perform the task of rotating a ball around a desired axis.

The scope of the thesis is to design an experiment, in which the algorithm receives feedback of robot performance. The algorithm has also been adapted to cope with periodic movements parametrized as motor primitives. Furthermore, due to the high dimensionality of the problem, certain assumptions have been made in order to limit the state-space to a reliable subset of it. The obtained results illustrate the good performance of the algorithm as the robot is able to perform the task focusing on important aspects previously set by the user, both for simulation and also for the real robot. The main bottleneck of the thesis has been the speed of both software and hardware, as much time was required to perform long run experiments, specifically in the implementation on the robot where manual supervision was needed.

Acknowledgements

First, I would like to thank the Department of Automatic Control of Lund University for providing the necessary tools and a workshop. Specially to my supervisors Anders Robertsson and Mahdi Ghazaei, who have been of great help during the development of this thesis.

I also thank my colleagues from Barcelona, for their support during the master's degree, but most of it for the memorable moments we have spent together during countless hours of work.

And last but not least, my Sitcom family, Cate, Isa, Franco, Joan and Anna, who have taken care of me during my stay in Lund, I will never forget all the laughs we had together. Special mention to Franco, without you, I would have finished the thesis in May.

Contents

1

Introduction

1.1 Background

Robots have widely demonstrated their usefulness, as they are able to do many tasks, more accurately, in less time and by using less resources in comparison with humans. Since 1961, when the first robot was installed by Unimation, the number of commercial robots has rapidly grown. Notwithstanding most part of the commercial robots are aimed for the industry. Industrial robots operate in static/well-defined environments, doing precise and repetitive tasks, being teleoperated by a human or following a program. However, they are unable to cope with unexpected situations, that is why human supervision is still necessary.

In recent years, robotics has expanded to a wide range of fields such as medicine, agriculture, logistics, and many others. These new environments differ from the industrial ones in their unpredictability, along with in the necessity of interaction with outside elements such as, humans or animals. Thus, there is an upcoming necessity to endow robots with intelligent capabilities, which allow them to perceive the environment and adapt to it. This concept is commonly known as Artificial Intelligence (AI).

Nowadays, intelligent robots are entering the market with remarkable success. Examples of these robot are, Roomba, a vacuum-cleaner robot developed by iRobot, which can clean the floor of a house without an a priori knowledge of it, or Pepper a social robot capable of detecting human emotions and interact while adapting to the mood accordingly. These examples give an idea about the potential of the combination between AI and Robotics.

In fact, AI and Robotics are considered top trending technologies by several market experts as Forbes. Moreover, the International Federation of Robotics (IFR) has forecasted an impressive upcoming growth in robot's sales, both in industrial and services robots as well as the creation of direct and indirect jobs.

1.2 Problem formulation and motivation

The main motivation behind this thesis is twofold. First, as the previous section has mentioned, AI and robotics are increasing sectors. Therefore, expanding my knowledge in those fields will result tremendously beneficial professionally and personally. Second, real robots are not easily available, so being able to work with one is a great opportunity.

The main goal is to apply AI techniques, in this case reinforcement learning techniques, to a robot. Thanks to this new capabilities the robot will be able to learn, by trial and error, how to perform a task in an optimal manner.

The robot, shown in Figure 1.1, is a 4-finger-gripper manipulator, in which every finger has 3 Degrees of Freedom (DoF). This thesis is addressing the task of spinning a ball in a given direction without dropping it. For simplicity, the spin will be performed around the z axis. This thesis is an extension of the proposed problem in Ghazaei's thesis [2], but for the case of 4 fingers instead of 3, and with continuous trajectories.

The main problem is the modelling of the system. Separately, both ball and robot, present an easy modelling, which is not the case for the combination of both. An approach for modelling of the ball-finger system is detailed in [2], which explains several aspects to be taken into consideration. First, the definition of the contact point, as the finger may touch the ball not only with the fingertip. Second, the type of contact, as it can be a point or a surface contact. Moreover, the dynamics of the systems, the finger motion when spinning the ball, may present different conditions, such as sticking (ball and finger same velocities) or sliding, these behaviors are highly dependent on the friction coefficient. Another aspect for consideration is if the type of impact is elastic or non-elastic.

Thus, due to the complex modelling, a model-free approach seems a more practical idea than a model-based one, and furthermore, it could be applied to other cases.

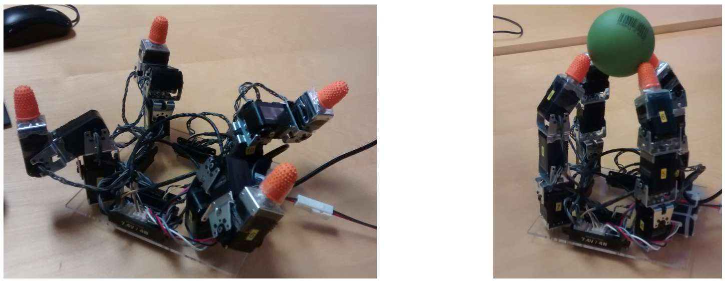


Figure 1.1: The 4-finger-gripper robot.

1.2.1 Resources and Tools

Different tools were used for the development of this thesis. The main ones are shown in Figure 1.2. Matlab environment [41] allowed the implementation and experimentation with the learning algorithm. V-REP [36], which is a robot simulation, offered useful insight about the performance of the algorithm and OpenCV libraries [44] were enormously practical for the visual feedback.

1.2.2 Outline

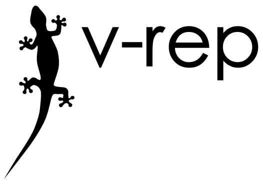
The report is structured in the following manner. Chapter 2 describes the theoretical framework in which the project is based, providing mathematical development of the used methods and it also motivates the choice of the methods and provides some insights about how the methods may have to be adapted to the problem.

Chapter 3 contains more information about the tools used in this thesis, including the robot, the software and the camera.

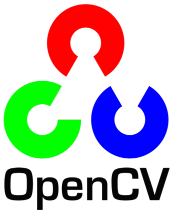
The adopted methodology is explained in Chapter 4. This chapter describes how the method described in Chapter 2 is applied to the problem and the experimental set up. It also justifies any possible changes from the initially made assumptions.

Chapters 5 and 6 present the achieved results as well as their interpretation. Finally Chapter 7 presents the conclusions of the overall project, and it suggests directions for potential extensions of this project

(a) Matlab



(b) V-REP



(c) OpenCV

Figure 1.2: Software Tools

2

Theoretical Background

2.1 Dynamic Movement Primitives

In recent years, several studies as [30], [31] and [32], have observed that complex movements in both vertebrates and invertebrates, are composed of elementary building blocks called motor primitives. Complex movements can be reconstructed by applying transformations to these primivites and by combining them.

Dynamic movement primitives (DMPs) were presented in [3] as a mathematical formalization of these primitives. DMPs present the dynamics of a second order system modulated by a nonlinear term f called *forcing term*. The idea behind is to use the third-order system as point attractor with convenient stability properties, whereas the forcing term enables generation of complex movements. The system is represented with the following equations:

$$\begin{aligned} \dot{z} &= \tau(\alpha_z(\beta_z(g-y)-z)+f) \\ \dot{y} &= \tau z \end{aligned} \tag{2.1}$$

Where α_z and β_z are chosen in order to ensure that the spring-damper system becomes critically damped e.g., by choosing $\alpha_z = 4\beta_z$, τ is a time constant of the system, g represents the goal position and $y, \dot{y}, \ddot{y}$ are position, velocity and acceleration respectively. The system (2.1) is called a *transformation system*.

There are two types of DMPs which are differentiated by the shape of the attractor dynamics. These types are known as *Discrete DMP*, in which the system is attracted to a point, and *Periodic DMP* in which the attractor is represented by a limit cycle. Both types present different definitions of the forcing term, as well as, different *canonical systems*. The canonical system models the generic behaviour of the model equations [5], representing the evolution of the *phase variable*.

2.1.1 Discrete DMP

For the discrete case, the forcing term is described as follow:

$$f = \frac{\sum_{i=1}^{N} w_i \psi_i(x)}{\sum_{i=1}^{N} \psi_i(x)} x(g-y_0) \tag{2.2}$$

where y_0 is the initial position of the system, w_i is the weighting for a given basis function

$$\psi_i(x) = \exp\left(-\frac{1}{2\sigma_i^2}(x-c_i)^2\right), \tag{2.3}$$

where c_i and σ_i define the centre and the standard deviation of a gaussian. The forcing term is dependent on x, the so called phase variable, whose dynamics are represented by the following equation:

$$\dot{x} = -\alpha_x \tau x, \tag{2.4}$$

where α_x is a positive constant, the system converge to 0 at an exponential decay.

Therefore, the forcing term is a set of Gaussian functions that are activated as the canonical system x converges to its target, their weighted summation is normalised, and multiplied by $x(g-y_0)$, which is both a diminishing and spatial scaling term respectively [39]. The diminishing term x ensures convergence of the forcing term to 0 over time, as the canonical system does. The spatial scaling term allows to preserve the shape of the trajectory for different goals as it scales the activation of the basis functions to be relative to the target distance.

Another design aspect to take into account, is that the centres of the basis function should be equally distributed over time, but their activation is hold by the phase variable. This is easily done by mapping the time and the phase variable.

Referring to Equation (2.1), τ denotes the duration of the movement and g represent the coordinates of the attractor point.

2.1.2 Periodic DMP

For the periodic case, the forcing term is:

$$f = \frac{\sum_{i=1}^{N} w_i \psi_i(\phi)}{\sum_{i=1}^{N} \psi_i(\phi)} r, \tag{2.5}$$

denoting r and ϕ the amplitude and phase variable respectively and

$$\psi_i(\phi) = \exp(h_i \cos(c_i - \phi) - 1)), \tag{2.6}$$

where h_i and c_i have the same meaning as in the discrete case. Notwithstanding, the basis functions are Von Mises functions, which are periodic Gaussian-like functions.

The choice of the canonical system for learning limit cycle attractors is a phase oscillator:

$$\dot{\phi} = 2\pi\tau \tag{2.7}$$

Note that τ refers now to the frequency of the system and g represent the anchor point for the oscillation trajectory.

Amplitude, frequency and baseline of the signal can also be modulated to accommodate any desired oscillation by varying τ, r and g as Figure 2.1 shows. Due to linear behaviour presented by the canonical system, the appropriate centre's location can be directly equidistant in the interval $[0, 2\pi]$.

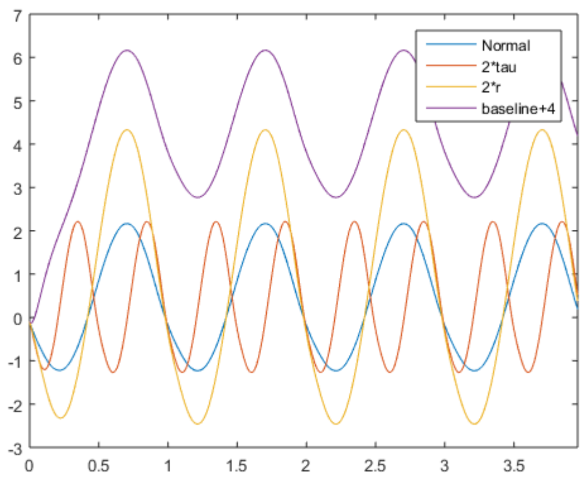


Figure 2.1: Periodic DMP behaviour with modulated parameters with respect to the reference one. Blue: Reference, red: double frecuency, yellow: double amplitud, purple baseline increment of 4.

2.1.3 Learning DMPs

The shape of the trajectory is defined by the forcing term, which includes unknown basis functions and weights. For the basis functions, the choice of their centres was previously described and the standard deviation is typically set equal for all functions. The number of basis function is selected by trial and error. The higher number, the better fitting is expected, but the more unknown parameters to estimate. Therefore, the designer should choose it accordingly to its purposes. The weight vector **w** has the role of defining the spatial-temporal path by weighting the basis function differently [19]. This weight vector, in contrast to the basis function, has to be learnt and cannot be set in advance.

The system is constructed to be linear with respect to the parameter vector, which allows applying a variety of learning algorithms to fit **w**. The goal is the system to follow a desired trajectory in terms of position, velocity and acceleration $(y_{target}(t), \dot{y}_{target}(t), \ddot{y}_{target}(t))$, where $t \in [1, \dots, T]$. The learning is performed in two phases: First, determining the high-level parameters (g, y_0, τ, besides r for periodic) and secondly learning the parameter vector **w**.

For the discrete-time case, τ is adjusted to the duration of the demonstration. In practise, extracting τ from the demonstration may require some thresholding in order to detect movement beginning and end. For instance, some studies as [5] suggest to choose τ as 1.05 times the duration. For periodic cases, τ is adjusted to the frequency of the periodic movement, r is an arbitrary value normally set to 1.

The learning of the parameter vector is done by means of supervised learning. In [40] the use of Locally Weighted Regression (LWR) was proposed due to its fast one-shot and independent learning of the kernels.

The formulation of the problem starts by rearranging Equation (2.1) as

$$f = \frac{1}{\tau^2}\ddot{y} - \alpha_z\big(\beta_z(g-y) - \frac{1}{\tau}\dot{y}\big), \tag{2.8}$$

and substituting trajectory values for the desired ones yields to

$$f_{target} = \frac{1}{\tau^2}\ddot{y}_{target} - \alpha_z\big(\beta_z(g-y_{target}) - \frac{1}{\tau}\dot{y}_{target}\big). \tag{2.9}$$

Hence, a function approximation problem is obtained where the parameters of f are to be adjusted such as f resembles as close as possible to f_{target}. LWR will minimize the cost function, defined as the locally weighted quadratic error,

$$J_i = \sum_{t=1}^{T} \psi_i(t)\big(f_{target}(t) - w_i\epsilon(t)\big)^2, \tag{2.10}$$

being $\epsilon(t) = x(t)(g-y_0)$ or $\epsilon(t) = r$ for discrete and periodic cases, respectively. This is weighted regression problem whose solutions is

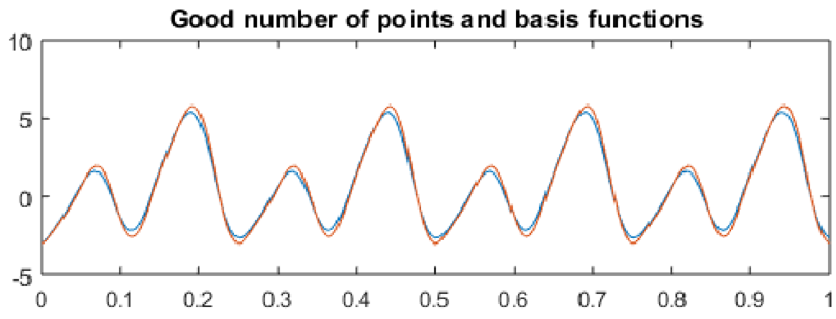
$$w_i = \frac{s^t\Gamma_i f_{target}}{s^T\Gamma_i s}, \tag{2.11}$$

where

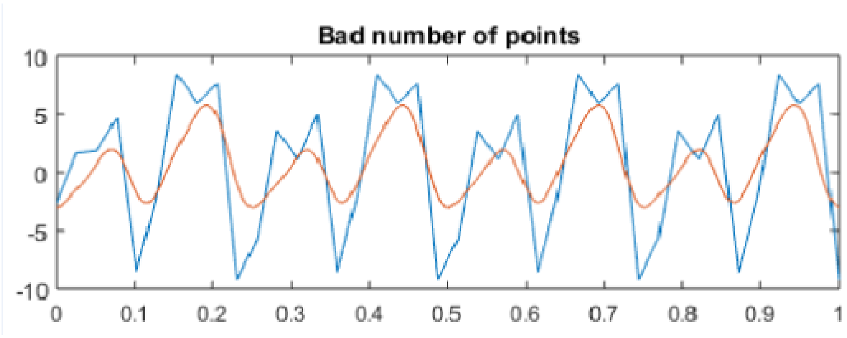
$$s = \begin{pmatrix} \epsilon(1) \\ \vdots \\ \epsilon(T) \end{pmatrix}, \; \Gamma_i = \begin{pmatrix} \psi_i(1) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \psi_i(T) \end{pmatrix}, \; f_{target} = \begin{pmatrix} f_{target}(1) \\ \vdots \\ f_{target}(T) \end{pmatrix}. \tag{2.12}$$

The above formula corresponds to the batch regression, in which the regression is performed once all data is gathered. Other option is to perform a Locally Weighted Projection Regression (LWPR), which minimizes the cost function incrementally as new data enters the system, the main difference between these two approaches is the computational complexity, which is polynomially $\mathcal{O}(n^2)$ for the case of LWR and linear $\mathcal{O}(n^2)$ for LWPR. For the case of this report the input data is not big enough to present any problem, thus LWR is sufficient.

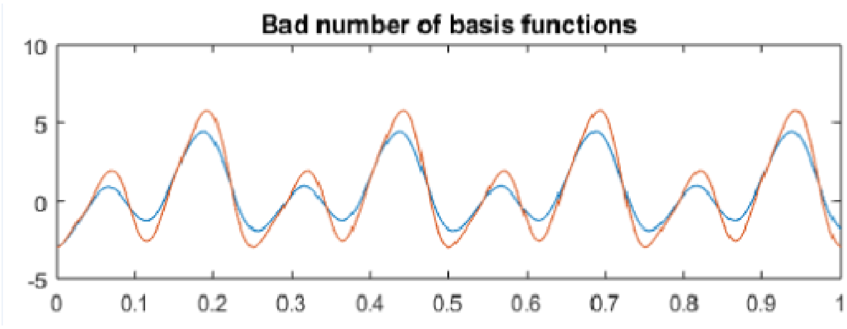
In order to perform a good fitting of the f_{target}, certain amount of basis function and points in the target trajectory are needed. Variations due to the number of points or basis functions can be seen in Figure 2.2.



(a)



(b)



(c)

Figure 2.2: Fitting with different number of basis functions and points. Case (a) both numbers are correct. Case (b) the number of points is too low. Case (c) the number of basis function is too low.

2.1.4 Multiple Degrees of Freedom

All previous explanations were focused on the case of the case of one DMP for a single DoF. However, robots normally have several degrees of freedom, thus the idea of the dynamical system has to be extended to cope with multiple DoF.

There are mainly three strategies to deal with the multiple DoF problem. The first approach consists on having independent transformation and canonical system for each DMP. The second option is to add coupling terms between the canonical systems, as in [13] where the coupling term maintains a lag between arms in a drumming example. The third alternative is to set a common canonical system for all the DoF as seen in Figure 2.3.

The main drawback of the first strategy is that there is no synchronization, as there is no exchange of information between dynamical systems. The second strategy also presents major problems such as the tuning of the coupling parameters, which increases the complexity of the modelling. Having in mind that the selected approach is periodic DMPs, and the aim of the project is to obtain a periodic trajectory for the robot end effector without high-complex modelling, the third strategy seems to be the correct alternative because of its simplicity and the temporal coupling provided. Besides, this strategy was used in several studies as [7], [20], [11] and [19] which perform similar tasks to the one addressed in this thesis.

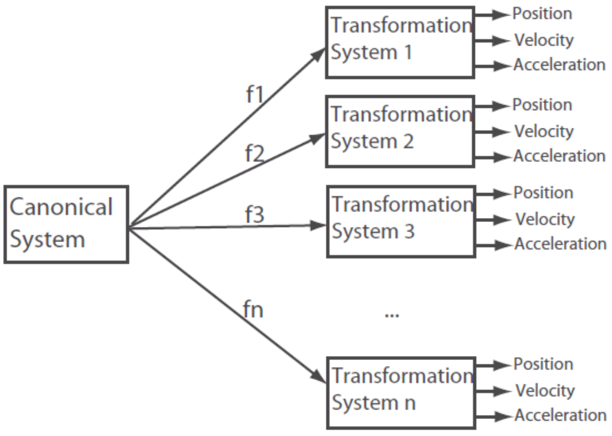


Figure 2.3: System overview of a multi-DoF dynamical system. All DoF share a common canonical system, while having their own transformation systems and forcing terms. Source of image [5].

2.1.5 Discrete+Periodic for transient states

Periodic DMPs have been proven to be effective tools to learn periodic trajectories. However, in reality all periodic motions start in a non periodic state. For example, when a human starts walking, the first step is different from the other ones, as it starts with zero velocity. In [11] the problem of a robot arm paddling the ball is studied, identifying the initial state as the ball resting on the paddle. A start-up phase in order to perform the task successfully was required, in which the robot needed to induce more energy to the ball. This start-up phase consists of a combination between Discrete and periodic DMPs in which the Discrete DMP gradually converges to the period parameter while its influence vanishes as the periodic movement is achieved.

Both [12] and [15] proposed different mathematical formulation to address this problem, by combining Discrete and Periodic DMPs. The main idea behind these formulations is the same.The canonical system for the new DMP formulation is an oscillator in the phase plane:

$$\begin{cases} \dot{\phi} = 2\pi\tau \\ \dot{r} = \eta(\mu^\alpha - r^\alpha)r^\beta \\ \phi(0) = \phi_0,\ r(0) = r_0 \end{cases} \tag{2.13}$$

where $\mu > 0$ represents the radius of the limit cycle, $\eta,\ \alpha,\ \beta$ are constants, $\tau > 0$ denotes the angular velocity of ϕ and ϕ_0 and r_0 are the initial conditions. The transformation system, as for previous cases, is a critically damped system, showed, in Equation (2.1), being the forcing term depending of ϕ and r and defined as:

$$f(\phi,r) = \frac{\sum_{j=1}^{M}\psi_j(\phi,r)\tilde{w_j} + \sum_{i=1}^{N}\varphi_i(\psi,r)w_i}{\sum_{j=1}^{M}\psi_j(\phi,r) + \sum_{i=1}^{N}\varphi_i(\psi,r)}, \tag{2.14}$$

where $W := (w_1,\ldots,w_j,\tilde{w_1},\ldots,\tilde{w_i}) \in \mathbb{R}^{N+M}$ contains the weights to be adjusted and $\Psi := (\varphi_1,\cdots,\varphi_N,\psi_1,\cdots,\psi_M)$ contains the basis functions, encoding ψ the transient part of the motion and φ the periodic pattern. The basic idea is that f depends on the position with respect to the limit cycle, parametrized with the variables ϕ and r, where ϕ denotes the phase of the motion while r corresponds to the distance to the limit cycle.

In order for ψ_j to encode only the transient part without having impact on the periodic pattern, the effect of ψ_j should vanish close to the limit cycle, thus there exist a $\mu_1 > \mu$ such that $\psi_j|_{\mathbb{R}\times(0,\mu_1)} = 0$. Alternatively, φ_i should dominate as close to the limit cycle and has insignificant effect when being far from the limit cycle. Hence, $\varphi_i|_{\mathbb{R}\times(0,\mu_1)} = 1$ and there exists a μ_2 such that $\varphi_i|_{\mathbb{R}\times(\mu_2,\infty)} \approx 0$. Since the movement should be smooth, both ψ_j and φ_i should overlap, by setting $\mu_2 > \mu_1$, there exists a region $r \in (\mu_1,\mu_2)$ where both basis functions are active. To achieve this behaviour, the basis function should be defined as

$$\varphi_i = g(r)\exp(l_i(\cos(c_i - \phi) - 1)), \tag{2.15}$$

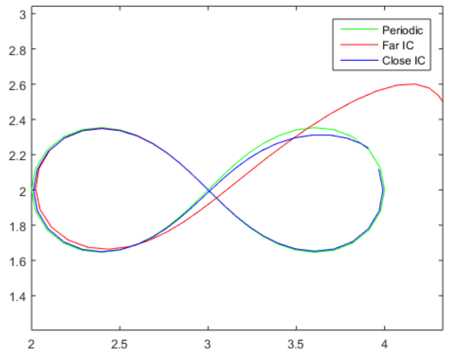


Figure 2.4: Behaviour of Periodic Dynamic Movement Primitives with different initial conditions. The closer the initial conditions are to the limit cycle the less impact the transient phase has.

$$\psi_j = a(r)\exp\left(-h_j\left\|\begin{bmatrix} r\cos\phi \\ r\sin\phi \end{bmatrix} - q_j\right\|_2\right), \tag{2.16}$$

where $q_J \in \mathbb{R}^2$ represents the centres of the basis functions on the phase plane

$$g(r) = \begin{cases} 1, & r \in (0, \mu_1) \\ \left(1 - \left(\frac{r-\mu_1}{\mu_2-\mu_1}\right)^3\right)^3, & r \in (\mu_1, \mu_2) \\ 0, & r \in (\mu_2, \infty) \end{cases}, \tag{2.17}$$

and

$$a(r) = \begin{cases} 0, & r \in (0, \mu_1) \\ \left(1 - \left(\frac{\mu_2-r}{\mu_2-\mu_1}\right)^3\right)^3, & r \in (\mu_1, \mu_2) \\ 1, & r \in (\mu_2, \infty) \end{cases}. \tag{2.18}$$

This formulation is applied in cases when the initial conditions highly differ from the periodic states. Nonetheless, it may be the case when the initial conditions are close to the attractor limit cycle, so the transient phase could be considered negligible as it can be observed in Figure 2.4 and the ordinary periodic DMP formulation is sufficient.

2.2 Reinforcement Learning

A commonly used methodology in robot learning is Reinforcement Learning. RL enables a robot to autonomously discover an optimal behavior through trial and error interaction with its environment [29]. Instead of detailing the solution of the problem, as in Supervised Learning, feedback is provided in terms of a cost function that measures the performance of the robot.

To illustrate RL methods and to facilitate the reader's understanding, some concepts and elements must be introduced. The *state s* provides an observation of the current dynamics of the system, which normally is composed of dynamic variables such as velocities or positions of internal and external element to the robot. The *actions a* represents all possible acts that the robot can execute that may change the system state, for example, motor or control commands. The *policy function* π maps the current observations (states) of the system to the robot commands (actions) to be taken. Last, the *reward r* is a value that represents the evaluation of taking a specific action at a current state. Thus, the goal of an RL system is to find the optimal policy π^* which maximizes the accumulated future rewards.

There are two different elements in RL systems. First, the *agent* which generates an action according to a policy function and previous states. Second the *environment* which modifies its state when an action is received. In robot learning the agent is identified as the robot or the control system. The agent-environment relationship can be seen in Figure 2.5, and it is described as follows: first, the agent observes the state s_t and receives a reward r_t. According to these values and the policy function followed at that instant, the agent generates an action a_t. Consequently, this action causes a reaction in the environment, changing the state into a new one s_{t+1} and giving a new reward r_{t+1}. The flow diagram of this interaction is showed in Figure 2.6.

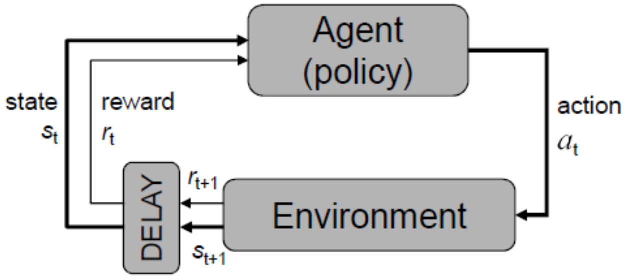


Figure 2.5: Diagram of Agent-Environment interaction.

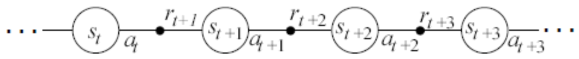


Figure 2.6: Performance sequence of states, action and rewards. Source [27].

2.2.1 Q-learning

Q-learning is a form of discrete-time model-free reinforcement learning technique. Within the reinforcement learning domain, it belongs to value function methodologies, which find the optimal policy by first searching the optimal value function and then deducing the optimal policy from it. A value function defines the sum of expected reward r from a given initial condition by following a certain policy:

$$V_{\pi}(s_t) = \sum_{m=k}^{\infty} \gamma^{m-t} r(s_m, a_m), \qquad 0 < \gamma < 1. \tag{2.19}$$

where γ is the discount factor that determines the importance of future rewards. This value is bounded in order to avoid divergence.

Specifically, Q-learning works by learning an action-value function $Q^{\pi}(s,a)$ that gives the expected utility of taking a given action a in a given state s and following the policy π thereafter [27]. The action-value function is defined as follows:

$$Q^{i+1}(s_t,a_t) = Q^i(s_t,a_t) + \eta[r + \gamma Q^i(s_{t+1}, \pi(s_{t+1})) - Q^i(s_t,a_t)] \tag{2.20}$$

where $\eta \in [0,1]$ represents the learning rate, defined as the level of trust of new information in comparison with the old one. Once the optimal function $Q^*(s,a)$ is obtained, the optimal policy π^* can be extracted without specification of the environment dynamics. Thus, the optimal policy is obtained by calculating:

$$\pi(s) = \arg\max_a Q(s,a) \tag{2.21}$$

A relevant characteristic of Q-learning is the off-policy learning capability. During the learning phase, any policy can be followed as long as all the state/action pairs are regularly visited and updated. In the learning phase, a trade-off between exploitation and exploration must be set, where a common method is ϵ-greedy, in which a random action will be taken with probability ϵ instead of the optimal one. The Q-learning pseudocode is summarized in Algorithm 1.

2.2.2 Path integral policy improvement

Reinforcement learning algorithms can be derived from different frameworks, e.g., dynamic programming, optimal control, policy gradients or probabilistic approaches [6]. *Path integral policy improvement* (PI^2) is a probabilistic reinforcement learning method which arose from the combination of stochastic optimal control theory and path integrals. The PI^2 approach makes an appealing theoretical connection between value function approximations using HJB equations and direct policy learning by approximating a path integral and was proposed in [1]. This section provides an outline of the important characteristics and prerequisites for the PI^2. For further explanation, see the complete development of the algorithm described in [1].

Algorithm 1: Q-algorithm [28]

```
Input :
  $Q_0$   Initial Q-Matrix
  $\eta$   learning rate
  $\gamma$   discount factor
  $\epsilon$   exploration parameter
Output:
  $Q^*$   Optimal Q-Matrix
while Q is not optimal do
    • Get current state: $s_t$
    • Choose action $a_{max}$ overall a, that maximize $Q(s_t, a)$
    • Apply action $a_t \leftarrow a_{max}$ with probability (1-$\epsilon$) (exploitation),
      or random action with probability $\epsilon$ (exploration)
    • Observe reward $r_{t+1}$ and new state $s_{t+1}$
    • Update $Q^{i+1}(s_t, a_t) = Q^i(s_t, a_t) +$
          $\eta[r + \gamma Q^i(s_{t+1}, \pi(s_{t+1})) - Q^i(s_t, a_t)]$
    • Update $s_t \leftarrow s_{t+1}$
end
```

Stochastic Optimal Control

The goal in the stochastic optimal control framework is to find the optimal control inputs of a stochastic system, which minimize a performance criterion. The dynamics of the controlled system is assumed to be of the form:

$$\dot{x}_t = f(x_t) + G(x_t)(u_t + \epsilon_t), \tag{2.22}$$

where $x_t \in \mathbb{R}^{n\times 1}$ denotes the state of the system, $G(x_t) \in \mathbb{R}^{n\times p}$ the control matrix, $f(x_t) \in \mathbb{R}^{n\times 1}$ the passive dynamics, $u_t \in \mathbb{R}^{n\times 1}$ the control vector and $\epsilon_t \in \mathbb{R}^{p\times 1}$ white noise with zero mean and variance Σ_ϵ. Many robotics systems can be modelled by this type of dynamics.

For the finite horizon problem $t \in [t_i : t_N]$, the aim is to find the control inputs $u_{t_i:t_N}$ which minimize the value function:

$$V(x_{t_i}) = V_{t_i} = \min_{u_{t_i:t_N}} E_{\tau_i}\Big[R(\tau_i)\Big] \tag{2.23}$$

where $E_{\tau_i}[.]$ is the expectation over all trajectories τ_i and R represents the finite cost function over a trajectory τ_i, with initial state x_{t_i} at initial time t_i and ending time t_N, defined as:

$$R(\tau_i) = \phi_{t_N} + \int_{t_i}^{t_N} r_t dt, \tag{2.24}$$

where $\phi_{t_N} = \phi(x_{t_N})$ is the terminal cost at ending time and r_t denotes the immediate cost at time t, described as:

$$r_t = r(u_t, x_t, t) = q_t + \frac{1}{2} u_t^T \mathrm{R} u_t \tag{2.25}$$

being $q_t = q(x_t, t)$ an arbitrary state-dependent cost function and R a positive semi-definitive weight matrix of the quadratic control cost.

Having this formulation of the problem, the stochastic HJB equation, provided by [26], is expressed as follows:

$$-\partial_t V_t = \min_u \left(r_t + (\nabla_x V_t)^T F_t + \frac{1}{2} \mathrm{trace}((\nabla_{xx} V_t) G_t \Sigma_\epsilon G_t^T) \right), \tag{2.26}$$

where $F_t = f(x_t, t) + G(x_t) u_t$, ∇_x and ∇_{xx} refers to the Jacobian and the Hessian, respectively, of the value function with respect the state x, while ∂_t is the partial derivative with respect to time. Inserting Equation (2.25) into (2.26) allows to find the minimun, after determining the gradient with respect to u and setting it to zero. The resulting optimal controls are given by:

$$u(x_{t_i}^*) = u_{t_i}^* = -R^{-1} G_{t_i}^T (\nabla_{x_{t_i}} V_{t_i}). \tag{2.27}$$

Optimal controls can be inserted in (2.26), resulting in a second order partial differential equation for the time derivative of the value function:

$$\begin{aligned} -\partial_t V_t = q_t + (\nabla_x V_t)^T f_t - \frac{1}{2} (\nabla_x V_t)^T G_t R^{-1} G_t^T (\nabla_x V_t) \\ + \frac{1}{2} \mathrm{trace}((\nabla_{xx} V_t) G_t \Sigma_\epsilon G_t^T), \end{aligned} \tag{2.28}$$

In order to find a solution to (2.28), a transformation of the value function can be applied such as:

$$V_t = -\lambda \log \Psi_t, \tag{2.29}$$

which together with the assumption

$$\lambda R^{-1} = \Sigma_\epsilon, \tag{2.30}$$

allows a simplification of the HJB equation into:

$$-\partial_t \Psi_t = \frac{-1}{\lambda} q_t \Psi_t + f_t^T (\nabla_x \Psi_t) + \frac{1}{2} \mathrm{trace}((\nabla_{xx} \Psi_t) G_t \Sigma_\epsilon G_t^T) \tag{2.31}$$

with the boundary condition $\Psi = \exp\big(-\frac{1}{\lambda}\phi_{t_N}\big)$. Applying the Feynman-Kac theorem, the solution for the exponentially transformed value function can be formulated as:

$$\Psi_{t_i} = \lim_{dt \to 0} \int P(\tau_i | x_i) \exp\left[-\frac{1}{\lambda}\left(\phi_{t_n} + \sum_{j=0}^{N-1} q_{t_j} dt\right)\right] d\tau_i. \tag{2.32}$$

Therefore, the stochastic optimal control problem has been transformed into an approximation problem of a path integral, where the optimal control can be directly derived as a form of an expectation:

$$u_{t_i} = \int P(\tau_i) u(\tau_i) d\tau_i. \tag{2.33}$$

where

$$u(\tau_i) = R^{-1} G_{t_i}^T \Big(G_{t_i} R^{-1} G_{t_i}^T\Big)^{-1} \Big(G_{t_i} \epsilon_{t_i} - b_{t_i}\Big), \tag{2.34}$$

where b_{t_i} is a complex term, whose derivation is explained in [1] and $P(\tau_i)$ represents the probability of the trajectory τ_i

$$P(\tau_i) = \frac{\exp^{-\frac{1}{\lambda} S(\tau_i)}}{\int \exp^{-\frac{1}{\lambda} S(\tau_i)} dt}, \tag{2.35}$$

where $S(\tau_i)$ is the general cost of the path given by:

$$\begin{aligned} S(\tau_{i,k}) =& \phi_{t_N,k} + \textstyle\sum_{j=i}^{N-1} q_{t_j,k} \\ &+ \frac{1}{2} \textstyle\sum_{j=i+1}^{N-1} \Big(\theta + M_{t_j,k} \epsilon_{t_j,k}\Big)^T R \Big(\theta + M_{t_j,k} \epsilon_{t_j,k}\Big). \end{aligned} \tag{2.36}$$

Thus, the optimal control is the expectation of the local controls when they are associated with their probabilities [6]. The probability is inversely proportional to the path cost, hence low-cost paths have a high associated probability, strongly influencing the outcome.

Parameterized Policies

PI^2 focuses on the direct policy learning, which means learning the policy parameter such as this policy becomes optimal. The stochastic policy is linearly parameterized as:

$$a_t = g_t^T(\theta + \epsilon_t), \tag{2.37}$$

where g_t is a vector containing the basis functions, θ is the parameter vector and ϵ_t is the exploration noise which its variance is set by the user as an exploration parameter.

The action a_t can be interpreted as an input to the control system such as a control command, a reference to the system to follow or a control gain.

For the case of this thesis, DMPs will be used as parameterized policies, where the input to the control system is a reference trajectory created by the DMPs. The usefulness of this parameterized policies has been proven in many studies as [1], [8], [18] or [21]. Notwithstanding, these studies focused on the discrete DMP,whereas this thesis proposed a formulation for the periodic case.

The dynamic system can be expressed in the form of the system (2.22):

$$\frac{1}{\tau}\begin{pmatrix}\dot{\phi}_t\\ \dot{y}_t\\ \dot{z}_t\end{pmatrix} = \begin{pmatrix}2\pi\\ z_t\\ \alpha_z(\beta_z(g-y_t)-z_t)\end{pmatrix} + \begin{pmatrix}0\\ 0\\ g_t^T\end{pmatrix}(\theta+\epsilon) \tag{2.38}$$

where

$$g_t = \frac{r}{\sum_{i=1}^{N}\psi_i(\phi)}\begin{bmatrix}\psi_1(\phi)\\ \vdots\\ \psi_N(\phi)\end{bmatrix}, \tag{2.39}$$

and θ contains the weight vector of the DMPs.

As it can be seen, the system presents a controlled part whose state is z_t and an uncontrolled part with ϕ and y, thus PI^2 is only applied to the controlled part. The PI^2 pseudocode is summarized in Algorithm 2.

Note that the parameter averaging takes into account several considerations, every parameter update $\delta\theta_{t_i}$ is weighted by two terms, its activation kernel ψ_{j,t_i} (Equation (2.6)) and the number of time steps left in the trajectory. This weighted average gives early points in the trajectory a higher importance as their parameters affect a large time horizon.

Importance characteristics

In PI^2, the exploration of the state-space is done by propagation of the DMPs rather than sampling the whole space. For high dimensional spaces, it is not possible to sample the whole state space. Therefore, PI^2 is able to perform in high dimensional spaces, which is typically the case of robots.

To apply PI^2, no discretization is needed, but instead it works in continuous state-action spaces, becoming suitable to perform in robotics systems.

In contrast with others reinforcement learning methods, such as policy gradient, PI^2 does not explicitly calculate the gradient, which is normally sensitive to

Algorithm 2: PI^2 algorithm [1]

```
Input :
  $r_t = q_t + \theta_t^T R \theta_t$;    immediate cost function
  $\phi_{t_N}$;                           terminal cost
  $a_t = g_t^T(\theta + \epsilon_t)$;     parametrised policy
  $\Sigma_{\epsilon_t}$;                  variance of the zero-mean noise
  $\theta_0$;                             initial parameters
  $K$;                                    number of roll-outs per update
Output:
  $\theta$;                               parameter vector
while trajectory cost R not converged do
    Create K roll-outs of the system with same initial state $x_0$ using
      stochastic parameters $\theta + \epsilon_t$
    foreach k in K do
        • $M_{t_j,k} = \frac{R^{-1} g_{t_j,k} g_{t_j,k}^T}{g_{t_j,k}^T R^{-1} g_{t_j,k}}$
        • $S(\tau_{i,k}) = \phi_{t_N,k} + \sum_{j=i}^{N-1} q_{t_j,k}$
            $+ \frac{1}{2} \sum_{j=i+1}^{N-1} \left(\theta + M_{t_j,k}\epsilon_{t_j,k}\right)^T R \left(\theta + M_{t_j,k}\epsilon_{t_j,k}\right)$
        • $P(\tau_{i,k}) = \frac{\exp^{-\frac{1}{\lambda} S(\tau_{i,k})}}{\sum_{k=1}^{K} \exp^{-\frac{1}{\lambda} S(\tau_{i,k})}}$
    end
    foreach i in Time steps N do
        • $\delta\theta_{t_i} = \sum_{k=1}^{K} P(\tau_{i,k}) M_{t_i,k} \epsilon_{t_i,k}$
    end
    $[\delta\theta]_j = \frac{\sum_{i=0}^{N-1} (N-i) \psi_{j,t_i} [\delta\theta_{t_i}]_j}{\sum_{i=0}^{N-1} (N-i) \psi_{j,t_i}}$
    Update $\theta \leftarrow \theta + \delta\theta$
    Create a noiseless roll-out to check the trajectory cost
    with the new parameters
    $R = \phi_{t_N} + \sum_{j=i}^{N-1} r_{t_i}$
end
```

noise and large derivatives. Besides the performed calculations are relatively simple making PI^2 computationally robust.

As it can be seen in the development of the method, only the control transition matrix of the model needs to be know. Furthermore, the designer can set the control system's structure so that the control transition matrix is known, entering a model-free domain.

Additionally, PI^2 has no tuning parameters except from the exploration noise ϵ, which offers an advantage comparing with others reinforcement learning methods where the designer has to tune many parameters to achieve a desired performance.

3

Robot and Software

3.1 The robot

This section explains the robot itself, both hardware and software parts.

The robot was originally created by Dr.Yamada [34], and posteriorly redesigned by the Department of Automatic Control at LTH. The robot, shown in Figure 3.1, is composed of four independent fingers, each one with 3 degrees of freedom, because of 3 revolute joints. For modelling each finger of the robots, we consider the following Denavit–Hartenberg parameters:

Table 3.1: DH parameters of the finger

Link	a_i	α_i	d_i	θ_i
1	L_1	$\pi/2$	0	θ_1
2	L_2	0	0	θ_2
3	L_3	0	0	θ_3

Where $L_1 = 38.5$ mm and $L_2 = 47.5$ mm. The value of L_3 depends on the dimensions of the end effector. In this case the end effector corresponds to a screw coupled magnetically to the servo motor such $L_3 = 51.6$ mm.

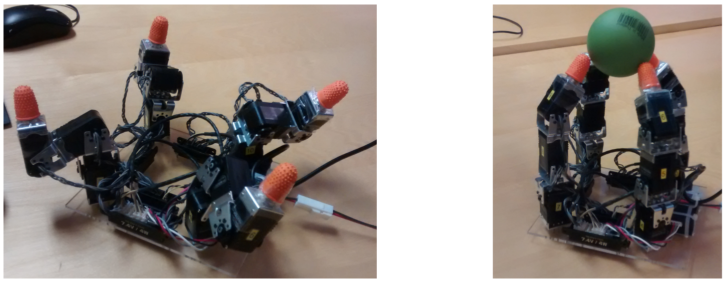


Figure 3.1: The robot.

Table 3.2: Base transformation parameters

Finger	x_i (mm)	y_i (mm)	ψ_i (°)
1	46	46	45
2	-46	-46	-135
3	-46	46	135
4	46	-46	-45

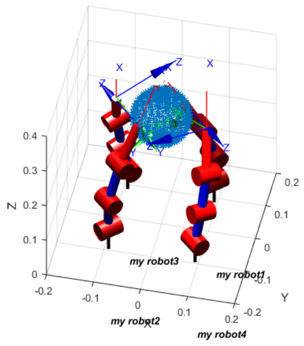


Figure 3.2: Model of the robot in the robotics toolbox.

The base of the generic finger is described by the transformation consisting of a translation along the z axis by h followed by a rotation of -90 degrees around the y axis with respect the world frame, where $h = 28.1$ mm.

Every finger is located in a different position with a different orientation around the z axis with respect the world frame. Hence, the base of each finger is specified by a transformation that premultiply the previously described generic base.

$$Base_i = \underbrace{\text{transl}(x_i, y_i, 0)\text{rotz}(\psi_i)}_{\text{specific}}\underbrace{\text{transl}(0,0,h)\text{roty}(-90)}_{\text{generic}}. \tag{3.1}$$

Table 3.2 shows the values of the specific transformation for each of the fingers. This modelling of the robot is used in both, Matlab and V-REP. In Matlab, these functions are supplied by the robotics toolbox [35] created by Peter Corke. This Matlab toolbox allows modelling, study and simulation of robotics arms, for example, kinematics and dynamics. Figure 3.2 displays the robot in this environment. However, this library is very simple and its only purpose is to generate trajectories and accomplish joint/Cartesian space transformations, and it is not used for simulating interactions with the environment. Therefore, this function will be addressed with V-REP.

The robot and the communication system were provided by the department of Automatic control at LTH. The setup of the robot was not changed. The libraries for

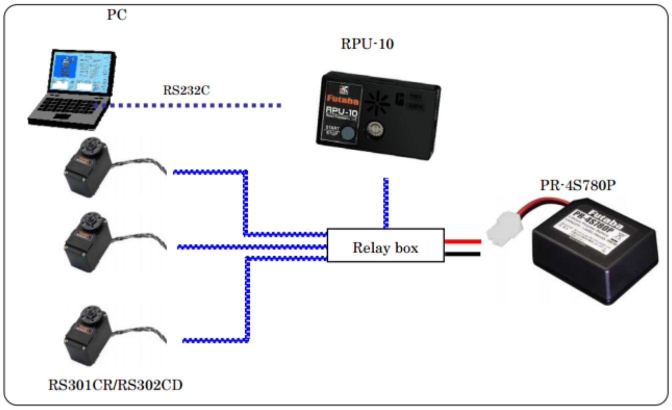


Figure 3.3: Scheme of connections between servo-motors and computer, Standard for serial communication transmission of data are RS232 and RS485. All servos and the power source are connected to a relay box, which also connects with the processing unit, which is in charge of exchange of information between servos and the control station (laptop). Source [47].

the communication were written in C, and they were composed of API functions oriented to send control commands and receive the state of the servos. The servo's controllers were internally implemented, therefore no modifications were possible. The control commands sent to the servo are reference position and movement duration, and therefore the control type is position control. The information feedbacked from the robot includes motor current (which is directly proportional with torque), position, speed, temperature, input voltage and input control time.

The hardware of the robot is composed of 12 servo motors RS301CR-H3, attached together through some connecting elements, a power source TB-RV71EH and a bus-powered USB device RSC-U485. The information about specifications of the servos and connections is available at [47], the resolution of the servos is 0.1° and the resolution time for control is 10 ms. However the communication process constrains the robot to a limited bandwidth, which is related to the serial port baudrate set to 115200 bps. A scheme for the servo connection is displayed in Figure 3.3, as well as the disposition of the servos is shown in Figure 3.4.

Figure 3.5 shows a diagram of a generic finger along with an image of the real finger of the robot, where the reader can see the connecting elements referred above.

Although all calculations are performed by software, it might be insightful for the reader to know the direct kinematics of each finger:

$$\begin{array}{l} x = -L_2\sin(\theta_2) - L_3\sin(\theta_2+\theta_3) \\ y = \sin(\theta_1)(L_1 + L_2\cos(\theta_2) + L_3\cos(\theta_2+\theta_3)) \\ z = h + \cos(\theta_1)(L_1 + L_2\cos(\theta_2) + L_3\cos(\theta_2+\theta_3)) \end{array} . \tag{3.2}$$

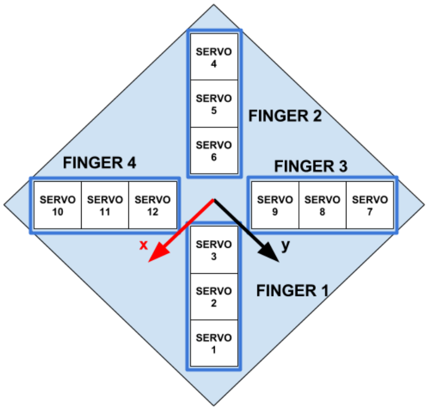
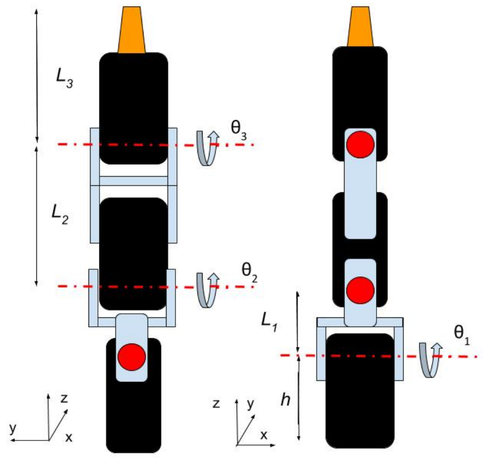
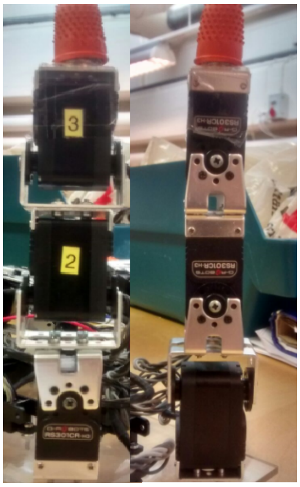


Figure 3.4: Servo arrangement; outer servos represent the base's servos of each finger.



(a)



(b) Real finger

Figure 3.5: Diagram of a generic finger (a) and real finger (b).

The kinematics of the finger can be considered atypical when comparing with usual examples of 3 DoF robots, because of the absence of a joint around the global z axis, which increases the complexity of the inverse kinematic problem, addressed in the next chapter.

From the above kinematics the geometric Jacobian matrix can defined as follows:

$$J(\theta) = \begin{bmatrix} 0 & -L_2c_2 - L_3c_{23} & -L_3c_{23} \\ c_1(L_1 + L_2c_2 + L_3c_{23}) & s_1(-L_2s_2 - L_3s_{23}) & s_1(-L_3s_{23}) \\ -s_1(L_1 + L_2c_2 + L_3c_{23}) & c_1(-L_2s_2 - L_3s_{23}) & s_1(-L_3s_{23}) \end{bmatrix} \quad (3.3)$$

where $c_1 = \cos\theta_1$ and $c_{23} = \cos(\theta_2 + \theta_3)$. The Jacobian matrix will be used to obtain end effector forces from motor torques.

3.2 Simulation Software V-REP

Having a simulation environment is beneficial when working with real robots, as it is used to create embedded applications for a robot without depending physically on the actual machine, thus saving cost and time. There exists a wide range of robot simulators, each of them aiming for different purposes. For this thesis, the desirable characteristics are the ability to simulate with a certain degree of accuracy the dynamics of a system, namely collision and reactions forces between ball and finger, and also between fingers.

Some surveys such as [33], present a list of the most common robot simulators along with their main characteristics and user satisfaction ratios. For this project, the motivation to use V-REP was twofold, first it offers several physics engines that will be discussed in this section, and secondly it offers several programming approaches including remote API control with several languages. Both features provide a high level of usability. According to [33], V-REP presents the highest user satisfaction ratio.

The following subsections contain information about the characteristics of V-REP, for simplicity only relevant characteristics will be mentioned. For further detailed explanation have a look to the V-REP manual available in [36], from where all this information comes.

3.2.1 Physics Engine

A physics engine is a computer software that provides simulation of certain physical systems, such as rigid body dynamics, soft body dynamics and fluid dynamics. While their main applications are in the video game industry, they are also commonly used for engineering purposes.

V-REP's dynamics module currently supports four different physics engines: the Bullet physics library, the Open Dynamics (ODE), the Vortex Dynamics engine and the Newton Dynamics engine [36]. The reason for this diversity in physics engine support, is that physics simulation is a complex task, that can be achieved with various degrees of precision, speed, or with support of diverse features.

Bullet and ODE feature collision detection and rigid body dynamics, but both are considered primarily a game physics engine. Newton offers stability and speed, features which convert it in an appropriate tool for real time physics engine. Vortex offers high fidelity physics simulation combined with realistic parameters. Vortex seems to be the most accurate physic engine, but in the free version of V-REP a license is required, and it can only run simulations for up to 20 seconds. Therefore, in practice only three physics engines are available, where among them, we focus on Newton and ODE.

3.2.2 Material properties

For each physics engine, V-REP presents several material properties such as friction, restitution, damping, compliance and many others. More information is available at [36]. A remarkable comment on this section, is that for ODE and Bullet, the material parameters provided by the user, does not correspond to real parameters of the material, which may trouble tuning. On the other hand, Newton only possesses few parameters which are bounded, which allows easier choices from the users at the cost of limited modelling. For these reasons, the choice of the dynamic engine is done empirically and may vary depending on the purpose of the simulation.

3.2.3 Joint Control

V-REP includes different types of joint control, but for this thesis all joints present the same type of control which is Torque/Force mode. V-REP presents a control loop property in which the user is able to control the joint choosing among three different control joints:

- **Custom control**: the joint will be controlled by its own script, allowing the user to use any imaginable algorithm.
- **PID control**: the joint will be controlled in position via a PID controller that will adjust the joint velocity in following way:

  $$v_i = \frac{K_p e_i + K_i \sum e_i \triangle t + K_d (e_i - e_{i-1})/\triangle t}{\triangle t}$$
  $$F_i \leq c$$

  where v_i represents velocity and e_i position error. In V-REP this controls differs from the common PID. To clarify, the PID sets the velocity reference and V-REP will try to reach it with an internal force control where the maximum force is bounded to the value c.
- **Spring-damper mode**: the joint will act like a spring-damper system via a force/torque modulation:

  $$v_i \leq c$$
  $$F = K e_i + C(e_i - e_{i-1})/\triangle t$$

  In this case, the velocity is the bounded parameter.

For the simulation of our system, PID control was chosen, as it was the type of control which offers better performance.

3.2.4 Communication with Matlab

V-REP offers a remote API allowing to control a simulation (or the simulator itself) from an external application or a remote hardware (e.g., real robot, remote computer, etc.). The V-REP remote API is composed by approximately one hundred specific functions and one generic function, which can be called from a C/C++ application, a Python script, a Java application, a Matlab/Octave program, an Urbi script, or a Lua script. Still the remote specific functions do not entirely cover all the specific functions of the simulation scripts in V-REP,and therefore it could be the case where V-REP itself performs certain calculus and send the data to Matlab.

The idea is that Matlab will set the joint target periodically while the simulation is running, and the control loop of V-REP will execute its commands. By default, the remote API calls are asynchronous, which means that simulation may advance without taking into account the progress of the API client. Thus, synchronous operational mode is used, in which the API client trigger each simulation step.

3.2.5 Designing dynamic simulations

V-REP allows several ways to model a system. Systems can be imported from CAD files or be directly created in V-REP. V-REP advises several design consideration when simulating a system in order to achieve better simulation performance. Therefore, these guidelines were followed as close as possible.

First, the design of the robot: the robot was built from combinations of primitive shapes (cuboids, cylinders, spheres), as primitives shapes are much more stable and faster during simulations.

Second, the hierarchical structure follows the suggested convention. V-REP classifies shapes according to two characteristics, non-static/static and respondable/non-respondable. During dynamic simulation, static shapes maintain a fix position with respect to their parents, whereas non-static shapes will be directly influenced by gravity or other constraints. Respondable shapes influence each other during dynamic collision, i.e., they produce mutual collision reaction. The model base object is a static respondable shape, which represents the first servo of each finger, whereas the rest of the shapes are modelled as non-static respondable shapes. Furthermore, all non-static shapes are connected through dynamical enabled joints, as it can be seen in Figure 3.6.

The robot was modelled using the actual dimensions. Nevertheless, these dimensions were smaller than suggested (less than 3 cm), so a scaling term was applied to the whole model. Masses as well as inertias were also accordingly scaled. This aspect was important to obtain a stable and realistic simulation. The model can be seen in Figure 3.7.

3.2.6 Simulation configuration

Simulations are run through a main script in V-REP, which is executed every simulation step. This main script is in charge of handling all the functionality of the

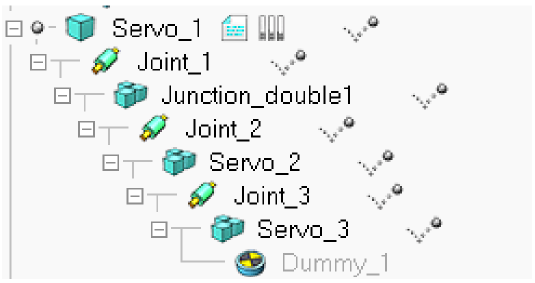


Figure 3.6: Hierarchical structure of a finger. The bouncing ball indicates dynamic enabled elements which the physics engine will take into account.

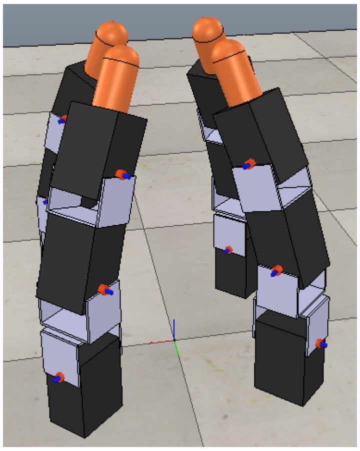


Figure 3.7: Model of the robot designed in V-REP.

simulator, as for example, running possible non-threaded child scripts associated to some objects, send control commands and take some measurements. The default and recommended value for the simulation step is 50 ms, but it can be customized.

The physics engine has a different time step, set by default to 5 ms. That means that for every simulation step the physics engine runs 10 times and the measure received from the simulation usually correspond to the last physics engine step. This implies that if a precise control is required from an remote API, the simulation's and physics engine's time steps should be equal. Unfortunately, it slows down the simulation.

4

Methodology

This chapter describes the methodology adopted in order to setup the experiments in an appropriate way, for both the simulation and the real robot.

4.1 Initial parameters

As mentioned in previous sections, PI^2 relies heavily on the initial parameters, from which it explores to find an optimal solution. Therefore, the choice of good initial parameters is a matter of great importance for this work. One can believe that spinning a ball optimally is easy, but this fact is far from reality. Human hands and robot present different architectures, so optimal initial parameters are unknown a priori. In previous studies, as pancake flipping, ball in a cup or ball paddling, the initial conditions were taught by demonstration when a human operator guided the robot through a good enough trajectory to complete its task. However, this approach is not applicable to this work.

To address the problem, a discrete high-level planing was done in order to obtain certain insight about the optimal finger coordination that maximises ball rotation without letting it fall.

4.1.1 High-level planning

An abstract high-level model of the system is built, in which states and actions are high level representations. This method is a modification of the one used in [2], which addresses the problem for a 3-finger configuration.

Each finger performs an action independently. These actions correspond to staying still, closing or opening, and moving left or right. Writing the action of finger i as a^i, the actions are encoded as:

Given that there are 4 fingers, there are $3^4 = 81$ possible sets of actions.

The proposed state of the fingers, are defined by their relative position with respect to the ball. Denoting r_0 the radius of the ball and ϕ_0 the nominal position of the finger and by using the polar coordinate from the centre of the ball, we can describe:

Table 4.1: Finger Actions.

a^i	Effect
0	Still
1	Open or Close
2	Left or Right

$$Left\ Open = \{\big((r,\phi)|r > r_0\big) \wedge \big(0 < \phi - \phi_0 < 15\pi/180\big)\} \tag{4.1}$$

Hence, each finger presents the following states denoted by s_i:

Table 4.2: Finger States

s^i	Definition
0	Right Open
1	Right Close
2	Left Open
3	Left Close

As we have 4 fingers, the total number of total states is $4^4 = 256$.

Once the states and the actions of the system are set, the next step is to create the dynamics of the system, which in this case is non-linear:

$$s(t+1) = f(s(t), a(t)) \tag{4.2}$$

where

$$f_t = {s^t}_2 \text{ XOR } {a^t}_2 \tag{4.3}$$

and where the subscript 2 denotes the binary representation of the number. This systems dynamics are possible thanks to the encoding of states and actions. Taking a look at the actions, Close and Open action is encoded as 01_2, which combined to a logic exclusive OR is able to change only the lower bit of the states, which corresponds to the open or close states, same occurs with left or right action.

The reward function is defined taking into account the dynamics of the ball. It is considered that the rotation is produced when all fingers in contact with the ball perform a displacement in the same direction. If one finger remains still or moves in the other direction, ball rotation is not considered. Furthermore, stability of the ball must be assured. A stable grip is recognised when more than 2 fingers or the two opposite fingers are in contact with the ball. Transition between states also represents a stability issue. A stable transition means that at least the two opposite fingers are closed in both $s(t+1)$ and $s(t)$ states.

The designed reward function encourages counter-clockwise (ccw) rotation of the ball and discourages clockwise (cw) rotation. Furthermore, it punishes unnecessary movements and more heavily movements which yield to instability.

$$r(s,a) = \begin{cases} 1 & \text{ccw rotation} \\ -1 & \text{cw rotation} \\ -1 & \text{some closed fingers} \\ & \text{still while others moving} \\ -2 & \text{all closed fingers move} \\ & \text{but in different directions} \\ -10 & \text{unstable grip or transition} \end{cases} \tag{4.4}$$

Given the definition of the actions, states and reward function, the Q-learning algorithm is applied with the following parameters for the Equation (2.20):

$$\gamma = 0.9, \qquad \eta = 0.3, \qquad \epsilon = 0.01 \tag{4.5}$$

After computing several times the learning algorithm with approximately 3000000 iterations, two optimal sequences were reached. Figures 4.1 and 4.2 display those sequences. Initial conditions are irrelevant, since our objective is to find a periodic sequence. The following tables show the sequence of states and actions to be taken:

Table 4.3: State-transition sequence 1 for optimal spinning of the ball.

t	s_t	a_t		s_{t+1}	r_{t+1}
1	[3,1,3,1]	[1,2,1,2]	$\Rightarrow$	[2,3,2,3]	1
2	[2,3,2,3]	[2,0,2,0]	$\Rightarrow$	[0,3,0,3]	0
3	[0,3,0,3]	[1,0,1,0]	$\Rightarrow$	[1,3,1,3]	0
4	[1,3,1,3]	[2,1,2,1]	$\Rightarrow$	[3,2,3,2]	1
5	[3,2,3,2]	[0,2,0,2]	$\Rightarrow$	[3,0,3,0]	0
6	[3,0,3,0]	[0,1,0,1]	$\Rightarrow$	[3,1,3,1]	0

Table 4.4: State-transition sequence 2 for optimal spinning of the ball.

t	s_t	a_t		s_{t+1}	r_{t+1}
1	[2,1,3,1]	[2,2,1,2]	$\Rightarrow$	[0,3,2,3]	1
2	[0,3,2,3]	[1,0,2,0]	$\Rightarrow$	[1,3,0,3]	0
3	[1,3,0,3]	[0,0,1,1]	$\Rightarrow$	[1,3,1,2]	0
4	[1,3,1,2]	[2,1,2,2]	$\Rightarrow$	[3,2,3,0]	1
5	[3,2,3,0]	[0,2,0,1]	$\Rightarrow$	[3,0,3,1]	0
6	[3,0,3,1]	[1,1,0,0]	$\Rightarrow$	[2,1,3,1]	0

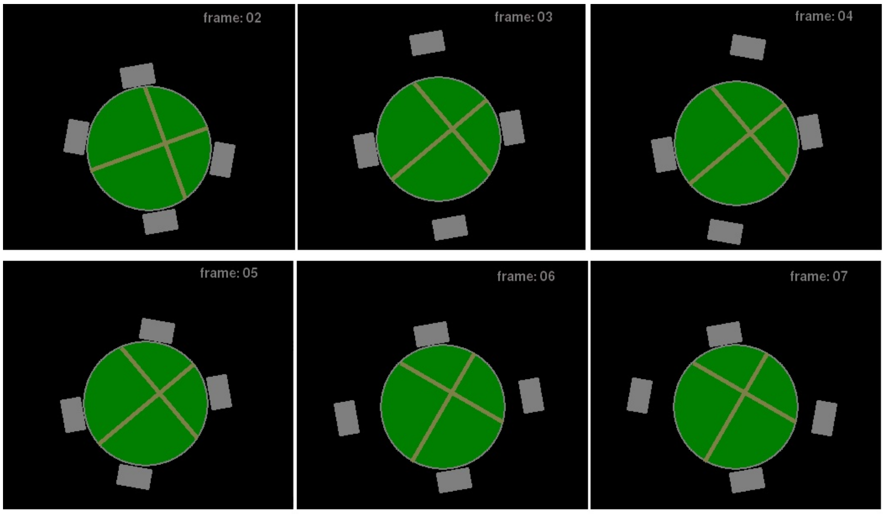


Figure 4.1: Optimal sequence 1.

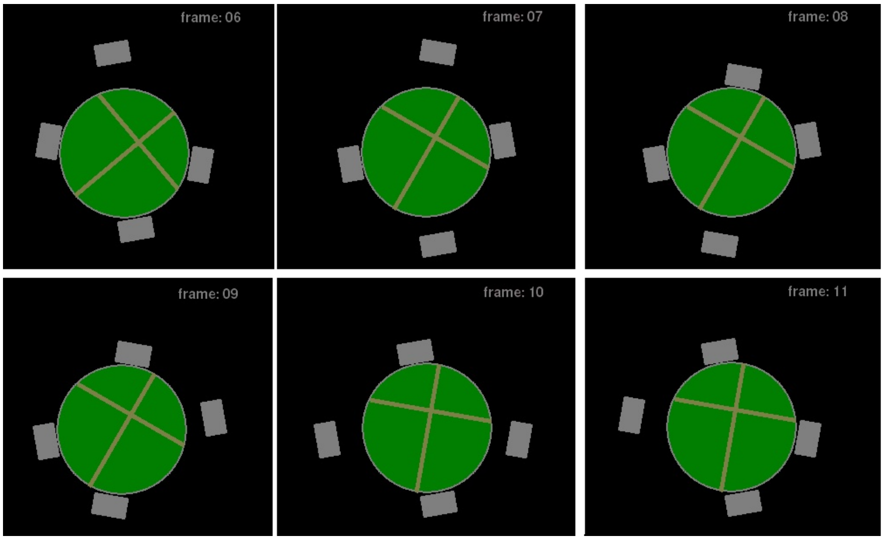


Figure 4.2: Optimal sequence 2.

In fact, these optimal sequences are not unique, as there can exist small variations. For example, in Sequence 1, when $t = 2$ it is possible to have either [2,3,2,3] or [2,1,2,1], which only means a delay in the movements of finger 2 and 4, and does not affect the total reward of the cycle.

As it can be observed, both sequences present a duty cycle of 2/6, i.e., 2 rotations in a sequence of 6 movements. Comparing both sequences, several conclusions can be extracted. First, Optimal sequence 1 is symmetric for opposite fingers, and thus the dimensionality of the problem can be reduced, presenting a major advantage to the problem. The main difference is that Sequence 1 relies in two finger's grip and Sequence 2 in three finger's grip. It could be the case that the stability conditions of the ball for this experiment do not accurately match the real ones. For Sequence number 2, the 3 finger's grip might be unstable configuration as the sum of the applied forces may not be zero. Hence, the project will focus on Sequence number 1.

4.1.2 Theoretical Approach: Inverse Kinematics

Once there is an overall understanding about the optimal trajectory, the next step is to create a continuous trajectory. As the learning algorithm is going to be applied in the joint space, there is a need to deduce a joint trajectory with enough points, in which the DMPs are successfully initialized.

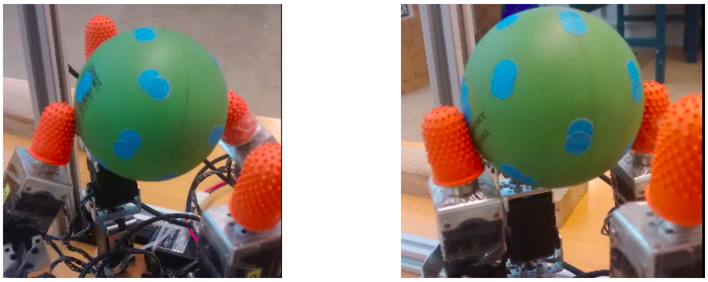
In order to transform from task space to joint space, the Inverse Kinematic (IK) problem must be solved. For the configuration of the finger, solving the IK problem through the algebraic approach is too complex. Fortunately, the robotic toolbox [35] provides two different methods to solve this problem. The typical one is to use the function *ikine*. However, it only outputs one solution among the possible ones. For this reason, the function *ikine_sym* was used, which provides a symbolic solution from which all possible configurations can be deduced. This function uses the Pieper's approach [38], which theoretically it works only for robots whose consecutive axis intersect. Surprisingly, all provided solutions were checked with the forward kinematics function, and were classified as valid as long as the tested points were inside the workspace.

These trajectories were constructed in Matlab, and were used in the simulation, as the results in the real world were unsatisfactory. The type of grip can be seen in Figure 4.3a.

4.1.3 Empirical Approach

The previous approach was built upon a simplistic model of the robot, so it did not consider its own shape. Using the real robot offers some awareness about a stable grip of the ball. For example, a stable grip was achieved when the robot contacts the ball, not with the fingertip, but with the finger surface as in Figure 4.3b.

Joining this knowledge with the obtained Sequence 1 from Section 4.1.1, a trajectory was generated in which the robot managed to rotate the ball. However, this trajectory was composed of a reduced number of points, which hindered the initialization of the DMPs. By linear interpolation, the number of points were increased


(a) Grip of the theoretical approach.

(b) Grip of the Empirical approach.

Figure 4.3: Ball grips.

to satisfied a good fitting by the DMPs. These initial conditions were applied to the real robot.

4.2 Algorithm Parameters

Thanks to the assumption of symmetry, the dimensionality of the problem can be reduced from 12 to 6 DoF. Therefore the number of DMPs was set to 6. The number of basis function for each movement primitive was set to 15. To select an appropriate number, two aspects must be taken into consideration. First, it has to be high enough to offer an appropriate fitting for the chosen initial conditions and second, it should be as low as possible to facilitate convergence. The DMPs will provide the reference trajectory to the robot.

For the exploration noise ϵ there are different approaches. The first one is to add noise only on the weight of the maximally activated basis function of the motor primitive, and keep it constant until another function gets higher activation. The other one is to add variable noise to all basis function. For this thesis, the first approach was used as it improves the learning speed. To ensure convergence, the variance of the exploration noise Σ_ϵ is multiplied by a decay factor γ where

$$\gamma = \max\left(\frac{N-i}{N}, \gamma_{min}\right), \tag{4.6}$$

N represents the total number of updates, i is the actual update, and γ_{min} is an arbitrary value set to 0.2. This combines advantages of high and low degrees of exploration, where in the first steps the algorithm presents quick convergence due to high exploration, and benefits of good exploitation in the final steps.

As mentioned, PI^2 was run in Matlab,and the code was an adaptation to cope with periodic DMPs, of the code provided in [42].

4.3 Simulation

V-REP showed bad behaviour in relation with contact surfaces, and therefore the purpose of the simulation was shifted to prove the usefulness of PI^2 for the optimization of the rotation problem, instead of focusing on holding the ball.

The simulation helped to address the problem of reward shaping, the design of a good reward function is ultimately important as it may quicken convergence to a the optimal solution. Specifying a good reward in robotics requires a fair amount of knowledge and may often be hard in practice.

In simulation the ball was attached to a spherical joint whose position in the workspace was fixed see Figure 4.4. The grey cylinder is a non-responsible static shape where the joint is attached, and does not present any dynamical behaviour, but its function is merely visual. The spherical joint is passive, allowing free rotation of the ball among the three axes. In order to have a realistic behaviour of the ball rotation, the angular drag factor was increased. As consequence the ball only rotated when it is in contact with the robot, when the robot was not touching the ball the angular drag compensated the inertia momentum of the ball impeding rotation.

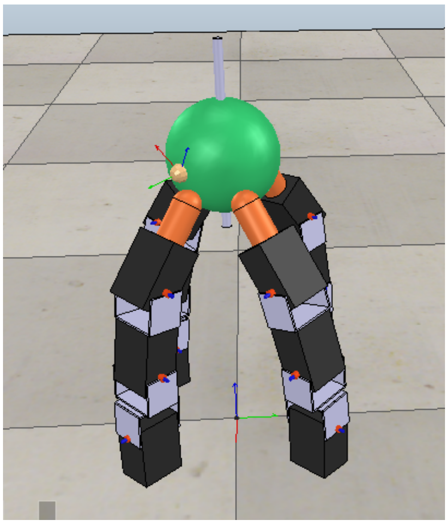


Figure 4.4: Simulation environment consisting in a fixed ball with free rotation axis and the 4-finger-gripper robot.

4.3.1 PI^2 in V-REP

This section explains how the PI^2 algorithm was applied to the simulation environment.

The initialization of the DMP's parameters and the implementation of the algorithm was performed in Matlab. The evaluation of the cost function was done by Matlab and V-REP, actuation was in a synchronous mode. Matlab acted as a client sending the action command to the V-REP model (server). V-REP executed this control commands and sent back some information about the state of the system. Matlab, with the new information, evaluates the cost function at this time instant. Matlab was also in control of start and ending of the simulation process.

The state of the system, such as angular velocities and interaction forces were provided by V-REP, thus their correctness depended on the physics engines. To obtain these parameters, the simulation time step was reduced to 15 ms, and the control command was sent every 3 simulation time steps. This was done in order to rely upon more than one value provided by the physics engine, so for every control command, the angular velocities and forces were retrieved by averaging the received data. The diagram of the process is represented in Figure 4.5. However, this method presented a drawback as it slowed down the simulation.

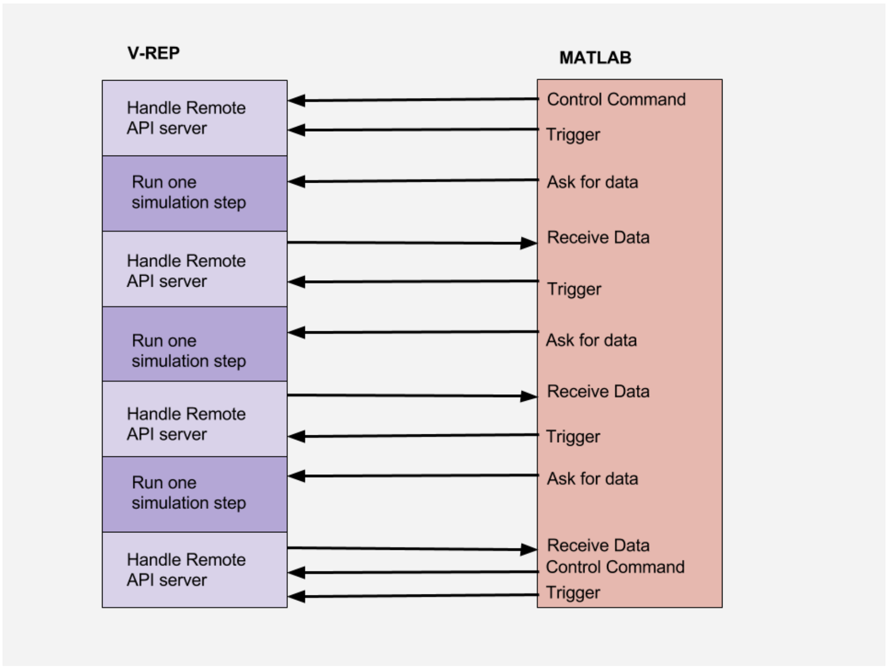


Figure 4.5: Synchronous communication between V-REP and Matlab; Three simulation steps are run for every control command to improve measurements. V-REP will always follow the last control command sent, in case no control command is received.

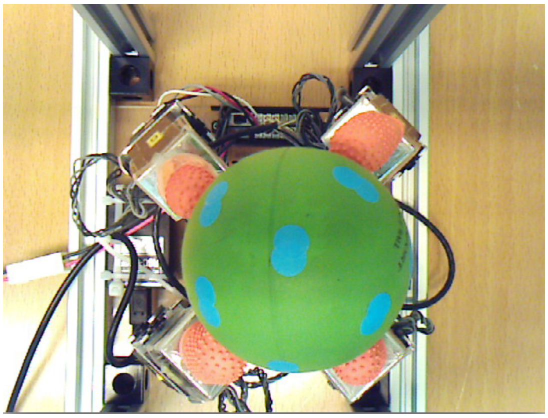


Figure 4.6: View of the workspace from camera.

4.4 Real Robot

The experimental setup for the real robot is more complicated in comparison with the simulation. The process of obtaining information about the system is more difficult and the development of the experiment might present a stochastic behaviour. This section explains how these problems are resolved.

4.4.1 Visual Feedback

In the real implementation, the states of the ball can not be obtained in a simple way as with V-REP. Therefore, a camera is needed to detect ball translations and rotations. The camera is located offering a top view of the workspace as in Figure 4.6. In order to fix the camera in a desired position, an aluminium structure was built, showed in Figure 4.7.

The camera should record a sequence of images, while the robot is performing, and from this sequence, the reward values have to be extracted. Hence, several computer vision techniques should be applied to extract the relevant information. This subsection explains how this process is executed. The algorithm was implemented in C++, using the OpenCV library [44]. OpenCV is an open source library aimed at real-time computer vision.

Due to the fact, that many variables composing the reward function come from the state of the ball, the first step is to segment the ball from the image. As the camera is fixed, one can consider that the only external element in the workspace is the ball. It is an important consideration because as it can be seen in Figure 4.8, the workspace is not composed of a wide range of colours. Hence, as the testing ball is green, the main feature for the segmentation is colour.

The ball surface is chromatically uniform, except from small inscriptions, and thus some landmarks need to be attached to it, in order to be able to estimate ball

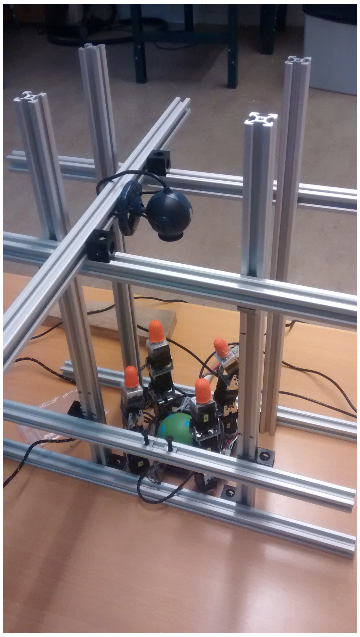


Figure 4.7: Aluminium structure, whose function is to fix the camera with a desirable view and to facilite the user to obtain similar initial conditions of the ball.

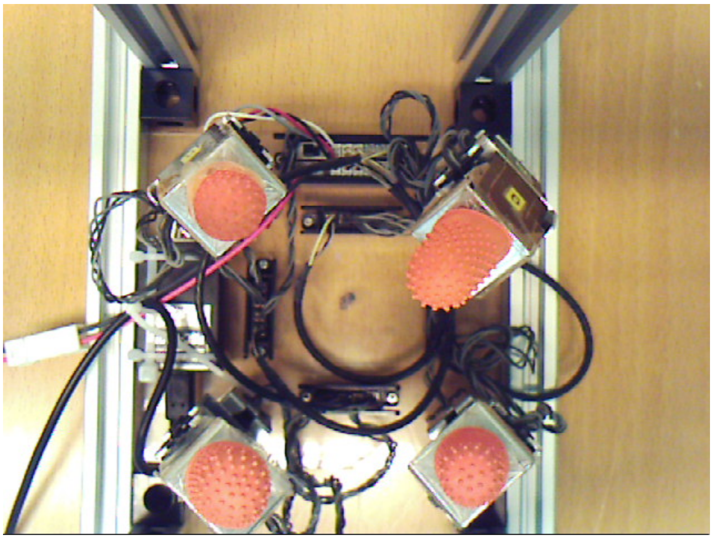


Figure 4.8: View of the workspace without the ball. The lack of green colour makes the color detection feasible.

Table 4.5: HSV limits for detection

	HighH	LowH	HighS	LowS	HighV	LowV
Green	75	38	233	53	233	60
Light-Blue	130	75	255	62	255	70

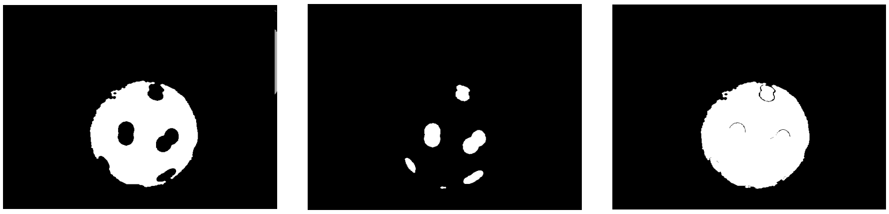

(a) Green colour segmentation (b) Light blue colour segmentation (c) Bitwise OR to segment the ball

Figure 4.9: Binarized images from camera.

rotations. Again, having in mind the effectiveness of the colour segmentation, some light-blue stickers are glued to the ball. Light-blue is the color that offers a good contrast with the workspace facilitating its detection, as can be seen in Figure 4.6. This segmentation is accomplished by following different steps: Firstly, the acquired image is converted into HSV color space, The HSV space is widely used in color segmentation as its offers a differentiation between colour and intensity. Secondly, the image is binarized using the *inRange* function. If all the HSV values of the pixel are within a certain interval previously selected, the pixel will be converted to 255, otherwise the value will be converted to 0. Afterwards, some morphological operations, such as opening and closing, are performed to eliminate noise.

These previous steps are done both for green and blue light color, Table 4.5 shows the bounding parameters for this detection. At this point, the two resulting images are representations of the green and light-blue objects within the workspace, Figures 4.9a and 4.9b . The ball is extracted from a bitwise-OR between both images, see Figure 4.9c.

It is possible to extract the properties from the objects by applying a contour detection to the binarized images. In order to make a robust detection, a minimun area threshold is applied to all objects, eliminating small objects that could have possibly survived the morphological operations.

In the case of the ball, once the contour is extracted, centre position and radius are acquired by means of the *minEnclosingCircle* function. This function provides the centre and the radius of the mininum enclosing circle of a set of points, which for this case form the contour of the ball. This function is highly useful as it perfectly matches the shape of the object to detect, even when some parts are hidden, see Figure 4.10. The same procedure is also applied to the landmark detection,

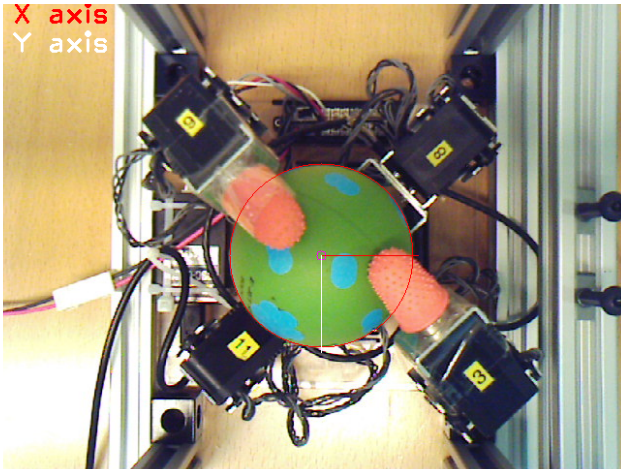


Figure 4.10: Ball detection with obstacles.

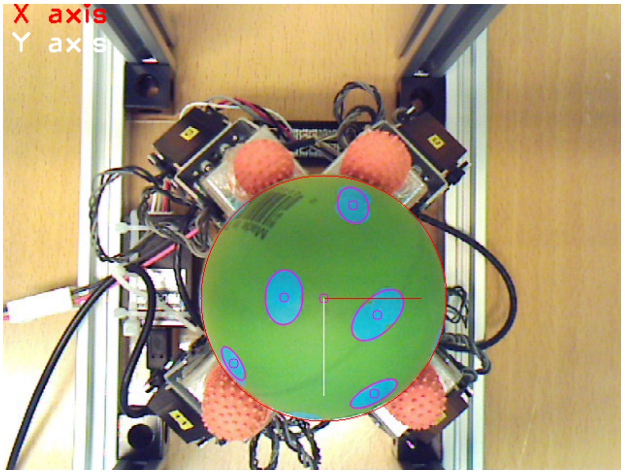


Figure 4.11: Detected ball and landmarks.

but instead of *minEnclosingCircle* function, *fitEllipse* function is used, as the only property of interest is the centre. The results are displayed in Figure 4.11. Only the landmarks whose centre are inside the ball are taken into account, this is done in order to eliminate possible errors.

From this process, the centre and radius of the ball are collected, which are directly related to the cost function. Moreover, the study of the evolution of position of the landmark is related with the angular velocities. Unfortunately, this study might be hindered as a consequence of disapearence and entrance of new landmarks during experimentation.

The tracking of the landmarks is performed by comparing distance between landmarks at consecutive frames. The assumption that the landmark at time t, comes from the nearest landmark at time $t-1$ is valid when Δt is small enough. Landmark coordinates are referenced to the centre of the ball, thus the landmark positions are only affected by ball rotations. The z coordinate is deduced thank to the known shape of the object, by applying the formula:

$$\begin{aligned} \phi &= \arccos \tfrac{x_t^2+y_t^2}{r_{b_t}} \\ z_t &= r_{b_t} sin(\phi) \end{aligned} \tag{4.7}$$

where r_{b_t} is the detected ball radius at time instant t. Then, from the tracking of the landmarks, the angular velocity of the ball can be estimated. Considering the equation of the rotation of a point:

$$r(t) = R(t)r_0, \tag{4.8}$$

where $R(t)$ is the rotation matrix and r_0 the vector of the initial point. The time derivative of this equation, explained in [45], is:

$$\dot{r}(t) = S(\omega)R(t)r_0, \tag{4.9}$$

where $S(\omega)$ is a skew-symmetric matrix. Developing (4.9) in the following manner:

$$\dot{r}(t) = S(\omega)r(t) = \omega \times r(t) = -r(t) \times \omega = -S(r(t))\omega. \tag{4.10}$$

We arrive to an equation where the unknown variable is ω. As $\dot{r}(t)$ corresponds to the linear velocity of the landmark, which can be easily calculated, and $S(r(t))$ is a skew-symmetric matrix of its position defined as:

$$S = \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix}. \tag{4.11}$$

Since there are as many Equations (4.10) as visible landmarks, we deal with a linear least squares systems of the form:

$$\begin{bmatrix} \dot{r}_{1x}(t) \\ \dot{r}_{1y}(t) \\ \dot{r}_{1z}(t) \\ \vdots \\ \dot{r}_{Nx}(t) \\ \dot{r}_{Ny}(t) \\ \dot{r}_{Nz}(t) \end{bmatrix} = - \begin{bmatrix} 0 & -r_{1z} & r_{1y} \\ r_{1z} & 0 & -r_{1x} \\ -r_{1y} & r_{1x} & 0 \\ \vdots & \vdots & \vdots \\ 0 & -r_{Nz} & r_{Ny} \\ r_{Nz} & 0 & -r_{Nx} \\ -r_{Ny} & r_{Nx} & 0 \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}. \tag{4.12}$$

In order to solve this system within the C++ script, the library Eigen [46] was used. This library provides templates for linear algebra: matrix and vector operations, numerical solvers and related algorithms.

4.4.2 Combined Feedback Servo-Visual

Excessive normal forces may damage the servo motors and consume more power, so for the real robot it is a significant aspect to be considered.

As mentioned, from the servo motors, position and torque can be extracted. Knowing them and the relation between torque and force,

$$\tau = J(\theta)^T F, \tag{4.13}$$

the force applied by the fingertip can be derived, by computing the inverse of the transpose Jacobian. However, for robustness purposes, instead of the inverse, the pseudo-inverse is used, Equation (4.14). The reason is that there could exist certain configurations in which the Jacobian matrix is not invertible, and consequently generates a singularity.

$$F = \left(J(\theta)^T\right)^{\dagger}\tau \tag{4.14}$$

The data received from the robot does not present a high accuracy. In fact, the values of the torques and angles have an offset. Therefore, the program has a start-up phase in which it reads these offsets in order to subtract them during the whole data acquisition process.

The force calculated in Equation (4.14) is referenced to the base of each finger. As a result, all forces must be converted to global coordinates by applying the specific transformations detailed in Subsection 3.1. In order to determine the normal force exerted to the ball, the normal vector to the ball needs to be know. To define the normal vector, it has to be assumed that the contact point with the ball would be the fingertip. As explained previously in the real case the contact point is unknown, but we could estimate it, as it would be near the fingertip.

The visual feedback provides us with the position of the centre of the ball. Nonetheless, its coordinates are in pixels and with respect to the camera frame. A calibration of the camera should be done in order to transform to global coordinates. The camera calibration is performed in a separate script. This script will provide some parameters to transform pixel coordinates to global coordinates.

For a correct calibration, the camera should be facing down in a perpendicular way and be located above the centre of the robot, which is the world's origin. First, the user selects the 4 corners of the base of the robot, whose dimensions are known, and as a result the script will calculate the origin's coordinates in pixels and the ratios millimetre-pixel in x and y directions. These 4 parameters define the calibration for the plane x-y.

The calibration of the z axis is done by using the radius of the ball. The radius is directly proportional to its height, but the mapping function is unknown. The user

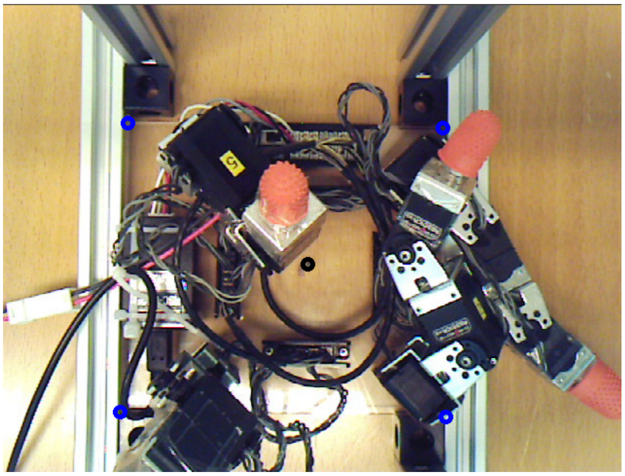
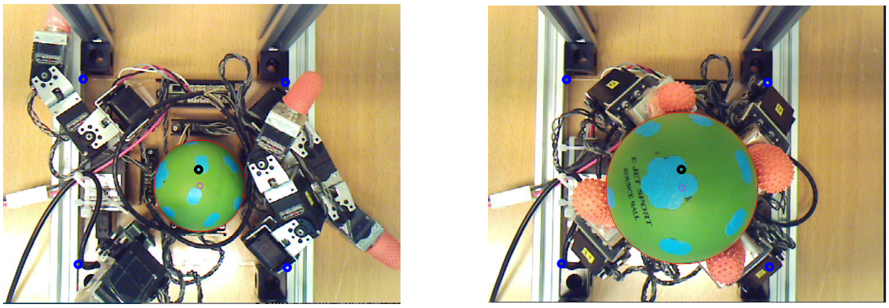


Figure 4.12: Calibration process: The user clicks in the four corners of the robot's base (blue dots) and the origin is computed (black dot).



(a) Low-case calibration.

(b) High-case calibration.

Figure 4.13: Height calibration.

should input two measures of the height as the camera detect the ball, corresponding to when the ball is on the floor or held by the robot. The script will retrieve 4 parameters in order to perform a linear interpolation. During experimentation the height is not expected to vary much, unless the ball drops. Therefore the linear interpolation will compute an acceptable estimation. Table 4.6 summarises the calibration parameters.

The global coordinates of the ball in millimetres are determined as follows:

$$\begin{aligned} x &= (x_p - c_x) r_x \\ y &= (y_p - c_y) r_y \\ z &= h_1 + (r - r_1) \tfrac{h_2 - h_1}{r_2 - r_1} \end{aligned} . \tag{4.15}$$

Table 4.6: Calibration parameters.

Parameter	Meaning
c_x	origin coordinate x in pixels
c_y	origin coordinate y in pixels
r_x	ratio mm/pixel in axis x
r_y	ratio mm/pixel in axis y
h_2	big height
h_1	small height
r_2	big radius
r_1	small radius

where x_p, y_p and r are the ball centre in pixels and the radius.

The global position of the fingertip is determined by applying direct kinematics and the transformation described in Section 3.1. With ball and fingertip positions the normal vector $\vec{n_i}$ is deduced, and the normal force of each finger is computed as follows:

$$\vec{f_{n_i}} = \frac{\vec{f_i} \cdot \vec{n_i}}{\|n_i\|_2} \vec{n_i}. \tag{4.16}$$

4.4.3 Program Structure

The program is structured similarly to the simulation. From Matlab the DMPs are initialized, PI^2 parameters and cost function parameters are set, and parameters updates are performed.

As mentioned earlier, the library files for controlling the robot were written in C and OpenCV works in C++. However, this can be specified to the compiler, so that C libraries can also be used in a C++ program. This is very advantageous as only one program is needed. The C++ program is in charge of reading the reference trajectory, and executing it. Meanwhile it takes data from the camera and the robot, after executing the trajectory it sends them to Matlab.

The communication between Matlab and C++ is done in a simple way, by writing and reading .txt files. This naive way of communication presents some advantages, for instance, there is no need of parallel execution of them, which serves to prevent overheating of the robot.

An operator must be present to direct the process. The operator has the role to inform Matlab, when the execution of the trajectory has finished, to set the ball in the same initial conditions for every roll-out, and inform the C++ program when it can start to execute the trajectory. Figure 4.14 displays a scheme of the method.

4.4.4 Repeatability of Initial Conditions

In the real environment the initial conditions are not exactly the same as in simulation. Repeatability of initial condition is important, as it affects the performance of

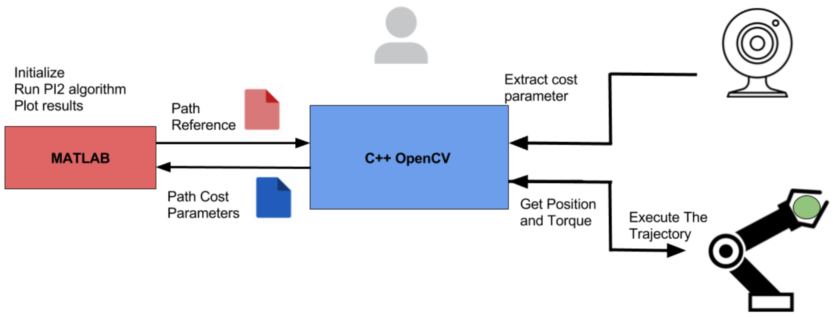
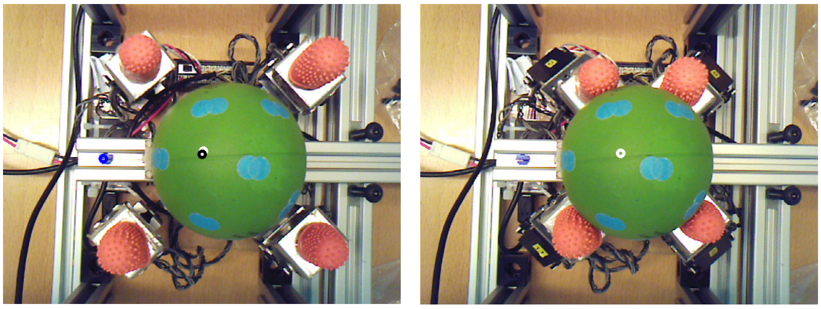


Figure 4.14: Scheme of the training program. The files icons represent .txt files in which the data is stored. All the process needs manual control and supervison from the user.

PI^2. For instance, if the ball is located in a different position before starting every roll-out the cost function may be different for equal trajectories, which is undesirable for the purpose of learning.

To reduce this problem, some elements have been added to the experiment to ensure minimun variations of initial conditions. The ball is inserted to the environment with an aluminium bar and the direction of the aluminium bar's insertion is constrained by some mechanical elements. Moreover, some reference points were marked in both the bar and image,so as to the user ensures overlapping of both points before sending the grasping command to the ball. Figure 4.15 shows the initial condition set-up.



(a) The blue dots should overlap to ensure repeatability.

(b) Robot grasps the ball, and the user can take the bar away.

Figure 4.15: Initial condition setting.

5

Experiments and Results

5.1 Simulation

For the simulation experiments the cost function is defined as:

$$r_t = a_1\omega_x^2 + a_2\omega_y^2 + a_3 f_r + a_4 \sum_{i=1}^{4} f_{n_i}, \tag{5.1}$$

which penalizes rotation around x and y axes as only pure rotation around z is desired. The term f_r denotes the sum of all the forces applied to the ball. The idea behind is that we want to maintain the ball in the same position and therefore, the sum of the interaction forces should be zero. f_{ni} represents the normal force applied by each finger. This term avoids penetrations and excessive forces, as it could be the case that the grasping is stable ($f_r = 0$), but the robot is applying high forces compressing the ball, which may damage the robot. Both f_r and f_{ni} are only considered when their absolute value surpass a certain threshold, Γ_r and Γ_n respectively. The cost function is normalised according to the number of cycles performed by the DMPs. For rewarding counterclockwise rotation around the z axis, two approaches were considered. First, the addition of a terminal cost inversely proportional to the angle rotated by the ball:

$$\phi_t = a_5\big(\theta_{max} - \theta_N\big), \tag{5.2}$$

where θ_{max} is a reference angle big enough to not be reached, it has the form $\theta_{max} = 2,5n$, being n the number of total cycles performed by the DMPs and θ_N is the total rotated angle of the ball, calculated by a reference point attached to the ball. And second, adding in Equation (5.1) the term $-a_5\omega_z$. This term will reduce the cost function by favoring high angular velocities around the z axis. For both cases, the weight a_5 should be high, if the user wants to prioritise this term over the others.

This section presents the result obtained from simulation using various parameters, $a_1, a_2, a_3, a_4, a_5, \Gamma_r, \Gamma_n$ used in Equation (5.1) and various number of updates and noise variance Σ_ϵ of the PI^2. Every update is composed of 10 roll-outs, reusing the best 5 for the next update. The duration of all experiments is of 4τ. This duration was selected in order to give more importance to the periodic part, while also having realistic simulation time. Long simulation runs present the disadvantage of becoming slower. For instance, in an experiment with 80 updates, the first roll-outs were computed in less than 10 seconds, which was not the case for the last roll-outs as they took almost 1 minute.

Test I: Terminal Cost

In the first scenario, the method for rewarding counterclockwise rotation is to add a terminal cost such as in Equation (5.2). This test aims to perform a reduction of all terms in the cost function, and their weights have thus been chosen accordingly, in order to have similar importance. Table 5.1 shows the parameters used for the simulation.

Table 5.1: Parameters used in Test I.

Parameter	$a_1\|a_2$	a_3	a_4	a_5	$\Gamma_r\|\Gamma_n$	Σ_ϵ	$Updates$
Value	5	0.01	10	1000	10	20	80

The results can be seen in the following figures:

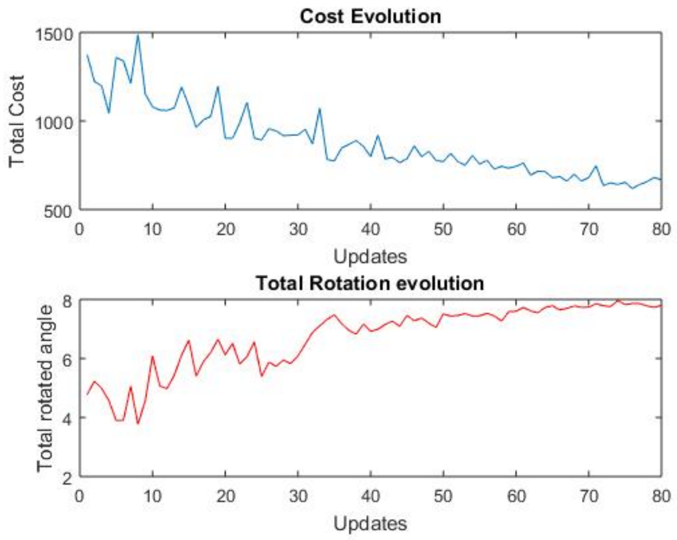


Figure 5.1: ***Test I:*** Total cost evolution (upper). Evolution of the total rotated angle, this term is associated with the terminal cost (lower).

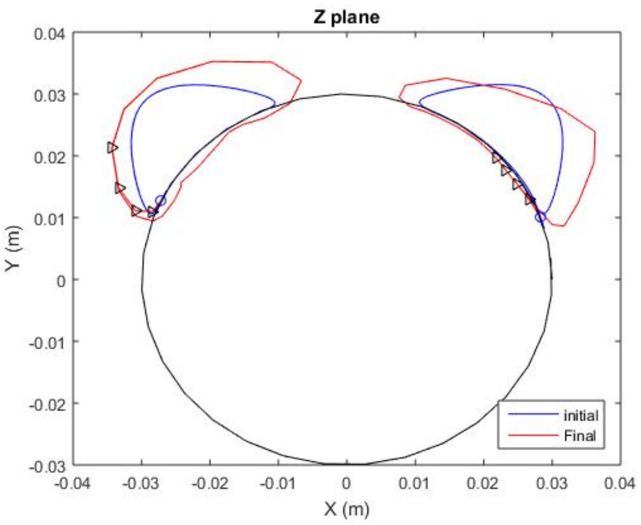


Figure 5.2: ***Test I:***Initial and final reference trajectories of the fingertips. The left one correspond to finger 1 and the right one correspond to finger 3. As the behavior of the fingers is symmetrical, finger 2 and finger 4 have same local reference trajectories as finger 1 and finger 3. Black triangles inform about the direction of the movement.

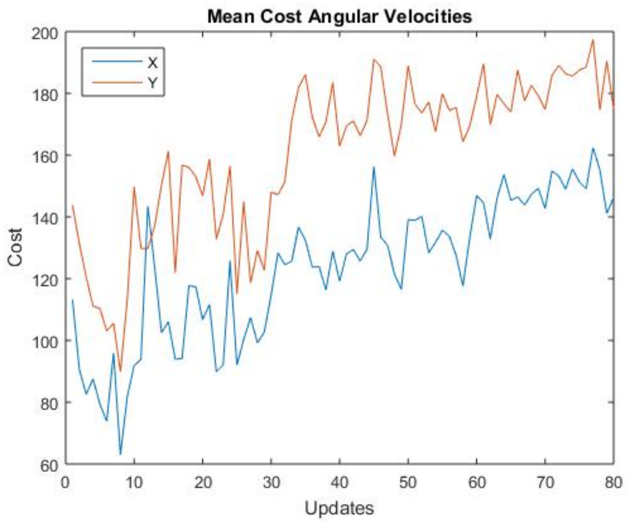


Figure 5.3: ***Test I:*** Cost of angular velocities.

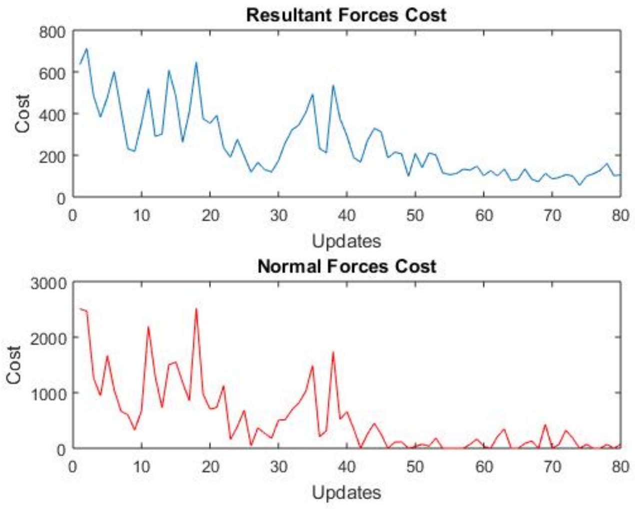


Figure 5.4: ***Test I:*** Forces evolution. The data corresponds to the total sum of the forces of all fingers at every time instant. Applied forces (upper), perpendicular forces to the ball (lower).

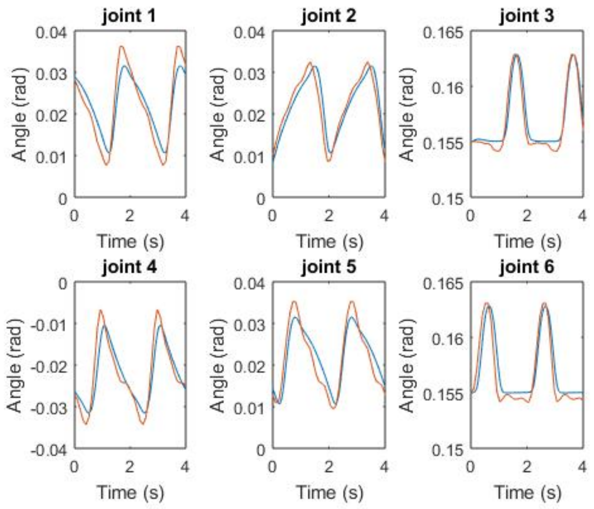


Figure 5.5: ***Test I:*** Joints reference trajectory: Blue initial trajectory, Red final trajectory. These plots show the two first periods. The first period is the transient part whereas all the forthcoming periods have very similar shapes.

Figure 5.4 presents the cost evolution and it can be seen that the cost is decreased nearly to a third of the initial value during the whole of the experiment. In the last updates the decrease is soften but still remains. The best update is number 76 out of 80.

The end effector trajectories, Figure 5.2, extend the contact with the ball while avoiding penetration. This is reflected in the contact forces, displayed at Figure 5.1, whose absolute value is successfully decreased. This test has performed well, reducing the total cost as well as the cost of the different terms, while increasing rotation.

Test II: Angular Term

In this case, there is no terminal cost to boost counterclockwise rotation but a term is added to the immediate cost function to maximize angular velocity as explained above. As the previous test, this test presents a balance between all term of the cost function. Table 5.2 summarizes the parameters of the experiment.

Table 5.2: Parameters used in Test II.

Parameter	$a_1 \vert a_2$	a_3	a_4	a_5	$\Gamma_r \vert \Gamma_n$	Σ_ϵ	$Updates$
Value	10	0.01	10	1000	10	10	80

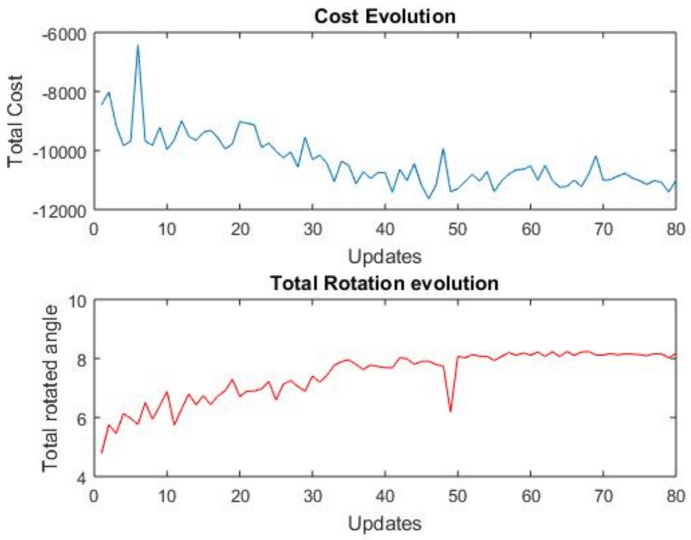


Figure 5.6: ***Test II:*** Total cost evolution (upper). Evolution of the total rotated angle (lower).

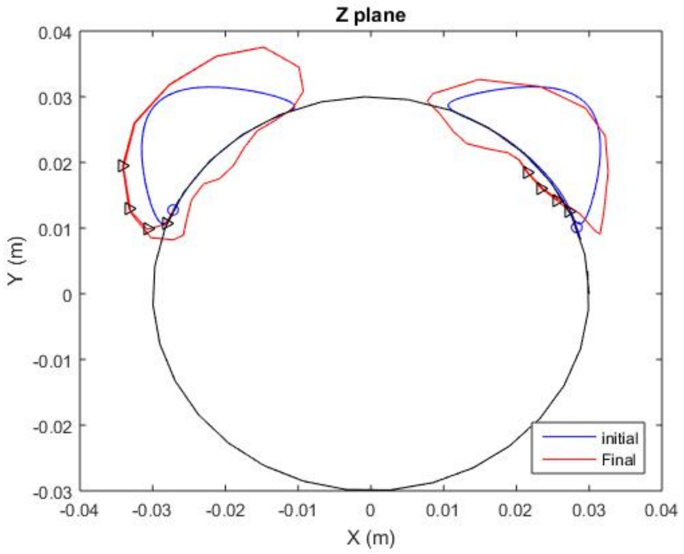


Figure 5.7: ***Test II:*** Initial and final fingertip reference trajectories. The black triangles represents the direction.

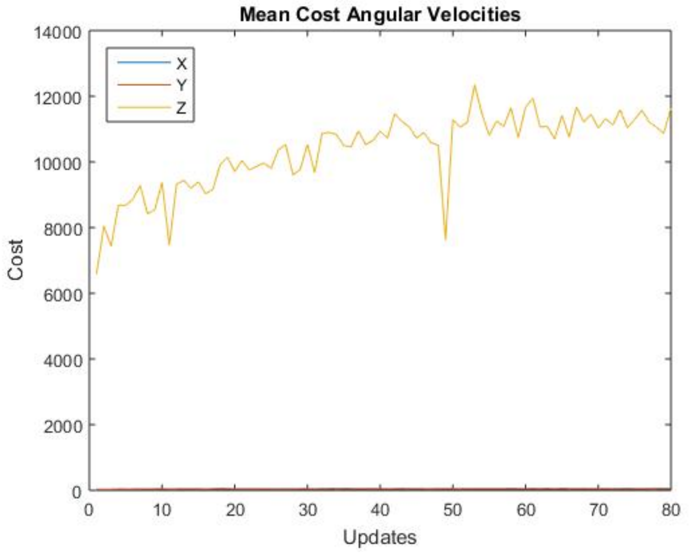


Figure 5.8: ***Test II:*** Cost of angular velocities. The ratios (final/initial) for x, y, and z are 1.17, 1.27 and 1.76 respectively.

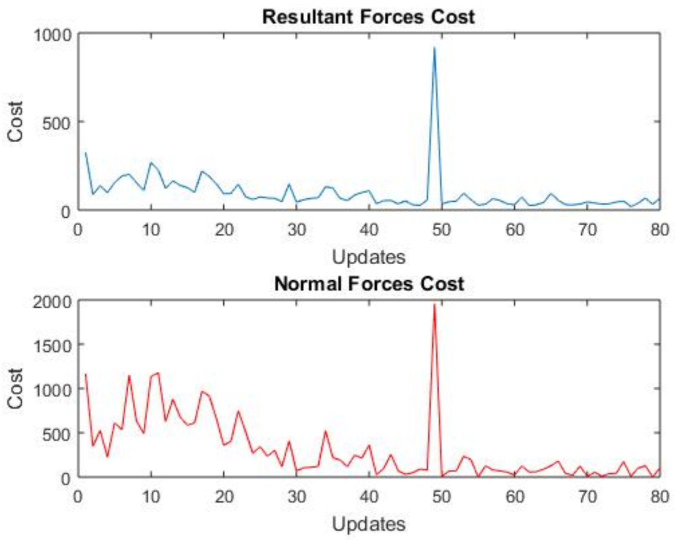


Figure 5.9: ***Test II:*** Total forces per update.

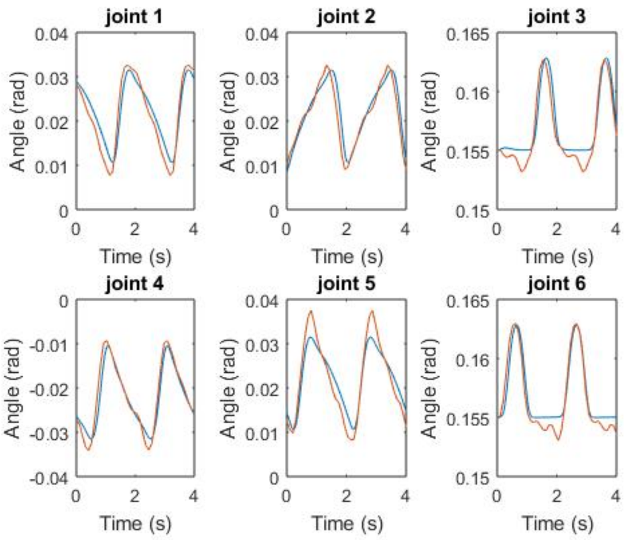


Figure 5.10: ***Test II:*** Joint reference trajectories. Blue initial trajectory, red final trajectory.

The cost evolution is presented in Figure 5.6. In this case the cost decreased during the first steps and later it fluctuates, being the best update at 46 out of 80. However the total rotated angle is a little higher than in the previous case.

The end effector trajectories and joint trajectories, Figures 5.7 and 5.10, present similar characteristics as in the previous example. The same happens with forces, see in Figure 5.9. The performance is satisfactory as costs are reduced.

Test III: High Cost Forces

In this test the weight of the penetration forces is increased, and therefore this test study the ability to spin the ball applying minimun forces, thus resulting trajectories are expected not to penetrate the ball. Terminal cost is the selected approach to deal with z rotation. The chosen parameters are:

Table 5.3: Parameters used in Test III.

Parameter	$a_1 \vert a_2$	a_3	a_4	a_5	$\Gamma_r \vert \Gamma_n$	Σ_ϵ	$Updates$
Value	10	10	100	2000	15	10	80

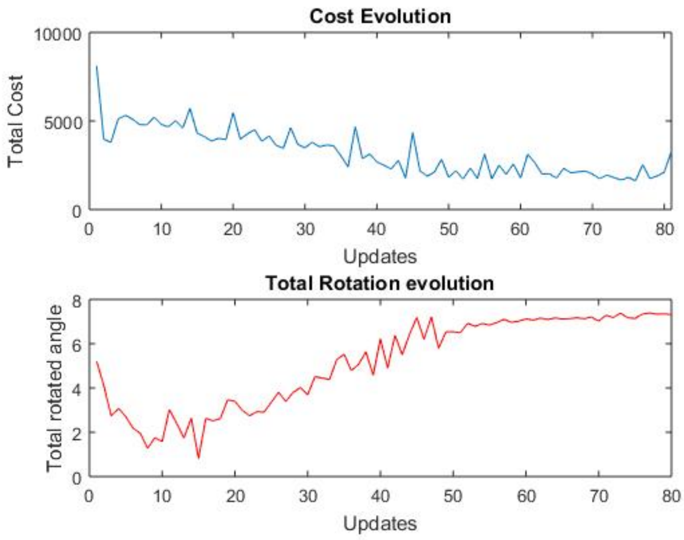


Figure 5.11: ***Test III:*** Total cost evolution (upper). Evolution of the total rotated angle (lower).

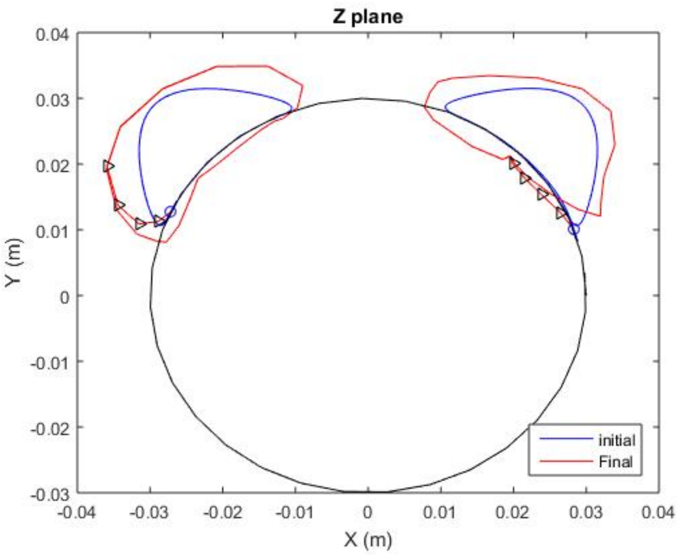


Figure 5.12: ***Test III:*** Initial and final fingertip reference trajectories.

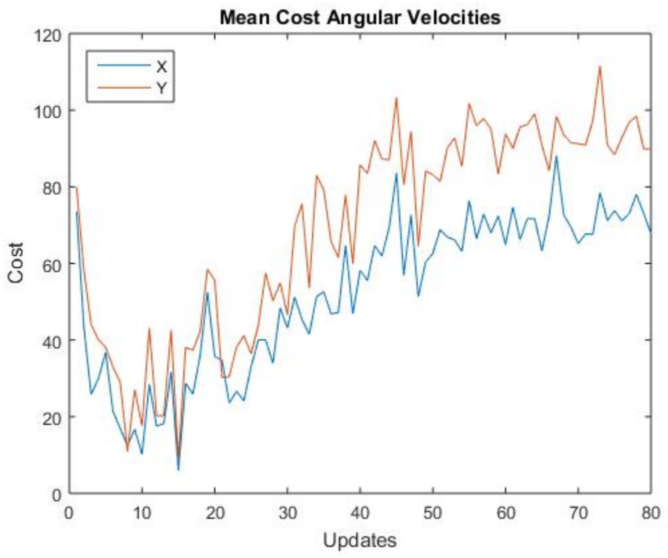


Figure 5.13: ***Test III:*** Cost of angular velocities.

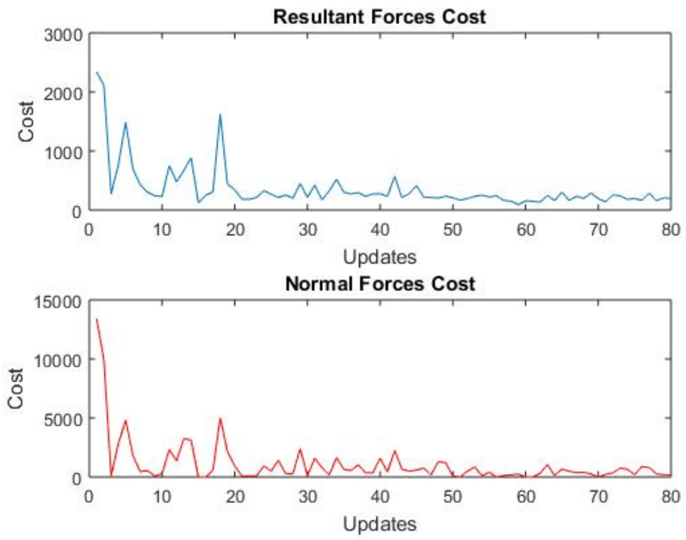


Figure 5.14: ***Test III:*** Total forces per update.

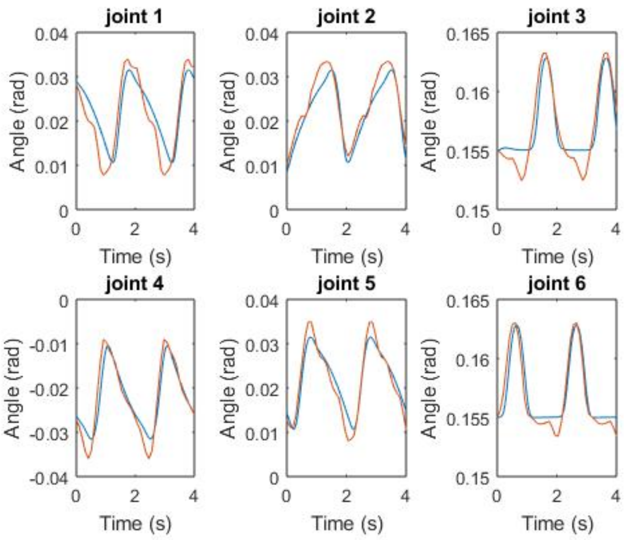


Figure 5.15: ***Test III:*** Joint reference trajectories. Blue initial trajectory, red final trajectory.

The total rotation angle is 7.4 rad as its maximum value, which is somehow smaller than in previous cases, see Figure 5.11. The cost is decreased during nearly all the experiment, the best update is number 76 out of 80.

As foreseen, the robot slightly penetrates the ball, as can be seen in Figure 5.12, and the forces are remarkably reduced, Figure 5.14. The joint trajectories are quite similar with the initial conditions during contact with the ball.

Test IV: High Variance

In this test the variance of the exploration noise is increased, and the weight of the penetration forces is slightly decreased, The main hypothesis is that the resulting trajectory will highly differ compare to the initial conditions, and due to the high variance and lower importance of forces, penetration will occur. Parameters are shown in Table 5.4. Particularly, for this experiment the parameter $\gamma : min$, previously mentioned in Equation 4.6, is set to 0.4 instead of 0.2 as in other cases.

Table 5.4: Parameters used in Test IV.

Parameter	$a_1 \vert a_2$	a_3	a_4	a_5	$\Gamma_r \vert \Gamma_n$	Σ_ϵ	$Updates$
Value	20	0.05	1	2000	20	40	100

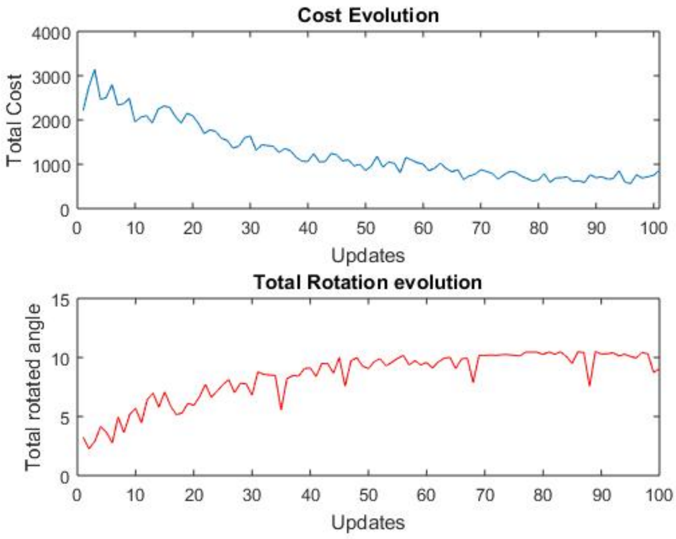


Figure 5.16: ***Test IV:*** Total cost evolution (upper). Evolution of the total rotated angle (lower).

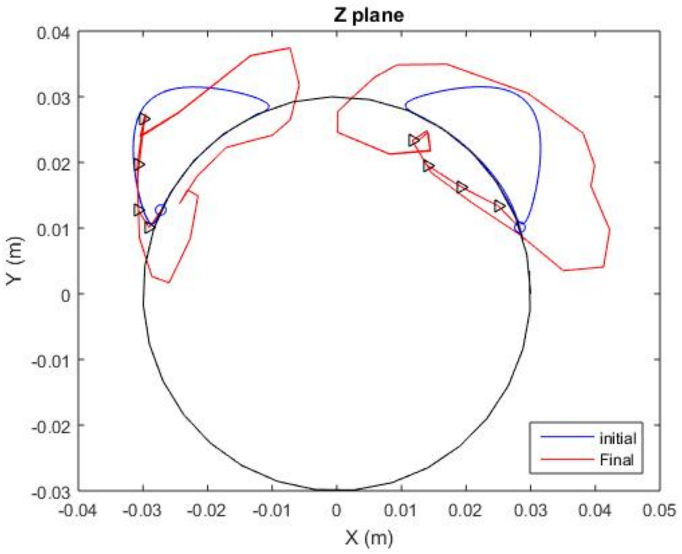


Figure 5.17: ***Test IV:*** Fingertip reference trajectories. The small loops in the contact part are produced due to the high variance.

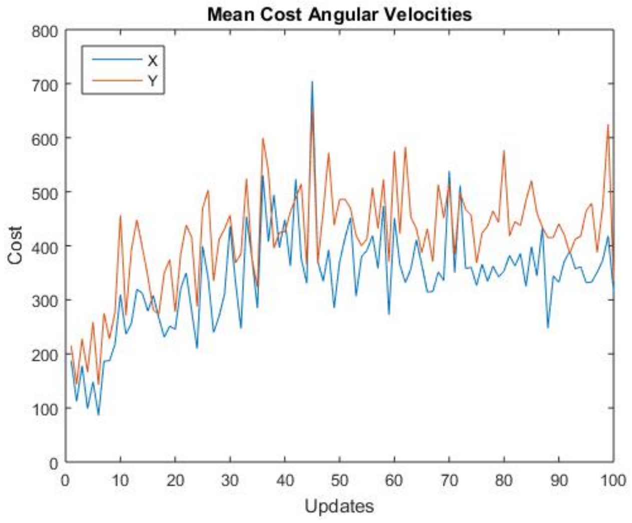


Figure 5.18: ***Test IV:*** Cost of the angular velocities.

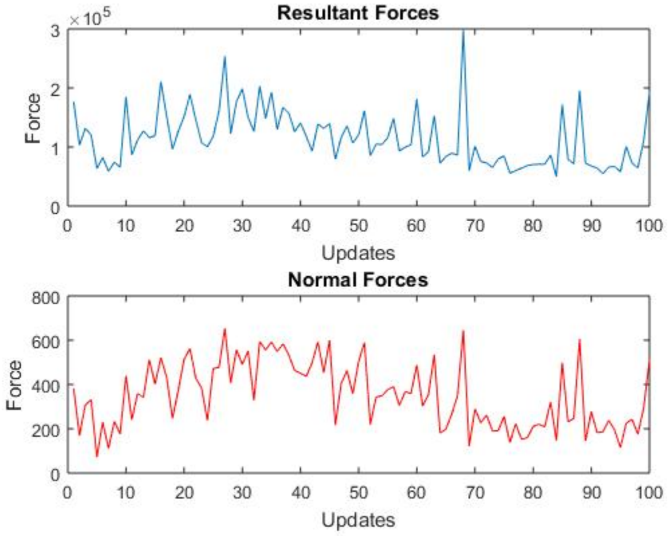


Figure 5.19: ***Test IV:*** Total forces per update.

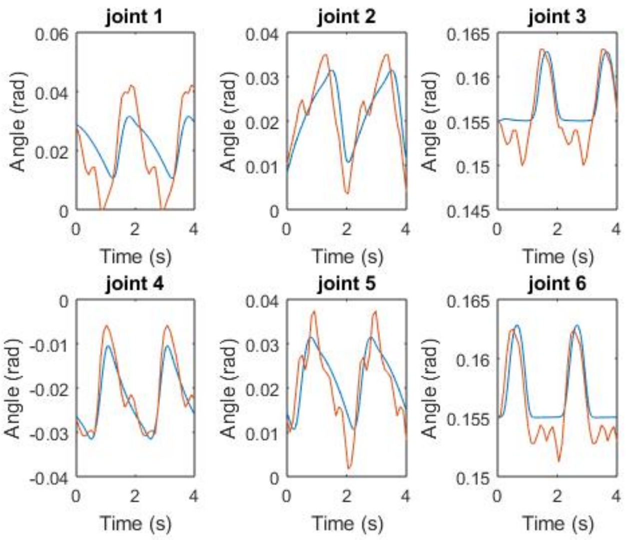


Figure 5.20: ***Test IV:*** Joint reference trajectories. Blue initial trajectory, red final trajectory.

The cost and total rotation are plotted in Figure 5.16. In this case the best cost is at update number 96 out of 100, but that does not mean that the cost is gradually decreasing, and in the last steps it is maintained almost constant. However, the rotated angle is close to 10, which is the highest results as it almost reaches to the reference angle set in Equation (5.2).

The end effector trajectories and the joint trajectories, plotted in Figures 5.17 and 5.20, present a significant differences from the initial conditions, with unnecessary deviation which does not make much sense. This is due to the high variance throughout all the experiment. As anticipated, the resultant forces are far from being reduced.

Test V: Focus on Velocities

In this test the cost is focused on the angular velocities, minimizing ω_x and ω_y and maximizing ω_z, due to that the total rotated angle of the ball is expected to be higher than in previous cases. Table 5.5 displays the used parameters.

Table 5.5: Parameters used in Test V.

Parameter	$a_1 \| a_2$	a_3	a_4	a_5	$\Gamma_r \| \Gamma_n$	Σ_ϵ
Value	100	0.01	1	20000	10	15

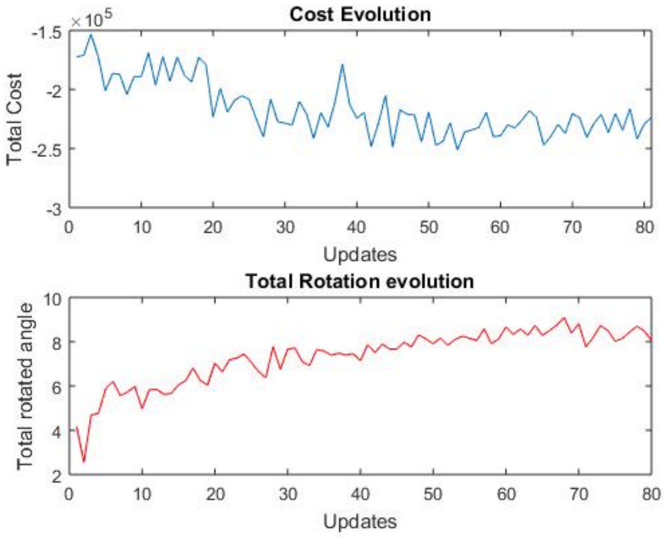


Figure 5.21: ***Test V:*** Total cost evolution (upper). Evolution of the total rotated angle (lower)

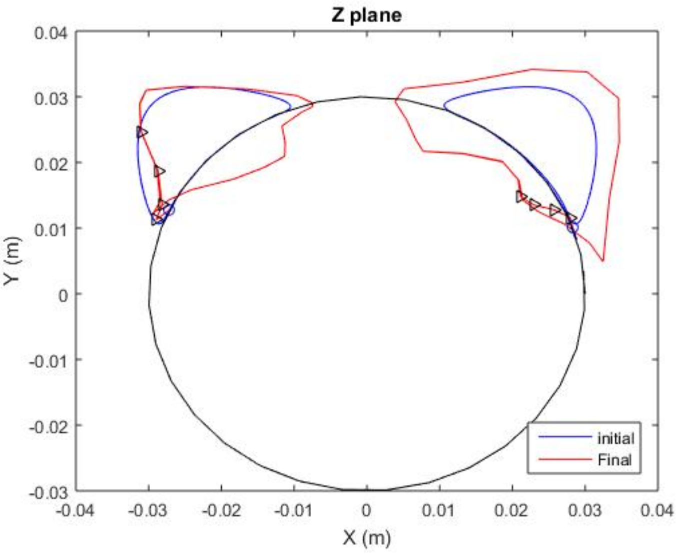


Figure 5.22: ***Test V:*** Fingertip reference trajectories.

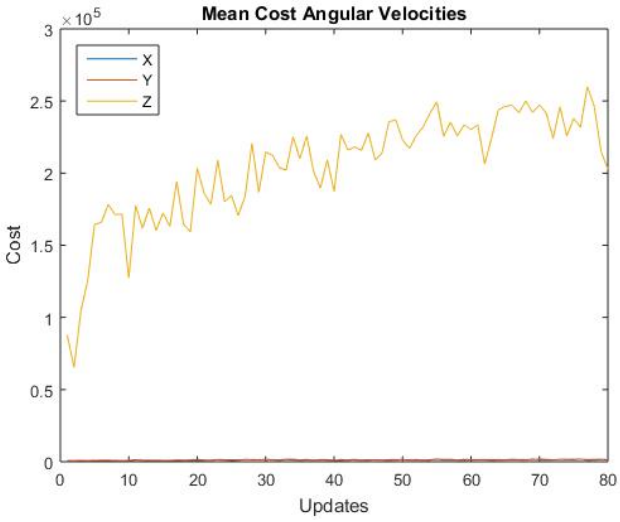


Figure 5.23: ***Test V:*** Cost of the angular velocities. The ratios (final/initial) for x, y, and z are 1.43, 1.33 and 2.3 respectively.

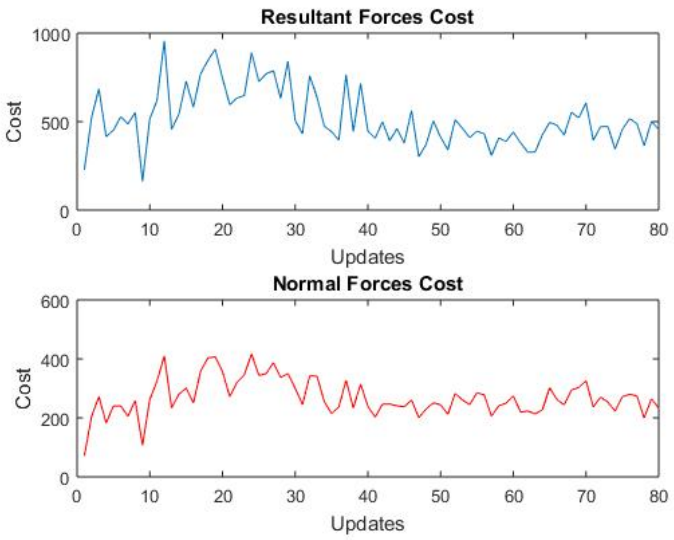


Figure 5.24: ***Test V:*** Total forces per update.

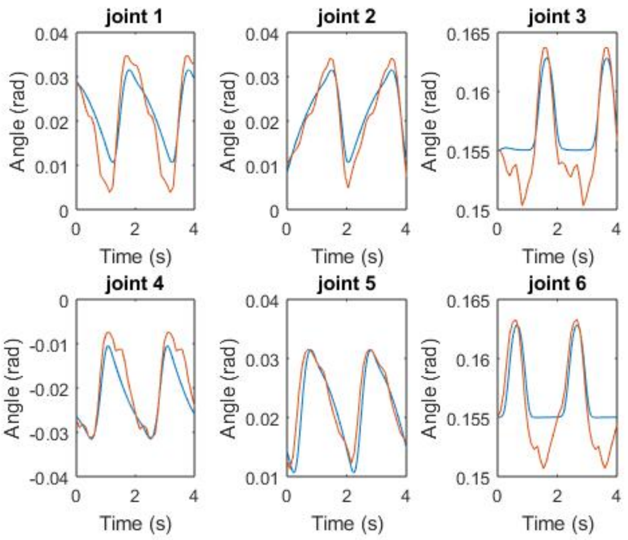


Figure 5.25: ***Test V:*** Joint reference trajectories. Blue initial trajectory, red final trajectory.

In this case, the cost is reduced only in the first steps, and thereafter it presents little variations but does not decrease. In fact, the best cost is achieved in the update number 54 out of 80. Thus the exploitation does not perform well. The maximum terminal cost achieved is 9 radians which is higher than in the balance cases.

As expected, the forces showed in Figure 5.24, do not decrease as their contribution to the total cost is practically negligible. Consequently, the trajectories of Figure 5.22 pierce the ball.

5.2 Real Robot

This section addresses the results obtained from the real robot. Experimentation on the real robot was limited by certain aspects such as time and hardware. Unlike simulation, this experiment requires the supervision of a user and the time required by each trajectory is higher. Continuous usage of the motors may cause overheating of them, thus certain short breaks between roll-outs were sometimes needed to avoid damage on the motors. These limitations affected the collection of data, so for the tests of this section, the number of updates and roll-outs were much lower than in simulation.

The duration of each experiment is 4τ as in the simulation. This duration corresponds to Matlab, where the reference trajectories are computed. Obviously, the robot takes much more time performing these trajectories. An average time of a roll-out performance is close to 35 seconds.

The cost function for the real robot presents some modifications with respect the one used in simulation, described by Equation (5.1). It is defined as follows:

$$\begin{aligned} r_t =& a_1\left[(c_x - c_{0x})^2 + (c_y - c_{0y})^2\right] + a_2\widetilde{r_b} + a_3\left(\omega_x^2 + \omega_y^2\right) \\ &+ a_4\sum_{i=1}^{4}\widetilde{f_{n_i}} - a_5\omega_z + a_6 d \end{aligned} \tag{5.3}$$

where

$$\widetilde{r_b} = \begin{cases} 1 & r_b > \Gamma_r \\ 0 & \text{Otherwise} \end{cases}, \widetilde{f_{n_i}} = \begin{cases} f_{n_i} & f_{n_i} > \Gamma_n \\ 0 & \text{Otherwise} \end{cases}, d = \begin{cases} 0 & \text{Data received} \\ 1 & \text{Otherwise} \end{cases} \tag{5.4}$$

denoting r_b as the radius of the ball, f_i as the module of the normal force applied to the ball by finger i, c_x and c_y as the positions of the centre of the ball and c_{0x} and c_{0y} the positions in the initial conditions.

The first term penalizes ball translations, the second term refers to height of the ball. As previously explained there is a direct proportional relation between the captured radius of the camera and the ball's height, and this term is a binary term which punishes the ball going down a threshold (i.e., dropping out). Third term discourages rotation around the x and y axes.

The forth term refers to the normal forces of each finger, where a threshold is applied due to the need of a minimun force in order to perform a stable grasping. The fifth term rewards counterclockwise rotation around the z axis. This term is in charge of maximizing the angular velocity. For the real robot, we can not compute the total rotated angle as in simulation, but it can be estimated by integrating the angular velocity. However, as maximizing ω_z will have a similar effect, this calculation is avoided.

For robustness purposes, the last term is added, which heavily penalizes the lack of data at instant t, being a_6 always a high value, for instance 100000. This is for the case of the ball falling and rolling out of the workspace.

Test I Real

For this test the parameters are chosen to have a balance importance, emphasising the rotation of the ball, see Table 5.6. Noise variance is much higher than in simulation because of the reference units. In simulation the input reference was in radians, whereas in real experiment it is in decimals of degrees. The number of total updates is set to 16, evaluating 5 roll-outs and re-using the best one of the previous updates.

Table 5.6: Parameters for Test I of the real robot.

Parameter	$a_1\|a_2$	a_3	a_4	a_5	$\Gamma_r\|\Gamma_n$	Σ_ϵ
Value	2	100	5	1000	80	5000

The results are presented by means of the following figures:

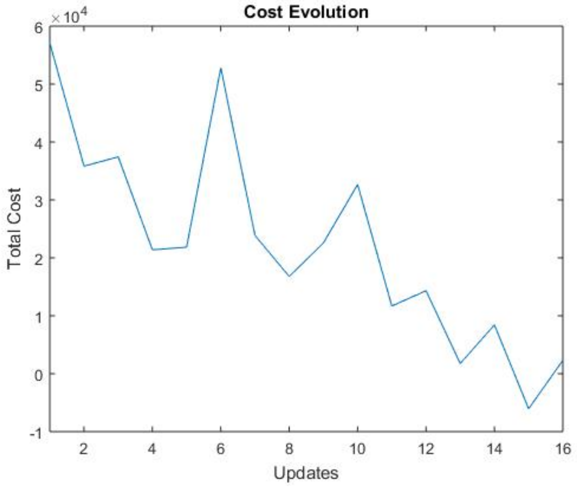


Figure 5.26: ***Test I Real:*** Cost evolution.

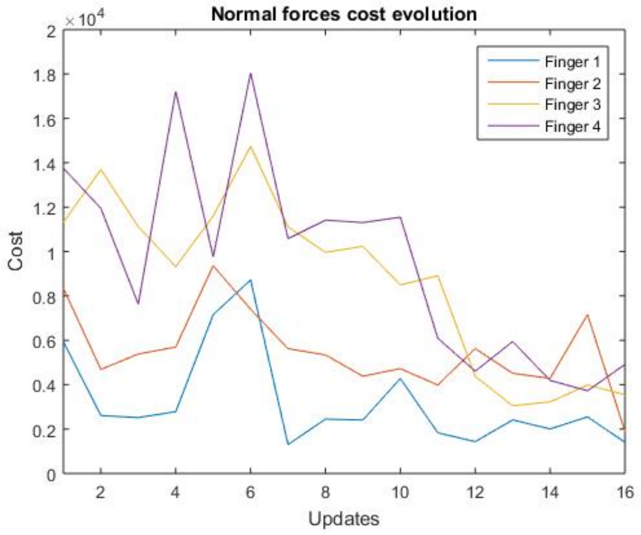


Figure 5.27: ***Test I Real:*** Evolution of the normal forces cost applied by each finger during learning.

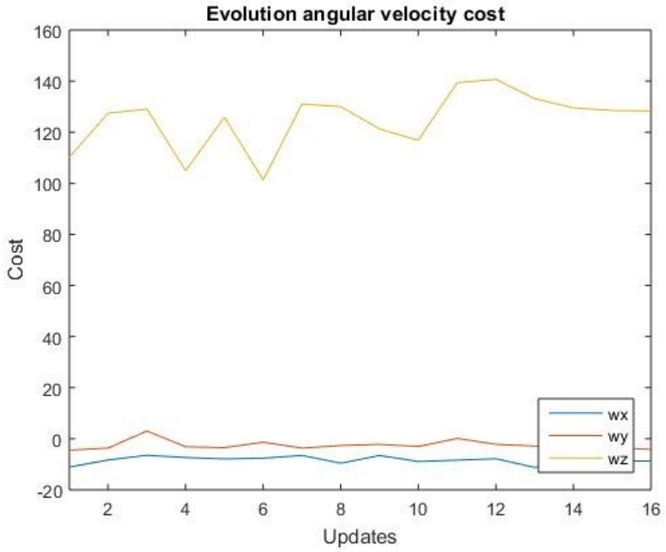


Figure 5.28: ***Test I Real:*** Evolution of the angular velocities cost.

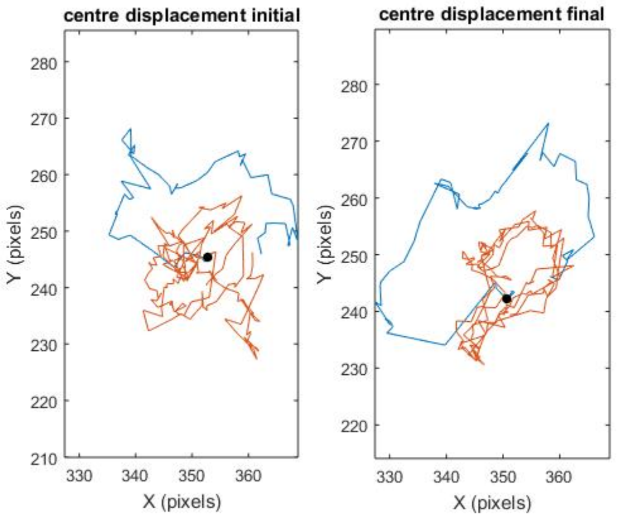


Figure 5.29: ***Test I Real:*** Ball displacement with the initial trajectory (left) and with the final trajectory (right). Blue represents the transient behavior and red represents the periodic behavior.

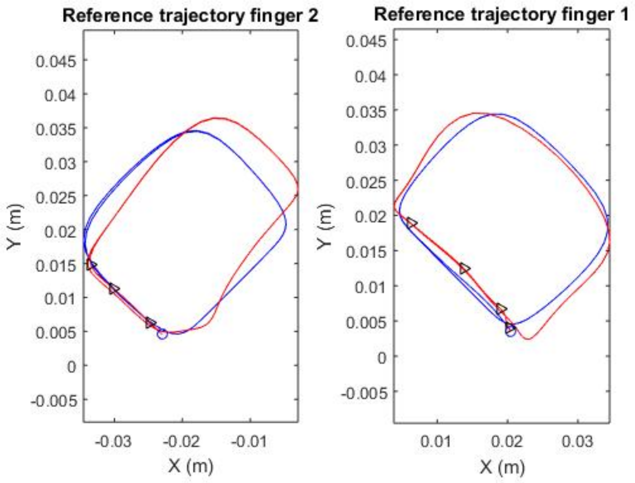


Figure 5.30: ***Test I Real:*** Trajectories of the end effector of the robot in the Z plane. (Blue): initial trajectory, (red): learnt trajectory. Black arrows indicate the flow of the movement.

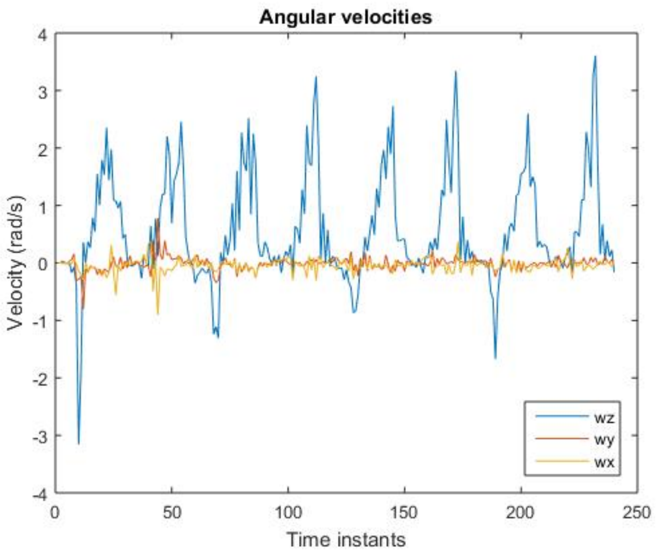


Figure 5.31: ***Test I Real:*** Angular velocities of the ball, in the final trajectory.

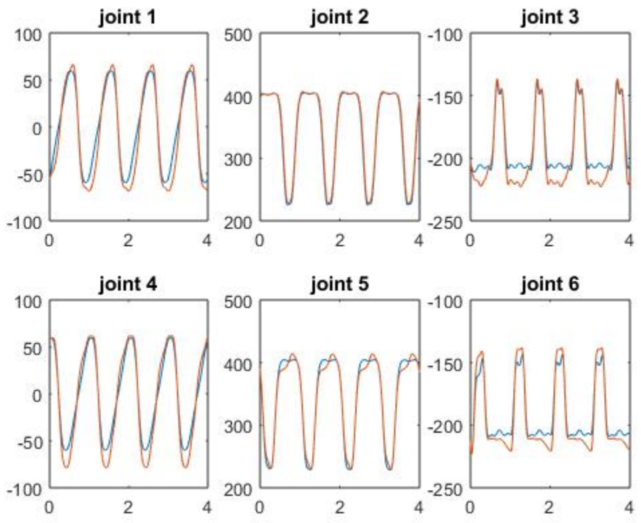


Figure 5.32: ***Test I Real:*** Joint reference trajectories: (Blue) initial trajectory, (red) final trajectory. Vertical axes represents decimals of degree and horizontal axes time in seconds.

Observing Figure 5.26, it can be seen that the cost is reduced, which could happen by several reasons. Figure 5.27 also displays a decrease in the cost of the forces, which is related with the total cost reduction. However the cost of the angular velocities hardly changes as Figure 5.28 shows. The displacement of the centre showed in Figure 5.29 seems to have a stable cyclic behavior as in the final case the centre trajectories seem to rotate around a point. Therefore the main cost reduction is due to the reduction of the force term.

Figure 5.31 displays the angular velocities received by the camera. It can be seen that the robot has 8 revolutions, which correspond to each pair of fingers in each cycle. It is also remarkable that before the first spin takes place, the angular velocities experimented a sudden change, due to the transient behaviour of the system.

Test II Real

In this test, the parameters are similar to the previous one, and just the weight of the angular velocities is changed. The number of total updates is set to 20, evaluating 6 roll-outs and re-using the best one of the previous updates.

Table 5.7: Parameters for Test II of the real robot.

Parameter	$a_1 \vert a_2$	a_3	a_4	a_5	$\Gamma_r \vert \Gamma_n$	Σ_ϵ
Value	2	1000	5	1000	80	5000

Table 5.7 presents the used parameters. In this case, all the angular velocities have the same weights. The values of ω_x and ω_y are usually lower than one, but as they are squared in third term of the cost function, see Equation (5.3), they have less impact than ω_z. The results are presented by means of the following figures:

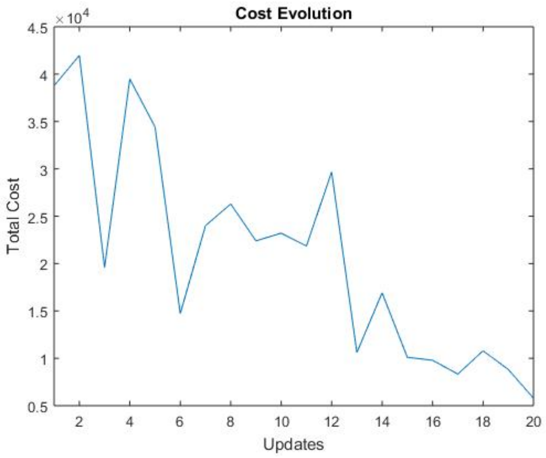


Figure 5.33: ***Test II Real:*** Cost evolution.

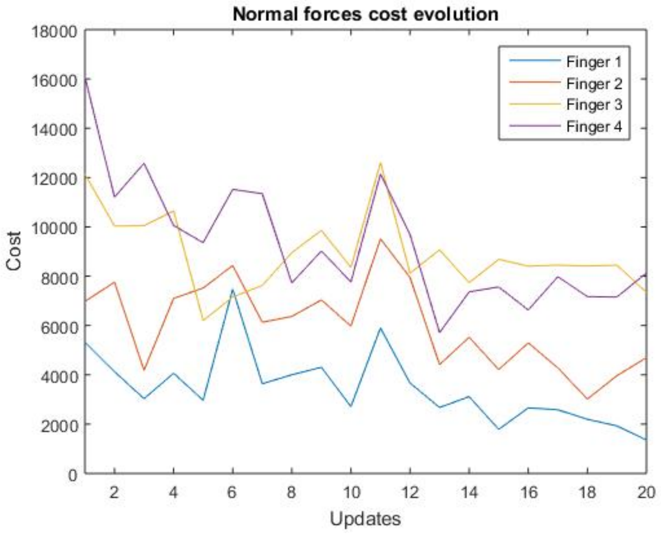


Figure 5.34: ***Test II Real:*** Evolution of the normal forces cost applied by each finger during learning.

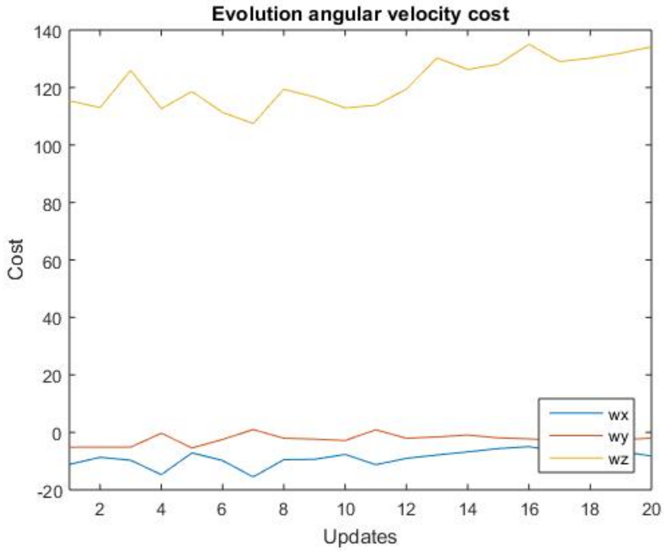


Figure 5.35: ***Test II Real:*** Evolution of the angular velocities cost.

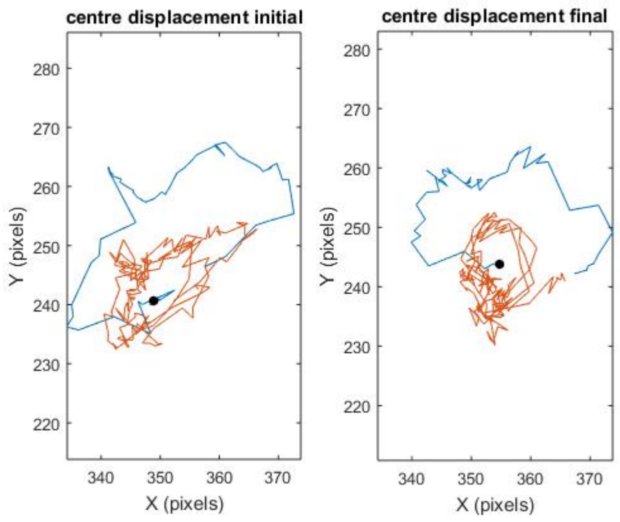


Figure 5.36: ***Test II Real:*** Ball displacement with the initial trajectory (left) and with the final trajectory (right). Blue represents the transient behavior and red represents the periodic behavior.

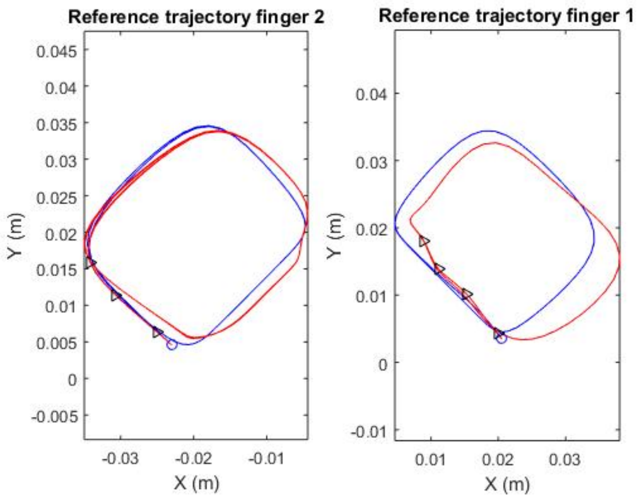


Figure 5.37: ***Test II Real:*** Trajectories of the end effector of the robot in the Z plane. (blue): initial trajectory, (red): learnt trajectory. Black arrows indicate the flow of the movement.

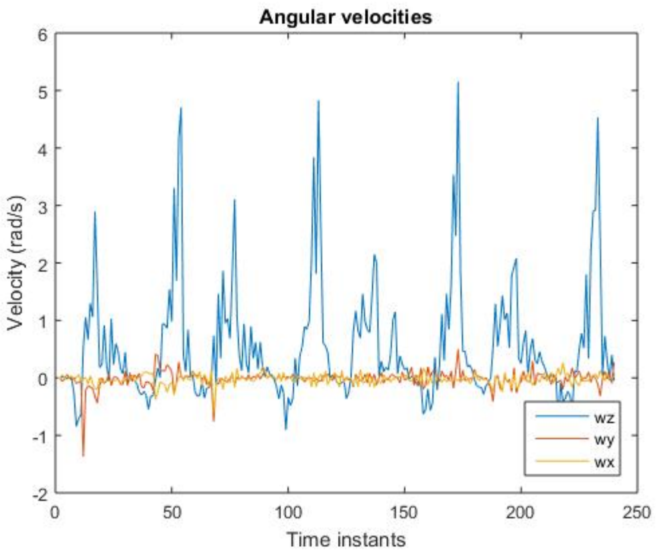


Figure 5.38: ***Test II Real:*** Angular velocities of the ball in the final trajectory.

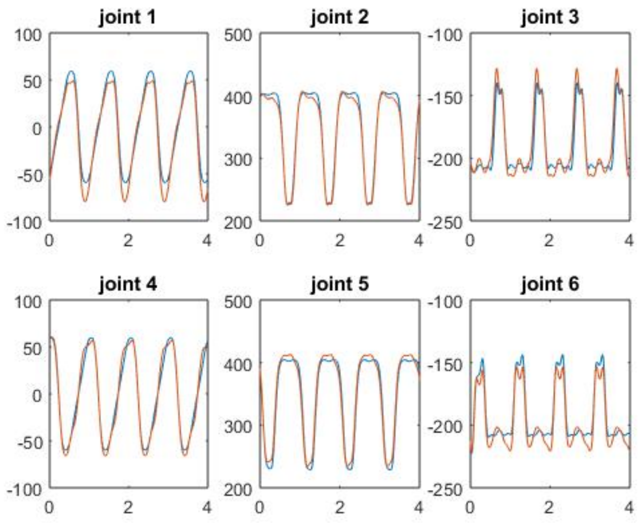


Figure 5.39: ***Test II Real:*** Joint reference trajectories: (Blue) initial trajectory, (red) final trajectory. Vertical axes represents decimals of degrees and horizontal axes time in seconds.

The results obtained in this test are analogous with those of the previous experiments. Thus, the change on the weight of undesirable angular velocities, does not play a determinant role on the experiment.

Test III Real

As previous tests show little increase in rotational velocities, but force reduction was accomplish. The purpose of this test is to maintain the ball in the same place, namely to increse the cost of ball displacement. The number of total updates is set to 20, evaluating 6 roll-outs and re-using the best one of the previous updates. The selected parameters are presented in Table 5.8:

Table 5.8: Parameters for Test III of the real robot.

Parameter	$a_1 \| a_2$	a_3	a_4	a_5	$\Gamma_r \| \Gamma_n$	Σ_ϵ
Value	10	100	1	100	80	5000

The results are presented by means of the following figures:

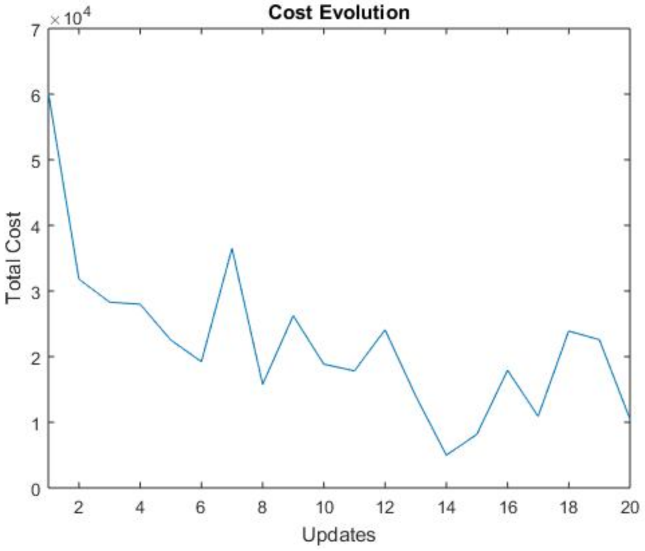


Figure 5.40: ***Test III Real:*** Cost evolution.

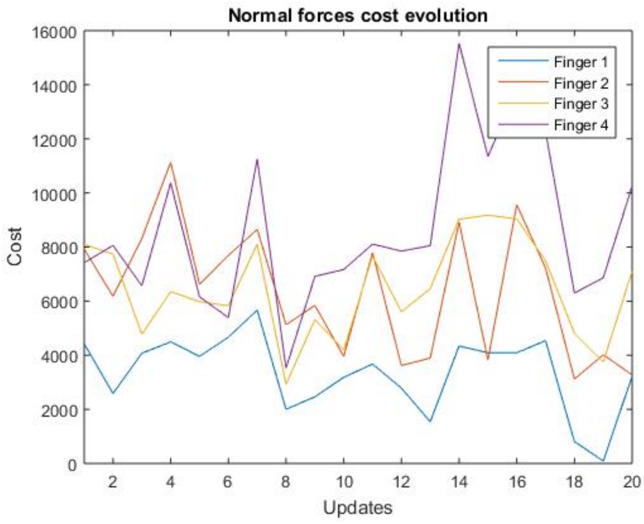


Figure 5.41: ***Test III Real:*** Evolution of the normal forces cost applied by each finger during learning.

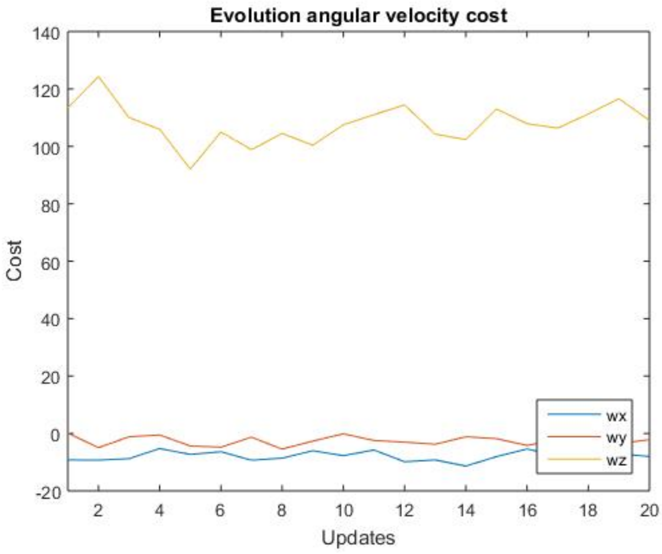


Figure 5.42: ***Test III Real:*** Evolution of the angular velocities cost.

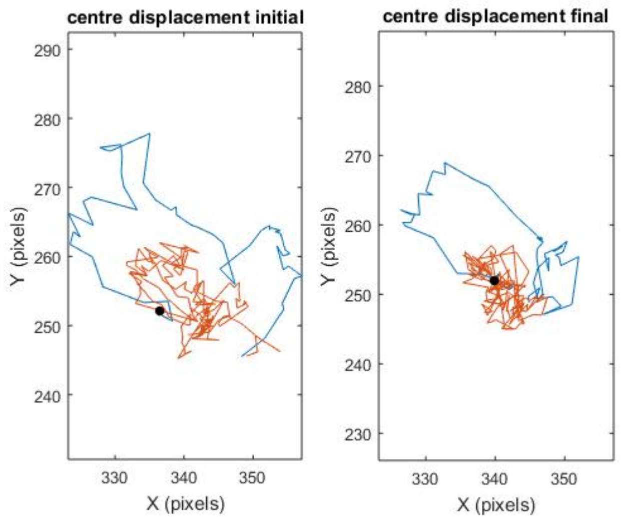


Figure 5.43: ***Test III Real:*** Ball displacement with the initial trajectory (left) and with the final trajectory (right). Blue represents the transient behavior and red represents the periodic behavior.

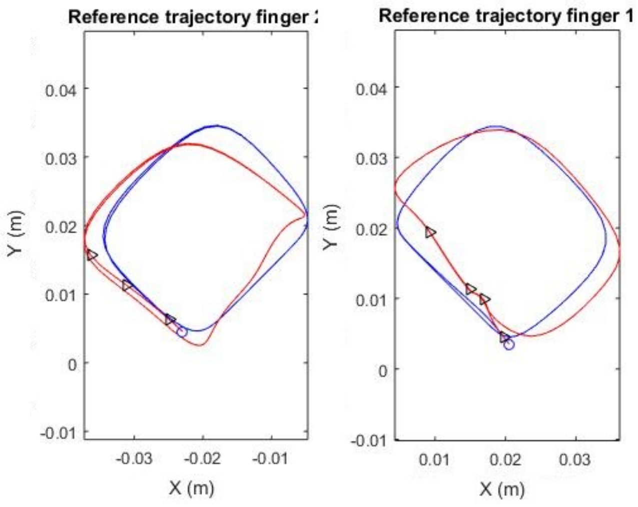


Figure 5.44: ***Test III Real:*** Trajectories of the end effector of the robot in the Z plane.According to previous comments (blue): initial trajectory, (red): learnt trajectory. Black arrows indicate the flow of the movement.

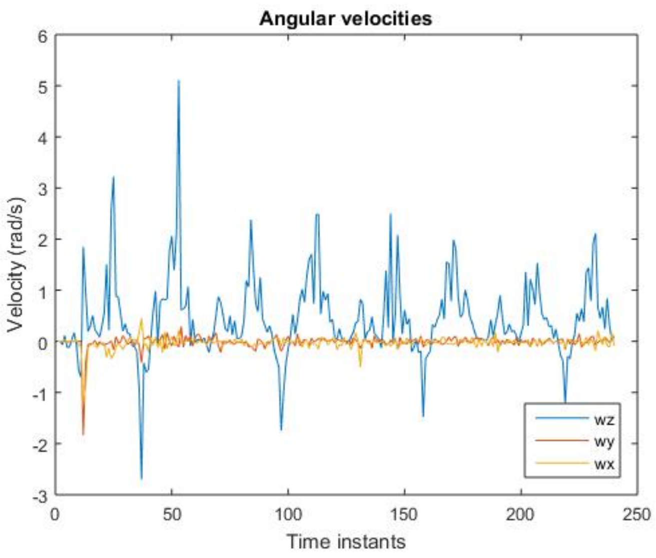


Figure 5.45: ***Test III Real:*** Angular velocities of the ball, with the final trajectory.

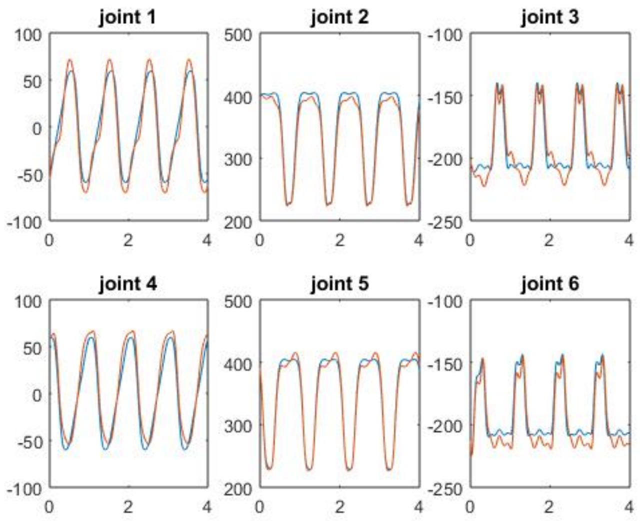


Figure 5.46: ***Test III Real:*** Joint reference trajectories: (Blue) initial trajectory, (red) final trajectory. Vertical axes represents decimals of degrees and horizontal axes time in seconds.

The cost of the centre is reduced as can be seen in Figure 5.43 in comparison with the initial trajectories. The same observation can be made in comparison with the other experiments where the importance of centre displacement was low. Figure 5.41 presents the evolution of the forces, and it can be seen that for 3 out of 4 fingers the cost is reduced. Still the total forces cost presents a small decrement, whereas angular velocities cost appears to remain constant, Figure 5.42. The test thus confirms the expected influences of the design parameters.

6

Discussion

6.1 Simulation results

In all cases, PI^2 successfully reduces the cost of the path, by either increasing the rotation of the ball or reducing other cost terms. Tests I, III and IV include the terminal cost term, and the evolution of the cost functions are smoother and present less fluctuations than Tests II and V, which present the term inside the cost function. This can be seen comparing Figures 5.4, 5.11, 5.16 with 5.6 and 5.21. However, this difference may have its origin on the reliability of measurements, as the total rotated angle is more robust than the angular velocities in the simulation case. Generally, the cost of the angular velocities is not reduced, which can be due to the correctness of the measures or to some dependence with ω_z.

These tests show the importance of the reward design, as the main hypothesis of Tests III, IV and V were validated. Thus the designer can tune the parameters of the cost function to achieve a desired behaviour. The resulting trajectories are different for all experiments, which is due to the high dimensionality of the problem, as PI^2 explores the space differently arriving to many suboptimal solutions.

The first updates of the algorithm may present an increase of the cost, due to the lack of information at the beginning of the experiment. However this effect is stimulated by the choice of a high exploration parameter, as it happens in Test III. In all experiments, the cost function and the rotation angle display a decrease of variation after half of the total updates are done. This is not only due to the decrease of exploration noise at late updates, but for the convergence to a suboptimal solution. It must be said that all results present some fluctuations after convergence, accentuated for Tests II and V. Hence it can be concluded that the continued updating causes parameter fluctuation around the suboptimal ones.

From Tests III and IV, some insight about the behaviour of the system can be extracted. In order to spin the ball, the finger used the friction force which is proportional to the normal force. Thus, the learning deals with a trade-off between increasing speed (high normal forces) and motor safety (low normal forces). PI^2 has proven to be able to optimize this trade-off, but the correct choice of the parameters depends on the user. These tests prove this statement, as in Test III, the weight of the forces is the highest one, and it presents the lowest rotation, and contrary in Test IV, the relevance of the forces were the lowest, and it presents the highest rotation, increasing nearly 20% in comparison with other tests and being able to arrive to the reference angle.

Intuitively, one can think that in order to maximize ball rotation, the part of the trajectory in contact with the ball should be enlarged. This thought is correct for Figures 5.2, 5.7 and 5.17. In Figure 5.12, the right trajectory shortened from one side but became larger from the other while the left trajectory enlarges on both sides. Thus the extension of the contact surface may enhance ball rotation. Notwithstanding this extension may be constrained by the robot itself, since the fingers may collide with each other.

Another important aspect is the direction of the trajectory when contact occurs. For the initial trajectory the change of sign of the tangential velocity with respect to the ball is produced near the contact surface with the ball. As the trajectories represent a point of the end effector, but not necessarily the contact point with the ball, changing this sign close to the ball may happen after contact, consequently there exists an instant where two fingers are stopping or counter spinning the ball. However, the resulted trajectory performs this change in a further place from the ball, avoiding this undesirable behaviour as showed in Figures 6.1a and 6.1b.

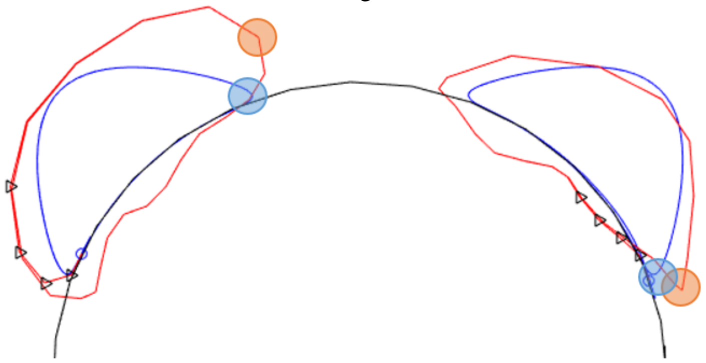
Figures 5.5, 5.10, 5.15, 5.20 and 5.25 display initial and final joint trajectories. It can be seen that final trajectories keep similarities with the initial ones, as they evolved from them. For Test IV, the large exploration parameters induce bigger differences among trajectories. In the case of joints 3 and 6, it can be seen that for all cases, the reference trajectory during contact does not differ, but it clearly does when no contact happens. This may be due to the fact that the cost function at these instants is dependent of the contact finger, and therefore the non-contact finger does not play an important role for this time interval. These images can help to understand the importance of initial conditions for the PI^2 algorithm.

6.2 Real Robot

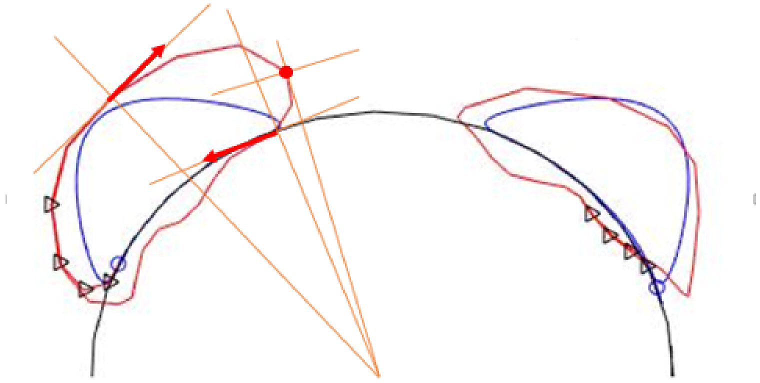
The real system presents certain stochasticity, as for the same control commands the path cost might slightly vary. Anyway, it does not present big problems as PI^2 is able to cope with stochastic systems.

As shown in Figures 5.26, 5.33 and 5.40, the cost is also decreased in the same way as in the simulation case. However the increase in the angular velocities increase is slight in comparation with simulation results, Figures 5.28, 5.35 and 5.42. This is because the initial conditions lead to an angular velocity close to the limit, as the movement of the fingers is already expanded almost until there is a collision between themselves.

Thus, the cost reduction is done by minimizing forces, undesirable angular velocities and displacements. In the initial conditions the fingers strongly grasp the subfigure



(6.1a) The circles represent the uncertainty region where the contact point is. The centres of the coloured circles correspond to the change of tangential sign. Blue: initial trajectory, red: final trajectory. This image corresponds to test 2. It can be seen that in the initial case the uncertainty region is in contact with the ball, which is not the case for the final trajectory.



(6.1b) The red arrows represents the velocity of the end effector in the tangential direction of the ball. The red dot means zero tangential velocity and it occurs when velocity sign at that direction changes.

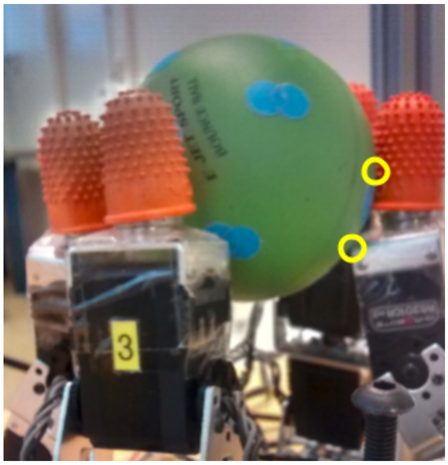


Figure 6.2: Contact with the ball, in the periodic phase of the movement. Notice that the finger makes contact with the ball at two points; at the end effector and at the body of the servo, which makes the grip more stable.

ball, which was done to ensure stability. During training, the ball happens to fall in a few roll-outs after several updates, which is because the finger forces were below the threshold limit not adding cost, but the force was not enough to hold the ball.

In the case of the real robot the transient behaviour was significant. Figures 5.29, 5.36 and 5.43 show the centre displacement differentiating between transient and periodic behaviour, being the higher displacement at the transient part. In Figures 5.31, 5.38 and 5.45, the angular velocities on the noiseless roll-outs can be seen. All of them present similar shape, and it can be observed that in the first period there exist some peaks in ω_x and ω_y caused by the transient part, in which the ball rotates and translates when it is first held by only two fingers until a position in which this holding is more stable. That happens because the finger used two contact points as shown in Figure 6.2.

The final trajectories, illustrated in Figures 5.30, 5.37 and 5.44, show how the robot has adapted to the ball, penetrating less and shifting to a new position where the grasp quality is better. The ball cannot be displayed as in the simulations because its position changes during execution.

7 Conclusions

This thesis has shown the usefulness of the PI^2 algorithm for a periodic case, as in all experiments the cost function has been decreased, and nearly all results fulfil the expectations. Even though experiments require a lot of time, the algorithm itself showed fast convergence.

PI^2 explores the state-space and updates the parameters minimizing the cost, thus arriving to trajectories with lower the cost. However, the convergence and the quality of the final result are highly dependent on the initial conditions. Providing good initial conditions will not only improve the results but also speed up convergence. Moreover there are some tricks to improve the convergence, such as reusing the elite roll-outs from previous updates or changing the exploration parameter along updates.

The design of the cost function is very flexible, as long as its terms are functions of the state of the system or of the control commands.

As conclusion, PI^2 presents good results and an easy design.

7.1 Future Improvements

This master thesis could have been extended in several aspects if not for the time constrains. A number of suggestions for improvements are listed here:

- The PI^2 algorithm can be modified in order to automatically adapt the exploration parameter, by using Covariance Matrix adaptation as described in [17].
- The communication between Matlab and the robot can be modified to facilitate user's supervision. For instance, a ROS node could be used which could interconnect both platforms and send starting signals between them. However, a supervisor would still have to position the ball at the initial point.
- Yet another alternative could be to implement the PI^2 algorithm in the embedded system using auto generated C code from Matlab. All the code could be embedded in a C++ program and run in ROS.

- Camera calibration can be been improved by compensating ofr lens imperfections. There are certain algorithms within the OpenCV library that can perform this task. Although this usually require predefined images like a chessboard to do it.

- A control term can be added to the DMP transformation equation. This control term could feedback online information about the robot, such as forces, enabling the robot to work in close-loop mode. The function would be to set a reference or new behavior of the system, for example to try to maintain the ball at a desired height or to react to external disturbances.

Bibliography

[1] E.A. Theodorou, J. Buchli & S. Schaal. *A generalized path integral control approach to reinforcement learning*. Journal of Machine Learning Research, vol. 11, pp. 3137-3181. 2010.

[2] M.M. Ghazaei. *On trajectory generation for robots*. Thesis No. TFRT-1116-SE. PhD thesis, Lund University, Department of Automatic control, pp. 132-167. 2016.

[3] S.Schaal. *Dynamic movement primitives- A framework for motor control in humans and humanoid robotics*. Adaptive Motion of Animal and machines. Vancouver, Canada. pp. 261-280. 2006.

[4] A.J. Ijspeert, J. Nakanishi & S. Schaal. *Learning attractor landscapes for learning motor primitives*. Advances in neural information processing systems, vol. 15, pp. 1523-1530. January 2002.

[5] A.J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor & S. Schaal. *Dynamical movement primitives: Learning attractor models for motor behavior*. Neural Computation, vol. 25, pp. 328-373. February 2013.

[6] J. Buchli, F. Stulp, E. Theodorou & S. Schaal. *Learning variable impedance control*. International Journal of Robotics Research, vol. 30, pp. 820-833. April 2011.

[7] E. Rombokas, M. Malhotra, E. Theodorou, E. Todorov & Y. Matsuoka. *Reinforcement learning and synergistic control of the ACT hand*. IEEE Transactions on Mechatronics, vol. 18, pp. 569-577. April 2013.

[8] E. Theodorou, J. Buchli & S. Schaal. *Reinforcement learning of motor skills in high dimensions: A path integral approach*. IEEE International Conference on Robotics and Automation. Anchorage, AK, USA. pp. 2397-2403. May 2010.

[9] A. Gams, T. Petric, B. Nemec & A. Ude. *Learning and adaptation of periodic motion primitives based on force feedback and human coaching interaction*. IEEE-RAS International Conference on Humanoids Robots. Madrid, Spain. pp. 166-171. November 2014.

[10] A. Ude, A. Gams, T. Asfour & J. Morimoto . *Task-specific generalization of discrete and periodic dynamic movement primitives*. IEEE Transactions on Robotics,vol. 26, pp. 800-815. September 2010.

[11] J. Kober & J. Peters. *Imitation and reinforcement learning*. IEEE Robotics and Automation Magazine, vol.17 , pp. 55-62. June 2010.

[12] J. Ernesti, L. Righetti, M. Do, T. Asfour & S. Schaal. *Encoding of periodic and their transient motions by a single dynamic movement primitive*. IEEE-RAS International Conference on Humanoids Robots. Osaka, Japan. pp. 57-64. November 2012.

[13] A. Gams, A. Ijspeert, S. Schaal & J. Lenarcic. *Online learning and modulation of periodic movements with nonlinear dynamical systems*. Autonomous Robots, vol. 27, pp. 3-23. July 2009.

[14] A. Gams, J. Kieboom, M. Vespignani, L. Guyot, A. Ude & A. Ijspeert. *Rich periodic motor skills on humanoids robots: Riding the pedal racer*. IEEE International Conference on Robotics and Automation. Hong Kong, China. pp. 2326-2332. May 2014.

[15] A. Gams, A. Ude & J. Morimoto. *Accelerating synchronization of movement primitives*. IEEE International Conference on Intelligent Robotics and Systems. Hamburg, Germany. pp. 2754-2760. September 2015.

[16] F. Stulp. *Adaptive Exploration for Continual Reinforcement Learning*. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Vilamoura, Portugal. pp. 1631-1636. October 2012.

[17] F. Stulp & O. Sigaud. *Path Integral Policy Improvement with Covariance Matrix Adaptation*. Proceedings of the 10th European Workshop on Reinforcement Learning. Edimburg, United Kingdom. pp. 1-14. June 2012.

[18] J. Fu & S. Chen. *Various Robot Motor Skills Learning with PI^2-GMR*. International Conference on Industrial Informatics-Computing Technology, Intelligent Technology, Industrial Information Integration. Wuhan, China. pp. 246-250. December 2016.

[19] J. Lundell, *Dynamic movement primitives and reinforcement learning for adapting a learned skill*. Master's Thesis. Luleå University of Technology. Department of Computer Science, Electrical and Space Engineering. August 2016.

[20] E. Theodorou, F. Stulp, J. Buchli & S.Schaal. *An Iterative Path Integral Stochastics Optimal Control Approach for Learning Robotic Tasks*. IFAC Proceedings, vol. 44, pp. 11594-11601. January 2011.

[21] F. Ficuciello & B. Siciliano. *Synergy-based Policy Improvement with Path Integral for Anthropomorphic Hands*. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Daejeon, South Korea. pp. 1940-1945. October 2016.

[22] D. Pongas, A. Billard & S. Schaal. *Rapid Synchronization and Accurate Phase-locking of Rhythmic Motor Primitives*. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Edmonton, Canada. pp. 2911-2916. August 2005.

[23] O. Kroemer. *Machine Learning for Robot Grasping and Manipulation*. Thesis No. TU-4584 PhD thesis, Techinische Unversität Darmstadt Department of Computer Science Intelligent Autonomous Systems. 2015. pp. 53-65. Available at http://www.ausy.tu-darmstadt.de/uploads/Member/OliverKroemer/KroemerThesis.pdf

[24] R. Platt. *Learning Grasp Strategies Composed of Contact Relative Motions*.7th IEEE-RAS International Conference on Humanoids Robots. Pittsburgh, USA. pp. 49-56. November 2007.

[25] T. Yoshikawa. *Multifingered robot hands: Control for grasping and manipulation*. Annual Reviews in Control, vol. 34(2), pp. 199-208. December 2010.

[26] R.F. Stengel. *Optimal Control and Estimation*.Dover books on advanced mathematics. Dover Publications.1994.

[27] A. El-Fakdi. *Gradient-Based reinforcement learning techniques for underwater robotics behavior learning*. Thesis No. GI-374-2011. PhD thesis , University of Girona, Computer engineering Department.December 2010. pp. 11-53.

[28] Watkins, C.J & P. Dayan. *Q-learning*. Machine learning, vol. 8, pp. 279-292. 1992.

[29] J. Kober, J.A. Bagnell,J. Peters. *Reinforcement learning in Robotics: A survey*. The International Journal of Robotics Research, vol. 32, pp. 1238-1274. September 2013.

[30] T. Flash, B. Hochner. *Motor primitives in vertebrates and invertebrates*. Current Opinion in Neurobiology, vol.15, pp. 660-666. January 2006.

[31] F.A. Mussa-Ivaldi, E. Bizzi. *Motor learning through the combination of primitives*.Philosophical Transactions of The Royal Society B Biological Sciences, pp. 1755-1769. 2001.

[32] P. Dario, G. Sandini, P. Aebischer. *Robots and Biological Systems: Towards a New Bionics?*.Chapter 1, Active Perception and Exploratory Robotics, pp. 11-20. 1993.

[33] S. Ivaldi, V. Padois, F.Nori. *Tools for dynamics simulation of robots:a survey based on user feedback*. IEEE-RAS International Conference on Humanoids Robots. Madrid, Spain. pp. 842-849. November 2014.

[34] T. Yamada. http://www.eng.gifu-u.ac.jp/chinoukikai/e/staff/yamada-takayoshi.html. 2014.

[35] P.I. Corke. *Robotics Toolbox for MATLAB*. 2001. URL: http://petercorke.com/wordpress/toolboxes/robotics-toolbox. Accessed: 01-04-2017.

[36] Coppelia Robotics. *V-REP: virtual robot experimentation platform*. 2017. URL: http://www.coppeliarobotics.com/. Accessed: 10-02-2017.

[37] Open Dynamic Engiles. *Manual: Concepts*. 2012. URL: https://www.ode-wiki.org/wiki/index.php?title=Manual:_Concepts. Accessed: 15-04-2017.

[38] Bill Goodwine, *Robotics and Automation Handbook*. Chapter 3, Inverse Kinematics, Thomas R. Kurfess, Editor, 2005.

[39] T.DeWolf.*Dynamic Movement Primitives*. URL:https://studywolf.wordpress.com. Accessed: 15-02-2017.

[40] S. Schaal, C.G. Atkeson, S. Vijayakumar. *Scalable Techniques from Nonparametric Statistics for Real time Robot Learning*. Applied intelligence vol. 17, pp. 47-60. July 2002.

[41] Mathworks. *Matlab*. 2017. URL: https://es.mathworks.com/products/matlab.html?s_tid=hp_products_matlab. Accessed: 08-02-2017.

[42] S. Schaal. *Path Integral Reinforcement (PI2) Learning Software*. 2011. URL: http://www-clmc.usc.edu/software/git/gitweb.cgi?p=matlab/pi2.git;a=summary. Accessed: 15-02-2017.

[43] S. Schaal. *Motor Primitives Based on Dynamic Systems Theory*. 2010. URL: http://www-clmc.usc.edu/software/git/gitweb.cgi?p=matlab/dmp.git;a=summary. Accessed: 15-02-2017.

[44] Open CV. *OpenCV: Tutorials*. 2012. URL: http://docs.opencv.org/2.4/doc/tutorials/tutorials.html. Accessed: 27-04-2017.

[45] Hamano. F. *Derivative of Rotation Matrix, Direct Matrix Derivation of Well-Know Formula*. Green Energy and Systems Conference. 2013.

[46] Jacob. B, Guennenbaud. G *Eigen, a C++ linear algebra library*. 2016. URL: http://eigen.tuxfamily.org. Accessed: 15-05-2017

[47] *RS301CR/RS302CD Instruction Manual*. Futada. Available at URL: http://manuals.hobbico.com/fut/rs301cr-302cd-manual.pdf. Accessed: 18-05-2017.