TennisCopter

Towards Table Tennis with a Quadrotor Autonomous Robot and Onboard Vision

Rui Manuel Ramos Teixeira da Silva

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Francisco António Chaves Saraiva de Melo

Examination Committee

Chairperson: Prof. José Carlos Martins Delgado
Supervisor: Prof. Francisco António Chaves Saraiva de Melo
Member of the Committee: Prof. Artur Miguel do Amaral Arsénio

October 2014
It always seems impossible until it’s done.

Nelson Mandela
Acknowledgments

First of all, I would like to express my gratitude to Professor Francisco Melo for his patience and for all the inspiration and constant guidance offered throughout this journey. It was a great honor having him as my supervisor.

I must also thank my parents. Their never-ending support and confidence in me made it all possible.

Finally, I have to acknowledge every person that positively marked my time at Instituto Superior Técnico.
Abstract

Robot table tennis is a challenging domain in robotics, artificial intelligence and machine learning. In terms of robotics, it requires fast and reliable perception and control; in terms of artificial intelligence, it requires fast decision making to determine the best motion to hit the ball; in terms of machine learning, it requires the ability to accurately estimate where and when the ball will be so that it can be hit. The use of sophisticated perception (relying, for example, in multi-camera vision systems) and state-of-the-art robot manipulators significantly alleviates concerns with perception and control, leaving room for the exploration of novel approaches that focus on estimating where, when and how to hit the ball. In this work, we move away from the hardware setup commonly used in this domain—typically relying on robotic manipulators combined with an array of multiple fixed cameras—and address the problem of robotic table tennis using a quadrotor drone with a light cardboard racket and an onboard camera. We adapt a recently proposed framework for learning complex robot tasks to our setup and show that, in spite of the perceptual and actuation limitations of our system, the overall approach enables the quadrotor system to successfully respond to balls served by a human user.

Keywords

Robotics, Table Tennis, Quadrotor, Motor Primitives, Imitation Learning, Reinforcement Learning
Resumo

O ténis de mesa robótico é um domínio complexo não só para a robótica, mas também para a inteligência artificial e machine learning. Em termos de robótica exige percepção e controlo rápidos e fiáveis; em termos de inteligência artificial requer tomada de decisão rápida no que toca à escolha do melhor movimento para acertar na bola; em termos de machine learning exige a capacidade de estimar de forma precisa a posição da bola num dado momento, para que esta possa ser batida. A utilização de sistemas de percepção sofisticados (baseados, por exemplo, em sistemas de visão com múltiplas câmaras) e manipuladores robóticos modernos, simplifica significativamente as questões de percepção e de controlo, passando a haver espaço para explorar novas abordagens focadas na estimativa de como, onde e quando acertar na bola. Neste trabalho afastamo-nos do hardware normalmente utilizado neste domínio—manipuladores robóticos combinados com um conjunto de câmaras fixas—endereçando o problema do ténis de mesa robótico usando um quadricóptero, que transporta uma câmara e uma raquete feita de cartão leve. Adaptámos ao nosso sistema uma abordagem proposta recentemente para a aprendizagem de tarefas robóticas complexas, e mostramos que apesar das limitações de percepção e actuação do sistema robótico usado, a abordagem permite que o quadricóptero consiga responder com sucesso a bolas lançadas por um humano.

Palavras Chave

Robótica, Ténis de Mesa, Quadricóptero, Primitivas Motoras, Aprendizagem por Imitação, Aprendizagem por Reforço
# Contents

1 Introduction .............................................. 1
   1.1 Motivation ........................................... 3
   1.2 Problem Description .................................. 4
   1.3 Contributions ........................................ 4
   1.4 Solution Outline ..................................... 4
   1.5 Thesis Outline ....................................... 6

2 Related Work ............................................ 7
   2.1 Historical Perspective on Robotic Table Tennis .......... 9
   2.2 Robotic Table Tennis .................................. 10
   2.3 Flight Control ........................................ 14

3 Background .............................................. 19
   3.1 Parrot AR.Drone 2.0 .................................. 21
   3.2 Quadrotor Dynamics ................................... 21
   3.3 Pinhole camera model .................................. 22

4 Solution .................................................. 25
   4.1 Software Architecture .................................. 27
      4.1.1 Logical Architecture ................................. 27
      4.1.2 Physical Architecture ............................... 29
   4.2 Learning Architecture .................................. 31
      4.2.1 Imitation Learning .................................. 31
      4.2.2 Generalization of Movements ....................... 32
   4.3 Computer Vision ........................................ 32
      4.3.1 Table Tennis Ball Detection and Tracking .......... 33
      4.3.2 Table Tennis Ball Pose Estimation ................. 34
   4.4 Imitation Learning ...................................... 38
      4.4.1 Trajectory Representation ......................... 38
      4.4.2 Acquiring and Initializing Initial Knowledge ....... 41
   4.5 Generalization of Movements ............................ 44
      4.5.1 Learning World Dynamics ........................... 45
List of Figures

1.1 Photo of a Parrot AR.Drone 2.0 ......................................................... 3
1.2 Illustration of the functioning of the motor primitives gating network .......... 5
1.3 General view of the solution outline ................................................. 6
2.1 Three of the first robotic ping pong players ........................................ 9
2.2 Acosta el al. setup playing against a human ........................................ 11
2.3 Design of Myazaki et al. setup, indicating the existing DoFs .................... 12
2.4 Design of Lai and Tsay setup ............................................................ 12
2.5 A group of quadrotors in formation .................................................. 16
2.6 A quadrotor passing a pole to another one ......................................... 17
3.1 Quadrotor dynamics ................................................................. 21
3.2 Example of the pinhole camera model .............................................. 22
3.3 Camera coordinates system compared with image coordinates system ........ 23
4.1 Logical architecture of the system .................................................. 27
4.2 Physical architecture of the system .................................................. 29
4.3 Learning architecture of the system ................................................. 31
4.4 Example of how color segmentation works ....................................... 33
4.5 Example of ping pong balls colors in RGB and HSV ............................ 33
4.6 Example of an image in both RGB and HSV formats ............................. 34
4.7 Example of a table tennis ball being detected and tracked in real-time ........ 35
4.8 Ground truth and predicted trajectories and velocities of the ball .............. 38
4.9 DMP with different duration, and final position and velocity ................... 42
4.10 Mobile application for controlling a Parrot ArDrone ............................ 43
4.11 Keys used in the keyboard controller ............................................. 43
4.12 Trajectory execution performance ................................................... 45
4.13 Ridge factor performance ............................................................. 48
4.14 Example of two possible game situations ........................................ 49
5.1 Reward function example ............................................................ 55
5.2 Performance of single motor primitives .......................................... 57
List of Tables

2.1 Overview of existing robotic table tennis systems .............................................. 15

4.1 Ball position estimation algorithm results on the Parrot Ar.Drone. ...................... 36

4.2 Ball pose estimation algorithm results on simulation environment ..................... 37

4.3 Trajectory execution controller performance on a simulated environment .......... 44
## Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CrKR</td>
<td>Cost-regularized Kernel Regression</td>
</tr>
<tr>
<td>DDP</td>
<td>Differential Dynamic Programming</td>
</tr>
<tr>
<td>DMP</td>
<td>Dynamical system Motor Primitives</td>
</tr>
<tr>
<td>DoF</td>
<td>Degree of Freedom</td>
</tr>
<tr>
<td>LBR</td>
<td>Linear Bayesian Regression</td>
</tr>
<tr>
<td>LWR</td>
<td>Locally Weighted Regression</td>
</tr>
<tr>
<td>MoMP</td>
<td>Mixture of Motor Primitives</td>
</tr>
</tbody>
</table>
1 Introduction

Contents

1.1 Motivation ......................................................... 3
1.2 Problem Description ........................................... 4
1.3 Contributions .................................................... 4
1.4 Solution Outline .................................................. 4
1.5 Thesis Outline ..................................................... 6
1.1 Motivation

Table tennis is known as a complex task for robotic systems, as it requires accurate control and reliable perception in real time. In particular, a robot that is able to play table tennis with some level of competence must successfully address

(P) Perception, \textit{i.e.}, reliably detect the position and velocity of the table tennis ball, once it is hit by the opponent;

(E) Estimation/decision, \textit{i.e.}, reliably predict the position in which the robot will hit the ball back to the opponent’s court (hitting point) and select the best motion to hit it (hitting motion);

(C) Control, \textit{i.e.}, reliably execute the hitting motion.

All issues above (perception, estimation and control) must be addressed in real-time, to ensure that the robot reaches the desired hitting point before the ball does.

Pioneer works on robot table tennis date back to 1983. The approaches proposed at that time focused mostly on the design of real-time robust perception and control (issues (P) and (C) above), and relied heavily on human knowledge in the form of explicit models of the task. This was due to the limited computational power available at the time.

Recent years witnessed a revived interest in robot table tennis. Unlike the pioneer works in the 80s, the latest approaches are able to take advantage of the much improved hardware and software, and focus on issues of generalization and adaptation. A number of researchers have proposed robot table tennis as an attractive domain on which to investigate the application of machine learning to robotics \cite{1-3}. In terms of the issues previously identified, machine learning techniques have been used mainly to address the problem of hitting point estimation and hitting motion selection, corresponding to issue (E) above.

In this work, we consider in detail the general framework of Mülling et al. \cite{4}. This framework has been tested on setups comprising by WAM\textsuperscript{TM} robotic arms, with joint level robust controllers, and by expensive multi-camera vision systems. The resulting system has proved able to return balls with success rates higher than 94%. We revisit the robotic table tennis problem and investigate the general applicability of this framework. In particular, we move away from the hardware setup used in \cite{4} and address the problem of robotic table tennis using a quadrotor drone (the Parrot AR.Drone 2.0) equipped with a light cardboard racket and an onboard camera. Mülling’s framework was adapted to the constraints imposed by our robotic setup (namely, the use of a single onboard camera as the only sensor, and the much slower response times) and we show that the overall approach is able to partly mitigate the perceptual and actuation limitations of the robot, while requiring minimum domain knowledge to be explicitly provided to the system.

\begin{figure}[h]
\centering
\includegraphics[width=0.4\textwidth]{figure1.jpg}
\caption{Photo of a Parrot AR.Drone 2.0. Picture taken from the official website (http://ardrone2.parrot.com/).}
\end{figure}
1.2 Problem Description

This work investigates the general applicability of a framework, which presented good results on a setup composed by both a highly accurate fixed manipulator and vision system, to the case where:

- A low-cost aerial robot is used.
- The computer vision system is composed by a single moving camera attached to the robot.
- The system should avoid the use of hand-coded knowledge about the world dynamics.
- The quadcopter is controlled by a commodity computer, where the commands are sent by a traditional WiFi connection.

New interesting and challenging problems arise from these new conditions. The dynamics of a quadcopter are completely different from those of a fixed manipulator, with the control being less precise and the sensors significantly less informative. The latter includes the computer vision system, consisting of only one moving camera, which makes the ball pose estimation process much harder. All these difficulties add considerable complexity to the learning process.

1.3 Contributions

The main contribution of this work consists on the adaptation of a framework for learning complex robot tasks, previously tested on robotic systems with accurate control and vision, to the case where an aerial robot with considerable perceptual and actuation limitations is used. Despite those limitations, we deployed the framework in both a simulation environment and a real Parrot AR.Drone 2.0, reaching hitting rates of over 30% and 20%, respectively. Thus, our low-cost aerial robotic system with onboard vision, and controlled by a commodity computer, is capable of playing rudimentary table tennis. By rudimentary table tennis we mean that our robot is able to hit a ping pong ball thrown at it, respecting some constraints regarding initial positions and velocities.

This main contribution led to two by-products. The first one includes a software and physical architectures that respect three classes of requirements:

1. Performance
2. Modifiability/Extensibility
3. Testability

These architectures are relevant contributions as they lead to a system with satisfactory performance, while being easy to test and able to accommodate future changes.

The second one is the development of a simulation system for the table tennis task using the quadcopter as a player. This simulation system adopted a widely used simulator, Gazebo, and its physics engine. By adopting a long-established simulator we increased the maintainability of the system, since more people are capable of modifying it.

1.4 Solution Outline

As discussed, this work studies how a state-of-art approach for robotic table tennis using fixed manipulators [4] can be mapped to the case where the player is a low-cost quadcopter, with a computer vision system comprising a single moving camera, controlled by a commodity computer.
This approach is inspired on how humans adapt to changing environments. Humans can perform complex motor tasks in new environments by generalizing their current motor abilities [4]. This adaptability has been extensively studied and it has become generally accepted that motor skill learning in robots should mimic humans in the sense that they should decompose complex motor tasks into smaller sub-tasks [4]. These subtasks are often referred as movement primitives—sequences of motor commands that accomplish a certain task [5].

The state-of-art approaches for motor primitives representation rely on Dynamical system Motor Primitives (DMPs), as they are more robust against environment perturbations and more advantageous when combined with learning processes [6].

In order to collect those motor primitives, imitation learning is used, as it is the most natural way to gather initial knowledge for quadcopter trajectories [7][8]. Imitation learning consists on having a human expert controlling the robot for a given environment situation [9]. After the imitation learning step, each motor primitive is stored in a library alongside with a set of parameters that encode relevant information concerning the hitting motion. We refer to this set of parameters as an augmented state that includes:

1. The current state, corresponding the ball pose when it is at 2.5 meters from the quadcopter.
2. A meta-parameter that contains information of where, when and how the ball is hit.

In order to hit new incoming balls, the system must be able to estimate the new meta-parameter, given the current state that its perceptual system provides. Once the meta-parameter is computed, the system selects the hitting motion to be performed using an episodic reinforcement learning algorithm—Cost regularized Kernel Regression (CrKR).

Having the information of where, when and how to hit the ball, the system must finally produce the trajectory the quadcopter should follow. This trajectory is created by generalizing the motor primitives stored in the library. The generalization is done through a Mixture of Motor Primitives (MoMP) algorithm.

This algorithm uses a gating network to activate each motor primitive, weighting it according to the two following aspects:

1. The similarity between the current state and the state observed during the learning of that motor primitive.
2. The estimated success of the motor primitive in the current state.

An illustration of the gating network can be seen in Figure 1.2.

![Figure 1.2](image_url)

**Figure 1.2**: Illustration of the functioning of the motor primitives gating network. Adapted from [4].
It is important to notice that there are two distinct learning processes. The first occurs offline, by using imitation learning to initialize the library of motor primitives. The second happens online by throwing ping pong balls at the player and letting it try to hit them. The player learns as it hits new balls. This second learning process will be called generalization of movements and can be further divided in two subprocesses:

1. Improvement of the policy that maps states to hitting motions.
2. Improvement of the weights associated with the gating network.

Figure 1.3 provides a general view of the whole process just discussed.

1.5 Thesis Outline

In the remainder of this document we will proceed as follows. First, in Chapter 2 we provide an overview of existing literature that is relevant for the research presented. Chapter 3 presents background knowledge on quadrotor dynamics and computer vision, necessary to understand the rest of the document. Chapter 4 describes in detail all the components of the solution. Chapter 5 contains the experimental evaluation. Then, Chapter 6 concludes the document while referring possible future work. Appendix A provides information regarding the simulation environment used.
Related Work

Contents

2.1 Historical Perspective on Robotic Table Tennis ................. 9
2.2 Robotic Table Tennis ........................................ 10
2.3 Flight Control ................................................ 14
In this chapter we review literature relevant to our own research. Despite being a recent research field, there is already a considerable number of existing approaches and systems for robotic table tennis. We start with an historical perspective, presenting some of the first works developed in the area, and proceed by reviewing other related work that will play a role in our research.

2.1 Historical Perspective on Robotic Table Tennis

Early approaches to robot table tennis go back to 1983, with the robot ping pong competitions launched by Billingsley, which underwent a special set of rules [10]:

1. The table was only 0.5 meters wide and 2 meters long.
2. The net was 0.25 meters high.
3. The maximum ball speed was limited to 10 meters/second.

These competitions prompted significant research into this problem. As a result, several systems were presented in the following ten years, by Knight and Lowery in 1986 [11]; Hartley in 1987 [12]; Hashimoto et al. in 1987 [13]; Andersson in 1988 [14]; Fässler et al. in 1990 [15].

The major bottleneck and difficulty at the time was the lack of fast real-time vision. Therefore, during this period the focus was mainly on the design of efficient vision systems and improved hardware, rather than on the development of algorithms and techniques that could elevate the robot playing skills to human-comparable ones.

(a) (b) (c)

Figure 2.1: Three of the first robotic ping pong players: (a) Design of the Knight and Lowery’s setup (image taken from [11]). (b) Photo from Hartley’s ping pong robotic player (taken from [12]). (c) Photo of Andersson’s robotic ping pong player (picture taken from [14]).

One of the first systems to demonstrate a reasonable performance was developed by Knight and Lowery [11]. Their setup consisted of an X-Y plotter placed at the edge of the table. The main carriage moved in the X-axis (width-wise), which in turn carried the bat carriage that moved in the Y-axis (vertically), as seen in Figure 2.1(a). The vision system comprised a mobile camera, mounted on the main carriage, and a set of mirrors for the capture of different view points. The justification for a mobile camera was that it reduced the field of view and consequently the background interference. This setup
was capable of tracking the ball in the X axis with good performance. However, it was not that good regarding the prediction of the interception point at the Y-axis.

In 1987, Hartley presented a Toshiba experimental project regarding the development of a system composed by a robotic arm that could strike some ping pong balls [12]. The main focus of this work was the hardware and its performance, as they were studying sensory control in real-time. The vision system was composed by two line CCD cameras (see Figure 2.1(b)). The idea was that one sensor detects the position of the ball as it passes through an under measuring plane and the other as it passes through an upper measuring plane, in order to intersect the trajectory of the ball. This setup presented several limitations. First, it was not intended to play against a human in an official table, as it was built to play against a wall. Second, the number of balls that it was capable of striking was relatively low. The maximum number of balls it was capable of striking in a row was four. Despite all this, it was still a relevant work as it was one of the first setups to use a robotic arm.

A major breakthrough was achieved by Andersson in 1988, when he presented the first robot table tennis system capable of playing against humans [14]. For that, Andersson used a six degrees of freedom (DoF) Puma 260 arm, with a 0.45 meters long stick, mounted between the table tennis racket and robot (see Figure 2.1(c)). In order to predict ball trajectories he used a vision system with four fixed cameras. The controller was based on an expert system that chose the strategy as a response to the incoming ball. This approach made full use of human knowledge in the form of explicit models of the task, thus making performance completely dependent on the system developer’s knowledge, and not allowing the system to improve its skills through practice or experience. Comparing with this setup, our approach is more challenging. First, it uses a single moving camera, making it harder to identify and track the ping pong ball. Secondly, unlike the approach of Andersson, ours incorporates a learning component.

The last robot table tennis competition took place in 1993 and was won by Fässler [15]. His setup consisted on a six DoF manipulator, with a 2-camera vision system. The focus of his work was on optimization and high speed performance, which resulted in a manipulator designed for maximum acceleration and minimal inertia of moving parts. Like most of the research of this time, Fässler’s setup did not have a learning component, and its computer vision system made use of more than one camera.

Recent years witnessed a revived interest in this task. Main breakthroughs were achieved, and nowadays we have systems that manage a 99% rate of success in returning balls against a ball launcher. We review some of these works in greater detail ahead. We also review other work which, although not specific to table tennis, addresses problems that are relevant to our research, namely flight control.

### 2.2 Robotic Table Tennis

After the end of the competition, one of the first works to appear was developed by Acosta et al. They proposed a low-cost ping pong player prototype with two rackets, which was capable of playing table tennis against a human [16] (see Figure 2.2). By using commodity components that could be found in the market, they achieved a cost reduction beyond the 10:1 factor, when compared to the robotic systems used at the time.

A decision that greatly contributed to the reduced cost was the use of a vision system that used only one camera and one acquisition module (like the Parrot AR.Drone 2.0 used in our work), thus not making use of stereoscopic vision which was the standard back then. The decision to use a single camera presented some challenges regarding the ball position identification. First of all, they had to proceed to a lens deformation correction, which was done through neural networks. Secondly, the algorithm they employed was based on the detection of both the ball and its shadow on the captured
images. In order for the system to be able to reliably identify the shadow of the ball on the table, they had to include a spotlight as part of their vision system, which was placed over the table. Then, taking into account the fixed position of the camera and spotlight, with some geometry they were able to estimate the spatial coordinates of the ball. They also proposed a simple game strategy learning component that consisted on dividing the table in several non-intersecting zones at which the robot would try to make the ball bounce. When evaluated in a real game against human opponents, this system was able to return balls with a success rate near 90%, when playing with ball velocities below 5m/s and small spin effects. The selection of the zone would be based on priorities that were increased when the other player failed to respond to a ball on that area of the table.

This setup bears some resemblance to ours, although with a key difference. Our system does not have a fixed camera, as it is incorporated in the moving aerial robotic system. This difference impairs the possibility of reusing some ideas from this work. In particular, since our camera is attached to the robot, Acosta et al. ping pong ball tracking method can not be used.

Later research, taking advantage of the much improved hardware and software, started to focus on the learning component of the systems, as well as on ball trajectory and hitting point predictions.

Myazaki et al., proposed a different approach, both regarding the robotic setup and the learning process [17]. Their hardware setup was mounted on the table with four DoF for the motion in a horizontal plane, and two other DoF for the paddle's tilt (see Figure 2.3). The paddle was square-shaped, with a 155 millimeters side, and moved in parallel with the table at a height of 195 millimeters. The vision system was stereo and consisted on two cameras set behind the paddle that captured the location of the ball's center of gravity from each image every 1/60 seconds. With respect to the learning process, it was achieved by means of three input-output maps, implemented by \( k \)-dimensional trees, using locally weighted regression. The three maps consist on:

1. A map for predicting the impact time of the ball hit by the paddle, and the ball position and velocity at that moment;
2. A map representing a change in ball velocities before and after the impact;
3. A map giving the inverse relation between the ball velocity just after the impact and the bouncing point and time of the returned ball.

Implementing the first map consisted on observing incoming balls and storing the ball's state when it passes through a virtual plane for measurements, and when it is hit by the robot. The second and third maps are implemented by observing the ball being hit by the paddle, with a certain altitude and velocity selected at random, and storing the ball's velocities before and after the impact, the paddle's altitude

---

[16] Memory-based method that performs a regression around a point of interest using only training data near that point, using weights that depend on how far away each data point is from the point of interest [16].
and velocity, the landing point, and time of the returned ball. For the generation of paddle movement, a feed-forward control scheme based on iterative learning control was proposed. More specifically, it used supervised learning to compute new trajectories that might occur during the game.

It is interesting to note that this setup is more complex than the previous one, as its hardware setup can move more freely. However, the system we propose presents additional challenges due to the larger freedom of motion, as well as a vision system based on a moving camera.

Lai and Tsay proposed an approach for a self-learning humanoid robotic ping pong player \cite{19}, depicted in Figure 2.4. In order to imitate the paddle motion of a human ping pong player, they use inverse kinematics to translate to joint positions the different poses observed during the demonstrations. They estimate the ball trajectory using a fuzzy ART network \cite{20}, and learn the behavior for each strike using a self-organizing map-based reinforcement learning network. Their setup consisted of a robot composed by a 7 DoF binocular head, two 7 DoF arms, two 7 DoF hands and a mobile base. The vision system was composed by four cameras. Their approach successfully allowed a humanoid robot to play against a human. It is also worth mentioning that this was the first approach to make use of reinforcement learning.

Figure 2.4: Design of Lai and Tsay setup. Taken from \cite{19}.

---

2Inverse kinematics refers to the use of kinematic equations of a robot in order to recover the angles of the joints needed to achieve a given end-point position.
More recent research started to include reinforcement learning components allowing the systems to learn by trial and error. Up until the work of Müller et al. [21], none of the existing research addressed the learning of robust and smooth hitting movements that were able to compensate for perturbations and noise in the environment, as well as inaccuracies in predictions. In their work, Müller et al. introduced a biomimetic approach for trajectory generation and movement adaptation for a robotic arm, taking into account theories on human motor control in the table tennis task [21]. They indicate that, for both humans and robots, the comfort-optimality criterion is more important than energy-optimality, when it comes to strike movement generation. Their approach models striking movements using a virtual hitting point hypothesis and four discrete movement stages, implemented by a state-machine:

1. **Awaiting stage**: Starts as soon as the robot player hits the ball, and lasts until the ball returned by the opponents crosses the net. It is at this point that the robot player decides to either use a forehand or a backhand movement.

2. **Preparation stage**: Starts when the ball returned by the opponent crosses the net, and the player starts moving the racket backwards in order to prepare the hitting movement. It is at this stage that the player chooses the point where he plans to intercept the ball—the virtual hitting point.

3. **Hitting stage**: Occurs when the ball reaches the virtual hitting point, where the player intercepts it.

4. **Finishing stage**: Starts when the player hits the ball, and the latter starts moving towards the opponent.

The movement trajectories are modeled using fifth order splines.

In order to predict the ball trajectories, they employ a dynamic model of a table tennis ball, which takes into account air drag and gravity, but not spin, as it is hard to observe from visual measurements. With this model, their system predicts the ball's position and velocity when it reaches the virtual hitting point. Then, through inverse kinematics, they translate that goal position and velocity into joint configurations of the robot, generating a movement. There is also an additional parameter of the movements, which allows the system to choose one of three possible stroke types, based on the area of the virtual hitting point, where the ball will be at. For evaluation purposes, they used a setup composed by a Barret WAM™ arm and a vision system with four fixed cameras. Their system was able to return 99% of the balls served by a ball launcher.

More recently, this same group of researchers presented an improved approach on how to select and generalize striking movements [4]—the approach we are following in our work. As already discussed, the first step of the approach consists on learning by imitation a set of elementary hitting movements that will be the basis of other more complex striking movements. Each of these movements is called a motor primitive, and is stored in a library associated with a set of parameters, called the augmented state.

Each motor primitive is formalized as a policy that maps an augmented state of the system and a vector with task-specific adjustable parameters to a hitting motion. As we already discussed, an augmented state is composed by a state and by a meta-parameter.

When the time to choose a trajectory comes, the system selects a mixture of motor primitives and its parameters according to the current context. This process of selecting the basic motor primitives is accomplished through a gating network that provides different weights to each motor primitive, according to their similarity to the input signal, and to the confidence in their success.

To tune the weights, an episodic reinforcement learning algorithm is used, by means of Cost-regularized Kernel Regression (CrKR) [22], and Linear Bayesian Regression (LBR) [23]. While CrKR creates new mappings between states and meta-parameters, LBR updates the gating network's weights. Intuitively, what this learning process does at each time-step is as follows:
1. Simulate a new game state and choose a new meta-parameter using the current knowledge.
2. Select an action using the gating network and execute it.
3. Update the knowledge about the mappings from states to meta-parameters.
4. Use LBR to update the weights of the gating network.

This research work also proposed a new approach regarding movement representation through DMPs, which are dynamic systems that allow the parameterization of the duration, and position and velocity of the movement at its final stage, while preserving the shape of the original movement.

It is important to notice the differences between the last two works. The former uses a specific model for the trajectories of ping pong balls to predict where the ball will intersect with the virtual hitting plane, and only presents three types of stroke movements, which can be parameterized to hit the ball at a given hitting point. The latter only uses a trajectory model for the ball tracking. The prediction where the ball will be at, as well as the velocity and orientation that the racket should have at the hitting moment, are estimated through learning techniques. Thus it is more robust against environment perturbations.

Regarding the results of this last approach, before learning the system was able to return 74.4% of the balls against a ping pong ball launcher. After learning, that number increased to 97%.

Comparing these two approaches to ours, we will be dealing with additional challenges as our robot has less sensory information, since it is not at a fixed position, and because our vision system is composed by only one mobile camera. However, since these approaches are, to the best of our knowledge, the state of art regarding robotic table tennis, our work will be based on them, specially on the last one. The preference for the last one comes from the fact that it depends less on exact measurements and more on features as seen in [4]. This is important because our drone has sensors significantly less informative.

Table 2.1 summarizes the systems discussed.

### 2.3 Flight Control

In recent years, there was a lot of research on automated flight control, both of helicopters and quadcopters. We review some of that research now, as part of it also includes a learning component using reinforcement learning, and is therefore relevant for the present work.

A first example is the research developed by Abbeel et al., which focuses on aerobatic helicopter flights. One of their first works consisted on applying apprenticeship learning and reinforcement learning to develop a controller for a helicopter that could achieve four complex aerobatic maneuvers:

1. **Forward flip** at low speed, in which the helicopter rotates 360 degrees forward around the axis that goes from the right to the left of the helicopter, while staying in place.
2. **Sideways roll** at low speed, which differs from the forward flip in the fact that the helicopter rotates around the axis that goes from the back to the front of the helicopter.
3. **Tail-in funnel**, in which the helicopter describes a circular trajectory with the tail pointed towards the center of the circle.
4. **Nose-in funnel**, which differs from the tail-in funnel in the fact that it is the nose of the helicopter that points to the center of the circle.

---

3Apprenticeship learning is another name for imitation learning.
Table 2.1: Overview of existing robotic table tennis systems.

<table>
<thead>
<tr>
<th>Author</th>
<th>Hardware</th>
<th>Vision</th>
<th>Learning Component</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Robotic arm</td>
<td>DoF</td>
<td>Stereo</td>
<td>Fixed</td>
</tr>
<tr>
<td></td>
<td>Author</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hartley [12]</td>
<td>X 7</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Andersson [14]</td>
<td>X 6</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Fässler [15]</td>
<td>X 6</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Acosta et al. [16]</td>
<td>1</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Myazaki et al. [17]</td>
<td>4</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Lai and Tsay [19]</td>
<td>X</td>
<td>7</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Mülling et al. [21]</td>
<td>X</td>
<td>7</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Mülling et al. [4]</td>
<td>X</td>
<td>7</td>
<td>X</td>
</tr>
</tbody>
</table>

Notes:

- The vision system comprises a single camera and mirrors.
- The vision system comprises a single camera and a spotlight.
- The hardware setup is a humanoid robot, with a robotic arm.

In order to accomplish these maneuvers, the authors collected data from an expert human pilot flying the desired maneuvers. This data is then used to learn a model of the dynamics of the helicopter during flight, from which they create a controller that matches the desired trajectories. The controller is developed by reinforcement learning, using Differential Dynamic Programming (DDP), to obtain the optimal policy, with the reward function found through apprenticeship learning via inverse reinforcement learning. The final controller was able to successfully complete the mentioned maneuvers.

The same approach was used for even more complex maneuvers, like inverted helicopter flight [8]. Comparing with the problem we address, we face additional challenges, as we must deal with vision to track the ball, and the trajectories that we must follow are not static, since they depend on the ball’s path. Despite these different challenges, the work of Abbeel et al. presents similarities with our own at the architectural level—they use human demonstrations to obtain initial motions, and then refine them using reinforcement learning.

More recently, several researchers addressed flight control on a new type of aerial vehicles—quadrotors (often also called quadcopters). Their popularity is rapidly increasing, as their flight endurance is growing and their cost decreasing. The fact that they are highly maneuverable and allow for a great number of applications also supports that popularity increase.

Vijay Kumar et al. presented a detailed introductory tutorial on modeling, estimation and control for multirotor aerial vehicles, including the quadrotors [27]. All the main concepts related to quadrotors’ setup and their rigid-body dynamics are discussed, with a lot of focus given to aerodynamics. Estimation of vehicle state is also studied, presenting the existing techniques for estimating attitude, velocity and

---

4 Iterative local search algorithm for optimal control [25].
5 Inverse reinforcement learning is the problem of extracting a reward function from observed optimal behavior [26].
position. In the end, the focus shifts to control, where attitude and trajectory control, as well as trajectory planning are reviewed.

In more recent work, the same authors address the organization and coordination of many agile micro quadrotors [28, 29], as seen in Figure 2.5. By using multiple micro aerial robots, the fundamental payload limitation is overcome and a new range of applications is possible, for example using grippers or cables. The usage of many quadrotors also presents the advantages of bigger battery duration and agility. However, coordinating an increasing number of quadrotors comes at a price—the combinatorial explosion of possible interactions between vehicles increases the required computational power [28]. To manage this complexity increase and guarantee communication, topology and the shape of formations, they consider a team architecture, in which the team is organized into labeled groups, each with labeled quadrotors.

Regarding the software infrastructure, they use a Vicon™ motion capture system to track the position of each quadrotor. This data is then streamed over a high-speed ethernet network to a desktop base station, where the high-level control and planning occurs. For each quadrotor there is a Matlab control node, running in an independent thread. These control nodes receive critical data from the Vicon™ system through shared memory, while control nodes communicate non-critical data with Inter Process Communication. Each control node is associated with a radio module that is used to communicate with the quadrotors, which have two independent radio transceivers. Each command sent by the control nodes to the quadrotors contains the changes that each robot must apply to:

1. Orientation.
2. Thrust.
3. Angular rates.
4. Attitude controller.

Much like our own work, this research also relies on quadcopters and a vision system. There also similarities at the architectural level—the robots’ sensors send their data to an off-board computer, which computes the control commands to be executed and sends them back to the robots. However, there are some important differences, the major being the usage of a Vicon™ motion capture system. They use a fixed camera vision system, as opposed to the mobile camera used by the Parrot AR.Drone 2.0. Another difference resides is the fact that in this work all the agents are aware of each others’ intentions and behavior. In ours, the robot ping pong player does not know where its adversary will try to put the ball, thus requiring better performance and good trajectory prediction.

---

Figure 2.5: A group of quadrotors in formation. Taken from Vijay Kumar’s TED Talk\(^6\)

---

\(^6\)Vijay Kumar’s TED Talk: [http://www.youtube.com/watch?v=4ErEBkJ_3PY](http://www.youtube.com/watch?v=4ErEBkJ_3PY) Last time accessed: 10th October, 2014.
Another research that shares similarities with our work is related with pole acrobatics using quadcopters \cite{30}. Dario Brescianini et al. designed a system that allows a quadcopter to balance a pole, throw it into the air, while having another quadcopter catch and balance it again (see Figure 2.6). Their work combined different algorithms and techniques:

1. Optimal control methods were used for the off-line design of the throwing and catching maneuvers.
2. Estimation techniques for hybrid systems were needed, because of the switching nature of the dynamics of the pole, which can be in free flight or in contact with the quadcopter.
3. For the catch maneuver, real-time trajectory generation was necessary.
4. In order to improve the system’s performance over time, an adaptation strategy was developed, which compensates for systematic errors. More specifically, their system learns iteratively, in a RL-like manner, some parameters regarding the catching height and position, and the offset error to be expected.

Their architecture does not differ significantly from the previous systems, as the algorithms run on desktop base stations, which send commands to quadcopters through radio signals, and because the vision system consists on a infrared motion capture system. As far as results are concerned, in a typical series of 9 throws, the system is able to execute 8 catches.

To conclude this chapter, we highlight how some of the most recent works share common elements. Mülling et al. were able to achieve successful results on the complex task that is robotic table tennis, with an architecture based on the conjugation of imitation and reinforcement learning \cite{4}. Relying on the same approach, Abbeel et al. developed an autonomous aerobatic helicopter, capable of performing complex maneuvers \cite{7}. The latter also shares similarities with the works of Kumar \cite{28} and Brescianini \cite{30}, as all have decentralized control with the computational processing occurring on separated computers and the commands being sent by radio frequencies.

Therefore, this architecture has proven to be able to achieve successful results on several distinct and complex tasks. This provides some guarantees of success when it comes to our own work, despite the fact that we are dealing with additional challenges, such as the limitations in perception and control.
## Background

### Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Parrot AR.Drone 2.0</td>
<td>21</td>
</tr>
<tr>
<td>3.2 Quadrotor Dynamics</td>
<td>21</td>
</tr>
<tr>
<td>3.3 Pinhole camera model</td>
<td>22</td>
</tr>
</tbody>
</table>
This chapter reviews basic concepts and notation concerning quadrotor dynamics and computer vision.

### 3.1 Parrot AR.Drone 2.0

In the context of this work we use a commercial quadrotor—Parrot AR.Drone 2.0. Equipment-wise, the AR.Drone 2.0 offers:

1. An HD camera, with 720p resolution at 30fps.
3. A 3 axis accelerometer, with ± 50mg precision.
4. A 3 axis magnetometer, with 6° precision.
5. Ultrasound sensors for ground altitude measurement.
7. 1GHz and 32 bit ARM processor.
8. 1GBit DDR2 RAM at 200MHz.

Along with all this equipment, the drone offers the capability of remote programming, by creating a dedicated WiFi network to which other devices can connect.

### 3.2 Quadrotor Dynamics

Quadrotors and helicopters in general have complicated dynamics, due to all the factors that can be taken into account such as blade dynamics or air flow. However, it is possible to simplify the dynamic model of these systems by considering that all movements can be described by the amount of acceleration supplied to each rotor. Additionally, while quadrotors have 6 Degrees of Freedom (DoF), there are only 4 parameters that can be controlled:

1. Throttle \((y)\)
2. Pitch \((\theta)\)
3. Roll \((\phi)\)
4. Yaw \((\omega)\)

These parameters are depicted on Figure 3.1.

![Figure 3.1: Example of quadrotor dynamics. The throttle, which is not depicted in the picture, consists simply on the acceleration applied to the rotors.](image-url)
Throughout this work, we consider the depicted robot reference frame. Additionally, for the purpose of control, we will not consider the yaw parameter. This is due to the fact that the quadrotor will mostly be facing towards the table, in order to hit the balls with the attached racket.

### 3.3 Pinhole camera model

The pinhole camera model describes the transformation between a 3D point in world coordinates to a 2D point in image coordinates. It assumes that a single point describes the camera aperture and that no lenses are used to focus light. Figure 3.2 illustrates this model.

![Pinhole Camera Model Diagram](image)

**Figure 3.2:** Example of the pinhole camera model. (a) shows a point $O$ in both 3D and plane coordinates. (b) presents the concept of focal length and the geometry behind this model. Figures adapted from [http://www.cs.unc.edu/~lazebnik/spring09/lec11_single_view.pdf](http://www.cs.unc.edu/~lazebnik/spring09/lec11_single_view.pdf).

In its simplest form, the pinhole model is defined by the transformation:

$$ (x, y, z) \rightarrow \left( \frac{fx}{z}, \frac{fy}{z} \right), \quad (3.1) $$

where $f$ is the focal length of the camera, and $x$, $y$, and $z$ are the world coordinates of the point. By using homogeneous coordinates it is possible to express transformation (3.1) as:

$$ \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} fx \\ fy \\ fz \end{bmatrix} = \mathbf{P} \mathbf{x} \triangleq \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}. \quad (3.2) $$

In Figure 3.2(a), $p$ is the point where the principal axis intersects the image plane (the principal point). The transformation in (3.1) assumes that $p$ is the origin of the image coordinate system. However, such coordinate system usually features its origin in a corner of the image, for example the bottom left corner (see Figure 3.3). Therefore, we modify (3.1) accordingly:

$$ (x, y, z) \rightarrow \left( \frac{fx}{z} + p_x, \frac{fy}{z} + p_y \right), \quad (3.3) $$

where $p_x$ and $p_y$ are the coordinates of $p$ in the image reference frame.
Once again, by using homogeneous coordinates it is possible to describe the transformation in matrix form:

\[
\begin{bmatrix}
    x \\
    y \\
    z \\
    1
\end{bmatrix}
\rightarrow
\begin{bmatrix}
    fx + zp_x \\
    fy + zp_y \\
    z \\
    1
\end{bmatrix}
= PX \equiv
\begin{bmatrix}
    f & 0 & p_x & 0 \\
    0 & f & p_y & 0 \\
    0 & 0 & 1 & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    x \\
    y \\
    z \\
    1
\end{bmatrix}, \quad (3.4)
\]

where \( P = K [I|0] \) and \( K = \begin{bmatrix}
    f & 0 & p_x \\
    0 & f & p_y \\
    0 & 0 & 1
\end{bmatrix} \) is called the camera calibration matrix.

However, this transformation still does not output 2D image coordinates in pixel units. In order to do it, the transformation must account for the pixel magnification factors specific to the camera. These are given by \((m_x, m_y)\), where \(m_x\) and \(m_y\) are, respectively, the number of pixels by distance unit, horizontally and vertically.

Taking the magnification factors in consideration, matrix \( K \) finally becomes:

\[
K = \begin{bmatrix}
    m_x & 0 & 0 \\
    0 & m_y & 0 \\
    0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    f & 0 & p_x \\
    0 & f & p_y \\
    0 & 0 & 1
\end{bmatrix}
= \begin{bmatrix}
    \alpha_x & 0 & \beta_x \\
    0 & \alpha_y & \beta_y \\
    0 & 0 & 1
\end{bmatrix}
\]

Thus, the final transformation then becomes:

\[
(x, y, z)\rightarrow \left( \frac{fm_x x}{z} + p_x m_x, \frac{fm_y y}{z} + p_y m_y \right) = \left( \frac{\alpha_x x}{z} + \beta_x, \frac{\alpha_y y}{z} + \beta_y \right), \quad (3.6)
\]

This transformation outputs actual 2D image coordinates in pixel units.
This chapter presents the solution proposed for our system. It starts by describing the logical, physical and learning architectures used, and the rational behind them. We discuss each component in detail and conclude with an overview of the system.

### 4.1 Software Architecture

The architecture developed for our system was driven by several requirements that can be divided in three main categories:

- **Performance**
- **Modifiability**
- **Testability**

Performance requirements are imposed by the nature of the table tennis task itself. In a professional ping pong game, ball velocities can reach 100 m/s, while beginners usually play at speeds of about 4 to 6 m/s. Taking into account that the full length of a ping pong table is 2.74 meters, it is possible to conclude that even when playing at a beginner level, a player must detect and track the ball, estimate its trajectory and hit it, all in less than one second.

Modifiability and testability requirements arise from the necessity of having a system that could change and evolve in the future. More specifically, we wanted the system to be designed so as to be possible to easily add and modify functionalities, and test their effect on the overall system performance.

#### 4.1.1 Logical Architecture

The overall logical architecture of our system is depicted in Figure 4.1. The computer vision module differentiates the perception processing in distinct visual features, and communicates with the Manager module through a Queue. The manager module communicates with the movements library, estimator and controller modules. However, only the communication with the controller module occurs through a Queue. The controller module outputs its results to a Queue.

It is important to analyze in more detail the interaction between modules. As it is possible to verify, the manager, computer vision and controller modules output their results to a message queue. This is a common software architecture, called publish-subscribe [31], which allows the decoupling of message senders and receivers.

Let us now review the role of each module in the architecture in greater detail.

![Figure 4.1: Logical architecture of the system.](image-url)
Computer Vision

The computer vision module has the responsibility of providing information regarding relevant visual features. In the simplest case, its function is to detect and track the ping pong ball, thus computing its position and velocity, at each time instant.

The visual features submodules allow for greater decoupling on how relevant visual information is handled. Thus, it makes it easy to add, modify and replace visual data processing modules.

Manager

The manager module is responsible for orchestrating the whole process. It collects information provided by the computer vision module and passes it to the estimator, getting an estimate of where, how and when to hit the ball. This information is passed to the motor primitives library, which creates a trajectory composing the stored motor primitives. The trajectory is then passed to the controller that will execute it, making the quadcopter hit the ball.

Movements Library

This module plays two main roles. First, it stores striking motion primitives, which as discussed previously represent the trajectory demonstrated by a human expert in order to respond successfully to a given environment state—a thrown ball. Secondly, it is capable of composing them into more complex movements, taking into account which are the best primitives for the given game situation, weighting them respectively.

Estimator

The estimator module makes estimates about the dynamics of the table tennis ball thrown at the quadcopter. More precisely it estimates how, where and when to hit the ball, taking into consideration the position and velocity of the incoming ball, information which is computed by the computer vision module.

Controller

The controller is responsible for executing a given trajectory, thus driving the quadcopter from one position to another, with time and final velocity constraints.

The highly modular architecture developed enables several adjustments to the system to be easily conducted. For example, without affecting much of the system, it is possible to:

- Modify/replace the computer vision algorithm that detects and tracks the ball.
- Modify/replace the quadcopter controller.
- Modify/replace the ball dynamics estimator.
- Add/modify/replace the motor primitives.

By applying the publish-subscribe pattern [31], we can also easily test the output of the computer vision and controller modules in isolation.

Since it allows all these adjustments, the logical architecture helps the fulfillment of the modifiability and testability requirements.
4.1.2 Physical Architecture

The implementation of our system makes extensive use of ROS\footnote{Robot Operating System: \url{www.ros.org} Last time accessed: 10th October, 2014.}, a largely used meta-operating system for robotics. The decision to use ROS relied mostly on the services it provides:

1. Hardware abstraction by offering high-level commands.
2. A publish-subscribe messaging infrastructure that allows a quick development of modular systems, without neglecting performance issues.
3. An extensive set of tools for configuration, logging, debugging and testing.
4. Easy integration with simulators.

ROS is also supported by a large and helpful community. This also played an important role in our decision.

A simple ROS application can be described as a set of ROS nodes communicating with each other through a set of ROS topics. ROS nodes are the computational processes where software modules run, and ROS topics are message queues that implement the publish-subscribe pattern.

Figure 4.2 depicts the overall physical architecture of the system. Each computational process is a ROS node, and each message queue is a ROS topic. All the modules presented on the logical architecture run in some process. There is also an external process which is not associated with any logical software module—the ArDrone Driver, which will be discussed further ahead.

Perception Processes

As seen before, computer vision relies on a set of visual features. Each visual feature runs on a separated process. This happens for two main reasons:

1. Performance
2. Flexibility

In fact, different visual features might need distinct processing power and/or publishing frequencies. By running each feature on a different process, we can easily assign independent process running priorities,
thus making the best of the available computational power. It also allows for greater flexibility when it comes to deciding the publishing frequency of each visual feature.

This code unit was implemented using the Python language, making extensive use of the OpenCV open-source library. This library efficiently implements the main computer vision algorithms. Therefore, we were able to use Python’s recognized simplicity and extensibility, while ensuring a good performance.

Managing Process

The manager, the estimator and the motor primitives library are executed within the same process. It is not necessary to separate them, as they do not make independent and parallel computations.

All three modules were implemented in the Python language. However, the motor primitives library module has some performance-critical parts, which were written in Cython\(^2\). Cython is an optimizing static compiler for the Python language that allows the usage of extensions implemented in the C programming language. The advantage of this solution is that we can enjoy the best of both worlds—the flexibility and productivity provided by Python, and the high performance granted by the C programming language.

Actuation Process

The controller module also runs on its own process. Once again, this decision is driven by both performance and modifiability requirements. The rational for the performance issues is the same as the one for the Perception processes. Regarding the modifiability requirements, by running the controller modules on separated processes, we allow the addition of new controllers in the future.

Publish-Subscribe Queues

The publish-subscribe queues were implemented using the messaging infrastructure provided by ROS. Specifically, ROS nodes publish the messages to a ROS topic, which is listened by other nodes that are interested in receiving those messages.

An important advantage of using ROS topics is that we can easily integrate the system with debugging, testing and simulation tools, since they have the same interfaces.

ArDrone Driver

By default, ROS can interface with several robotic hardware platforms. Since that did not happen with the Parrot Ar.Drone, the ROS community developed the \texttt{ardrone_autonomy} ROS driver\(^3\). ROS drivers establish an interface between ROS and the SDK of a specific robotic hardware platform.

Often the SDKs of these robotic platforms allow for great levels of control and performance. However, this comes at a cost. They often force us to use a specific low-level programming language, recognized as less productive, and do not allow for much integration with existing libraries and tools. Community-developed drivers, at the expense of a minimal performance decrease, allow us to control a robotic platform using a high-level programming language and existing tools and libraries.

\(^3\)ardrone\_autonomy homepage: \url{http://wiki.ros.org/ardrone_autonomy} Last time accessed: 1st October, 2014.
The physical architecture shown relies on ROS, a meta-operating system for robotics. ROS is widely-used, even in some industrial applications. Therefore, its messaging platform respects performance concerns, which is useful to us. It also allows us to satisfy the modifiability and testability requirements, since it promotes modular application design and usage of the publish-subscribe architectural pattern. By employing this pattern, we can also make use of several available tools to analyze and test the output of some of the modules.

4.2 Learning Architecture

As already seen, our goal is to have a low-cost aerial robot that learns how to play rudimentary table tennis. To achieve the learning component, we use a learning architecture comprising two distinct, yet interconnected steps.

An overview of the two steps is depicted in Figure 4.3. The first step of the learning process consists on Imitation Learning. The second is the Generalization of Movements, which includes Learning the World Dynamics and Policy Search.

4.2.1 Imitation Learning

In order to hit a table tennis ball thrown at it, the quadcopter must execute one of multiple possible trajectories. These trajectories can be learned in several different ways. A rather naive approach consists on letting the quadcopter learn the world and its own dynamics, starting with no knowledge whatsoever. However, this has proven to be very inefficient, given the big solution space that we would be dealing with.

A more popular approach consists on providing hand-coded trajectory examples relying on dynamical models of the ball-robot system. This approach is able to yield reasonable results. However, since the dynamic models are never perfect, hand-coded examples may result in unfeasible trajectories [32].

The most widely used approach consists on having an expert demonstrating the trajectory by controlling a real quadcopter. Trajectories are collected and stored for later use, using a specific trajectory representation. This “teacher process” takes place only once, and is used to create a library of trajectories that are then refined to construct the robot’s actual movement. This is the approach we use, following [4].
4.2.2 Generalization of Movements

The second step of our learning process is called generalization of movements. It is in this step that the robot tunes its ball dynamics estimates and evaluates the success of each trajectory stored in the previous phase. As depicted in Figure 4.3, this process can be divided in two new subprocesses.

In the subprocess Learning World Dynamics we tune the estimator of the behavior of incoming table tennis balls. This estimator takes as input the position and velocity of the ball at a given time instant, returning the three following parameters:

1. The position where the drone should hit the ball.
2. The velocity at which the drone should hit the ball.
3. The duration of the trajectory.

Regarding the subprocess Policy Search, it is used to evaluate the success of each motor primitive, and weight them accordingly.

Both these processes are tuned using an episodic approach. To learn world dynamics we employ an episodic reinforcement learning algorithm—CrKR. When it comes to policy search, we use LBR.

Given its episodic nature, the tuning process happens in a cycle. We throw balls at the quadcopter and it tries to hit them. Its performance is evaluated by a reward, which measures its success. Upon each successful iteration the player should become a little better, as it is gaining new information.

The decision to use episodic reinforcement learning is supported by the fact that it allows our player to learn with few training strokes, in opposition to other algorithms.

4.3 Computer Vision

This section presents in detail the computer vision system developed for our robotic table tennis player. First we analyze the requisites that the computer vision must fulfill, taking into consideration the goals and requirements of the overall system. We then discuss ball detection and tracking. We address how the ball pose estimation is computed, and conclude with an overview of the system.

Requirements

As we already saw, the goal of our system is to have a low-cost aerial table tennis player, that can run on commodity computers. In order to achieve that, our computer vision algorithm must be fast and provide reasonable results at the same time. Such performance requirement guided our implementation of both ball detection and tracking, and ball pose estimation.

We also want a system that can be extended and modified in the future. Such requirements are also fulfilled by the computer vision system as seen in the software architecture section, with the processing of visual features.

At this moment, the system only processes one visual feature—the table tennis ball.

In order to achieve the proposed performance requirements for the computer vision system, we rely on the undemanding assumption that the color of the visual items we want to track is unique in the environment. In the case of the table tennis ball, we assume that no other object in the environment has its characteristic orange color.
Figure 4.4: Example of how color segmentation works for two ping pong balls. (a) consists on the image captured from the Parrot Ar.Drone 2.0 camera. (b) is the result of applying the color segmentation algorithm. The HSV range threshold used consisted on $[0, 60]$ for Hue, $[158, 255]$ for Saturation and $[150, 255]$ for Value.

RGB : [229, 170, 84]  
HSV : [36, 63, 90]  
RGB : [189, 128, 22]  
HSV : [38, 88, 74]  
RGB : [171, 94, 14]  
HSV : [31, 92, 67]  
RGB : [144, 81, 9]  
HSV : [32, 94, 56]

Figure 4.5: Example of ping pong balls colors in RGB and HSV. It is possible to verify that there is a bigger variation in the color ranges when using RGB. With HSV the color values remain relatively similar, thus making it easier to define the threshold ranges.

4.3.1 Table Tennis Ball Detection and Tracking

From the assumption of the uniqueness of the ping pong ball color, we chose to use fast color segmentation as a first step for ball detection and tracking. This algorithm works by segmenting the pixels of the image: pixels whose color belong to a given range are assigned the value 1 (white), while the others are assigned the value 0 (black). This effect can be seen in Figure 4.4.

Color segmentation becomes even faster and easier to implement, if the input images are in a specific class of color space formats. This class does not contain the commonly used RGB format. In RGB similar colors are not linearly related which, as shown by Figure 4.5, leads to similar colors having very different RGB codes. Therefore, it is usually hard to find a color range for the color segmentation algorithm, which covers the objects’ possible colors as well as lightning changes.

HSV, on the other hand, separates color information (chroma), from intensity or lighting (luma). This makes it very robust when it comes to lighting changes. The conversion of an image from RGB to the HSV format is also computationally efficient. Given all these advantages, we apply this transformation...
before running the color segmentation algorithm. Figure 4.6 shows how the same image looks in RGB and HSV formats.

As we already saw, the color segmentation algorithm outputs a binary image—an image with only two colors. White for the pixels that had the color we were looking for and black for the others. Therefore, the set of white pixels forms the shape of the object or objects we were looking for. These shapes are commonly called blobs. As seen on Figure 4.4 when color segmentation is successfully applied to a table tennis ball, it outputs an image with a circular blob. Additionally, given the fact that there will only exist one ping pong ball at each time, we assume that there will only exist one blob (in light of the assumed uniqueness of the ball color).

Given the binary image with the ping pong ball blob, it is necessary to identify it. A simple and computationally cheap way to do it is to compute the outer contours of the binary image. The outer contours are the pixels that establish the boundaries between the white and black pixels. Having the outer contours we can now apply an iterative minimum enclosing circle algorithm [33,34]—an algorithm that iteratively finds the center and radius of the minimum area circle that covers all the given points. Having the radius and center of the blob of the ping pong ball, the detection and tracking phase is completed. Figure 4.7 shows an example of a table tennis ball being both detected and tracked.

In Algorithm 1 we provide the pseudo-code for the steps described.

**Algorithm 1** Computer vision pseudo-algorithm for detecting a ping pong ball.

1: **Input:** image captured from the camera.
2: Convert image from RGB to HSV.
3: Apply color segmentation algorithm.
4: From the orange colored pixels, compute the minimum enclosing circle.
5: **Return:** circle center and radius.

As already stated, this algorithm was implemented using the commonly used OpenCV library.

### 4.3.2 Table Tennis Ball Pose Estimation

In computer vision, the most common way to estimate the pose of an object is to apply a triangulation algorithm, which requires stereo vision. However, since we have only available the single camera mounted on the robot, stereo-based approaches are not possible.

Still, it is possible to make position estimations of objects relative to the quadcopter’s camera, if the two following conditions are met:

![Figure 4.6: Example of an image in both RGB and HSV formats.](image)
Figure 4.7: Example of a table tennis ball being detected and tracked in real-time. Images taken directly from the processed video feed.

1. The object whose position must be detected has a known size.
2. The focal length of the camera is known.

The first condition is obviously fulfilled, as table tennis balls have a fixed size of 4 centimeters. Moreover, since the balls have a spheric shape, they have the same size independently of the angle at which we look at them. The second condition is also satisfied, as the focal length parameter can be computed from the camera calibration.

Given that the two conditions are met, it is possible to estimate the distance of a given object to the camera with the following equation:

\[ d = \frac{f \times H_{\text{real}} \times H_{\text{image}}}{H_{\text{object}} \times H_{\text{sensor}}}, \]  \hspace{1cm} (4.1)

where:

1. \( f \) is the focal length of the camera.
2. \( H_{\text{real}} \) is the height of the real object.
3. \( H_{\text{image}} \) is the height of the captured image.
4. \( H_{\text{object}} \) is the height of the object in the image.
5. \( H_{\text{sensor}} \) is the height of the camera sensor.

\( H_{\text{image}} \) and \( H_{\text{object}} \) are in pixel units. The other variables can be represented in any distance units, as long as they are consistent.

Using Equation (3.6) it is now possible to get the \((X, Y, Z)\) world coordinates of the center of the ball from its 2D pixel image coordinates \((x_p, y_p)\). We have:

\[ x_p = \frac{f m_x X}{Z} + p_x m_x \quad y_p = \frac{f m_y Y}{Z} + p_y m_y, \] \hspace{1cm} (4.2)

which leads to:

\[ X = \frac{x_p - p_x m_x}{f m_x} Z \quad Y = \frac{y_p - p_y m_y}{f m_y} Z \] \hspace{1cm} (4.3)

Both \( X \) and \( Y \) now depend on \( Z \). \( Z \) can be computed by noticing that:
Table 4.1: Ball position estimation algorithm results on the Parrot Ar.Drone. Errors represented in centimeters.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Original Position</th>
<th>Estimated Position</th>
<th>Component Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>Y</td>
<td>Z</td>
</tr>
<tr>
<td>1</td>
<td>15.00</td>
<td>6.00</td>
<td>60.00</td>
</tr>
<tr>
<td>2</td>
<td>0.00</td>
<td>-2.50</td>
<td>90.00</td>
</tr>
<tr>
<td>3</td>
<td>0.00</td>
<td>15.50</td>
<td>120.00</td>
</tr>
<tr>
<td>4</td>
<td>-30.00</td>
<td>-2.50</td>
<td>150.00</td>
</tr>
<tr>
<td>5</td>
<td>30.00</td>
<td>6.00</td>
<td>165.00</td>
</tr>
<tr>
<td>6</td>
<td>15.00</td>
<td>15.50</td>
<td>215.00</td>
</tr>
<tr>
<td>7</td>
<td>0.00</td>
<td>15.50</td>
<td>235.00</td>
</tr>
</tbody>
</table>

\[ d = \sqrt{X^2 + Y^2 + Z^2}, \] (4.4)

which leads to:

\[ d^2 = X^2 + Y^2 + Z^2 = \left(\frac{x_p - p_x m_x}{f m_x}\right)^2 Z^2 + \left(\frac{y_p - p_y m_y}{f m_y}\right)^2 Z^2 + Z^2 \] (4.5)

Finally,

\[ Z = \frac{d}{\sqrt{\left(\frac{x_p - p_x m_x}{f m_x}\right)^2 + \left(\frac{y_p - p_y m_y}{f m_y}\right)^2 + 1}}. \] (4.6)

In order to estimate the velocity of the ball at any given time instant it is only necessary to compute the quotient between the distance traveled by the time elapsed, since the previous time instant.

It is important to notice that this approach has expectable drawbacks. Distance estimations rely heavily on the size in pixels of the ball. Since pixels are a discrete measurement unit, this might lead to sudden variations in size, and consequently in distance. It is expected that this aspect will lead to errors in the estimates of both positions and velocities of the ball.

Illustrative Examples

The ball pose estimation algorithm was tested both with a real Parrot AR.Drone 2.0, as well as on a simulation environment in ROS.

With the real quadcopter, the tests consisted on manually placing a ping pong ball on specified marks, at specific distances from the drone’s camera. The results obtained are summarized on Table 4.1. Despite the possibility of errors introduced by the manual placement of the ball, it is still possible to draw conclusions. The data shows that the farther away the ball is positioned, the bigger the position estimate error will be. Additionally, it is in the Z component where the error is most noticeable. This was expected given the dependency of the distance estimation algorithm on the number of pixels the object occupies in the image. As already discussed, since pixels are a discrete measurement unit, this leads to sudden variations, which directly affect the Z value.

The simulation environment allows further testing, namely the performance of the algorithm for an actual ping pong ball trajectory. Twenty random ball trajectories were analyzed, with the results being summarized on Table 4.2. From the data it is possible to verify that, as expected, the error is more pronounced on the velocity estimates, because since the sampling period is relatively small, even a small error on a position measurement may lead to a big error on the velocity estimate. Additionally, as
Table 4.2: Ball pose estimation algorithm results on a simulation environment. Position errors are represented in centimeters, while velocities are in centimeters per second. The mean errors were computed taking into account the error at each time step of the trajectory.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>Ẋ</th>
<th>Ẏ</th>
<th>Ž</th>
<th>Mean Position Error</th>
<th>Mean Velocity Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.40</td>
<td>4.71</td>
<td>23.85</td>
<td>13.71</td>
<td>126.66</td>
<td>581.95</td>
<td>24.70</td>
<td>604.21</td>
</tr>
<tr>
<td>2</td>
<td>0.56</td>
<td>4.54</td>
<td>24.61</td>
<td>2.17</td>
<td>116.23</td>
<td>468.97</td>
<td>25.42</td>
<td>491.72</td>
</tr>
<tr>
<td>3</td>
<td>0.84</td>
<td>5.47</td>
<td>25.54</td>
<td>14.05</td>
<td>108.72</td>
<td>450.20</td>
<td>26.63</td>
<td>471.39</td>
</tr>
<tr>
<td>4</td>
<td>0.51</td>
<td>4.56</td>
<td>23.48</td>
<td>7.15</td>
<td>120.00</td>
<td>484.62</td>
<td>24.27</td>
<td>507.41</td>
</tr>
<tr>
<td>5</td>
<td>0.49</td>
<td>4.91</td>
<td>22.88</td>
<td>11.87</td>
<td>122.98</td>
<td>477.76</td>
<td>23.84</td>
<td>503.23</td>
</tr>
<tr>
<td>6</td>
<td>0.66</td>
<td>4.45</td>
<td>23.60</td>
<td>12.34</td>
<td>110.29</td>
<td>491.51</td>
<td>24.33</td>
<td>511.64</td>
</tr>
<tr>
<td>7</td>
<td>0.90</td>
<td>4.90</td>
<td>23.70</td>
<td>25.90</td>
<td>104.42</td>
<td>479.38</td>
<td>24.66</td>
<td>498.00</td>
</tr>
<tr>
<td>8</td>
<td>0.41</td>
<td>4.52</td>
<td>23.07</td>
<td>7.70</td>
<td>97.38</td>
<td>431.77</td>
<td>23.86</td>
<td>451.39</td>
</tr>
<tr>
<td>9</td>
<td>0.44</td>
<td>5.29</td>
<td>24.47</td>
<td>13.04</td>
<td>125.42</td>
<td>480.84</td>
<td>25.46</td>
<td>509.63</td>
</tr>
<tr>
<td>10</td>
<td>0.59</td>
<td>4.65</td>
<td>24.43</td>
<td>16.77</td>
<td>113.71</td>
<td>466.88</td>
<td>25.23</td>
<td>489.31</td>
</tr>
<tr>
<td>11</td>
<td>1.42</td>
<td>5.23</td>
<td>23.74</td>
<td>32.90</td>
<td>136.66</td>
<td>508.46</td>
<td>24.83</td>
<td>538.85</td>
</tr>
<tr>
<td>12</td>
<td>1.76</td>
<td>5.37</td>
<td>28.05</td>
<td>35.58</td>
<td>122.31</td>
<td>488.48</td>
<td>29.06</td>
<td>513.78</td>
</tr>
<tr>
<td>13</td>
<td>1.35</td>
<td>4.56</td>
<td>22.55</td>
<td>11.98</td>
<td>121.13</td>
<td>448.88</td>
<td>23.41</td>
<td>477.14</td>
</tr>
<tr>
<td>14</td>
<td>0.55</td>
<td>5.62</td>
<td>23.95</td>
<td>24.73</td>
<td>138.82</td>
<td>527.27</td>
<td>25.13</td>
<td>553.36</td>
</tr>
<tr>
<td>15</td>
<td>1.03</td>
<td>5.05</td>
<td>24.73</td>
<td>11.84</td>
<td>126.09</td>
<td>491.04</td>
<td>25.66</td>
<td>519.67</td>
</tr>
<tr>
<td>16</td>
<td>1.02</td>
<td>5.25</td>
<td>23.43</td>
<td>35.43</td>
<td>125.17</td>
<td>453.52</td>
<td>24.59</td>
<td>484.03</td>
</tr>
<tr>
<td>17</td>
<td>0.34</td>
<td>5.40</td>
<td>24.48</td>
<td>7.94</td>
<td>108.33</td>
<td>420.80</td>
<td>25.45</td>
<td>445.86</td>
</tr>
<tr>
<td>18</td>
<td>0.81</td>
<td>4.66</td>
<td>24.59</td>
<td>14.11</td>
<td>131.75</td>
<td>470.70</td>
<td>25.42</td>
<td>502.63</td>
</tr>
<tr>
<td>19</td>
<td>0.44</td>
<td>4.73</td>
<td>22.34</td>
<td>11.31</td>
<td>116.03</td>
<td>469.77</td>
<td>23.27</td>
<td>497.23</td>
</tr>
<tr>
<td>20</td>
<td>0.85</td>
<td>5.30</td>
<td>25.57</td>
<td>30.84</td>
<td>108.64</td>
<td>466.71</td>
<td>26.55</td>
<td>486.45</td>
</tr>
</tbody>
</table>

Mean: 0.77 4.96 24.15 17.07 119.04 477.98 25.09 502.85

already discussed, Ž is the component that presents the most error. It is also important to notice that the errors are clearly consistent, with each component having the same magnitude on each experiment.

Figure 4.8 presents both the ground truth and predicted trajectories and velocities of the ball, for one of the experiments. From the plots it is also possible to draw some conclusions. First, the error on the velocity’s Ž component is more noticeable when the ball is more distant, typically at the beginning of the trajectory. This occurs due to the fact that when the balls are distant from the quadcopter, their pixel size on the image is smaller and more prone to induce errors. Secondly, the initial velocity spikes on component Ž can be justified by the sudden position variations of the corresponding component.

⋄

The algorithms just reviewed make it possible to detect a table tennis ball and estimate its position relative to the quadcopter’s camera.

Regarding ball detection, the main benefits of the algorithm used are its simplicity of implementation and fast performance. It is fast and simple to implement while allowing for good performance thanks to the reduced number of operations and their simplicity.

We note as disadvantages that even though it uses the HSV color model, the algorithm is still sensitive to extreme lighting changes. Another disadvantage is that it requires the environment not to have other objects with the same color as the table tennis ball.

In relation to ball pose estimation, once again the main benefit is its high computational performance. This high performance is due to simple computations required, which are computationally inexpensive.
The main disadvantages are related to the lack of precision due to the algorithm’s considerable reliance on the pixel size of the object, which is discrete and often not informative enough. However, this error is consistent throughout the experiments, always presenting the same order of magnitude on each component. Therefore, if computer vision estimates are used by learning algorithms that do not take into account a hard-coded ping pong ball dynamics model, a reasonable performance is still expectable.

4.4 Imitation Learning

Imitation learning is a popular approach to provide initial knowledge to a given agent. We use it to build initial estimates regarding the ping pong ball dynamics, and the trajectories the robot should follow in order to hit it. This section discusses in further detail how the imitation learning process was carried out, and the trajectory representation chosen. We conclude with an overall discussion.

4.4.1 Trajectory Representation

Several approaches have been proposed in the literature for representing movements. Most of them make use of via-point based formulations using splines to interpolate between them [35], Gaussian Mixture Models [36] or Hidden Markov Models [37]. While these approaches can provide good results in many situations they still have some disadvantages. Trajectories represented using models based on
via-points and splines are not easily adjustable, thus making it difficult to use in new situations [4]. Hidden Markov Models and Gaussian Mixture Models can be hard to train when dealing with high dimensional data [4].

Thus a new approach called Dynamical system Motor Primitives (DMPs) was suggested [6,38]. DMPs are robust against perturbations in the environment because they do not use time explicitly. They are also capable of dealing with changes in the final position, speed, and duration of the motion, without modifying the overall shape of the movement. Additionally, they combine well with imitation and reinforcement learning.

Dynamic system Motor Primitives

As seen before, our system uses trajectories provided by the human expert as demonstrations of desired control. Once translated into the preferred representation, they behave as motor policies that determine the control signal as a function of time and robot state.

In the original formulation [6,38], the representation of a motor policy \( \pi \) as a DMP resorts to two differential equations. They are referred as canonical and transformed system, respectively.

The canonical system determines the phase \( z \) of the movement, generated by:

\[
\dot{z} = h(z).
\]

(4.7)

The transformed system is defined as:

\[
u = \pi(x) = d(y, g_f, z, w).
\]

(4.8)

It generates the desired movement for a single DoF. It is a function of the starting position \( y \), the goal position \( g_f \), the phase variable \( z \) and an internal parameter vector \( w \).

The original formulation has the limitation of forcing the movements to have zero final velocities. This is not very appropriate for a table tennis task, as it makes it hard to hit the ball as far as possible. An extension to this approach was proposed by Kober et al. [3], which supported non-zero final velocities. However, it still presented two drawbacks. First, considerable changes in the final position and velocity could lead to inaccuracies in the final velocity. Secondly, when the demonstrated movement for a certain DoF had small amplitude, i.e., the change in position was close to zero, the controller could induce huge accelerations when scaling the final position.

Mülling's Extension

To overcome the aforementioned drawbacks, Mülling et al. proposed an extension [4] in which the canonical system is defined as:

\[
\tau \dot{z} = -\alpha_z z,
\]

(4.9)

where \( \tau \) is a temporal scaling factor and \( \alpha_z \) is a constant defined to make the system stable. The quantity \( z \) is initialized as 1 and converges to 0 at the end of the movement.

For hitting movements, a DMP is represented by the transformed system:

\[
\tau \dot{v} = \alpha_v \left( \beta_y (g - y) + \dot{g} \tau - v \right) + \ddot{g} \tau^2 + \eta f (z),
\]

\[
\tau \dot{y} = v,
\]

(4.10)
where \( y \) and \( v \) are the planned position and velocity generated by the policy, while \( y_0 \) and \( g_f \) are, respectively, the start and desired final position. The constants \( \alpha_y \) and \( \beta_y \) are chosen in order to make the system critically damped. The quantity \( \eta \) is the normalized amplitude of the movement, and is defined as:

\[
\eta = \frac{\exp(g_f - y_0)}{\exp(a)},
\]

where \( a \) is the reference amplitude—the amplitude of the movement learned by imitation learning.

Additionally, in the formulation of Mülling et al., the transformed system can be seen as representing a body being pulled by a “moving target”. The state \( g \) of the moving target is

\[
g = \begin{bmatrix} g(t) \, \dot{g}(t) \, \ddot{g}(t) \end{bmatrix},
\]

and is described by the following equations:

\[
g = \sum_{i=0}^{5} b_i \left(-\frac{\ln(z)}{\alpha z}\right)^i,
\]

\[
\dot{g} = \sum_{i=1}^{5} ib_i \left(-\frac{\ln(z)}{\alpha z}\right)^{i-1},
\]

\[
\ddot{g} = \sum_{i=2}^{5} (i^2 - i) b_i \left(-\frac{\ln(z)}{\alpha z}\right)^{i-2},
\]

where, at the end of the movement, \( g \) is equal to the goal position \( g_f \), \( \dot{g} \) is equal to the desired final velocity and \( \ddot{g} \) is zero. Coefficients \( b_i, i = 0, \ldots, 5 \), are computed by solving the fifth order polynomial that starts with the initial position, velocity and acceleration, and ends at the desired goal position and velocity and zero acceleration. By using a fifth order polynomial, there is more control over the behavior of the moving target, ensuring a more accurate end result.

The transformation function \( f \) is defined as:

\[
f = \sum_{j=1}^{N} \frac{w_j \psi_j(z) z}{\sum_{j=1}^{N} \psi_j(z)}.
\]

Gaussian basis functions \( \psi_j(z) = \exp \left(-\rho_j (z - \mu_j)^2\right) \) are employed, with center \( \mu_j \) and bandwidth \( \rho_j \). \( N \) is the number of parameters \( w_j \). The transformation function allows the introduction of non-linearity to the generation of the movement. It is also relevant to notice that as \( z \) converges to zero, the effect of \( f \) reduces.

**Trajectory Initialization**

After collecting the observed trajectories, it is necessary to represent the trajectory of each DoF as a DMP. This consists on finding the set of internal parameters \( w \) that minimizes the error between the reproduced and demonstrated trajectory. For a collected DoF trajectory \( \{\theta_1, \ldots, \theta_T\} \), with \( \theta_t = [\theta_t, \dot{\theta}_t, \ddot{\theta}_t] \), it is possible to compute the reference signal as:

\[
f_t^{\text{ref}} = \tau^2 \ddot{\theta}_t - \alpha_y (\beta_y (g - \theta_t) + \tau \dot{\theta}_t) - \ddot{\theta}_t \tau^2
\]

For each parameter \( w \), we minimize the cost function:

\[
e_n^2 = \sum_{t=1}^{T} \psi_t w_n^2 \left(f_t^{\text{ref}} - z_t w_n\right)^2,
\]

which in matrix form yields:
Algorithm 2 Computation of Motor Primitive (Adapted from [4])

**Input:** augmented state $x$

Compute parameters $b_i$ so that $g$, $\dot{g}$ and $\ddot{g}$ verify the boundaries conditions

**for** $t = 1$ to $T_f$ **do**

**for** $i = 1$ to $L$ **do**

Compute the phase variable $z$

$\dot{z} = -\alpha z$

**for** $j = 1$ to $\text{DoF}$ **do**

Compute the moving target system

$g = \sum_{i=0}^{5} b_i \left( -\frac{\tau \ln(z)}{\alpha z} \right)^i$

$\dot{g} = \sum_{i=1}^{5} i b_i \left( -\frac{\tau \ln(z)}{\alpha z} \right)^{i-1}$

$\ddot{g} = \sum_{i=2}^{5} (i^2 - i) b_i \left( -\frac{\tau \ln(z)}{\alpha z} \right)^{i-2}$

Compute the transformed system

$\eta = \exp(a \tau \dot{v})$ \(\exp(a \tau \dot{v})\)

$\tau \dot{v} = \alpha_y \left( \dot{z} y (g - y) + \ddot{g} v - v \right) + \ddot{g} \tau z + \eta f(z)$

$\tau \dot{y} = v$

end for

end for

end for

$$e_n^2 = (f^{\text{ref}} - z w_n) \Psi_n (f^{\text{ref}} - z w_n)^T,$$ (4.16)

where $f^{\text{ref}}$ is the vector $f^{\text{ref}}_t$ for each time instant $t$, $\Psi_n = \text{diag} \{ \psi_n^1, \ldots, \psi_n^T \}$ and $z = [z_1, \ldots, z_{T_f}]^T$. As seen in [38] this problem can be solved using Locally Weighted Regression (LWR), and each optimal weight $w_n$ is computed by:

$$w_n = \left( z^T \Psi_n z \right)^{-1} z^T \Psi_n f^{\text{ref}}.$$ (4.17)

Algorithm 2 summarizes the computation of a motor primitive.

**Illustrative Examples**

We ran some tests to verify that the DMP trajectory representation allows to modify the duration, final position and velocity, while keeping the trajectory shape. Figure 4.9 shows how Mülling’s extension to DMPs deals with a given trajectory. The left column depicts the effects of changing the trajectory duration. It is possible to verify that all three components of the generalized trajectory (position, velocity and acceleration) keep the shape of the demonstration. The same occurs with the central and right columns, which present the effects of changing the final position and velocity, respectively.

**4.4.2 Acquiring and Initializing Initial Knowledge**

In order to provide initial knowledge to the quadcopter, we had a human expert execute the correct commands that lead to the right trajectory, for a given world state, and then stored that trajectory.

For the human expert to be able to command the quadcopter, an adequate controller is necessary. The Parrot AR.Drone already comes with a mobile application that can be used as a controller. In order to assess the usability of the mobile controller as the input device to control the robot, we asked two humans to use it to control the quadcopter and attempt to hit a ping pong ball thrown at it. Both experts struggled to accomplish this task. For this reason, the mobile controller ended up not being used and,
instead, a keyboard controller was implemented and used (see Figure 4.11(a) and Figure 4.11(b)). Both users felt more comfortable using it, rather than the mobile application, since they are accustomed to this kind of setup when playing video games.

As will be described in Chapter 5, we conducted several experiments in order to gather initial knowledge. In each experiment the quadcopter was randomly positioned at the top of the table. A ping pong ball was then thrown from a fixed position towards the area around the robot, with a random initial velocity. The human user would then control the robot using the keyboard interface, and try to hit the ball. The demonstrated hitting trajectories were stored alongside the associated augmented states.

Each trajectory was collected on the simulation environment, by recording the world coordinates positions \([x_{w_t}, y_{w_t}, z_{w_t}]\) over the time interval \(t \in [1, T_f]\), where \(T_f\) is the duration of the trajectory.

Even though the keyboard controller was considered more comfortable by the human users, it still is not the perfect way to acquire the initial trajectories. This is due to the fact that it is very hard to acquire the characteristic velocity variations of hitting movements. This issue is further discussed in Chapter 6.

**Trajectory Execution**

The AR.Drone 2.0 includes flight stabilizers and automatically compensates for gravity. As such, the control module needs only to provide, at each time instant \(t\), a command \((\dot{X}, \dot{Y}, \dot{Z})\) corresponding to the desired velocity for the robot at time \(t\).

Trajectory execution is run by an open-loop controller. The decision not to use a closed-loop controller was due to two main reasons:
Figure 4.11: \(a\) The keys used in the keyboard controller use a typical layout. The keys in blue, T and L, allow the takeoff and landing of quadcopter. Keys in red allow the control of the quadcopter trajectories, with W, A, S, D moving the quadcopter in front, left, back and right, while I and K move the quadcopter up and down. \(b\) Human controlling the Parrot Ar.Drone using a keyboard controller.

1. It requires more complex computations, which may impact performance requirements.
2. It relies on information about the robot state, which as previously discussed, is only available from very noisy measurements.

Our use of an open-loop controller means that errors become more pronounced as the trajectory duration increases. However, as hitting trajectories present relatively small durations (between half a second and 0.7 seconds) they should not lead to significant errors. It is also expected that the learning process is capable of dealing with and compensating these errors. Finally, we expect the errors to be consistent, with the trajectories ran by the open-loop controller always undershooting the original ones. This expectation was drawn from multiple experiments, where we verified that for every time step \(t\) of the trajectory, the position of the quadrotor was always less than expected. In a way, Table 4.3 and Figure 4.12 depict this effect.

**Illustrative Examples**

To evaluate the performance of the trajectory execution controller, a human expert executed a typical hitting trajectory in the simulator, with duration of about 0.7 seconds, whose ground-truth was stored. This trajectory consisted on going forward and upwards, while going a little bit to the right. The trajectory was reproduced ten times with the open-loop controller, and the ground-truth of the reproductions was also stored. Finally, we measured the average position and velocity errors at the end of the reproduced trajectories, when compared with the original one.
Table 4.3: Trajectory execution controller performance on a simulated environment. Errors are computed at the end of the trajectory. Position errors are given in meters and velocity errors in meters per second.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>Final Position Error</th>
<th>Final Velocity Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.004</td>
<td>-0.038</td>
<td>-0.066</td>
<td>0.008</td>
<td>-0.128</td>
<td>-0.277</td>
<td>0.076</td>
<td>0.305</td>
</tr>
<tr>
<td>2</td>
<td>0.004</td>
<td>-0.039</td>
<td>-0.069</td>
<td>0.011</td>
<td>-0.136</td>
<td>-0.290</td>
<td>0.079</td>
<td>0.320</td>
</tr>
<tr>
<td>3</td>
<td>0.004</td>
<td>-0.038</td>
<td>-0.066</td>
<td>0.009</td>
<td>-0.130</td>
<td>-0.282</td>
<td>0.076</td>
<td>0.311</td>
</tr>
<tr>
<td>4</td>
<td>0.004</td>
<td>-0.033</td>
<td>-0.066</td>
<td>0.009</td>
<td>-0.125</td>
<td>-0.280</td>
<td>0.074</td>
<td>0.307</td>
</tr>
<tr>
<td>5</td>
<td>0.004</td>
<td>-0.040</td>
<td>-0.071</td>
<td>0.009</td>
<td>-0.133</td>
<td>-0.286</td>
<td>0.082</td>
<td>0.315</td>
</tr>
<tr>
<td>6</td>
<td>0.004</td>
<td>-0.039</td>
<td>-0.069</td>
<td>0.009</td>
<td>-0.134</td>
<td>-0.290</td>
<td>0.080</td>
<td>0.319</td>
</tr>
<tr>
<td>7</td>
<td>0.004</td>
<td>-0.039</td>
<td>-0.068</td>
<td>0.009</td>
<td>-0.129</td>
<td>-0.276</td>
<td>0.079</td>
<td>0.305</td>
</tr>
<tr>
<td>8</td>
<td>0.005</td>
<td>-0.039</td>
<td>-0.068</td>
<td>0.010</td>
<td>-0.123</td>
<td>-0.254</td>
<td>0.078</td>
<td>0.282</td>
</tr>
<tr>
<td>9</td>
<td>0.004</td>
<td>-0.039</td>
<td>-0.068</td>
<td>0.010</td>
<td>-0.132</td>
<td>-0.285</td>
<td>0.078</td>
<td>0.314</td>
</tr>
<tr>
<td>10</td>
<td>0.004</td>
<td>-0.038</td>
<td>-0.066</td>
<td>0.006</td>
<td>-0.087</td>
<td>-0.165</td>
<td>0.076</td>
<td>0.187</td>
</tr>
<tr>
<td>Mean</td>
<td>0.004</td>
<td>-0.038</td>
<td>-0.068</td>
<td>0.009</td>
<td>-0.126</td>
<td>-0.268</td>
<td>0.078</td>
<td>0.297</td>
</tr>
</tbody>
</table>

Table 4.3 summarizes the results obtained for the error at the end of the reproduced trajectories. Figure 4.12 illustrates a comparison between the original and one reproduced trajectory.

It is possible to verify that, as expected, the error in all components is consistent. The reproduced trajectory always undershoots the original one. It is also possible to see that the error in the $X$ and $Y$ components is not sufficiently large to cause the quadcopter to miss a hit, as the error is smaller than the racket size. On the other hand, the error on the $Z$ component is more pronounced and could possibly lead to missing hits.

The system developed uses a state-of-art approach for capturing and representing initial knowledge. By using imitation learning, the initial trajectories are guaranteed to be feasible and correct. By using DMPs, the system is able to adapt known trajectories to new conditions, like duration, final position and velocity, while keeping their shape.

We use an open-loop controller to execute the trajectories. While some precision is lost, when compared with close-loop controllers, there is a clear performance gain as the computations required by the later class of controllers are more expensive. Additionally, since the errors produced by the open-loop controller are fairly consistent, it is expected that, to some extent, the learning modules will be able to accommodate and mitigate them.

### 4.5 Generalization of Movements

The second element of our learning architecture consists on the Generalization of Movements. Humans perform complex tasks in new environments by generalizing their motor abilities to new situations. State-of-art approaches also suggest that robotic motor skill learning should mimic this human process, by learning to perform complex motor tasks from smaller and simpler motor skills.

In our module of Generalization of Movements, we adopt the above perspective on learning of complex motor skills. We start by refining the world dynamics estimator, from successful attempts to hit the ball. At the same time, we compute a stochastic policy $\mu$ that maps the observed state $s$ to a hitting motion $\delta$. This policy is learned using an episodic reinforcement learning algorithm, dubbed CrKR.
Using the computed hitting motion, the robot instantiates each motor primitive \( \pi_i, i = 1, \ldots, L \), in the library and combines them to perform a complex movement \( \pi \). We refer to the resulting complex movement as a Mixture of Motor Primitives (MoMP). A policy generated by MoMP can then be represented as:

\[
\pi(x) = \frac{\sum_{i=1}^{L} \gamma_i(x) \pi_i(x)}{\sum_{j=1}^{L} \gamma_j(x)},
\]

where \( L \) is the number of motor primitives used, and \( \gamma_i \) is the weight associated with \( \pi_i \).

Finally, depending on the success of the hitting motion (encoded as a reward value \( r \)), each motor primitive is evaluated using Linear Bayesian Regression (LBR), and the associated weights are adjusted accordingly.

Algorithm 3 summarizes the complete learning process of our system. In the continuation, we describe the main elements of the Generalization of Movements learning process.

### 4.5.1 Learning World Dynamics

In the table tennis task, the world dynamics that the quadcopter must learn consists of the behavior of the ping pong ball. More specifically, the learning of the world dynamics is reflected by the estimates for where, when and how to hit an incoming ping pong ball, given the ball pose estimates when it is at the distance of 2.5 meters, provided by the computer vision system.

So, the robot must compute a stochastic policy \( \mu \) between states \( s \) and meta-parameters \( \delta \), where:

- \( s \) consists in the ball pose when it is at 2.5 meters from the quadcopter, and is given by \( s = (x_b, y_b, \dot{x}_b, \dot{y}_b, \dot{z}_b) \).

- \( \delta \) consists in the position where to hit the ball, the velocity at which the ball should be hit, and the duration of the trajectory, i.e., \( \delta = (x_{hp}, y_{hp}, z_{hp}, \dot{x}_{hp}, \dot{y}_{hp}, \dot{z}_{hp}, t) \).

When a new state \( s_{new} \) is observed, the estimate should take into consideration, not only the similarity between the new state and the states of the training examples already seen, but also the success that was achieved on each one of them. Since this mapping determines how the robot will approach incoming
Algorithm 3 Generalization of Movements Algorithm (Adapted from [4])

Input:
Library of Motor Primitives \(\{\pi_1, \ldots, \pi_L\}\)
State information from demonstrations:
\(S^0\) (states observed)
\(D^0\) (meta-parameters measured)
\(C^0\) (costs incurred)
Kernel parameters \(k, \lambda, \phi\)
Reward function \(r\).

for each trial do
  LEARNING WORLD DYNAMICS
  Determine current state \(s\).
  Find \(\delta = \mu(s)\) with CrKR.
  Set \(x = [s, \delta]\).
  Compute motor policy using MoMP:
  \[
  \pi(x) = \frac{\sum_{i=1}^{L} \gamma_i(x) \pi_i(x)}{\sum_{j=1}^{L} \gamma_j(x)}
  \]
  Execute motor policy \(\pi(x)\).
  Find reward \(r\) for executing \(\pi(x)\).
  POLICY SEARCH
  Update \(S\), \(C\) and \(D\), using \(s, \delta\) and \(c = \frac{1}{N}\).
  Use LBR to update \(\theta_i, i = 1, \ldots, L\).
end for

balls, it should also include some exploration, i.e., when presented with a state \(s_{new}\) very similar to the state of a training example:

- If that training example lead to a very successful \(\delta\), the algorithm assumes it is close to an optimal policy and does not provide a very different mapping.
- Otherwise, it will explore new motions, with the training point having reduced influence.

CrKR

The stochastic policy \(\mu\) mapping \(s\) to \(\delta\) is computed by an episodic reinforcement learning algorithm dubbed Cost-regularized Kernel Regression (CrKR) proposed by Kober et al. [22]. This algorithm is an adaptation of standard regularized kernel regression, taking into account the cost of different samples when computing the mapping between \(s\) and \(\delta\). In particular, \(\mu\) is computed so as to maximize the expected reward:

\[
J(\mu) = E\{r(s, \delta) | \mu\},
\]

where \(r(s, \delta)\) is the reward achieved by selecting \(\delta\) in state \(s\).

The estimates are based on a Gaussian distribution with mean \(\bar{\delta}\) and variance \(\Sigma\) for the policy \(\mu\). This can be summarized as \(\mu(\delta|s) = N(\delta|\bar{\delta}, \Sigma)\). The mean and variance are calculated by:

\[
\bar{\delta} = k(s)^T (K + \lambda C)^{-1} D
\]

\[
\Sigma = k(s, s) + \lambda - k(s)^T (K + \lambda C)^{-1} k(s),
\]

where, assuming the system has \(N\) training points, and noticing that, in our case, \(s\) and \(\delta\) are respectively \(1 \times 5\) and \(1 \times 7\) vectors:
• $k(s)$ is a $N \times 1$ vector, and measures the distance between state $s$ and the state of the training examples with a kernel.

• $K$ is a $N \times N$ matrix with the pairwise distances between the states of the training examples.

• $\lambda$ is a constant value and acts a ridge factor. It trades off the influence of the reward versus prediction accuracy in CrKR.

• $C = \text{diag} \{ r_1^{-1}(s_1, \delta_1), \ldots, r_N^{-1}(s_N, \delta_N) \}$ is a $N \times N$ matrix that contains the cost of each training.

• $D$ is a $N \times 7$ matrix whose lines consist on the $\delta_i$ of the training examples.

• $k(s,s)$ is a constant value and is the Gaussian kernel of state $s$ with itself.

In our implementation, $k$ and $K$ are computed using a Gaussian kernel to compare states \[23\]. Gaussian kernels are used to measure the similarity between two vectors in an input space, and can be defined as:

$$k(x, x') = \exp(-x^T U^{-1} x'),$$

(4.22)

where $x$ and $x'$ are column vectors, and $U$ is a diagonal matrix that defines the width to be considered for each component.

Illustrative Examples

The CrKR algorithm deals with several parameters and constants. The values chosen for these parameters and constants affect the overall performance of the world dynamics estimator, and consequently the performance of the complete system.

It is important to refer that CrKR has been implemented considering normalized states and meta-parameters. This is due to the fact that this data contains components with different magnitude orders. This is specially noticeable on the computer vision ball pose estimates (see, for example, Figure 4.8).

The data normalization was attained by collecting $N = 35$ hitting trajectories with imitation learning, and storing the observed augmented state. Then, for both the states $s_i$ and meta-parameters $\delta_i, i = 1, \ldots, N$, a sample mean ($\hat{\mu}_s$ and $\hat{\mu}_\delta$) and a sample standard deviation ($\hat{\sigma}_s$ and $\hat{\sigma}_\delta$) is computed. A new state $s_{\text{new}}$ is first normalized before being used to estimate the new meta-parameter $\delta_{\text{new}}$. This normalization is done using the standard score:

$$\hat{s}_{\text{new}} = \frac{s_{\text{new}} - \hat{\mu}_s}{\hat{\sigma}_s}$$

(4.23)

As the matrices $S$ and $D$ are also normalized, CrKR will output a normalized meta-parameter $\hat{\delta}_{\text{new}}$. To denormalize it, the following formula is applied:

$$\delta_{\text{new}} = \hat{\delta}_{\text{new}} \hat{\sigma}_\delta + \hat{\mu}_\delta$$

(4.24)

Regarding the ridge factor, it defines how exploratory the algorithm should be, by weighting the importance that should be given to the cost of training examples. In order to find the most suited value for the ridge factor an experiment was made, where we used:

• A library of motor primitives initialized with 10 motor primitives.

• A set of approximately 170 hits, from 400 trials.
The results depicted in Figure 4.13 were obtained by cross-validation, and test the performance of CrKR for different ridge factor values, for different training set sizes.

The data shows that the ridge factor value that leads to faster convergence was \( \lambda = 0.5 \), closely followed by \( \lambda = 0.25 \). \( \lambda = 2 \) clearly lead to the worst results.

Since, as seen before, we use a Gaussian kernel in CrKR, we must adjust the corresponding \( U \) matrix. The most important components of this matrix are the sideways position and velocity of the ball, relative to the quadcopter, \( i.e. , x_b \) and \( \dot{x}_b \). Intuitively, this can be explained by the fact that those are the components that may lead the quadcopter to completely fail to hit the ball, by moving in the wrong direction. Regardless of the frontwards and upwards speed, and height of the table tennis ball, the quadcopter most probably will have to move forward and upwards. This phenomenon is better illustrated in Figure 4.14. Figure 4.14(a) presents two movements observed during demonstration, where the difference lies in the sideways position and velocity of the ball, with respect to the drone. Figure 4.14(b) depicts a new game situation. This new game situation should be considered more similar to the one faced by the blue quadcopter, because the expected movements in both cases is the same.

Taking this into account, while using the current computer vision system, the \( U \) matrix of the Gaussian kernel was defined as:

\[
U^{-1} = \text{diag}(0.45, 0.1, 0.35, 0.05, 0.05)
\]

It is important to notice that the bigger the value in a \( U^{-1} \) component, the smaller the distance width allowed.
Figure 4.14: The most important component to consider when comparing two states is the sideways position of the ball, relative to the quadcopter. (a) shows two possible learned movements, where the only difference in the state vectors are the sideways position and velocity of the ball. (b) presents a new game situation. It is clear that the new situation is more similar to the one seen by the blue quadcopter, and thus the green quadcopter should move to the left.

4.5.2 Policy Search

The second learning subprocess consists on tuning the weights of the gating network, which mixtures the motor primitives. This is achieved by computing the weighting function $\gamma(x)$ that determines the contribution of each primitive in the final motion.

Intuitively, for a given incoming ball, the weight $\gamma_i$ to be assigned to motor primitive $i$ should be selected so as to generate a more reliable complex movement. This reliability is measured in terms of the reward function, which, as previously discussed, evaluates the success of the hitting movement. In other words, $\gamma_i$ can be seen as the "probability" that motor policy $\pi_i$ is the right policy to respond to the augmented state $x$. The computation of $\gamma_i$ takes into consideration the two following aspects:

- The similarity of the new augmented state with those seen during the imitation learning phase.
- The success of the motor primitive in those augmented states.

Naturally, more successful primitives in more similar states will be weighted more favorably than primitives that are not as successful or that were used in less similar circumstances.

Linear Bayesian Regression

LBR is an episodic regression algorithm that we use to approximate the value associated with each motor primitive. Formally, we represent the value of a motor primitive $\pi_i$ at an augmented state $x$ as a function:
Algorithm 4: Linear Bayesian Regression (Taken from [4])

1: for all \( \pi_i \) in library do
2: \[ c_i = \frac{\gamma_i}{\sum_j \gamma_j} \]
3: \[ V_{i}^{N+1} = \left( V_i^N \right)^{-1} + \beta \phi(x_{N+1})^T c_i \phi(x_{N+1}) \]
4: \[ \theta_i^{N+1} = V_{i}^{N+1} \left( \left( V_i^N \right)^{-1} \theta_i^N + \beta \phi(x_{N+1})^T c_i r \right) \]
5: end for

\[ J_{\pi_i}(x) = E\{r(x) | \pi_i\}. \quad (4.26) \]

We represent \( J_{\pi_i} \) as a linear combination of a set of predefined features \( \phi_j, j = 1, \ldots, L \), i.e.,

\[ J_{\pi_i}(x) \approx \sum_{j=1}^{L} \theta_{i,j} \phi_j(x) = \theta_i^T \phi(x), \quad (4.27) \]

where \( \theta_i \) is a parameter vector. Using standard Bayesian regression, and given examples of rewards \( r \), LBR computes the parameters \( \theta_i \) that minimize the error between \( \theta_i^T \phi(x) \) and \( E\{r(x) | \pi_i\} \).

Algorithm 4 presents the pseudo-code of this algorithm.

Having computed the optimal \( \theta_i \) parameters, it is now possible specify how the weights \( \gamma_i \) are computed:

\[ \gamma_i(x) = \xi \exp\left\{ \theta_i^T \phi(x) \right\}. \quad (4.28) \]

where \( \phi(x) \) denotes a feature vector at \( x \), \( \xi \) is a normalization constant. Expression (4.28) indicates that motor primitives with a higher associated value, \( J_{\pi_i} \), have a larger contribution in mixture.

Illustrative Examples

In our implementation, \( V_i \) is initialized as the \( N \times N \) identity matrix, and \( \theta_i \) as a zero-filled vector, with only the \( i \)-th component being initialized to one. A Gaussian kernel is also used to measure the similarity between augmented states. Following the same reasoning used to determine the \( U \) matrix in CrKR, we tuned the kernel covariance matrix experimentally, obtaining:

\[ U^{-1} = \text{diag}(0.189, 0.042, 0.147, 0.021, 0.021, 0.29, 0.145, 0.145, 0.017, 0.006, 0.012, 0.0232). \]

4.6 Discussion

In this chapter we presented our approach towards table tennis with a quadrotor autonomous robot and onboard vision. Our approach features a commercial quadrotor (Parrot AR.Drone 2.0), controlled by a commodity computer, with the commands being sent through a regular WiFi connection.

We started by reviewing the requirements imposed by the idealized system, and the proposed architectures to achieve them. Simple experiments showed that the logical, physical and learning architectures are capable of dealing with the performance, modifiability and testability requirements that we departed from.

We analyzed the computer vision system used, which consists of a single camera attached to the
quad rotor and verified that this adds considerable complexity to the computer vision module, especially regarding the ping pong ball pose estimation. We tested the computer vision system both in simulation and in the real quadcopter, and illustrated the performance of our proposed algorithm, highlighting its main advantages (simplicity and efficiency) and shortcomings (accuracy).

We presented DMPs as an efficient representation of trajectories that can be easily integrated with imitation and reinforcement learning. We discussed the use of imitation learning to provide the system with an initial set of hitting motions that the system can then generalize to new situations. Trajectory execution is accomplished with a simple open-loop controller.

Finally, we illustrate the performance of each module separately, discussing the main steps in determining the relevant parameters used.
5
Evaluation

Contents

5.1 Evaluation in Simulation .............................................. 55
5.2 Evaluation with Real Quadcopter ................................. 58
This chapter presents the experimental evaluation of the complete system. The system was tested both in a simulation environment, as well as in a real quadcopter. The evaluation procedure and results obtained for each case are reported.

5.1 Evaluation in Simulation

Evaluating a robotic system in a simulation environment is a standard procedure, as it allows faster and safer testing of the algorithms used. Faster because it enables the automation of certain tasks and imposes less limitations. For example, the batteries of the robot do not exhaust after 30 minutes, as occurs with the real robot. Safer because if anything goes wrong, neither the real robot nor any human are put at risk. The simulation environment used was ROS-based, and we refer to Annex A for more information regarding the characteristics of the simulation world and engine used.

Our system should be capable of learning how to hits the balls thrown at it and, as such, the evaluation procedure was created with the focus on assessing that the system actually learns. Tests were made both with the computer vision system and with ground truth data. The tests were also made using both a single motor primitive, and a library with ten motor primitives.

Setup

We collected a total of 35 trajectories on the simulation environment, demonstrated by a human expert, from which we built a library of 35 motor primitives, represented as DMPs. The motor primitives employed on the tests were carefully selected from this initial pool of motor primitives.

In order to evaluate the success of the motor primitives, we designed a reward function consisting on the exponential of the negative displacement between where the ball first hit the table and the line segment that cuts in half the table court that belongs to our quadrotor player (see Figure 5.1). A displacement of 5 meters was considered for balls that were hit but that fell out of the table.

![Figure 5.1](image)

**Figure 5.1:** Reward function example. The three cross marks represent three possible places where the ball might land. The blue mark leads to a positive displacement, and consequently to the biggest reward. The red mark represents a ball that does not hit the table, thus getting a displacement of 5 meters. The yellow mark while having a negative displacement, still provides a better reward than the red case.
5.1.1 Single Motor Primitive

The tests with a single motor primitive aim at testing how well can the motor primitives adapt to new situations. To test that, four series of 30 balls were thrown at the quadcopter. While the initial position of both the ball and the quadcopter were always approximately the same, the initial velocity of the ball was subject to four levels of zero mean Gaussian noise, namely:

1. No noise, \( i.e., \) constant initial velocity.
2. Small noise, with \( \sigma = 0.05 \).
3. Medium noise, with \( \sigma = 0.1 \).
4. High noise, with \( \sigma = 0.2 \).

The first experiments considered that the ball position was observed perfectly, using ground truth information from the simulator. Figure 5.2(a) depicts four experiments, for four different single motor primitives. As expected, the hitting performance drops as the Gaussian noise increases (blue plots). However, it also shows that the system is still capable of hitting a reasonable number of balls, when the position of the ball at the end of the table diverges from the original trajectory by at least 5 centimeters. Note that with high noise, all motor primitives were able to hit on more than 10% of the balls, even though the average distance to the original trajectory was near 15 centimeters.

The same kind of tests was repeated, now using the computer vision system developed. Figure 5.2(b) depicts four experiments, for the same four different single motor primitives as before. As expected, the performance drops when compared with the case where perfect vision is used. However, as in the previous case, even when high noise is used, the system is still capable of hitting more than 10% of the balls. Note in particular Motor Primitive 4, where the average distance to the original trajectory was near 15 centimeters. This shows that single motor primitives are able to adapt to new situations. These results also demonstrate, as expected, that while the learning processes can deal with the vision system estimation errors, a precise perception system will ultimately lead to better results.

5.1.2 Multiple Motor Primitives

Having assessed that single motor primitives can successfully generalize to new situations, it is now possible to test the overall performance of the system.

For this test, a library containing a total of 10 motor primitives was used. The 10 primitives were carefully selected from the 35 initial motor primitives collected. Figure 5.3(a) illustrates the 3D-coordinates of the points where the ping pong ball was hit, for each of the 35 collected trajectories, including the 10 chosen to be part of the library of motor primitives. As can be seen from the figure, the points are spread across a volume of \( 0.4 \text{m} \times 0.12 \text{m} \times 0.07 \text{m} \). The ten chosen trajectories are well spread across this volume.
Figure 5.2: Performance of four single motor primitives for different levels of Gaussian Noise added to the initial velocity of a given ping pong ball. (b) is the performance using the developed computer vision system, while (a) is the performance using ground-truth vision.

Figure 5.3(b) presents the performance of the player, both using the developed computer vision system and ground truth. It is possible to verify that in both cases the system quickly improves its
Figure 5.3: Overall performance of the system in the simulation environment. (a) depicts the 35 trajectories used for data normalization, including the 10 trajectories used for initializing the motor primitives library. (b) presents the performance of the player, both using the developed computer vision system and ground truth vision.

performance. With the developed computer vision system, the player is able to hit 25% of the balls in just 100 hits. Using ground truth vision, the player reaches a success rate of almost 40%.

5.2 Evaluation with Real Quadcopter

Evaluating a real aerial robotic system is considerably harder and more time-consuming than in simulation, as the real batteries run out very quickly and must be recharged. Therefore, the amount of testing that can be done with a real quadcopter is significantly reduced. However, enough tests were performed to verify that our system applied to the real robot also works.

Setup

For the robot table tennis task we used a Parrot AR.Drone 2.0 as the robotic player, with a vision system composed of a single onboard camera. The quadcopter was equipped with its protective indoor hull at all times in order to guarantee its safety in case something wrong happened. Additionally, the indoor hull was used to sustain the table tennis racket, which was made out of light cardboard. The racket has a diameter of 16 centimeters, in accordance with the International Table Tennis Federation (ITTF) rules. Cardboard does not have a high restitution coefficient and, as such, the ball hardly bounces off the racket. Therefore, in this section, success is measured by the capability of the drone to hit the ball or not. Figure 5.4(a) shows the racket used and Figure 5.4(b) depicts the whole set in flight.

At the time of this writing, we did not have a standard sized ping pong table available to use. Therefore, two regular tables with 1.58m of length and 0.76m of width, thus forming a 3.16m × 0.76m ping pong table.

In the real robot experiments, we opted by using the demonstrations provided in the simulation environment. This decision is supported by the fact that, even with the keyboard controller, it is hard to hit the ball with the real drone. It is easier on the simulation environment since it allows the slowdown of simulation time.
We proceeded in a similar way to the tests on the simulation environment. First, we tested the system using single motor primitives. Then, the tests used a 10-motor primitives library.

It is important to notice that all the balls were thrown manually by a human. Therefore, we cannot guarantee the similarity of the balls’ initial velocity and position.

**Camera Parameters**

In the real robot, the camera parameters used were as follows:

\[
\begin{align*}
    f_x &\triangleq f_{m_x} = 561.999\text{px} \\
    f_y &\triangleq f_{m_y} = 561.782\text{px} \\
    p_x m_x &= 307.433\text{px} \\
    p_y m_y &= 190.144\text{px}
\end{align*}
\]  

(5.2)

These values were obtained in the camera calibration file provided by the ArDrone ROS driver.

### 5.2.1 Single Motor Primitive

For the tests on the real quadcopter, we used the same two of the four single motor primitives utilized on the simulation environment tests. Two runs of 20 ball throws were performed for each motor primitive. One without any training, and another with the motor primitives library having around 20 additional training examples.

The motor primitives averaged a 20% success rate without training. With just a few training samples that rate increased to the 32.5% mark.

### 5.2.2 Multiple Motor Primitives

For the tests with multiple motor primitives the human ball thrower was asked to vary the direction and velocity of the ping pong ball. However, he was asked to cover a smaller area than the one on simulation.

Once again, there were two runs of 20 balls each. In the training run, the motor primitives library was initialized with approximately 80 training examples.
Without training the success rate averaged around the 10% mark, increasing to approximately 20% when training examples were used.

As already discussed, the tests performed with the real quadcopter are not easily reproduced. Additionally, there are no guarantees regarding the randomness of the initial direction and velocity imposed by the human ball thrower.

Still, the results presented suggest that the system's performance increases when training examples are used. Recalling our initial purpose of assessing the applicability of the framework of Mülling et al. [4] in a different hardware, our results show that, in spite of the perception and actuation limitations of our robotic setup, the proposed architecture is in fact able to successfully enable a quadcopter to learn to respond to balls thrown by a human player. Both the tests with single and multiple motor primitives suggest that conclusion.

An automatic ball thrower is necessary in order to achieve a more rigorous testing setup.
Conclusions and Future Work

Contents

6.1 Conclusions ................................................. 63
6.2 Future Work ................................................. 63
In this chapter we review all the work done and the results achieved, and discuss possible directives for future work.

6.1 Conclusions

This work addressed the problem of robotic table tennis using a quadrotor with a light cardboard racket and an onboard camera, using as little hard-coded models of the world as possible. This represents a step back from the most recent line of work of the area, which mostly focused on the estimation/decision component of the task, while relying on precise manipulators for control and accurate stereo vision systems for perception. By using a considerably less precise robot with a significantly less accurate perception system, we faced new and interesting challenges.

To overcome these challenges, we successfully adapted an architecture designed for accurate hardware. This architecture comprises a two-steps learning process, where imitation learning is used to gather initial knowledge from a human expert, and reinforcement learning is applied to refine what was learned.

Our system achieved very satisfactory results, with hitting rates of about 30% in a simulation environment, and 20% with a real quadrotor. These results show that the learning processes are able to deal, to some extent, with the inherent errors associated with the vision system and the open-loop controller used. The errors from the vision system are successfully handled because they are consistent. Thus, errors observed while learning are approximately the same as the ones verified during the testing phase. Regarding the controlling errors, these were surpassed by the exploratory nature of the learning algorithm used to estimate the hitting point. In fact, the player explores the solution space and quickly learns how to successfully hit the ball.

Additionally, we were able to fulfill requirements of modifiability and testability, with the logical and physical architectures developed. These architectures allow the system to be easily extended in the future, by the addition of new functionalities or modification of the current ones.

6.2 Future Work

The system developed presents a good performance given the limitations imposed by the robot and its sensors. However, there is still room for improvement. Additionally, our work raises several interesting research problems that can be addressed in future work.

First of all, the vision system can be further improved by using a Kalman filter with minimal models of the world. As discussed, the algorithm proposed for ball pose estimation relies heavily on the number of pixels that the ball occupies on the image. Due to not perfect lighting conditions, this measure may change during the trajectory of the ball, making it possible for the ball to look at a distance \( d \) at time \( t \), and at distance \( d_{t+1} < d \) at time \( t_{t+1} \). A simple model to avoid this would be to assume that the pixel size of the object may never decrease. Even though this adds hand-coded knowledge to the system its impact in computational performance is minimal and would lead to smoother position estimates.

There is also room for improvement on the estimation algorithm. For example, it would be interesting to develop a CrKR algorithm for each meta-parameter component, using Gaussian Kernels that do not depend on all components of the perceived state, but only on those that matter for the specific meta-parameter component.

Additionally, we deployed and conducted some initial experiments with a visual-servoing module to return the quadcopter to an initial position, at the end of the table. Such a module allows to execute a trajectory without explicitly relying on any position knowledge, neither relative nor absolute. It works by
simply trying to reduce the distance of selected features in the image, between the current image and a reference one.

While our initial experiments still require some work, they already suggest the existence of an interesting and research worthy problem—the conjugation of both visual-servoing and DMPs. Since visual servoing only considers the current error between images, there would probably need to exist a second input. It would be interesting to apply this approach to our system, as it reduces the dependency on position information, which as previously discussed, is hard to compute with good precision, given the sensorial information available to the quadrotor.

Finally, there is interesting work to be done regarding the imitation learning phase of our setup. As previously discussed, humans find the keyboard controller more comfortable to use than the mobile application, when it comes to hitting ping pong balls. However, keyboard controllers are still not the most natural way to have a human demonstrating a table tennis hitting movement. Additionally, the movements demonstrated end up being a little artificial, as it is hard to model the velocity’s variations throughout the movement. Therefore, future work could explore the use of a Wii Remote controller, or similar device, to demonstrate trajectories. Since this remote controller can be held like a real table tennis racket, it may allow a more natural movement, where variations of velocity can be better recorded.
Bibliography


Simulation Environment
Figure A.1: Illustration of the simulation world developed. (a) presents in detail the standard sized table tennis table. (b) depicts an overall view of the simulation world.

A.1 Simulator

For this work, Gazebo was the simulator chosen. Gazebo is the recommended simulator to use alongside with ROS as they can be easily integrated. This easy integration is due to the publish subscribe architecture that ROS enforces. Once again, this proves to be a good decision.

The decision to use Gazebo was heavily influenced by this factor, as it allows the usage as-is of all ROS nodes developed, without having to proceed to any change whatsoever. Gazebo also offers all the functionalities we could need, like easy object modeling, a physics engine or a GUI interface. Additionally, there were already some research works that made use of this platform. More specifically, there is a package called Tum Simulator that already offers a Parrot Ar.Drone 2.0’s visual and physics model.

A.2 Simulation World

To test the table tennis task, a simulation world was developed. This world is composed by a standard sized ping pong table and table tennis ball, both in accordance to ITTF rules. Additionally, a racket was attached to the quadcopter model provided by the Tum Simulator package. Figure A.1 depicts the developed world, and Figure A.2 provides a closer look at the racket attached.

World Dynamics

To mimic as realistic as possible the dynamics of a ping pong ball, Gazebo’s physics engine was used. More specifically, the physics engine allows the parametrization of restitution and friction coefficients, as well as air drag effects. We followed some of the values experimentally determined by Mülling [21]. For the ping pong ball and table interaction the coefficients used were $\varepsilon_T = [\varepsilon_{T_x}, \varepsilon_{T_z}] = [0.73, 0.92]$, where $\varepsilon_{T_x}$ is the coefficient of friction of the ball on the table and $\varepsilon_{T_z}$ is the coefficient of restitution. The ping pong ball and racket interaction has a coefficient of restitution of $\varepsilon_R = 0.78$. 
The ping pong ball's velocity damping had to be a linear value, because that was the only option provided by Gazebo’s physics engine. The damping value was determined empirically, was chosen to be $\varepsilon_D = 0.000625$. This translates to a 0.000625% decrease of the ping pong ball's velocity at each time step. For simulation purposes, a 0.001 seconds time step was used.