Robot Skills: From Imitation to Exploration Learning

Carlos Eduardo Sampaio de Freitas Cardoso

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor(s): Prof. Dr. Alexandre José Malheiro Bernardino

Examination Committee

Chairperson: Prof. Dr. João Fernando Cardoso Silva Sequeira
Supervisor: Prof. Dr. Alexandre José Malheiro Bernardino
Members of the Committee: Prof. Dr. Manuel Fernando Cabido Peres Lopes

May 2016
World domination is such an ugly phrase. I prefer to call it world optimisation.

Eliezer Yudkowsky, Harry Potter and the Methods of Rationality
Acknowledgments

I would like to thank my supervisor Alexandre Bernardino for the tireless support, encouragement and guidance during this project.

Secondly I would like to thank my family for their continuing encouragement.

I want to thank my friends and colleagues at Casa do Povo for the much needed breaks from work, for their friendship and support throughout my academic life.

I direct my gratitude to Ricardo Nunes who built the table tennis paddle, and also to Lorenzo Jamone for collaborating with me in the publication that resulted from this work.

Lastly I would like to say thank you to everyone at VisLab for making me feel welcome, and for being always helpful and a great group of people.
Abstract

Table tennis is a task with a long history as a robotics challenge. In table tennis the existence of complex interactions between the physics of the ball and of the robot makes it hard to create an analytical model that can match the skill of a human player. In addition handcrafted models can be very sensitive to the parametrization of the environment, requiring manual tuning for any alterations. By using a machine learning approach the robot can adapt autonomously, learning the task and model simultaneously.

This dissertation can be divided into two parts. First we discuss regression with Gaussian Processes (GPs) and follow with Cost-regularized Kernel Regression (CrKR), a state of the art Reinforcement Learning (RL) algorithm. We explore strategies for online sparsification of the collected samples to overcome the regression's $O(n^3)$ computational complexity, allowing the robot to learn beyond a fixed budget of samples.

In the second part we address Imitation Learning (IL). Expanding from the state-of-the-art Dynamic Motion Primitive (DMP) framework we developed Quadratic Program Motion Primitive (QPMP), our novel method to imitate trajectories. QPMP takes advantage of the existence of extremely fast solvers for finding the minimum of convex functions and adds increased flexibility to the existing methods with the definition of custom optimisation constraints to the generated trajectory. We present our results in a table tennis simulation and in a robotic platform. By combining demonstrations by an human expert and autonomous learning, we achieve a robot that can easily adapt to its environment and task.

Keywords

Robotic Table Tennis, Reinforcement Learning, Imitation Learning, Dynamic Motion Primitives
Resumo

O ténis de mesa é uma tarefa onde as interacções complexas entre a física da bola e a do robót tornam muito difícil criar um modelo analítico que possa competir com a precisão e habilidade de um jogador humano. Esta complexidade torna o ténis de mesa um desafio aliciante, com uma longa história de demonstração de algoritmos em robótica. Adicionalmente os modelos analíticos tendem a ser muito sensíveis às parametrizações, e requerem reprogramação quando existem alterações. Uma abordagem baseada em aprendizagem automática permite ao robô adaptar-se de forma autónoma, e aprender o modelo e a tarefa em simultâneo. Esta dissertação pode ser dividida em duas partes. Na primeira parte é discutida a regressão através de processos gaussianos (GPR) e é feita uma ponte para a Regressão Kernelizada regularizada por Custo (CrKR) um algoritmo do estado-da-arte de aprendizagem por reforço (RL). Explorámos novas estratégias online para aumentar a esparsidade das amostras recolhidas. Estas permitem a continuação da aprendizagem para além de um número de amostras após o qual a complexidade computacional de $O(n^3)$ da regressão se torna onerosa. A segunda parte aborda métodos de aprendizagem por imitação (IL). Introduzimos as Primitivas Dinâmicas de Movimento (DMP) e o nosso método de imitar trajetórias, as Primitivas de Movimento por Programação Quadrática (QPMP). Este método aproveita a existência de algoritmos extremamente eficazes para minimizar funções convexas e tem flexibilidade acrescida comparativamente aos métodos existentes. Permitindo a definição de restrições na generalização de trajetórias demonstradas. Apresentamos resultados obtidos em simulação e numa plataforma robótica real.

Palavras Chave

Ténis de Mesa, Aprendizagem por Reforço, Aprendizagem por Demonstração, Primitivas de Movimento
Contents

1 Introduction .................................................. 1
   1.1 Motivation ............................................... 2
   1.2 State of The Art ........................................ 2
      1.2.1 Table Tennis as a Robotics Challenge ............ 2
      1.2.2 Learning Table Tennis ............................... 3
      1.2.3 Imitation Learning of Trajectories ................. 3
   1.3 Original Contributions .................................. 4
   1.4 Thesis Outline ........................................... 4

2 Reinforcement Learning of Motion Parameters ............. 7
   2.1 Gaussian Process Regression ............................ 8
      2.1.1 Marginal Likelihood Maximization .................. 11
      2.1.2 Properties ......................................... 12
      2.1.3 Summary ........................................... 12
   2.2 Cost-regularized Kernel Regression .................... 13
      2.2.1 Selection of the Cost Function ...................... 14
      2.2.2 Algorithmic Complexity ............................ 15
      2.2.3 Cost weighted Sparsification ....................... 15
         2.2.3.A ANMS Filter .................................. 17
      2.2.4 Summary ........................................... 17

3 Trajectory Imitation and Generalization .................. 19
   3.1 Imitation with DMPs .................................... 20
      3.1.1 Original Formulation ............................... 21
      3.1.2 Hitting Movements using DMPs ...................... 22
   3.2 Quadratic Optimization for Motion Primitives ......... 25
      3.2.1 Formulation ........................................ 25
      3.2.2 Solution ........................................... 26
   3.3 Summary ................................................ 28

4 Experimental Results ....................................... 29
   4.1 Experimental results .................................... 30
4.1.1 Evaluation of QPMPs and DMPs .................................................. 30
4.1.2 Adaptation to different initial/final states and movement duration ........ 30
4.1.3 Inclusion of joint limits ............................................................... 31
4.1.4 Addition of intermediate way-points .......................................... 32
4.1.5 Rejection of noise and disturbances ........................................... 33
4.1.6 Effect of noise in the demonstration ............................................ 33

4.2 Experimental Setup and Simulation .............................................. 34
  4.2.1 Simulation Architecture .......................................................... 35
  4.2.2 Reinforcement Learning .......................................................... 36
    4.2.2.A Optimization of the Kernel Parameters ................................. 36
  4.2.3 Imitation Learning ................................................................. 38
  4.2.4 Results ....................................................................................... 38

4.3 Robotic Platform ................................................................. 40
  4.3.1 Implementation in the Robotic Platform .................................... 40
  4.3.2 Vision System for State Tracking ............................................. 40
  4.3.3 Mahalanobis Distance Validation Gate ..................................... 43
  4.3.4 Ground Collisions ................................................................. 43
  4.3.5 Experiments on the Robotic Platform ....................................... 43

5 Conclusions and Future Work ................................................. 45

Bibliography ................................................................. 47
# List of Figures

2.1 A linear function learned using GPR .................................................. 9
2.2 A SE-ARD Kernel composed with two SE Kernels .......................... 11
2.3 A linear function learned with CrKR ............................................... 14
2.4 Kernel Parameter sensitivity of CrKR .............................................. 15
2.5 Distribution of the samples collected by CrKR with online sparsification .. 16
2.6 Representative sample selection in CrKR with ANMS filter .............. 18
3.1 Generalization of primitives in the joints of a robot arm .................... 21
3.2 Gaussian functions and weights of a motor primitive ....................... 24
4.1 DMP and QPMP adaptation in one degree of freedom (DOF) .......... 31
4.2 QPMP adaptation with joint constraints ...................................... 32
4.3 QPMP adaptation with via-points .............................................. 33
4.4 Rejection of disturbances by QPMPs and DMPs ............................ 33
4.5 Savitsky-Golay filtering for QPMP in the presence of noise .............. 34
4.6 Simulation Software Architecture .............................................. 35
4.7 Mapping of the ball state to the robot state .................................. 36
4.8 Learning steps for table tennis with CrKR .................................. 37
4.9 Biorobs workspace with and without ground plane ....................... 38
4.10 Cost and success rates for different \( \lambda \) when running CrKR with sparsification in the Gazebo Simulation .......................... 39
4.11 Variation of the learned goal with the ball state .......................... 39
4.12 Plot joint angles learned with CrKR as a function of the ball state ... 40
4.13 Robotic Platform Software Architecture .................................... 41
4.14 Retro-reflective ball used in the experiments ................................ 41
4.15 The Extended Kalman Filter (EKF) prediction in the real system .... 43
4.16 Setup of the Robotic Platform ................................................... 44
4.17 Diagram of the execution sequence for a new trial ....................... 44
List of Tables

4.1 Optimized SE-ARD Kernel Hyperparameters for the simulation. . . . . . . . . . . . . . . 37
Abbreviations

**PbD**  Robot Programming by Demonstration

**LfD**  Learning from Demonstration

**ML**  Machine Learning

**RL**  Reinforcement Learning

**IL**  Imitation Learning

**GP**  Gaussian Process

**GPR**  Gaussian Process Regression

**CrKR**  Cost-regularized Kernel Regression

**DMP**  Dynamic Motion Primitive

**QPMP**  Quadratic Program Motion Primitive

**QP**  Quadratic Program

**SE**  Squared Exponential

**SE-ARD**  Squared Exponential - Automatic Relevance Determination

**EKF**  Extended Kalman Filter

**LKF**  Linear Kalman Filter

**ROS**  Robot Operating System

**HSV**  Hue, Saturation and Value

**DOF**  degree of freedom

**RANSAC**  Random sample consensus

**FPS**  Frames Per Second

**Mo-cap**  Motion capture

**URDF**  Unified Robot Description Format
ANMS  Adaptive Non-Maximal Suppression

YARP  Yet Another Robot Platform
## List of Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E$</td>
<td>Expected Value</td>
<td>8</td>
</tr>
<tr>
<td>$\mu$</td>
<td>Mean</td>
<td>8</td>
</tr>
<tr>
<td>$\text{Cov}$</td>
<td>Covariance</td>
<td>8</td>
</tr>
<tr>
<td>$k(x, x')$</td>
<td>Kernel evaluated between two data points</td>
<td>8</td>
</tr>
<tr>
<td>$I$</td>
<td>Identity Matrix</td>
<td>8</td>
</tr>
<tr>
<td>$K$</td>
<td>Covariance kernel evaluated on the input training data $k(X, X)$</td>
<td>9</td>
</tr>
<tr>
<td>$k_*$</td>
<td>Covariance kernel evaluated between the input point $x_*$ and the training data $X$</td>
<td>9</td>
</tr>
<tr>
<td>$S$</td>
<td>Matrix training inputs $x_i$</td>
<td>13</td>
</tr>
<tr>
<td>$\Gamma$</td>
<td>Vector of the training outputs $y_i$</td>
<td>13</td>
</tr>
<tr>
<td>$C$</td>
<td>Diagonal cost matrix $c_i$</td>
<td>13</td>
</tr>
<tr>
<td>$|x - x_{\text{target}}|$</td>
<td>L2 norm</td>
<td>15</td>
</tr>
<tr>
<td>$f$</td>
<td>Nonlinear forcing function</td>
<td>21</td>
</tr>
<tr>
<td>$z$</td>
<td>Canonical system</td>
<td>21</td>
</tr>
<tr>
<td>$\alpha_y$, $\beta_y$</td>
<td>gains of the canonical system</td>
<td>21</td>
</tr>
<tr>
<td>$\tau$</td>
<td>Temporal scaling term</td>
<td>21</td>
</tr>
<tr>
<td>$d$</td>
<td>Demonstration</td>
<td>22</td>
</tr>
<tr>
<td>$\psi_i$</td>
<td>Gaussian basis functions</td>
<td>22</td>
</tr>
<tr>
<td>$\xi(z, g)$</td>
<td>scaling term</td>
<td>22</td>
</tr>
<tr>
<td>$w_i$</td>
<td>Weights of the Gaussians</td>
<td>22</td>
</tr>
<tr>
<td>$|x_{\text{ball, t_{hit}}} - x_{\text{paddle, t_{hit}}}|$</td>
<td>L2 norm</td>
<td>36</td>
</tr>
<tr>
<td>$Q$</td>
<td>Observation error variance</td>
<td>42</td>
</tr>
<tr>
<td>$y$</td>
<td>Measurement Residual</td>
<td>42</td>
</tr>
<tr>
<td>$S$</td>
<td>Residual Covariance</td>
<td>42</td>
</tr>
<tr>
<td>$K$</td>
<td>Kalman Gain</td>
<td>42</td>
</tr>
<tr>
<td>$P$</td>
<td>Covariance Estimate</td>
<td>42</td>
</tr>
<tr>
<td>$\delta_t$</td>
<td>Time Step of the EKF</td>
<td>42</td>
</tr>
<tr>
<td>$F$</td>
<td>State Transition Matrix</td>
<td>42</td>
</tr>
<tr>
<td>$H$</td>
<td>Observation Matrix</td>
<td>42</td>
</tr>
<tr>
<td>$\delta_{\text{mah}}$</td>
<td>Mahalanobis distance</td>
<td>43</td>
</tr>
<tr>
<td>$C_R$</td>
<td>Restitution coefficient</td>
<td>43</td>
</tr>
</tbody>
</table>
Introduction

Contents

1.1 Motivation ......................................................... 2
1.2 State of The Art .................................................. 2
1.3 Original Contributions ......................................... 4
1.4 Thesis Outline ................................................... 4
1.1 Motivation

The perception of robots as humanities future helpers, capable of performing tasks with the same dexterity as a human has been driving the research of robots with increasing mechanical complexity. Today robots take the place of humans in the most tedious and dangerous jobs. They can already be seen vacuuming peoples homes and searching for life on mars. However these are still relatively simple robots, with a high degree of specialization for a single task.

More general robots with many degrees of freedom and a large number of sensors have become common in research (e.g. iCub, PR2) and are also showing signs of increasingly more industrial use (e.g. Baxter). However this increase in mechanical sophistication was lagged by control methods, many tasks that are relatively simple for humans such as driving a car or repairing a broken object today remain an immense challenge to robots. The increased complexity of the robots leads to an increase in the required human effort to model the robot and its environment for each new task.

An appealing alternative to extensive hand modeling is to learn the task and model together by demonstration or by trial and error. Reinforcement Learning approaches allow the robot to learn the task and model from available sensor data, integrating into the learned model even physical properties that are unknown or very hard to incorporate analytically. The system can also be made less sensitive to disturbances by unknown properties or unforeseen situations since these can be learned while performing the task. When left to learn by its own experiments the robot can even discover new solutions for accomplishing the task that surpass the creativity of its human creators.

Learning the appropriate motor commands for a given task can be regarded at different levels. At a higher level we have to find what are the goals or parameters that will lead to success in each new situation. At a lower level we have the trajectory that the robot will execute to reach the goals. At a even lower level we have the sensorimotor control, the torques that have to be applied by the motors given the state feedback and a reference. This dissertation will focus on the first two problems, using for the last one traditional control methods.

1.2 State of The Art

1.2.1 Table Tennis as a Robotics Challenge

Robot table tennis has been the subject of much research ever since John Billingsley put forward the idea in 1983 [8]. In the last decades a variety of table tennis playing robots have been designed, ranging from systems based on commercial robot arms like the PUMA 260 with 4 video cameras of R. L. Andersson [3], the low cost monocular vision robot of Acosta et al. [1]. Or a system using high speed cameras to measure and model the effect of spin on the trajectory of the ball [21]. These robots have been able to return a few volleys with humans but none have reached the dexterity necessary to actually compete with a professional human player.
1.2.2 Learning Table Tennis

When performing a complex motor task like playing table tennis, most current robots rely on well modeled environments with carefully estimated parameters. An expert must then reprogram the new motor behaviours for any change in the robot its environment or task. Given the diversity of situations that cannot be fully anticipated, learning techniques are going to play a major role in increasing the adaptability of complex motor skills and behaviours and in reducing the duplication of human effort.

To make a robot learn motor skills from its own interactions with the environment we can look to Reinforcement Learning (RL) methods. RL is a general approach to action-based reward-directed learning under uncertainty [45].

Learning problems that can be classified as RL have been studied deeply in the last decade. These problems range from learning the dynamics models with PILCO [14] to the recent developments on deep Learning to learn super-human skills in Atari video games, Mnih et al [36]. These deep learning methods are only now being used to learn continuous high-dimensional action spaces, opening the possibility of learning end-to-end from the camera input to the motor control of the robot [32] [33]. However, learning end-to-end in a model-free method requires a large number of trials, hard to obtain in a mechanical system that can suffer wear and for tasks that require human interaction. A more data-efficient approach for continuous low-dimensional input and output spaces is the Cost-regularized Kernel Regression (CrKR) proposed by Kober et al. [27]. This approach uses a cost-function to guide the exploration and a cost weighted mean allows better exploration far from known data-samples than kernel ridge regression [27]. A robot arm exploring from a small number of initial demonstrations with CrKR achieved a return rate 90% of the volleys served by a human player, Mulling et al. [37].

1.2.3 Imitation Learning of Trajectories

Using RL techniques to learn with in a low dimensional action-space results in target goals that must be achieved by the robot. To reach these goals a trajectory dependant on the target and the robots current state must then be generated according to chosen criteria. One practical way to allow robots to learn motor skills is through the "programming by demonstration" paradigm [4] [5] [7]. The main idea is that the motor skill is demonstrated to the robot (typically through kinesthetic teaching) and a representation of the movement is learned from the recorded data. Using such a representation the robot can replicate the demonstrated motion under different conditions e.g. with a different start or end position, thus showing not only the ability to learn from a human teacher but also to generalize and adapt what has been learned. Moreover, the learned primitive can be progressively refined though practice using, for example RL techniques [45]. One key issue is therefore to find a proper representation for such movement primitives. A number of solutions have been proposed during the last decade, resorting for instance to neural networks [46], probabilistic estimation [10] and many other techniques. One of the most promising approaches draws from the theory of dynamical systems, giving raise to solutions such as Stable Estimator of Dynamical Systems (SEDS [24]), sequenced Linear
Dynamical Systems \cite{16}, Implicit Dynamical Systems \cite{30} and, most notably, Dynamic Movement Primitives (DMPs \cite{22, 23}). This latter formulation in particular has proven to be a very effective tool for imitation learning, and has been therefore widely used in robotics and inspired many extensions to add velocity goals \cite{25, 37} and allow uncertainty and way-points in the execution \cite{40}. However, the Dynamic Motion Primitives (DMPs) approach still has a few drawbacks. One is that, despite allowing quick adaptation to new start/end positions, it may generate undesired motion profiles (especially in terms of accelerations) that are not under full control. Another one is that it lacks the possibility to impose constraints in the motion, such as joints position and velocity limits.

Summarizing, Imitation Learning (IL) provides more human-like motor control when compared to generating trajectories by minimum jerk or other criteria. Trajectory “learning by demonstration” is specially appropriate when we are already using expert demonstrations for learning trajectory goals in CrKR. The idea is that there will be no discontinuity between the shape of the trajectories generated by the robot and the shape of the initial demonstrations avoiding a possible mismatch between demonstration and exploration.

1.3 Original Contributions

In this section we list the contributions introduced by this work.

**Setup for robot table tennis** A setup based on a simple robot arm (biorob) and mid-end Optitrack motion tracking cameras. This setup was also reproduced in a simulated environment.

**Online Cost Regularized Kernel Regression** We use Adaptive Non-Maximal Suppression (ANMS) filtering for selecting online a sparse, low-cost, subset of the collected data to learn with CrKR beyond a fixed budget of samples.

**Novel approach to imitation learning using quadratic programming** A method based on quadratic programming that is an alternative to DMPs and adds the flexibility to parametrize the generated trajectory. This work resulted in the following publication:


1.4 Thesis Outline

In Chapter 2 we introduce Gaussian Process Regression (GPR) and CrKR to learn the mapping between a system state and a desirable robot state according to a cost function. This mapping is found through autonomous exploration by the robot. We also propose a online sparsification based on ANMS filtering.

We follow with trajectory imitation from expert demonstrations in section 3. These trajectories will have a shape matching those of the demonstrations used for initializing CrKR. We describe the DMP
framework and introduce our new solution, Quadratic Programming Motion Primitives (QPMP) based on the minimization of a convex function.

In Chapter 4 we present experiments demonstrating the properties of our methods. We describe the results obtained in a simulation environment and our robotic platform, some videos and materials related to these experiments can be seen in our repository [11].
2 Reinforcement Learning of Motion Parameters

Contents

2.1 Gaussian Process Regression ........................................... 8
2.2 Cost-regularized Kernel Regression ............................. 13
Many dexterous motor skills performed by humans are very difficult to automate. From performing
tennis swings to playing drums and biped locomotion, these skills require complex models that have to
be finely crafted and tuned, lacking the capability to generalize to different machines or environments.

Machine Learning, specially Reinforcement Learning strives to give robots the capability to learn
complex tasks from experience, like humans and other animals possess. Robots that learn from
experience can adapt to changes in the environment or to their own dynamics without manual repro-
gramming.

There are many ways of learning a function from data, one that is particularly elegant from a
mathematical viewpoint, and that has recently been successfully applied for many robot learning
tasks, is probabilistic inference [17]. Probabilistic inference methods take a group of hypotheses on
the properties of the underlying distribution and weight those based on how well the predictions fit the
data.

In this chapter we introduce Gaussian Processes (GPs), a simple class of models of functions
suitable for probabilistic inference in both regression and classification problems. A Reinforcement
Learning method related to Gaussian Processes, CrKR, is presented in conjunction with a novel
strategy to continue learning beyond the point where the cubic time complexity with the number of
samples becomes a limitation.

2.1 Gaussian Process Regression

A GPs is by definition any distribution over functions such that any finite set of function values
\( f(x_1), f(x_2), \ldots, f(x_N) \) has a joint Gaussian distribution [42].

The GPs before training is specified by its mean function

\[
E[f(x)] = \mu(x)
\]

and by its covariance function or \textit{kernel}:

\[
\text{Cov}[f(x), f(x')] = k(x, x').
\]

In practice it is common to assume the mean function is zero everywhere. The choice of the covariance function is of a greater importance since it implicitly encodes assumptions about the underlying
function to be modeled.

GPR is a powerful non-parametric probabilistic method for model learning. The learning problem
for GPR can be seen as, given \( n \) training points \( (x_i, y_i)_{i=1}^n \), to find the latent function \( f(x_i) \) that
transforms the input \( x_i \) into a prediction \( y_i = f(x_i) + \epsilon_i \) where \( \epsilon_i \) is Gaussian noise with zero mean
and variance \( \sigma_i^2(x_i) \), i.e. \( \epsilon \sim \mathcal{N}(0, \sigma_i^2(x_i)) \).

The prediction of a GPR is found by calculating the mean and the variance at a new evaluation
point \( x_* \):

\[
\hat{f}(x_*) = k^T_*(K + \sigma^2_n I)^{-1}y
\]

\[
\sigma^2_f(x_*) = k(x_*, x_*) - k^T_*(K + \sigma^2_n I)^{-1}k_*
\]

(2.3)
where $K$ is the matrix where each entry corresponds to the covariance kernel evaluated on the input training data $K_{i,j} = k(x_i, x_j)$, $k_*$ is a vector where each entry is the covariance function evaluated between the input point $x_*$ and the remaining training data. The variance of the input data is $\sigma_n^2$ and $y$ is the vector of the training outputs.

![Figure 2.1: Gaussian Process after conditioning on five noisy samples from a linear function. The pink region and the error bars show the 95% confidence interval for the Gaussian Process and of each of the data points.](image)

The kernel defines how the model generalizes to new data. In this way it provides a prior and specifies the kind of structure that can be captured by the GP. Many common ML methods such as Linear regression, splines and Kalman filters can all be seen as special cases of GPs with specific kernels [17]. The specification of a kernel that represents the particular structure of the problem being modeled is one of the main challenges in applying GPs for model learning. However in practice the Squared Exponential (SE) kernel has become the de facto default kernel when the structure of the function is unknown or hard to specify. It has very appealing properties such as being a universal
kernel. A universal kernel can approximate an arbitrary continuous function on any compact subset of the input space [35]. It is defined by only two parameters that are easy to interpret. First the lengthscale or bandwidth of the kernel $l$ determines the smoothness of the function and how far away from the training data the GP can extrapolate. The output variance $\sigma^2$ determines the average distance of the function away from it’s mean. In practice it acts as a scale factor.

$$k_{SE}(x, x') = \sigma^2 \exp(-\frac{(x - x')^2}{2l^2})$$  \hspace{1cm} (2.4)$$

For multidimensional problems we can multiply many SE kernels, each with a bandwidth adjusted to the corresponding dimension. This multiplication of kernels with different bandwidths makes the so called Squared Exponential - Automatic Relevance Determination (SE-ARD) kernel in eq. (2.5).

$$k_{SE-ARD}(x, x') = \prod_{d=1}^{D} \sigma_d^2 \exp(-\frac{(x_d - x'_d)^2}{2l_d^2}) = \sigma_f^2 \exp(-\frac{1}{2} \sum_{d=1}^{D} \frac{(x_d - x'_d)^2}{l_d^2})$$  \hspace{1cm} (2.5)$$
The high flexibility of the SE-ARD kernel means that more data may be needed to learn, the called "curse of dimensionality" [6]. In principle a kernel with more prior structure is preferable as it will learn faster, the "blessing of abstraction" [20].

2.1.1 Marginal Likelihood Maximization.

A very important feature of GPs is the ability to compute the marginal likelihood of a test data set given the model or evidence. Models can be compared using the marginal likelihood, which allows automatically tuning the parameters of a model and its fit to the data

$$p(y|x, \theta) = (2\pi)^{-\frac{n}{2}} \times |K|^{-\frac{n}{2}} \times \exp\{-\frac{1}{2}(y^T K^{-1} y)\}$$

(2.6)
We can write in log form and note that it is differentiable

\[ \log p(y|x, \theta) = -\frac{1}{2} y^T K^{-1} y - \frac{1}{2} \log |K| - \frac{y}{2} \log 2\pi \]

\[ \frac{\partial}{\partial \theta_j} \log p(y|x, \theta) = \frac{1}{2} y^T K^{-1} \frac{\partial K}{\partial \theta_j} K^{-1} y - \frac{1}{2} \text{tr}(K^{-1} \frac{\partial K}{\partial \theta_j}) \]

\[ = \frac{1}{2} \text{tr}((\alpha \alpha^T - K^{-1}) \frac{\partial K}{\partial \theta_j}) \]

where \( \alpha = K^{-1}y \) (eq. 2.7)

The open parameters of \text{GPR} are the \text{hyper-parameters} of the kernel, these can be the \( \sigma \) and \( l \) in the particular case of the \text{SE} kernel. The usual practice is to maximize the log marginal likelihood using common optimization tools such as quasi-Newton methods or take advantage of its differentiability (eq. 2.7) and use gradient methods Plagemann et al. [41].

2.1.2 Properties

GPs offer many appealing properties, when compared to other nonparametric models.

- The posterior distribution can be calculated in closed form given the kernel and some samples.
- Modeling assumptions can be expressed by the choice of the kernel or covariance function.
- The posterior, given a fixed kernel, can integrate over a wide range of hypotheses reducing overfitting, no need for complex regularization schemes like the ones used for neural networks [17].
- The possibility to calculate the marginal likelihood of the data given the model allows comparing between models to choose the most appropriate one.
- The predictive distribution of a GPs at a set of test points is a multivariate Gaussian distribution. This means GPs can be composed with other models.

2.1.3 Summary

GPs are suitable for regression and classification problems with a moderate number of training samples. The main imitaton is the need for matrix inversion with \( O(n^3) \) complexity. When compared to other inference methods GPs have many appealing properties, such as the possibility of tuning the parameters of the kernel directly from the training data without requiring cross validation. The choice of the kernel can encode prior knowledge possibly making the learning very data efficient. However choosing a kernel for a particular problem is not trivial, it is equivalent to learning a useful representation of the input. Kernel parameters can be set automatically by maximizing the marginal likelihood but to choose the parametric form of the kernel requires some understanding of the underlying problem. Recently some methods for automatic model construction were demonstrated by Duvenaud et al. [18].
2.2 Cost-regularized Kernel Regression

Gaussian processes assume that the training points are sampled from the underlying process with Gaussian noise. However, some tasks are interactive in nature and must be actively explored by the robot system. If we consider a newborn animal taking its first steps before knowing how to run, or a robot learning how to play-ball-in-a-cup, we can see that interaction is required, the agent is seeking a goal by interacting with an uncertain environment. The agent needs to use the acquired experience to improve its performance over time. CrKR is a kernelized version of the reward-weighted regression [27], that is suitable for RL [27] of complex tasks with continuous input and output spaces. A common application of CrKR is to learn a small set of continuous meta-parameters that generalize a motor primitive [27, 37].

Algorithm 1 Cost-regularized Kernel Regression algorithm

Preparation steps:
- determine initial state $s_0$, output $\gamma_0$ and cost $c_0$ of the demonstration.
- initialize the corresponding matrices $S, \Gamma, C$.
- choose a kernel $k$, initialize $K$.
- set the exploration/exploitation trade-off $\lambda$.

for all iterations $j$ do
- Determine the state $s_j$ specifying the situation.
- Calculate the meta-parameters $\gamma_j$ by:
  $\bar{\gamma}_i(s_j) = k(s_j)^T (K + \lambda C)^{-1} \Gamma_i$,
- Determine the variance:
  $\sigma_j^2(s_j) = k(s_j, s_j) - k(s_j, S)^T (K + \lambda C)^{-1} k(s_j, S)$,
- Draw the meta-parameters from a Gaussian distribution $\gamma_j \sim N(0, \sigma^2)$
- Execute the motor-primitive using the new meta-parameters
- Calculate the cost $c_j$ at the end of the episode
- Update the matrices $S, \Gamma, C$ with the new sample
end for

CrKR has been successfully applied to many robot learning tasks such as dart throwing, ball throwing and table tennis in many different robots, outperforming techniques like finite difference gradient and reward weighted regression [26].
Figure 2.3: CrKR after learning from an initial demonstration and performing five exploration iterations. The pink region shows the 95% confidence interval for the CrKR exploration, the error bars show the cost of each tryout.

The inference is performed in a way that is similar to Gaussian Process Regression (GPR). In eq. (2.3), the term $\sigma^2 I$ is replaced by $\lambda C$ to make a cost weighted regression. The open parameter $\lambda$ expresses an exploration exploitation trade-off and also a scaling of the cost function to what could be seen as analogous to a variance in a GPR. Given a state of the system $s_j$ the policy for our action is predicted mean $\gamma_j(s_j)$ combined with the predicted variance $\sigma^2(s_j)$ to add exploration.

2.2.1 Selection of the Cost Function

Great care must be taken in the choice of a cost function appropriate to the task. The cost function is employed in CrKR as a predictive variance used to guide the exploration. Cost functions that represent some notion of closeness to the target tend to perform better than binary success/failure functions in robotic tasks. Some desirable properties of the learned policy may be discovered by the
Figure 2.4: CrKR is specially sensitive to the choice of the kernel and the tuning of its parameters. The kernel affects both the inference but also for the exploration of new data-points. We plotted three learning iterations all with the SE-ARD kernel. In the first figure the bandwidth of the kernel is too wide to explore the ripples of the function. In the second figure we have the opposite situation, the ‘short’ inference leads the exploration to explore with zero mean and high variance when away from low cost samples leading to slow learning. In the last case we used the kernel parameters obtained by maximizing the marginal likelihood as described in 2.1.1.

autonomous agent without needing to be explicitly included in the cost function. As an example a batting robot may select policies that avoid high accelerations even if the the cost function does not consider the generated trajectory but only the accuracy of the final hit for the computation of the cost value. For easier reasoning we decided to use the saturating cost function in (2.8).

$$c(x) = 1 - \exp\left(-\frac{1}{2\sigma^2} \|x - x_{\text{target}}\|^2\right)$$

(2.8)

This cost function is locally quadratic but saturates for large deviations from the desired target. In Kording et al. Körding and Wolpert [29] it was shown that this saturated cost mimics well some properties of the human reasoning.

2.2.2 Algorithmic Complexity

Like GPR the prediction step involves the inversion of a square matrix $(K + \lambda C)^{-1}$ of order $n$ where $n$ is the number of samples. The time complexity of most matrix inversion algorithms is $O(n^3)$ and the space for storing the matrix is $O(n^2)$.

This leaves us with an algorithm that requires increasingly more computational resources during learning. This limits in practice the number of samples that can be used for learning. The final learned policy is highly dependant on the coverage of the input space and the quality of the initial demonstrations.

2.2.3 Cost weighted Sparsification

The $O(n^3)$ complexity in GPR becomes specially relevant when dealing with on-line learning, and real time applications. A possible solution to this problem was proposed by Nguyen-Tuong et al. [39]. The proposed solution consists in learning in the same way as a normal GPR until a fixed budget of samples is attained. For the subsequent samples, a independence measure is used to replace the least informative data point, if the new sample improves the coverage of the input. A time decay was also proposed to deal with changing dynamics of the system, such as when a load is added to a robot arm. In this case the algorithm will adapt to the new dynamics and forget the old. This on-line
Figure 2.5: Sample distribution on a 2 dimensional input space before and after running CrKR with on-line sparsification through our heuristic.

sparsification method although designed with GPR in mind is straightforward to apply to CrKR. The time decay concept can be replaced by a cost decay where high cost samples are rejected.

The independence measure is stated as:

\[
\delta = k(d(m+1), d(m+1)) - k^T a
\]

\[
a = K^{-1} k
\]

This independence is computationally very expensive to recalculate for each new sample if we keep high amount of samples in our budget. The solution found in [39] uses incremental updates to reduce the complexity of checking which sample to replace in the dictionary.

Using a SE-ARD kernel with a appropriate bandwidth we can use a fast heuristic to determine which sample to replace. By summing the distances to the other samples in kernel space we choose to remove samples in well represented areas of the input space and replace them with samples in areas that need better sampling.

We compute a measure of the closeness of the samples.

\[
\delta^* = \sum_{j=1}^{N} \sum_{j=1}^{N} K(s_j, s_j) \quad (2.9)
\]

When a new sample is obtained we use it to replace each of the previous samples and recompute the measurement \(\delta^*\). The sample that yields the higher \(\delta^*\) is the one that is kept. This is equivalent to finding the sample whose location has more samples in its vicinity, and keeping only the samples in less explored locations (2.9).

When we are learning by experience such as with CrKR we also have to concern ourselves with the fact that some experiments are more relevant than others. When using our sparsification approach in a naive way we can end up discarding good experiments to save poor ones in less explored regions. This will impair the learning, since CrKR takes advantage of the fact that samples close in the input space should have similar ideal output to guide the exploration. A solution to this problem is weighting
the independence measures with the cost associated to that experiment, $c_i + \delta^*_i$. This heuristic method performed well in our experiments but requires SE kernels with a specific bandwidth.

### 2.2.3. A ANMS Filter

The usual approach to overcome the cubic complexity of regression algorithms is to find a representative subset of the data-points. Since in our case the points have weights associated we can formulate the problem as finding the sparsest set of $N$ low cost points. This is a similar problem to the selection of keypoints in images for computer vision applications. We experimented with a filter commonly used for for spatial feature distribution Adaptive Non-Maximal Suppression (ANMS) [9, 19]. ANMS works by sorting the entire dataset according to the weights. Then starting from the lowest cost data point all data points are added in sequence by order of cost. For each point the distance to the closest point already evaluated is saved and added to a list. The $N$ points with highest distances in this list are selected for the regression. We provide a pseudo-code implementation of the algorithm in Algorithm 2.

**Algorithm 2** Naive Implementation of ANMS Filtering

**Preparation steps:**
Sort $P$ by cost, let $p_1, p_2, \ldots$ denote the points in that order.
result = [($\infty$, $p_1$)] //$\{(\text{radius}, \text{keypoint})\}$

for all $p_i \in P$ do
  $r_i$ = distance to nearest point already in result.
  result.pushback(($r_i, p_i$))
end for
Sort result by radius, return first $k$ entries in result.

Compared to the sparsification methods previously discussed this method allows retaining a larger number of training samples of which some are used for the regression. This avoids discarding samples that when collected were not relevant but can become necessary due to the collection of good samples in different locations of the input space. The filter can also work with any distance measure, giving flexibility in the choice of the kernel used for regression. The naive implementation showed in Algorithm 2 has a quadratic run time complexity. Sub-quadratic algorithms that make use of approximate nearest neighbor search exist [19]. These have a more complex implementation but become advantageous when the number of samples becomes sufficiently large.

### 2.2.4 Summary

CrKR provides an effective strategy to learn by trial and error. In its original formulation the high computational complexity of the matrix inversion limits the amount of samples that can be used for learning. We also can note that the exploratory nature of RL means that some of the collected experiments will be of a lower quality than others. In particular the trials made in the beginning of the training will generally be less informative than the ones collected when there is a better corpus of low cost examples to generate the new experiment. Substituting the worse samples may be a good strategy to overcome both these issues and to learn within a limited budget. We can use try to...
Figure 2.6: In this figure we can see how CrKR can continue to learn beyond a fixed budget by selecting the data samples with an ANMS filter. The system is learning an analytical simulation of the ball trajectory and the robot kinematics of a table tennis task. The vertical line shows the learning epoch where the budget of 250 samples was filled, beyond this line ANMS was used to select the most representative of all collected samples for regression. Each epoch consists of 50 training samples, the results are the average, the maximum and the minimum of 10 runs with the same parameters.

choose a representative sparse subset composed by the lowest cost training samples as a criteria to define what points are used for the regression. With an ANMS filter for sample selection the robot can continue improving each time a new experiment results in a valuable sample beyond the limitations imposed by the time complexity of the regression.
Contents

3.1 Imitation with DMPs .................................................. 20
3.2 Quadratic Optimization for Motion Primitives .................. 25
3.3 Summary ............................................................... 28
Learning from Demonstration (LfD) or Robot Programming by Demonstration (PbD) also called IL or Apprenticeship Learning forgo decomposing and programming the desired behaviours explicitly to the robot system. By instead relying on demonstrations made by a human expert we can achieve a more expedite deployment of a robot to a new task, also allowing the programming of robots by people without a traditional programming background. In the context of playing table tennis we want to find the motor trajectory that strikes the ball with the highest probability of success. An appealing alternative to trying to define what is the optimal criteria for such a trajectory is to simply imitate a demonstration by a human expert. In this chapter we describe DMP, a state of the art framework for IL of motor trajectories and introduce our novel solution to this problem based on constrained quadratic optimization. The methods proposed in this chapter were published in [12].

3.1 Imitation with DMPs

As previously introduced, DMP is a widely adopted formalism to represent motor actions in a way that allows flexible adjustment without custom parameter tuning. After a primitive is learned from a demonstration (or a series of demonstrations) it can then be generalized to different initial and goal states while maintaining its overall shape. The original DMPs have two formulations, one for discrete movements and another for periodic motion [13]. Here we focus on the discrete formalization, that is needed to represent point-to-point tasks.
3.1.1 Original Formulation

The DMP system controls the motion of a scalar variable \( y \) through a point attractor \( g \) that makes the trajectory converge asymptotically to the goal and a nonlinear forcing term \( f \) that encodes the characteristics of the demonstration. The two terms are coupled by a canonical system \( z \) that acts as a replacement of time.

\[
\begin{align*}
\tau^2 \ddot{y} &= \alpha_y (\beta_y (g - y) - v) + f(z, g) \\
\tau \dot{y} &= v \\
\tau \dot{z} &= -\alpha_z z 
\end{align*}
\]  

(3.1)

The gains \( \alpha_y, \beta_y \) are chosen to make the 2nd order system critically damped, ie. \( \beta_y = \alpha_y/4 \). The temporal scaling term \( \tau \) allows the primitive to execute the movement faster or slower while preserving its shape.

Let us consider a demonstrated trajectory as \( d(t) \). The objective of learning a DMP is to compute an approximation of \( f \) such that the observed profile of the trajectory is as close as possible to the demonstration. Rewriting the first equation of (3.1) and replacing the motion variable \( y \) by the
demonstration $d$ we have:

$$f_{\text{target}}(z, g) = \tau^2 \ddot{d} - \alpha_y (\beta_y (g - d) - \tau \dot{d})$$  \hspace{1cm} (3.2)$$

This term can be represented by a normalized linear combination of Gaussian basis functions \[23\], as follows:

$$f_{\text{target}}(z, g) = \frac{\sum_{i=1}^{N} \psi_i(z) w_i}{\sum_{i=1}^{N} \psi_i(z)} \xi(z, g)$$  \hspace{1cm} (3.3)$$

$$\psi_i(z) = \exp\left(-\frac{1}{2\sigma_i^2} (z - c_i)^2 \right)$$

$$\xi(z, g) = z(g - y(0))$$

The Gaussian basis functions $\psi_i$ have centers $c_i$ along the exponential of the Canonical system so that they are equally spaced in time. The scaling term $\xi(z, g)$ makes the accelerations converge to zero near the goal and to normalize their values according to the amplitude of the movement.

The weights $w_i$ can be learned from the samples of the original demonstration at sample times $z_p$, $p \in \{0, \cdots, P\}$, using for example Locally Weighted Regression \[22\]:

$$w_i = \frac{s^T \Gamma_i f_{\text{target}}}{s^T \Gamma_i s}$$  \hspace{1cm} (3.4)$$

where

$$s = \left[ \xi(z_0) \cdots \xi(z_P) \right]^T$$

$$f_{\text{target}} = \left[ f_{\text{target}}(z_0, g) \cdots f_{\text{target}}(z_P, g) \right]^T$$

$$\Gamma_i = \begin{pmatrix} \psi_i(z_0) & \cdots & 0 \\ 0 & \ddots & 0 \\ 0 & \cdots & \psi_i(z_P) \end{pmatrix}$$  \hspace{1cm} (3.5)$$

The canonical system has $\alpha_z > 0$ so that it converges asymptotically to zero. This ensures that, if the weights are bounded, the forcing function’s interference eventually vanishes as the movement ends, ensuring that the point attractor is free to converge to the goal. The initial value of $z$ is usually set to 1 but any positive value can be chosen.

### 3.1.2 Hitting Movements using DMPs

For some situations such as striking a moving ball and throwing darts it is necessary to adapt to an end goal with a desired final velocity $v_f$, different from zero. A DMP formulation that adapts to goals with velocity was proposed by Kober et. al \[25\]. The proposed system used a shifting goal with linear velocity. This system however had some drawbacks: First, fast changes of the goal position and velocity lead to error in the final velocity. Second, when the final and starting position are close to each other the system generates large accelerations. To overcome these shortcomings Mulling et al \[37\] proposed an improved system, which is defined as follows:

$$\tau \dot{v} = \alpha_y (\beta_y (g - y) + \dot{g} \tau - v) + \dot{g} \tau^2 + \eta f(z)$$

$$\tau \dot{y} = v$$  \hspace{1cm} (3.6)$$

Here the goal follows a polynomial that starts and ends with zero accelerations and the correct starting and final position and velocity, computed as:
\[ g = \sum_{j=0}^{5} b_j \left( -\tau \frac{\ln(z)}{\alpha_z} \right)^j \quad \dot{g} = \sum_{j=1}^{5} j b_j \left( -\tau \frac{\ln(z)}{\alpha_z} \right)^{j-1} \]
\[ \ddot{g} = \sum_{j=2}^{5} (j^2 - j) b_j \left( -\tau \frac{\ln(z)}{\alpha_z} \right)^{j-2} \] (3.7)

The 5th order polynomial coefficients are found by setting the boundary conditions such that it has the desired initial and final positions and velocities. The accelerations are set to zero at the start and at the end of the movement.
Figure 3.2: Here we can see how the weights combine with the gaussian functions to generate the acceleration of the motor primitive. In the first panel we plotted the thirty gaussian functions with centers shifted in time and the corresponding weights found by our optimization. Each Gaussian was multiplied by its weight as shown in the second panel. Finally by adding together all the gaussians we get the acceleration profile of the motor primitive as seen in the third panel.
3.2 Quadratic Optimization for Motion Primitives

The approach in Mulling et al. [37], implemented through (3.6) and (3.7), achieves the final goal position and velocity with accuracy. However, the polynomial trajectory is computed independently of the demonstrated trajectory and may lead to accelerations profiles significantly different from the demonstration. Moreover, the acceleration vanishes at the end of the movement. This distorts the demonstrated trajectory at the most important phase in hitting movements. To have a greater control on the resulting trajectory profiles, we formulate the problem as a convex Quadratic Program (QP). The imitation problem is addressed by computing the trajectory that best imitates the acceleration profile of the demonstration while constrained to start and end at specific positions and velocities. Additional constraints can be incorporated, for instance via-points and limits in the joint positions, velocities or accelerations during the whole movement trajectory.

3.2.1 Formulation

We use a dynamical system as (3.6), but instead of having separate reference trajectory and forcing terms, we generate a single desired trajectory directly taking all constraints into account (imitation of the demonstration, initial and final states, joint limits, waypoints) and plug it into a reference tracking dynamical system. This is no longer a DMP but still ensures robustness to noise and perturbations. To compute the desired trajectory we use a Gaussian kernel expansion as in the DMPs forcing term (3.3). Let \( g(t), \ t \in [t_0, t_f] \) denote the desired trajectory to be executed between \( t_0 \) and \( t_f \). Its acceleration is represented as:

\[
\ddot{g}(t) = \sum_{i=1}^{N} w_i \psi_i(t) \tag{3.8}
\]

The Gaussian basis functions \( \psi_i(t) \) are the same ones used for DMPs but time warped to fit the desired time interval. Since we do not need a canonical system with exponential decay, we can have the basis functions spaced linearly in time and with constant width. The expression for the \( \psi_i(t) \) is the same as in (3.3), but now the input is time \( t \) instead of phase \( z \), and centers \( c_i \) and variance \( \sigma \) are shifted and warped according to the transformation from the demonstration time interval to the imitation time interval.

The desired trajectory position and velocity can be obtained by integration:

\[
\dot{g}(t) = v_0 + \sum_{i=1}^{N} w_i \psi_i'(t) \tag{3.9}
\]

\[
g(t) = p_0 + (t - t_0)v_0 + \sum_{i=1}^{N} w_i \psi_i''(t)
\]

where \[1\]

\[
\psi_i'(t) = \int_{t_0}^{t} \psi_i(\tau) d\tau \quad \psi_i''(t) = \int_{t_0}^{t} \psi_i'(\tau) d\tau
\]

\[1\]The cumulative functions \( \psi_i'(t) \) and \( \psi_i''(t) \) are, in practice, approximated by numerical integration.
Given a demonstration \( d(t) \) and the desired initial and final times, positions and velocities, respectively \((t_0,p_0,v_0,t_f,p_f,v_f)\), the optimization problem is thus written as:

\[
\text{minimize} \quad \int_{t_0}^{t_f} \left( \sum_{i=1}^{N} w_i \psi_i(t) - \tau^2 \ddot{d}(t) \right)^2 dt
\]

subject to

\[
p_0 + (t_f - t_0)v_0 + \sum_{i=1}^{N} w_i \psi_i''(t_f) = p_f
\]

\[
v_0 + \sum_{i=1}^{N} w_i \psi_i'(t_f) = v_f
\]

where \( \tau \) is the time scale due to different durations of the demonstration and imitation.

After the trajectory is computed, it can be used as a shifting goal in the acceleration-based controller:

\[
\ddot{y} = \alpha_y (\beta_y (g - y) + \dot{g} - v) + \ddot{g}
\]

\[
\dot{y} = v
\]

The time scaling factor is included directly in the solution of the minimization problem. Every time a trajectory has to be computed, the new \( \psi_i(t) \) terms are recomputed to cover the desired time span, so it is no longer necessary to scale by \( \tau \) in the dynamic system.

Note that the computed trajectory plays the role of both the forcing term and the shifting goal. Because the trajectory already incorporates the imitation, there is no need to use two different entities to implement the dynamical controller.

### 3.2.2 Solution

To solve the optimization problem we rearrange (3.10) as a QP with linear constraints in standard form. Then, any existing modern QP solver can be used to obtain the result.

Expanding the cost function in (3.10) and defining \( \psi_{ij} = \int \psi_i(j(t) \ddot{d}(t) \text{ and } D^2 = \int \ddot{d}^2(t) \) we get:

\[
J = \int dt \left( \sum_{j=1}^{N} w_j \psi_j(t) - \tau^2 \ddot{d}(t) \right)^2 =
\]

\[
J = \sum_{j=1}^{N} \sum_{i=1}^{N} w_{ij} \psi_{ij} - 2\tau^2 \sum_{j=1}^{N} w_j \theta_j + \tau^4 D^2
\]

where \( w_{ij} = w_i w_j, \psi_{ij}(t) = \psi_i(t) \psi_j(t) \) and the integrals are between \( t_0 \) and \( t_f \).

Because the last term does not depend on the \( w_i \)’s it can be left out of the optimization cost \( J \). Therefore, representing in matrix form we get:

\[
J = w^T \Psi w - 2\tau^2 w^T \theta \quad (3.12)
\]

where

\[^2\text{In practice the terms } \psi_{ij} \text{ and } \theta_j \text{ are approximated by numerical integration}\]
\[
\mathbf{w} = [w_1 \ldots w_n]^T \quad \mathbf{\theta} = [\theta_1 \ldots \theta_N]^T \\
\Psi = \begin{pmatrix}
\psi_{11} & \cdots & \psi_{1N} \\
\vdots & \ddots & \vdots \\
\psi_{N1} & \cdots & \psi_{NN}
\end{pmatrix}
\]

(3.13)

The constraints of (3.10) can be written as:

\[
\mathbf{w}^T \psi''(t_f) = p_f - p_0 - (t_f - t_0)v_0 \\
\mathbf{w}^T \psi'(t_f) = v_f - v_0
\]

(3.14)

where

\[
\psi''(t_f) = \begin{pmatrix}
\psi''_1(t_f) \\
\vdots \\
\psi''_N(t_f)
\end{pmatrix} \quad \psi'(t_f) = \begin{pmatrix}
\psi'_1(t_f) \\
\vdots \\
\psi'_N(t_f)
\end{pmatrix}
\]

(3.15)

Now we can write (3.10) in standard QP form:

\[
\begin{aligned}
&\text{minimize} & & \frac{1}{2} \mathbf{w}^T \mathbf{\Sigma} \mathbf{w} + \tau^2 \mathbf{\theta}^T \mathbf{w} \\
&\text{subject to} & & \begin{pmatrix}
\psi''(t_f) \\
\psi'(t_f)
\end{pmatrix} \mathbf{w} = \begin{pmatrix}
p_f - p_0 - (t_f - t_0)v_0 \\
v_f - v_0
\end{pmatrix}
\end{aligned}
\]

(3.16)

These values can be used in a standard QP solver. For adapting to new targets only \(p_f, p_0, v_f, v_0\) and \(\Psi\) change in the resulting QP. The basis functions \(\Psi\) must be recomputed for changes in the final time. The \(\mathbf{\theta}\) vector, obtained from the original demonstration remains unchanged and does not have to be recalculated. This vector therefore is the invariant that defines the primitive.

The minimization must be solved each time we want to execute a new motor action. However existing QP solvers can find a solution very fast. If we are controlling an arm with many degree of freedoms (DOFs), solving for each individual DOF is an intrinsically parallel problem and a solution can be found in the same time as with only one primitive given there is an execution thread available for each DOF.

Additional constraints can also be added to ensure that joint accelerations, velocities and positions are bounded within safe physical limits. For a discretization of the time interval at instants \(t_i\) the following constrains can be added:

\[
\ddot{q}_{\text{min}} \leq \mathbf{w}^T \dot{\psi}(t_i) \leq \ddot{q}_{\text{max}} \\
\dot{q}_{\text{min}} \leq v_0 + \mathbf{w}^T \dot{\psi}'(t_i) \leq \dot{q}_{\text{max}} \\
q_{\text{min}} \leq p_0 + (t_i - t_0)v_0 + \mathbf{w}^T \ddot{\psi}''(t_i) \leq q_{\text{max}}
\]

(3.17)

Constraints can also be added to limit jerk:

\[
\dddot{q}_{\text{min}} \leq \mathbf{w}^T \psi(t_i) - \mathbf{w}^T \psi(t_{i-1}) \leq \dddot{q}_{\text{max}}
\]

(3.18)

However each of these limitations requires adding 2 constraints for each sample point and may increase the computation time. In our case the problem is still solved in the order of tenths of a second for trajectories with 1000 points. Some solvers allow the use of lazy constraints. We experimented
with a lazy constraint version running in the proprietary solver Gurobi. The solver introduces the constraints only when the current iteration's solution violates them. In this way most constraints do not have to be used reducing the computation for each iteration of the solver.

3.3 Summary

We have shown an alternative way to adapt movement primitives to new execution conditions. Expanding from modern developments on DMPs, we take a novel approach by posing the problem of imitation as a constrained global optimization. Our experiments show that our method maintains the flexibility of the DMP formulation (i.e. possibility to change initial and final positions, velocities, time) while adding important additional features (i.e. possibility to add limits and way-points during the whole trajectory). Moreover, the generated trajectories always show the traits of the original demonstration, by closely matching its acceleration profile. Other constraints can be easily added, such as limitation of the maximum jerk in the joints motion. Solving the QP requires an extra computation step with respect to the DMP formulation. However since the problem is formulated as a convex quadratic program for which fast solvers exist, the solution can be found sufficiently fast to be used even in tasks that require low reaction times.
Experimental Results

Contents

4.1 Experimental results ................................................. 30
4.2 Experimental Setup and Simulation ................................. 34
4.3 Robotic Platform ....................................................... 40
In this chapter we present the experimental results obtained throughout this dissertation. First the Dynamic Motion Primitives (DMPs) and Quadratic Program Motion Primitives (QPMPs) introduced in chapter 3 are evaluated for simple trajectories in one DOF. More specifically we show the possibilities provided by the constrained optimisation compared to traditional DMPs.

Next, we evaluate learning with Cost-regularized Kernel Regression (CrKR) and the Adaptive Non-Maximal Suppression (ANMS) sparsification scheme described in chapter 2. This sparsification method allows CrKR to continue after a number of samples were the regression becomes too expensive for online learning.

We describe the simulated experimental setup in section 4.2 and show the results for the learning methods introduced of the previous chapters. Finally, in section 4.3 we describe the state tracking system and the robotic platform.

4.1 Experimental results

4.1.1 Evaluation of QPMPs and DMPs

To test our proposed methods we generated a ball hitting movement, defined in joint space, with our 6 DOF arm to serve as a demonstration for both a DMP with polynomial shifting goal (the system proposed in [37]) and for our QP formulation. For these experiments use a python implementation with the default CVXOPT solver to optimize our QP. CVXOPT was preferred for it's ease of integration with our implementation, the choice of an optimal solver remains an open task. For both the DMP and the QP we use Euler integration with a 0.001s time step. After the sample that corresponds to the final time both the DMP and the QP accelerations are set to zero.

We performed three different experiments (Sections 4.1.2, 4.1.3 and 4.1.4) that consisted in adapting the demonstrated motion to different execution conditions, comparing our QP method to DMPs. Moreover, we discuss how some important features of the DMP approach are also preserved, namely the rejection of noise and disturbances during execution (Section 4.1.5). Finally, we discuss how both approaches deal with noise in the demonstration (Section 4.1.6). Each experiment shows one useful feature of our proposed method. For the sake of clarity we present these results in separated experiments, but in a real application all features can be used simultaneously. Due to space constraints in the plots we display only the trajectory of one joint of our 6 DOF arm, but results for the other joints are analogous. The results presented in this section have been published in [12].

4.1.2 Adaptation to different initial/final states and movement duration

Both the DMP and the QPMP allow adapting the demonstrated primitive to new initial and final states composed of position, velocity and time. As it can be seen in Fig. 4.1 the DMP and the QP generate similar trajectories. However, the QP distributes the acceleration difference evenly throughout the trajectory. This allows a more accurate replication of the demonstrated movement shape, especially near the end of the trajectory. The non-zero final accelerations may increase the velocity error at the final instant. When a precise final velocity is critical to the task a new constraint can be
added to switch off accelerations at a predetermined final fraction of the trajectory.

![Graph showing trajectory generated by adapting the demonstration to a different initial position and a different final velocity. The final time is set to be at 0.8s instead of the 1.0s of the demonstration.](image)

The movement generated by the global optimization (without additional constraints) imitates the shape of the demonstration better, especially in the final part of the motion.

### 4.1.3 Inclusion of joint limits

Taking into account the physical limits of the joints is of crucial importance in practical operations, as every robot has position, velocities and acceleration limits that cannot be violated. Moreover, in some specific cases there might be the need for setting limits that are more conservative than the maximum, nominal ones; therefore, it is extremely helpful to have the possibility of changing these limits on-the-fly, depending on the task requirements, without the overall shape of the movement being deviated too much from the demonstration. As shown in Fig. 4.2 with the DMP formulation the adaptation to new goals can generate trajectories that violate joint limits (even if the demonstration was bounded). Conversely, with our approach these limits can be included as constraints of the QP by adding inequalities in eq. 3.18. The results in Fig. 4.2 show that our proposed method can generate a trajectory that is feasible and still a good approximation of the original demonstration.
Figure 4.2: Trajectory generation with joint limits. Here the joint limits were added as constraints in the QP. The position has a physical limit at ±2 rad. To adapt to a new end goal ($v_f = 10 \text{rad.s}^{-1}$, $p_f = -10 \text{rad}$) the QP can take the limits and a safety margin into account to generate a good imitation while a DMP generates an infeasible trajectory.

4.1.4 Addition of intermediate way-points

The flexibility of the QP formulation permits adding new constraints to fit the specific demands of the task. An interesting possibility is adding intermediate way-points while trying to maintain the shape of the movement. The way-points can be added in the same way as the final goal but at an intermediate time; they can be defined as a specific position and velocity occurring at a desired time instant. Among many, one practical application could be to perform obstacle avoidance while keeping the demonstrated shape of the movement. Fig. 4.3 shows how the QP formulation allows to generate a trajectory with the same initial and final position of the demonstration, but with a different way-point in the middle; notably, the shape of the trajectory highly resembles the demonstrated one. Although a small discontinuity in the acceleration profile seems to occur near to the way-point, this can be mitigated by adding a jerk limiting constraint in the QP problem. Moreover, since the discontinuity happens only after the way-point and the rest of the movement is smooth, if way-points are sufficiently spaced in time the resulting trajectory is not jerky and can be executed with precision.
4.1.5 Rejection of noise and disturbances

The acceleration-based controller of the DMP (see eq. 3.11) is stable when following a polynomial goal [37]. In our case, we are following an arbitrary reference instead of a polynomial: after transients, the system acts as an all-pass filter and does not diverge when given bounded inputs.

We performed an experiment in which we introduced i) noise in the sensory measurements (positions and velocities) and ii) a step disturbance in the acceleration, which can simulate, for example, the presence of an unexpected load. The plots in Fig. 4.4 show how the controller is able to reject the noise and recover the desired trajectory when the disturbance vanishes.

4.1.6 Effect of noise in the demonstration

The demonstration of a movement usually consists of a position signal, which is then differentiated to obtain the acceleration profile. The noise in the position signal is amplified by differentiation and
may result in a very noisy acceleration profile unsuitable for imitation. DMPs use the demonstrated position and velocity in addition to the acceleration for learning the forcing term. This helps to mitigate the jerky motion caused by noise in the acceleration. Our QPMP system loses some of the high frequency noise content by representing the demonstrated accelerations as Gaussian Basis functions, however if the position signal is very noisy the stored demonstration will still have very high accelerations, being unsuitable for imitation. Preprocessing the demonstration offline through a smoothing filter generates a smooth acceleration profile suitable for imitation. We used a Savitzky-Golay filter \[43\] to smooth the position signal, achieving good results in the imitation (Fig. 4.5).

Figure 4.5: Differentiation of the position signal with smoothing by a Savitzky-Golay filter results in better imitation for both the QPMP and the DMP. To the demonstration’s position signal we added a noise of amplitude 0.01 \([\text{rad}]\) this will be amplified by the differentiation to 10 \([\text{rad/s}]\) in the velocity and to 10000 \([\text{rad/s}^2]\) in the acceleration. The QPMP and the DMP used the acceleration smoothed by the Savitzky-Golay filter. Filtering is very desirable for the DMP to avoid jerky motion, and necessary for any useful imitation with the QP.

4.2 Experimental Setup and Simulation

In order to test the algorithms described in this work we designed a robot table tennis experiment. The goal of this experiment is to make a serial manipulator hit a ping pong ball thrown by a human player. First, we implemented the table tennis setup in a Gazebo \[28\] simulation environment. Gazebo is a multi-robot simulator with dynamics simulation and advanced 3D graphics. The simulator has Plugs for sensor feedback and plausible interactions between objects. The Gazebo simulator integrates well with robotic middlewares such as ROS or YARP that abstract away the details of communication, allowing code to be shared between the simulated environment and the robotic platform. In our simulation we use a URDF model of the Biorob V3 X6 serial manipulator with the end-effector modified to include a table tennis paddle. The simulation also includes a model of a ping pong table and ball. The position of the ball is taken directly from the simulations internal variables. The architecture of the simulated platform is described in Fig. 4.6.

Despite specifying model for the inertial and surface parameters of the ball and table, the physics of the ping pong ball proved challenging to simulate accurately. As such we had to avoid collisions...
with the table and focus on the task of hitting a ball thrown without rebounds. Replacing the physics engine by another such as Simbody [44] may achieve a more accurate model of the collisions but this was not attempted in this work.

4.2.1 Simulation Architecture

In our Gazebo simulation all logical blocks were implemented as Robot Operating System (ROS) nodes with communication also provided by ROS. This increased the flexibility for choosing the most adequate libraries for each part of the system while still having seamless integration. For the optimization of QPMPs we used the Gurobi™ solver through the JUMP library [34]. Gurobi allows the use of cuts and lazy constraints to improve the performance of the trajectory generation for low reaction times. The solver can save computational resources by adding the joint limit constraints only when they are violated by the generated trajectory. For the Game state detector we used OpenRAVE [15] for direct and inverse kinematics of the robot. Whenever our robot model needed adjustments we could automatically convert the URDF model to a Collada model and regenerate the direct and inverse kinematics. OpenRAVE includes IKFast to find an extremely fast closed form inverse kinematics model. For our robot we generated a 5 dimensional inverse kinematics, thus encoding our problems invariant in the axis of the normal to the racket surface. In the final system the inverse kinematics are learned together with the task by the CrKR. However the analytical solution was still useful for preliminary experiments were we tried to learn in Cartesian space instead of the joints space.

![Diagram of Simulation Architecture](image)

**Figure 4.6:** Using the ROS robotics middleware increased the flexibility of our software architecture. Each logical module could be implemented with the most convenient set of programming language/tools as a separate process with communication provided by ROS topics. In our simulation the robot, game and vision system are replaced with Gazebo. The grey boxes represent separate processes implemented as ROS nodes, and the yellow notes represent prior information collected from demonstrations. Once the ball crosses a set section of the table, while traveling towards the robot, the game state detector sends the ball state including velocity in the robot referential to the CrKR + ANMS node. This state is used by the CrKR + ANMS node to predict a new goal for the manipulator based on previous trials. The QPMP IL then uses the goal to generate the new trajectory based on the demonstration primitive. The Game state detector evaluates the cost of the trial and updates CrKR.
4.2.2 Reinforcement Learning

To learn the mapping between the state of the incoming ball and the goal for our robotic manipulator we applied CrKR and the ANMS filtering described in chapter 2. In this case the input to the system is the measured state of the ball’s position and velocity in Cartesian space. The output is the position and velocity of each joint of the robotic manipulator at a desired hit time. This goal can then be passed to a trajectory generation algorithm that handles the creation of an appropriate trajectory to reach the desired manipulator state. This mapping can be seen in Fig. 4.7.

\[ \text{Ball State} \quad s_j = \begin{bmatrix} x_j \\ y_j \\ z_j \\ \dot{x}_j \\ \dot{y}_j \\ \dot{z}_j \end{bmatrix} \]

\[ \text{Strike State} \quad \gamma_j = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \\ \theta_6 \\ \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \\ \dot{\theta}_4 \\ \dot{\theta}_5 \\ \dot{\theta}_6 \\ t_{\text{hit}} \end{bmatrix} \]

**Figure 4.7**: Through RL we try to learn a mapping from the measured state of the ball (with Cartesian position and velocity) to a desired position and velocity of the robot in joint space and a desired time. Our robot has 6 joints, so we are trying to learn a function that is \( \mathbb{R}^6 \rightarrow \mathbb{R}^{13} \).

During the trials the state of the ball is continuously tracked until it crosses a set plane parallel to the net of the ping pong table. At this point, if the ball is moving towards the robot it is considered as a new trial and used to estimate the new target for the robot. During the execution of the swing the game state tracker stores information to evaluate the cost of the swing, storing:

- The ball state at the goal time \( t_{\text{hit}} \).
- The Cartesian position of the paddle. This was computed by applying the forward kinematics to the state of the joints at the goal time \( t_{\text{hit}} \).
- The state of the ball in the next bounce from the table after time \( t_{\text{hit}} \).

The swing is classified as successful if the ball, after time \( t_{\text{hit}} \), bounces in the table and is traveling away from the robot. Only the successful trials were used as datapoints to update the regression. The position of the ball and of the paddle at time \( t_{\text{hit}} \) were used to compute the cost \( c_j \) associated with the trial, according to the saturated cost function in equation 4.1.

\[
 c(x) = 1 - \exp\left( -\frac{1}{2a^2} \| x_{\text{ball},t_{\text{hit}}} - x_{\text{paddle},t_{\text{hit}}} \|^2 \right) 
\] (4.1)

The sequence of steps used for exploration learning during a trial can be seen in Fig. 4.8.

4.2.2.A Optimization of the Kernel Parameters

The SE-ARD Kernel parameters were found by running a gradient method optimizer on a set of data points obtained by running the exploration with a handpicked parametrization. The state of the
ball that is used as input to our regression is always collected when the ball crossed the same section, $y = y_{\text{const}}$, parallel to the net. Since the $y$ position is constant the bandwidth $l_y$ does not affect the regression and was not optimized. For a better fit we used a separate SE-ARD kernel for each of the six joints position and velocity and another for the time. This results in 91 optimized parameters that can be seen in table 4.1.

Table 4.1: Optimized SE-ARD Kernel Hyperparameters for the simulation.

<table>
<thead>
<tr>
<th></th>
<th>$\log(\sigma)$</th>
<th>$\log(l_x)$</th>
<th>$\log(l_y)$</th>
<th>$\log(l_z)$</th>
<th>$\log(l_\dot{x})$</th>
<th>$\log(l_\dot{y})$</th>
<th>$\log(l_\dot{z})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_2$</td>
<td>2.1273</td>
<td>1.2685</td>
<td>9.0250</td>
<td>7.6990</td>
<td>2.4929</td>
<td>15.257</td>
<td>2.4338</td>
</tr>
<tr>
<td>$q_3$</td>
<td>1.6188</td>
<td>1.7495</td>
<td>9.0250</td>
<td>4.4027</td>
<td>3.2870</td>
<td>16.080</td>
<td>2.3113</td>
</tr>
<tr>
<td>$q_4$</td>
<td>2.8038</td>
<td>2.4539</td>
<td>9.0250</td>
<td>2.4958</td>
<td>2.4279</td>
<td>4.9273</td>
<td>1.6993</td>
</tr>
<tr>
<td>$q_5$</td>
<td>2.3789</td>
<td>0.9398</td>
<td>9.0250</td>
<td>3.3529</td>
<td>2.0231</td>
<td>5.7367</td>
<td>1.8172</td>
</tr>
<tr>
<td>$q_6$</td>
<td>0.4579</td>
<td>1.2671</td>
<td>9.0250</td>
<td>1.6557</td>
<td>0.5183</td>
<td>0.9754</td>
<td>0.5670</td>
</tr>
<tr>
<td>$\dot{q}_1$</td>
<td>0.3506</td>
<td>0.6660</td>
<td>9.0250</td>
<td>1.1826</td>
<td>1.0494</td>
<td>1.4521</td>
<td>1.0937</td>
</tr>
<tr>
<td>$\dot{q}_2$</td>
<td>1.2185</td>
<td>0.9375</td>
<td>9.0250</td>
<td>0.4983</td>
<td>0.8269</td>
<td>1.2930</td>
<td>0.3462</td>
</tr>
<tr>
<td>$\dot{q}_3$</td>
<td>4.5015</td>
<td>1.3626</td>
<td>9.0250</td>
<td>1.1300</td>
<td>1.0822</td>
<td>4.2214</td>
<td>0.9110</td>
</tr>
<tr>
<td>$\dot{q}_4$</td>
<td>7.1339</td>
<td>1.2509</td>
<td>9.0250</td>
<td>1.0626</td>
<td>0.9950</td>
<td>2.8507</td>
<td>3.9328</td>
</tr>
<tr>
<td>$\dot{q}_5$</td>
<td>8.8359</td>
<td>11.941</td>
<td>9.0250</td>
<td>1.7504</td>
<td>2.1314</td>
<td>3.1324</td>
<td>7.7091</td>
</tr>
<tr>
<td>$\dot{q}_6$</td>
<td>10.440</td>
<td>6.1633</td>
<td>9.0250</td>
<td>2.7898</td>
<td>3.2118</td>
<td>4.9104</td>
<td>11.826</td>
</tr>
</tbody>
</table>
4.2.3 Imitation Learning

Since our simulation model matches the robotic platform a trajectory was captured by kinesthetic teaching to serve as basis for the motion primitive. The trajectories were then calculated using QPMP independently for each DOF. The robot joint limits were added to the optimization as lazy constraints to add safety to the trajectory execution without significantly increasing the computation time for most trajectories. The solver will fail to generate the trajectory if it is not feasible due to the constraints. In these situations no trajectory is executed and the trial is discarded.

![Diagram of robot workspace](image)

**Figure 4.9:** A representation of the workspace of the robot attained by sampling random configurations with OpenRAVE. In the left we can see the complete workspace of the robot while on the right the table plane causes a significant reduction in the number of attainable configurations of the end effector.

The ping pong table acts as a ground plane that significantly limits the configurations that the serial manipulator can achieve without collision (Fig. 4.9). To ensure that no collision occurs with the table we sample the forward kinematics of the generated trajectory using OpenRAVE. The table was included in the robot model so that any situation where the arm would collide with the table is detected internally as a self-collision.

4.2.4 Results

The simulation was run starting with an initial set of five demonstrations and a motion primitive. The system throws the ball starting with position and velocity sampled from a uniform distribution that was tuned to generate trajectories lying mostly within the bounds of the robots reachability. After each training epoch of 50 samples, the strength of the predictions is verified by running a fixed test set of 50 trials that have the same distribution as the training samples. The costs are computed with the saturated cost function of eq. 4.1. Only the successful hits are used for training with success defined as bouncing in the table while moving away from the robot.

The results obtained can be seen in Fig. 4.10. We achieved a success rate of 78% according to the criteria previously described. It should be noted that this rate is dependent on the trajectory of the balls thrown to the robot, some trajectories in the test set may be unreachable for the serial manipulator so a 100% success rate is not possible. The learned policy can be visualized by variating one of the input state dimensions and plotting the resulting policy in joint space (Fig. 4.11) or in Cartesian space by calculating the forward kinematics (Fig. 4.12). Another important consideration is...
**Figure 4.10:** The cost and success rate by epoch while learning in the Gazebo simulator for different $\lambda$. After each training epoch of 50 random samples a evaluation is performed on a fixed test set of 50 trials. A trial is considered successful if the ball is hit and bounces the table while moving away from the robot. The cost is calculated by the expression 2.8. The plots are averaged over 5 different runs for each value of the exploration/exploitation trade-off $\lambda$.

That due to timings and the physics of the simulation the same ball, thrown with the same initial state, will have some variability in the trajectory executed. This makes the test set have some inherent variability itself. Consequently the test set evaluated twice with the same state of the system, at the same stage of learning, will still result in different costs and success rates.

**Figure 4.11:** In the upper plot we have the mean of the learned joint positions and velocities of the serial manipulator with respect to the ball state variation in the $x$ axis. The other state variables of the ball are kept fixed. These positions can also be seen plotted in Cartesian space in Fig. 4.12. In the lower plot we see the corresponding variation in the estimated hit time.
4.3 Robotic Platform

For the experiments with the real robot the state of the ball cannot be measured directly like in the simulation. To track the ball position we used a vision system based on Motion capture (Mo-cap) cameras. The ball position captured by the vision system must then be filtered to get an accurate state estimation that includes velocity. This resulting state is transformed to the referential of the robot and sent in the same way as in the simulated environment described in section 4.2.1.

4.3.1 Implementation in the Robotic Platform

For the robotic platform both the serial manipulator and the Mo-cap system had different software requirements. We used a dedicated computer for the Mo-cap and another for the robot control system, each running a specialized operating system (MS Windows™ for the Mo-cap and a real-time Linux kernel for motor control). The resulting integrated robotic platform is a distributed system with three networked computers. The Mo-cap computer streams the measurements in a proprietary protocol to a main Linux computer that translates to a ROS topic. The remaining communication is all done within ROS see Fig. 4.13.

4.3.2 Vision System for State Tracking

To find the ball’s state we evaluated two alternative systems. First we experimented with an Xtion Live Pro™ depth camera in a simulated environment. The image was segmented by color in the HSV color space. HSV is a color space widely used for segmentation in computer vision since it separates color information from intensity or lighting. After segmentation the depth values corresponding to the selected pixels were fit to a sphere using the Chernov’s Hyperfit algorithm [1], that showed better accuracy than RANSAC. These preliminary experiments were performed in a simulated environment so
they lack validation, nonetheless the Hyperfit algorithm showed promise for real time sphere tracking applications.

Our alternative setup used a Optitrack Flex 13™ Mo-cap system. This vision system consists of six infrared cameras with led illumination. The cameras detect groups of retro-reflective spherical markers for measuring the pose of objects. In the end we decided to choose this system for the robotic platform since it has a higher frame rate (120 FPS) and the accuracy was estimated to be 3mm according to the calibration software.

The Mo-cap software is designed so that groups of three or more markers can be associated for estimation of the pose of an object. In our case since we want to track a ping pong ball we needed to make the ball itself retro-reflective in order to be detected. This was achieved by covering the ball in retro-reflective tape, that was cut using a planar projection. To get the ball completely covered in retro-reflective tape without overlapping (which would dampen the bouncing and increase air drag) it was necessary to find a suitable planar projection. To this effect we decided to use a sinusoidal projection with 24 segments (Fig. 4.14) as suggested in Kulchenko and Todorov [31].

The robots referential is used in simulation and in the robotic platform as the inertial frame of reference. In order to increase robustness to changes in the position of the robot relative to the
cameras, we attached three markers the robot arm, so that the pose relative to the cameras is known, and all referential transforms can be dynamically updated.

To find the state of the ball with position and velocity we achieved better results with an EKF instead of a Linear Kalman Filter (LKF), due to the presence of nonlinear air drag. The EKF was implemented as suggested in [38], to which we added outlier rejection with a Mahalanobis distance validation gate. Ideally a realistic filter would need to model the aerodynamic drag, the physics of the bounces on the table and the spin. In our system it would be hard to measure the spin of the ball so it is unmodeled.

The EKF has the prediction step:

\[
\hat{s}_{k | k-1} = f(\hat{s}_{k-1 | k-1}, u_k) \\
P_{k | k-1} = F_k P_{k-1 | k-1} F_k^T + Q_k
\]

where \( \hat{s} \) is the state of the ball (\( \hat{s} = [s \ s' \ s''] \)), \( P \) is the predicted covariance, and \( Q \) is the observation error covariance. The control vector \( u \) is not used in our case.

The filter’s update step is:

\[
\tilde{y}_k = z_k - h(\hat{s}_{k | k-1}) \\
S_k = H_k P_{k | k-1} H_k^T + R_k \\
K_k = P_{k | k-1} H_k^T S_k^{-1} \\
\hat{s}_{k | k} = \hat{s}_{k | k-1} + K_k \tilde{y}_k \\
P_{k | k} = (I - K_k H_k) P_{k | k-1}
\]

Where \( \tilde{y} \) is the measurement residual, \( S \) is the residual covariance, \( K \) is the Kalman gain, \( \hat{s} \) is the updated state estimate, and \( P \) is the updated covariance estimate.

In these equations we set the discrete transition function \( f(\hat{s}_{k-1 | k-1}, u_k) \) to the following model obtained through sympletic Euler integration:

\[
\dot{s}_k = g - \frac{1}{2m} c_{\text{drag}} \rho_{\text{air}} S_{\text{section}} \|\dot{s}_{k-1}\| \dot{s}_{k-1} \\
\ddot{s}_k = \ddot{s}_{k-1} + \delta_t \dot{s}_k \\
s_k = s_{k-1} + \delta_t s_k
\]

Where \( g \) is the acceleration of gravity in the z axis, \( \delta_t \) denotes the time step, \( m \) is the Ping Pong ball mass (2.7 grams), \( \rho_{\text{air}} \) is the density of the air, \( S_{\text{section}} \) is the cross sectional area of the ball (\( \pi(0.02m)^2 \)), and \( c_{\text{drag}} \) is the drag coefficient (0.47 for a sphere).

Together with the observation function, in our case \( h(\hat{s}_{k | k-1}) = [s_{k | k-1} \ 0 \ 0] \) since only the ball’s position can be measured. We can find the \( F \) state transition and \( H \) observation matrices by computing the Jacobians:

\[
F_{k-1} = \frac{\partial f}{\partial s} \bigg|_{\hat{s}_{k-1 | k-1}, u_k} \\
H_k = \frac{\partial h}{\partial s} \bigg|_{\hat{s}_{k | k-1}}
\]

In each update step a small (order of \( 10^{-9} \)) amount of stabilizing noise is added to the measurements to prevent the covariance matrix from becoming statistically inconsistent.
4.3.3 Mahalanobis Distance Validation Gate

The motion capture system will occasionally detect false positives. This is due to the Mo-cap system being designed to detect collections of markers instead of single ones. In addition the surface of the table is slightly reflective confusing the system. To filter out samples that make no sense we use a validation gate that makes the EKF forgo the update step and do only the prediction. The validation gate calculates the Mahalanobis distance \( \delta_{mhln} \) and excludes the sample if it's distance is larger than a set threshold. A counter for the rejected samples is used, if too many samples are rejected in sequence there may be a discontinuity in the tracking, so we reset the filter and restart the tracking.

\[
\delta_{mhln} = y^T S^{-1} y
\] (4.6)

4.3.4 Ground Collisions

When the prediction collides with the table we multiply the predicted velocity by an estimated restitution coefficient \( C_R \) in eq. 4.7. This will approximate the collision if the time step is small. The results of tracking the ball on the robotic platform with the EKF filter and the additions we described can be seen in Fig. 4.15.

\[
\dot{s}_k = -C_R \dot{s}_{k-1}
\] (4.7)

**Figure 4.15:** Plot showing the EKF predictions based on collected samples of the ping pong ball bouncing on the table. The blue dots are samples of the ball position captured by the Optitrack system at a rate of 120 FPS. The green triangles show the prediction for the ball position 0.5[s] ahead. The initial predictions are off while the filter is initialized. The missing samples are samples rejected by the Mahalanobis distance validation gate.

4.3.5 Experiments on the Robotic Platform

Our robotic platform has a Biorob serial manipulator set on top of a table with three attached spherical markers for pose tracking. The measured pose of the serial manipulator is used to translate the ball position from the camera’s referential to the robot referential, automatically adjusting for any alterations in the robot location. To the end effector of the serial manipulator, we attached a 3D printed
paddle. This paddle was printed in ABS plastic with perforations to reduce air drag and decrease weight. This setup can be seen in Fig. 4.16.

Figure 4.16: The real setup showing the Biorob arm and the six cameras of the Optitrack Mo-cap system. Three markers are attached to the basis of the robot to automatically translate to the robot referential when the robot changes position relative to the cameras. In the upper right corner we can see the 3D printed paddle. The paddle does not need markers since its position can be found using the robot state and its forward kinematics.

With this robotic platform we performed preliminary experiments with low speed trajectories learned from demonstrations with QPMP. We tested the tracking and automatic referential adjustment resulting in the plot seen in Fig. 4.15. At this point we are in the process of refining control for QPMP with higher speed trajectories to reach the targets as pictured in Fig. 4.17 and to validate the results obtained in simulation. Videos and material related to these results can be seen in [11].

Figure 4.17: A schematic of the execution a trajectory in the robotic platform in response to an incoming ball. First the marker’s position and robot pose are captured by the Mo-cap system. This pose and position are sent to a translation ROS node that together with an EKF node send the ball’s state in the robot referential to the learning systems. The learning process occurs after the trial and was previously detailed in Fig. 4.22.
Conclusions and Future Work
In this work we explored methods for both imitation learning and active learning of motor tasks. Expanding from the known methods used for Reinforcement Learning (RL) in this class of robotics problems, we proposed the addition of a sparsification method based on Adaptive Non-Maximal Suppression (ANMS). This sparsification capable of choosing online a low cost and representative subset of the collected samples. Using this subset for regression allows the robot to continue learning past a set budget of samples beyond which the matrix inversion becomes too computationally expensive. Also taking modern developments in DMPs as a starting point, we take a novel approach by posing the problem of imitation as a constrained global optimization. Our experiments show that our QPMP method maintains the flexibility of the DMP formulation (i.e. adaptation to different start and end positions, velocities and time) while adding important additional constraints on the trajectory such as the joint limits of the robot and intermediate way-points.

Future work will focus first on validating the results in our robotic platform. Our ANMS filtering increases the amount of samples that can be collected, however the regression uses only a limited subset. Approximate inference methods can make use of all data. This is not always optimal since the samples are collected from random exploration and the task may be multi-modal. An interesting premise for further investigation would be to use a filter with an additional distance metric in the output space, in order to select samples that are relevant to each other. This could be combined with approximate methods to allow a larger number of samples for regression. To further develop our QPMP framework we can look to the capabilities added to traditional DMPs over the last decade. These include the online adaptation to moving goals, mixtures of primitives and the use of RL techniques to also improve the trajectory shape.
Bibliography


