Università degli Studi di Padova

Department of Information Engineering
Master Degree in Automation Engineering

Master Thesis

Study of Control Strategies for Robot Ball Catching

Candidate:
Nicolo' Rizzoli

Supervisor:
Ch.mo Prof. Ruggero Carli

Supervisors at Universidade de Aveiro:
Prof. Filipe Miguel Teixeira Pereira da Silva
Prof. Paulo Miguel de Jesus Dias

A. A. 2017–2018

Abstract

Human-Robot Interaction is an important and challenging part of robotics as it requires high accuracy and sophisticated technology, along with safety. It can be used for testing and evaluating advanced robotic technologies. The evaluation of vision and robotic systems is done with the ball catching using a robotic arm, task that requires high velocity and precision in a small time. Playing ball catching between a human and a robot is an example of such a form of safe interaction, not involving physical contact.

The main goal of this thesis is focused towards the study of a possible scenario for ball catching task by a robotic manipulator using off-the-shelf technologies. In the pursuit of that objective, two main problems are addressed: to study different strategies to control the robotic arm to catch the ball (predictive and prospective control); to implement a simulator in ROS that acts as a real robot, that include a vision system for ball detection and tracking using a Microsoft® Kinect sensor. The study of control strategies is supported by MATLAB®, while the simulator development uses Robot Operating System (ROS) with open-source platform, distributed architecture and C++ language programming. Several simulation tests are conducted to validate the proposed solutions and to evaluate the system's performance in various situations, using data obtained from both Microsoft® Kinect, both mathematically.

Contents

List of Figures

List of Tables

To my family and friends, always with me...

Chapter 1

Introduction

The interception of a moving object along its trajectory is an inherently challenging task given the spatiotemporal constraints for a successful accomplishment, requiring a tight interplay among visual, planning, control and mechanical systems in order to "get the hand to the right place at the right time" [41]. For example, catching a flying ball involves bringing the hand to the interception point at the right time, adjusting the hand posture to receive the incoming ball and closing the hand to ensure a stable grasp. A failure in catching the ball may result from a small error in any of these actions.

The generation of efficient and skilled interceptive actions, either in humans or robots, involves the search for solutions to several problems, including delays, noise, uncertainty and redundancy. Delays are present in all stages of the control system since the robot system only has access to out-of-date information about the world. Noise in both sensors and actuators limits the ability to perceive accurately and act precisely. Furthermore, sensory noise contributes to variability in estimating the ball's trajectory and, consequently, contaminates the planning process leading to variability of movement endpoints. The notion of "uncertainty" occurs in several meanings whenever referring to an incomplete knowledge of a quantity, to the experimental inaccuracy with which it is measured or to some ambiguity in its definition. The uncertainty either about the state of the world or of the task plays is an important factor in making the control problem difficult.

In addition to the aspects already mentioned, there are multiple possible solutions to accomplish the task. For example, the trajectory of a flying ball can be intercepted by the catcher's hand at an infinite number of spatial positions within a given temporal window. Moreover, each spatial location can be reached by means of many hand motions and joint trajectories. When intercepting a moving object along its trajectory, redundancy in the spatial locations and in the timing of interception may be reduced or exploited, depending on the specific control strategy and the particular task constraints. In the same line of thought, the control of the end-effector motion may be the result of a trade off between spatial accuracy (decreasing with the end-effector speed) and temporal accuracy (increasing with end-effector speed), as well as a trade off between variability due to sensory noise and variability due to motor noise. This dissertation was proposed by the Institute of Electronics and Informatics Engineering of Aveiro (University of Aveiro) in the scope of current activities aiming to design and evaluate robotic systems for human-robot interaction and for advanced studies on robot learning. The main focus of these projects is the use of off-the-shelf components integrated in a development environment supported by the Robot Operating System (ROS).

1.1 Motivation

The motivation behind this work is based on the observation that the new generation of robot systems will be increasingly present in our daily lives and engaged in new challenges tasks involving the interaction with humans. Moreover, a considerable part of the robotics market begins to be dominated by low-cost collaborative manipulators and off-the-shelf hardware components, what dictates a compromise in their accuracy and repeatability. This work aims to provide advances towards a robotic system able to perform a safe and entertaining task that directly engages a robot with a human partner: catching a flying ball. In line with this, the design of appropriate solutions to the different problems encountered should take into account the specific capabilities and limitations of the used hardware. The intention is to use a system consisting of a Robai Cyton manipulator arm and a Kinect depth sensor. Another motivating factor has been the existence, from the viewpoint of human studies, a dichotomy between predictive and prospective ball catching. The former control strategy relies on determining in advance a ballistic model of the object's motion. The later strategy is based on a close coupling between visual information and movement described by a feedback control law.

1.2 Objectives

This dissertation addresses both theoretical questions related with the strategy for intercepting the ball and experimental developments associated with the design of a testbed for a robotic ball-catching task. The main objectives are twofold: first, to evaluate different strategies for allocating time for perception and action that might increase the chances of a successful catch. The primary motivation is to gain insights on key problems associated with this challenging task, leading to novel implementations and a more effective robot motion visually guided.
The second central objective is to develop a platform for conducting research on advanced topics, such as optimal state estimation, robot learning or prediction-based control. For that purpose, a set of computational tools need to be integrated in a coherent architecture, aiming to provide a testbed for conducting replicable testing of new algorithms and technologies in a well-defined scenario. Although having in mind an application with a physical robot, the use of a virtual environment provides a powerful way for simulating and training the real situation, while avoiding the tedious, expensive and often hard to reproduce experience on a real physical robot. The simulated arm system is based on a real model. The intention is to have all the topics and nodes ready to be used with the real physical arm when it will be available (Cyton Gamma 1500 or another industrial manipulator). In this way, topics can be easily swap between simulated and real robot. This is a change done during the work, due to problems with the real Cyton Gamma 1500 available in the laboratory in University of Aveiro, that cannot be fixed before the deadline of Erasmus+ Programme.
At the current stage of development, computer simulations were adopted to allow the easy and quick test of many ideas for the same problem, test them and then decide which one will be implemented on the real robotic system. For that purpose, the robot software development is supported by the Robot Operating System (ROS) framework. Thus, the application developed in a ROS-enabled simulation environment can be deployed to the physical robot without significant modifications in the code.

1.3 Dissertation Structure

This dissertation is organized into six chapters: the Introduction of the work (Chapter 1), including the topics studied, is followed by the state-of-the-art in Chapter 2, that introduces the latest advances in ball catching tasks with various works. It also includes the experimental setup that explains hardware and software, from PC, robot and vision sensor to ROS, MATLAB® and C++.
The analysis of the robot with its inverse kinematics, control, singularities and manipulability are in Chapter 3.
Chapter 4 is one of the most important: it analyzes the possibles approaches (predictive and prospective) with simulations in MATLAB®. The results of the latter Chapter are implemented in ROS, with the explanation in Chapter 5, including vision, ball detection and tracking.
Chapter 6 regards the conclusions of the dissertation with future works to improve it, followed by Appendix A with mathematical derivations and Appendix B, with instructions to install packages and get the systems ready to work.

Chapter 2

Background and Context

This chapter provides a review of the literature and the context of the dissertation work. Section 2.1 presents an overview of human studies with particular emphasis on two different approaches in which the control strategies can be divided: predictive strategies relying on ballistic models and prospective strategies relying on continuous visual feedback [9]. From the viewpoint of robotics, numerous approaches for robot ball catching have been developed with their own solutions to the problems of perception and action. Section 2.2 briefly discusses the most relevant works found in the literature, focusing on how they predict the trajectories of the moving object and how they generate the robot's motions. Section 2.3 regards previous studies that are described within the experimental setup and summarizes the proposed approach to achieve the objectives outlined in Chapter 1. Finally, Section 2.4 provides the context for the work, with description of development phases and the main assumptions.

2.1 Overview of Human Studies

A subject that is relevant for this dissertation is the study of human ball-catching, namely for explaining control strategies used for catching flying objects. Currently, there is a dichotomy between predictive approaches, also referred to as model-based approaches or ballistic control, and prospective approaches, also referred to as online approaches or feedback control. On the one hand, predictive ball-catching relies on a ballistic model of the ball's motion, estimated from initial observations, to make a prediction of the most probable catching point and time. A predictive strategy seems to be necessary whenever the motion commands have to be started well before the point of contact, i.e., when the time of flight of the ball is short compared to the reaction time of the catcher. For example, this is typical for batting in baseball or cricket ([9]; [23]). On the other hand, prospective strategies are based on continuous feedback from the task performed without explicit need for a model of the object's motion. Instead, motion commands aim to minimize the distance or relative velocity between the object and the catcher. This type of strategy is viable when there is enough time to perform several corrections before the point of contact, being useful when the trajectory is difficult to predict from initial conditions, such as for outfield players in baseball [9].
A continuous debate exists to better understand whether humans use prospective or predictive control to intercept an object falling under gravity ([3]; [52]; [17]). Prospective control involves using continuous information to regulate action. The ratio of the size of the gap to the rate of gap closure (tau, τ) has been proposed as the information used in guiding interceptive actions prospectively [24]. Movement modulation is expected to be generated where variability decreases over the course of an action given the more accurate timing information. In contrast, predictive control assumes that a pre-programmed movement is triggered at an appropriate time instant. For a falling

object, it is commonly argued that an internal model of gravitational acceleration is used to predict the motion of the object and determine movement initiation. When catching a ball at a moderate velocity in the range $5 - 6\,\mathrm{m/s}$, the time for the entire throw is less than 1 s. In these cases, there does not seem to be enough time for a voluntary continuous feedback loop to correct the hand position. Instead, an initial estimate of the ball's trajectory seems to be made such that the hand moves roughly toward the point of interception. This position can be corrected one or more times to produce an accurate catch whenever there is time to react [13]. Evidence towards this is that one can distinguish a distinct trajectory towards an initial predicted catching position and additional shorter distinct trajectories that correct the initial one.
Katsumata & Russell (2012) [17] attempted to test predictive and prospective control strategies by disrupting visual information of a falling ball and by examining consistency in movement initiation and duration. The participants in the study were asked to bat a ball dropped from three different heights (1, 1.3 and 1.5 m), under both full-vision and partial occlusion conditions. The most important findings provide evidence for predictive control in initiating the swing and prospective control, based on τ, in guiding the bat to intercept the ball. More recently, the studies of Zhao & Warren [54] and de la Malla & Lopez-Moliner [27] also investigated the combination of predictive and online visual information throughout the perception-action cycle.
Other studies have investigated the human gaze behavior during ball-catching (Cesqui et al. 2015 [6]; Lopez-Moliner & Brenner [26]). In ball games, humans do not direct ones gaze at the ball all the time, since other aspects of the game need to be judged, such as the other players' position and movement. Lopez-Moliner and Brenner [26] studied whether there are times at which obtaining information about the ball is particularly beneficial for catching it if they have to perform a secondary task. Authors verified the existence of a notable flexibility in using information at any relevant time, meaning that one must determine when visual information would be useful for any particular task. Faisal and Wolpert (2009) [11] studied how subjects trade-off sensory and movement uncertainty by deciding when to initiate their actions. The task was formulated in a probabilistic framework aiming to understand the trade-off between the time allocated to perception and action: uncertainty in perceptual estimates decreases with a longer perception phase, but it leaves less time for action resulting in less precise movements. Findings revealed that the decisions about when to start moving are statistically near optimal given their individual sensory and motor uncertainties. Since movements are often guided by multiple sensory inputs with different uncertainty or variability, it has been suggested that optimal utilization of sensory inputs can minimize the effects of noise on the movement.
It appears that the acquisition of perceptual-motor skills involve not only refinement of information extraction, but also the progressive use of earlier relevant sources of information. Since moving to catch a ball takes time, it is presumably advantageous to make predictions well before the catch and refine them as the ball approaches. Initially, visual information results from observation of the thrower's arm movement before the ball is released. At the end, the catcher observes the ball's flight shortly before the ball is caught. In the same line of thought, evidence suggests that manipulating participants access to earlier information modifies their hand movements and gaze behaviors. Stone et al. (2015) [49] manipulated participant access to earlier information of a thrower's actions and from ball flight, while recording whole body kinematic and kinetic data to investigate effects on postural control during performance of interceptive actions. Twelve participants attempted to make or simulate performance of one-handed catches in three experimental conditions: only thrower's

actions, only flying ball and both advanced visual information and ball in flight. Findings revealed that movements were initiated earlier when advanced visual information was available prior to ball flight, resulting in more controlled actions and superior catching performance.

2.2 Robot Ball-Catching

From the viewpoint of robot ball-catching, numerous approaches have been developed with their own solutions to the problems of perception and action. In general, a ball catching task requires solving three challenging problems:

1. to detect and track the ball's trajectory;
2. to accurately predict the ball's trajectory for determining the catching position and time;
3. to adapt adequately the trajectory planning of the robot arm in order to intercept and catch the ball on time.

The pioneering work described in Hove and Slotine (1991) [15] and Hong and Slotine (1995) [14] used the 4-DOF "WAM" arm equipped with a gripper and an active vision system. The catch point selection is based on the the closest point of the ball's trajectory to the robot base, while the gripper is oriented perpendicular to the trajectory. A Cartesian path is generated as a 3rd order polynomial and executed relying on an inverse kinematics method running in the control loop. Nishiwaki et al. (1997) [34] presented a 5-DOF arm attached to a humanoid robot with a basket at the end-effector for catching the ball and an active vision system. The inverse kinematics is solved by a neural network that should provide a human-like behavior. A robotic ball catcher consisting of the 7-DOF DLR-LWR-II arm with a small basket at the end-effector and an off-the-shelf stereo vision system is described in Frese et al. (2001) [12]. The catching point is calculated with a heuristic assuming two goals:

1. to choose a point near the robot for reaching the target point on time;
2. to choose a point far away from the robot for avoiding joint limits.

An inverse kinematics algorithm computes the catch configuration, taking into account a perpendicular orientation of the catching basket with respect to the ball's trajectory. Finally, an interpolator generates the joint trajectories based on a trapezoidal velocity profile.
Riley and Atkeson (2002) [42] investigated the use of motion primitives for human-like path generation. A humanoid robot arm equipped with a baseball glove is used in order to determine the catching point, but without considering the end-effector's orientation. In this work, the catching position is chosen to be the intersection of the ball's trajectory with a horizontal plane at a given height. An inverse kinematics algorithm computes the catching configuration. Bäuml et al. (2011) [2] used the mobile humanoid robot Rollin' Justin for catching up to two simultaneously thrown balls with its hands. All DOF are used for the reaching motion, including the arms, torso and mobile platform, while the head joints are used to keep the ball in the cameras' field of view. The system operates completely wireless using only onboard sensing and an external compute cluster for path planning coupled by WLAN. Very few works address the monocular case in ball detection and trajectory estimation. Cigliano et al. 2015 [7] presented a robotic ball catcher employing only a single moving camera and coping with rolling, bouncing and flying balls in the same framework

without changing either the estimator or the control law. Authors use an industrial manipulator arm equipped with a CCD camera mounted directly on the manipulator end-effector. Therefore, the camera is also moving during the tracking/estimation since it is mounted in an eye-in-hand configuration
Most works model the trajectory of the flying ball as a parabola and, subsequently, they predict the ball's trajectory recursively through least squares optimization (Zhang and Buehler, 1994 [53]; Hong and Slotine, 1995 [14]; Riley and Atkeson, 2002 [42]). Another aspect that is also addressed is the incorporation of air drag with the ballistic model (Frese et al., 2001 [12]). The same authors apply a common solution to determine the desired catching point: they intercept the robot's reachable-space with the ball's trajectory in order to find the closest point to the current position of the end-effector. In what concerns the generation of the end-effector's trajectories, many researchers represent them by using polynomials satisfying boundary conditions. For example, Namiki and Ishikawa (2003) [32] minimize the net torque and angular velocities to satisfy constraints on the initial and final position, velocity and accelerations of the end-effector. Learning human demonstrations is another approach for trajectory planning (Riley and Atkeson, 2002 [42]; Park et al., 2009 [40]; Kim et al., 2010 [18]). In order to complete the catch, the closure of the fingers should be fast and precise. Usually, finger closure is triggered as soon as distance between the object and the end-effector is below a given threshold (Hong and Slotine, 1995 [14]; Lampariello et al., 2011 [22]; Namiki and Ishikawa, 2003 [32]; Riley and Atkeson, 2002) [42]. Bäuml et al. (2010) [1] proposed to solve a nonlinear optimization problem that minimizes the robot's joints accelerations during the catching task.
More recently, addressing the main challenges of the task demands the implementation of learning abilities in robots. Kim et al. (2014) [19] presented a learning framework to teach a robot to catch objects in flight either through observation of human demonstrations or through exploration. Authors consider the problem of catching fast objects with uneven shapes using time invariant dynamical systems as a method for encoding robot motions. Based on the same framework for extracting feasible postures to intercept general shaped objects, Salehian et al. (2016) [47] used a dynamical system (DS) based control law to generate the appropriate reach and follow motion. These authors proposed a method to approximate the parameters of Linear Parameter Varying (LPV) systems using Gaussian Mixture Models, based on a set of feasible demonstrations generated by an off-line optimal control framework.

2.3 Proposed Approach

This section describes the proposed approach for the thesis.

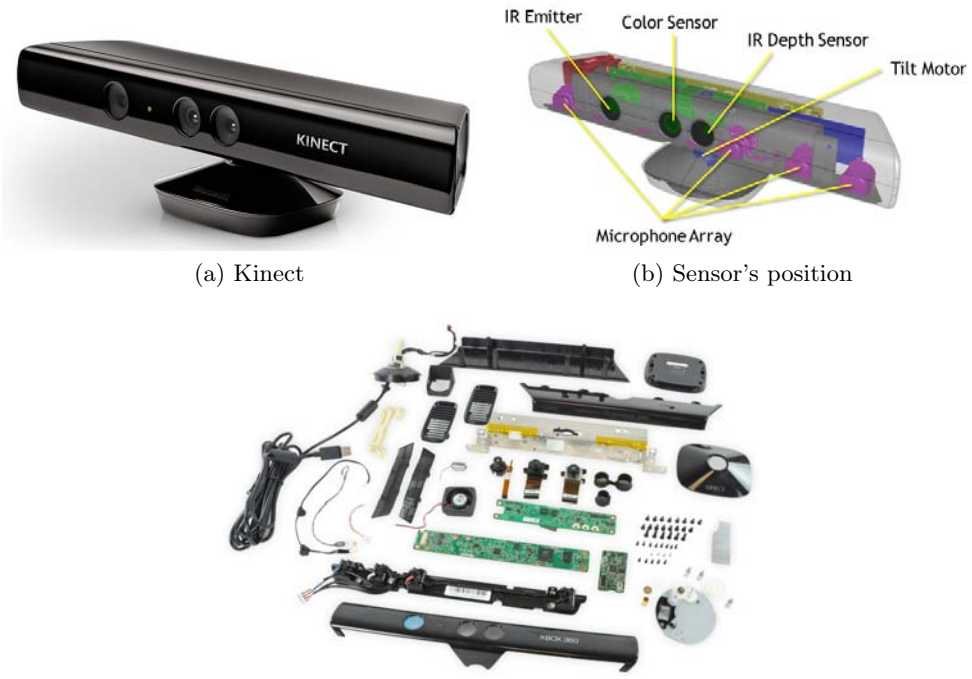
2.3.1 Context and Scope of the work

This dissertation was proposed in the context of current activities aiming to design and evaluate robotic systems for use by or with humans. An objective of the study was the use of off-the-shelf components, to make it possible to replicate the setup with a minimum effort. Two previous studies ([31]; [45]) provide an important contextualization of these earlier activities. The most recent and directly related [45] sought to develop a testbed for a ball catching task involving an upper-body humanoid robot and a human partner. The study focused on the development of the hardware and software infrastructures by employing off-the-shelf components, namely an educational/research manipulator Cyton Gamma 1500 [43] and a Kinect depth sensor [21].
The development environment is supported by the Robot-Operating System [44]

framework under Linux, using C/C++ programming language. Several computational tools have been developed in previous works, including detection and tracking algorithms, estimation methods based on Kalman filtering, planning and simple point-to-point motion control of the dual-arm humanoid torso. This section describes the main hardware and software components used throughout these earlier studies, the most important findings and their limitations, that has been outlined in this work.

2.3.2 Vision Sensor: Microsoft® Kinect

The Microsoft® Kinect Sensor [21] is a well known camera that implement a depth sensor and a RGB camera together in a compact box.
Kinect is a line of motion sensing input devices by Microsoft for Xbox 360. The first-generation Kinect was first introduced in November 2010, as a combination of Microsoft built software and hardware. Kinect V1 hardware included a range chipset technology by Israeli developer PrimeSense, which developed a system consisting of an infrared projector and camera and a special microchip that generates a grid from which the location of a nearby object in 3 dimensions can be determined. This 3D scanner system called Light Coding employs a variant of image-based 3D reconstruction.



(a) Kinect (b) Sensor's position

(c) Kinect exploded view

Figure 2.1: Microsoft® Kinect Sensor

The Kinect sensor, in Figure 2.1, is a horizontal black bar connected to a small base with a motorized pivot, ready to be positioned on a plane. The device features an RGB camera, depth sensor and multi-array microphone running proprietary software, which provide full-body 3D motion capture, facial recognition and voice recognition capabilities. It also provides a 3-axis accelerometer and comes with a power supply cable.
Inside the case, a Kinect sensor contains:

- **RGB camera**, that stores three channel data. This makes capturing a color image possible. The default RGB video stream uses 8-bit VGA resolution (640×

480 pixels) with a Bayer color filter, but the hardware is capable of resolutions up to 1280 × 1024 at a lower frame rate.

- **Depth sensor**, including an infrared (IR) emitter and an IR depth sensor. The emitter emits infrared light beams and the depth sensor reads the IR beams reflected back to the sensor. The reflected beams are converted into depth information measuring the distance between an object and the sensor. This makes capturing a depth image possible. The depth sensor consists of an infrared laser projector combined with a monochrome CMOS sensor, which captures video data in 3D under any ambient light conditions, with adjustable sensing range. The monochrome depth sensing video stream is in VGA resolution (640 × 480 pixels) with 11-bit depth, which provides 2048 levels of sensitivity. The Kinect can also stream the view from its IR camera directly (i.e. before it has been converted into a depth map) as 640 × 480 video, or 1280 × 1024 at a lower frame rate.

- **Multi-array microphone**, which contains four microphones for capturing sound. It is possible to record audio as well as find the location of the sound source and the direction of the audio wave using the four microphones. Each channel processes 16-bit audio at a sampling rate of 16 kHz.

- **3-axis accelerometer**, configured for a $2g$ range, where g is the acceleration due to gravity. It is possible to use the accelerometer to determine the current orientation of the Kinect.

The main Kinect specifications are resumed in Table 2.1.

Item	Specification
Viewing angle	43 ° vertical by 57 ° horizontal field of view
Vertical tilt range	±27 °
Depth stream	QVGA (320 × 240) at 30 frames per second (fps)
Color stream	VGA (640 × 480) at 30 frames per second (fps)
Audio format	16 kHz, 16-bit mono pulse code modulation (PCM)
Depth distance	from 0.8 m to 4.5 m

Table 2.1: Kinect Specifications

Kinect streams out color, depth, and skeleton data for each frame and can interact in a defined area with some limitations depending on the environment where it is used:

- **Color Space**: the color sensor captures a color image of everything visible in the field of view of the color sensor. A frame is made up of pixels. The number of pixels depends on the frame size. Each pixel contains the red, green, and blue value of a single pixel at a particular (x, y) coordinate in the color image.

- **Depth Space**: the depth sensor captures a grayscale image of everything visible in the field of view of the depth sensor. A frame is made up of pixels and each pixel contains the Cartesian distance, in millimeters, from the camera plane to the nearest object at that particular (x, y) coordinate. The (x, y) coordinates

of a depth frame do not represent physical units in the room; instead, they represent the location of a pixel in the depth frame.

- **Skeleton Space**: the depth image captured is processed by the Kinect runtime into skeleton data. Skeleton data contains 3D position data for human skeletons for up to two people who are visible in the depth sensor. The position of a skeleton and each of the skeleton joints (if active tracking is enabled) are stored as (x, y, z) coordinates. Unlike depth space, skeleton space coordinates are expressed in meters. The x, y, and z-axes are the body axes of the depth sensor with a right-handed coordinate system that places a Kinect at the origin with the positive z-axis extending in the direction in which the Kinect is pointed. The positive y-axis extends upward, while the positive x-axis extends to the left. Placing a Kinect on a surface that is not level (or tilting the sensor) to optimize the sensor's field of view can generate skeletons that appear to lean instead of be standing upright. Kinect is capable of simultaneously tracking up to six people, including two active players for motion analysis with a feature extraction of 20 joints per player. However, PrimeSense has stated that the number of people the device can see (but not process as players) is only limited by how many will fit in the field-of-view of the camera. In the thesis, only one person is tracked for hand recognition.

- **Interaction Space**: the interaction space is the area in front of the Kinect sensor where the infrared and color sensors have an unblocked view of everything in front of the sensor. If the lighting is not too bright and not too dim, and the objects being tracked are not too reflective, it is possible to get good results tracking human skeletons. While a sensor is often placed in front of and at the level of a user's head, it can be placed in a wide variety of positions. The interaction space is defined by the field of view of the Kinect. Kinect sensor records video at a frame rate from 9 Hz to 30 Hz, depending on resolution. It has a practical ranging limit of 0.8 m − 4.5 m, that is better at 1.2 m − 3.5 m distance. The area seen by Kinect is roughly 6 m^2, although the sensor can maintain tracking through an extended range of approximately 0.4 m − 6 m. Figure 2.2 represents the working range. The horizontal field of the Kinect sensor at the minimum viewing distance of ∼ 0.8 m is therefore ∼ 87 cm, and the vertical field is ∼ 63 cm, resulting in a resolution of just over 1.3 mm per pixel.

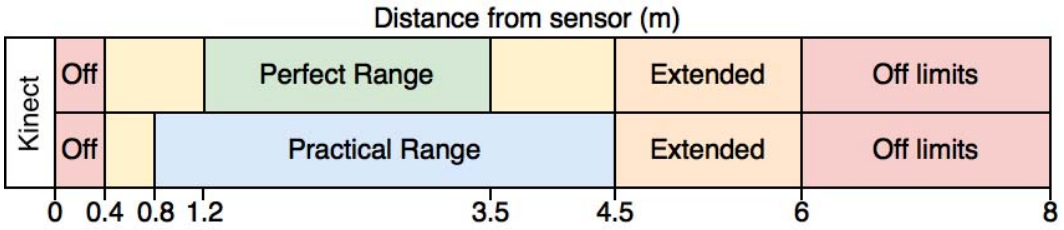


Figure 2.2: Working range

The Kinect sensor's motorized tilt mechanism requires more power than the USB ports can supply. Therefore, the device makes use of a proprietary connector combining USB communication with additional power, requiring the included power supply cable that splits the connection into separate USB and power connections; power is supplied from the mains by way of an AC adapter.

Instruction for the installation are available in Appendix B.3.
The following tips help to get started using the Kinect:

- Connect the power supply for the Kinect to an external power source; if the Kinect has only power from the USB connection, it will be minimally functional and light the LED, but it must be connected to an external power source to be fully functional.
- The Kinect is protected from overheating by a fan, controlled by the sensor's firmware, which turns off the camera at 90 °C.
- Reasonable lighting, neither extremely dark nor extremely bright, is important for capturing images with the RGB camera. Incandescent, fluorescent, and natural lighting provide no special obstacles, but do not point an intense or constant light source at the camera because this can blind the RGB sensor.
- The depth sensor functions adequately in typical and reduced lighting, although in near darkness there is increased noise in the signal.
- The depth sensor reads depth information from reflected light. Objects that are highly reflective (mirrors and shiny metal) or highly absorptive (fluffy and/or dark materials) may not be registered by the depth sensor as successfully as other objects.
- When the Kinect is perfectly connected to the PC, it is recognized as 3 devices: NUI Camera, NUI Motor and NUI Audio. Sometimes the PC can recognize only NUI Camera, but a simple plug and unplug solves the problem.

2.3.3 Robotic Arm: Cyton Gamma 1500

The robotic arm used is the Cyton Gamma 1500 by Robai Corporation [43]. The arm has 7-DOF and a motor to control the gripper. Due to that is difficult to compute the kinematics. It can be controlled both in position and velocity. All the joints are revolute and it is available a package that includes all the specification of the arm in ROS.
Humanoid robot arms, with many degrees of freedom, can reach around obstacles and through gaps, reconfigure for strength, and manipulate objects. These robots have kinematic redundancy, like that of the human arm, that enables placement of a hand or tool at a position and orientation in an unlimited number of ways. The Robai Cyton Gamma can perform advanced control by exploiting its kinematic redundancy. The overall structure and the dimension of each joint are in Figure 2.3. Among all the specifications, one of the most important in ball catching task are the joint velocities. The value is important to choose the best robot to use, because it is needed the highest velocity possible to catch the ball in a small time. The problem with higher velocity could be the robustness of the arm.
The specifications of Cyton Gamma 1500 are in Table 2.2, while joints limits are in Table 2.3 where (A) means Articulate, while (S) means Spin.
The Cyton Gamma 1500 requires an input voltage of $100-240$ V in AC and works with ambient temperature between 10 °C and 35 °C under normal atmospheric pressure conditions. Prongs can be chosen between standard or wide and there is a 3 finger hand as optional.
The thesis is based on this arm: the intention is to study the kinematics of the arm and to create a simulation considering the Cyton Gamma 1500 model, both in ROS

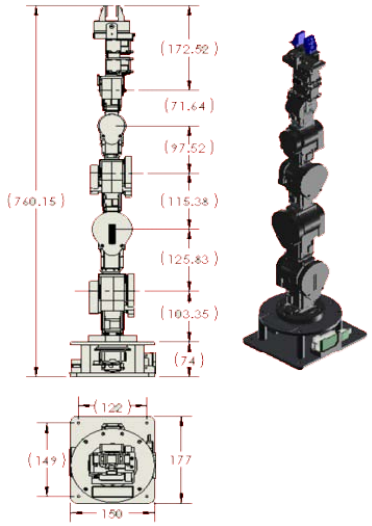


Figure 2.3: Cyton Gamma 1500

Specifications	Values
Total weight	3000 g
Payload at full reach	1200 g
Payload at mid reach	1500 g
Arm length	76 cm base to tip
Reach	68 cm
Maximum linear arm speed	45 cm/s
Maximum speed (free move)	70 cm/s
Repeatability	±0.5 mm
Gripper fingers	2 (opening range 3.5 cm)

Table 2.2: Cyton Gamma 1500: Specifications

Joint Name	Number	Displacement	Angle limits	Velocity limits
Shoulder Roll (S)	Joint 0	300 °	[−150 °, 150 °]	75 deg/s
Shoulder Pitch (A)	Joint 1	210 °	[−105 °, 105 °]	75 deg/s
Shoulder Yaw (A)	Joint 2	210 °	[−105 °, 105 °]	75 deg/s
Elbow Pitch (A)	Joint 3	210 °	[−105 °, 105 °]	65 deg/s
Wrist Yaw (A)	Joint 4	210 °	[−105 °, 105 °]	110 deg/s
Wrist Pitch (A)	Joint 5	210 °	[−105 °, 105 °]	330 deg/s
Wrist Roll (S)	Joint 6	300 °	[−150 °, 150 °]	330 deg/s

Table 2.3: Cyton Gamma 1500: Joint limits

and in MATLAB®. After these verifications, the idea is to move to the real arm using the nodes created for simulation. Problems with the robot does not allow the use of the real arm, therefore it is maintained the model in the simulation but it is not used in reality for this thesis.

2.3.4 Software Development Tools

The Robot Operating System (ROS) [44] is a software framework that provides a set of tools and libraries to ease the development of robotic applications.
One of its strong points is the support of nodes and their interaction via a network of topics, which can be published by nodes and subscribed to by other nodes. This helps in building a network of complex algorithms that each provides a part of the overall computations, each using the data that is sent via the topics, resulting in a robust design. This data can consist of sensor information, calculated environment details or planned tasks. Furthermore, ROS supports a wide range of sensors, varying from simple force, torque or touch sensors to 3D environment sensing sensors, like

range finders or cameras. The sensor-driver nodes handle interfacing with these sensors and send the sensors data and control commands through topics. Usually, a node written in C++ subscribes to a fixed set of topics, which are determined by using their data structures in the code. As a result the data structures of these topics are checked by the C++ compiler, resulting in a robust design. Due to its complexity and extensiveness, ROS is not capable to provide a hard real-time software environment. The arrival of data on a topic, the scheduled time of a node and so on cannot be guaranteed, as this depends on too many unknown factors. On the other hand, most of the time these things will go as expected or designed, making ROS suitable for soft real-time use.
It provides hardware abstraction, device drivers, libraries, visualizers, message-passing and package management. ROS is licensed under an open source, BSD license and is the unifying element of this project.
The distribution of ROS used is the latest available with long term support for Ubuntu Xenial (16.04 LTS), named ROS Kinetic Kame. The final release is available since May 2016 and will be update until May 2021, the same end of life date of Ubuntu Xenial. It includes various packages that allow the interaction among the PC, the Cyton Gamma 1500 and the Kinect Sensor.
Instruction for the installation of ROS in Ubuntu Xenial are available in Appendix B.1.
The primary goal of ROS is to support code reuse in robotics research and development. ROS is a distributed framework of processes (nodes) that enables executables to be individually designed and loosely coupled at runtime. These processes can be grouped into packages, which can be easily shared and distributed. ROS also supports a federated system of code, called repositories, that enable collaboration to be distributed as well. This design, from the filesystem level to the community level, enables independent decisions about development and implementation, but all can be brought together with ROS infrastructure tools. The other goals of ROS can be summarized in:

- **Thin**: ROS is designed to be as thin as possible, so that code written for ROS can be used and integrate with other robot software frameworks.
- **Language independence**: the ROS framework is easy to implement in any modern programming language. It is already implemented in Python and C++.
- **Easy testing**: ROS has a built in unit/integration test framework called rostest that makes it easy to bring up and tear down test fixtures.
- **Scaling**: ROS is appropriate for large runtime systems and for large development processes.

The ROS Master stores topics and services registration information for ROS nodes. Nodes communicate with the Master to report their registration information. As these nodes communicate with the Master, they can receive information about other registered nodes and make connections as appropriate. The Master will also make callbacks to these nodes when this registration information changes, which allows nodes to dynamically create connections as new nodes are run.
Nodes connect to other nodes directly; the Master only provides lookup information, much like a DNS server. Nodes that subscribe to a topic will request connections from nodes that publish that topic, and will establish that connection over an agreed upon connection protocol. The most common protocol used in a ROS is called TCP ROS, which uses standard TCP/IP sockets.

The two sides are decoupled. All the nodes publish without knowledge of whether anyone is subscribed. All the filters subscribe without knowledge of whether anyone is publishing them. The two nodes can be started, killed, and restarted, in any order, without inducing any error conditions.
An example of publish/subscribing between nodes and topics is depicted in Figure 2.4. This architecture allows for decoupled operation, where the names are the primary

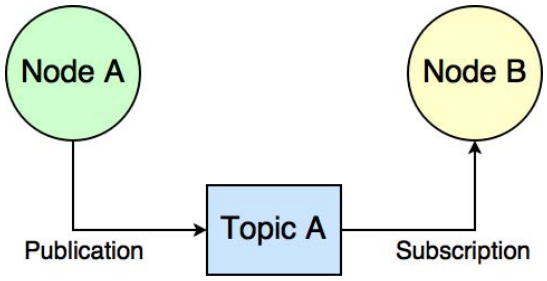


Figure 2.4: Nodes and topic publication and subscription

means by which larger and more complex systems can be built. Names of nodes, topics, services, and parameters are very important in ROS.

OpenCV

OpenCV (Open Source Computer Vision) [35] is a library of programming functions aimed at real-time computer vision. Originally developed by Intel's research center in Russia, it was later supported by Willow Garage (ROS organization) and is now maintained by Itseez. The library is cross-platform, free for use under the open-source BSD license and automatically downloaded with ROS.
Officially launched in 1999, the OpenCV project aim was to advance CPU-intensive applications, part of a series of Intel's projects including real-time ray tracing and 3D display walls. In the early days of OpenCV, the goals of the project were described as:

- Advance vision research by providing not only open but also optimized code for basic vision infrastructure.
- Disseminate vision knowledge by providing a common infrastructure that developers could build on, so that code would be more readily readable and transferable.
- Advance vision-based commercial applications by making portable, performance, optimized code available for free, with a license that did not require code to be open or free itself.

OpenCV is written in C/C++, but there are bindings in Python, Java and MATLAB®. All the new developments and algorithms in OpenCV are developed in the C++ interface. It is cross-platform and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications; the library can take advantage of multi-core processing. Enabled with OpenCL, it can take advantage of the hardware acceleration of the computer platform.
The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning

algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. The library is used extensively in companies, research groups and by governmental bodies.
OpenCV's deployed uses span the range from stitching street view images together, detecting intrusions in surveillance video in Israel, monitoring mine equipment in China, helping robots navigate and pick up objects at Willow Garage, detection of swimming pool drowning accidents in Europe, running interactive art in Spain and New York, checking runways for debris in Turkey, inspecting labels on products in factories around the world on to rapid face detection in Japan.
ROS uses both versions of OpenCV: 2.4 and 3.0. The C++ code needs only some changes to use it with both versions.

OpenNI and OpenNI tracker

OpenNI [36], where NI stands for Natural Interaction, is a non-profit organization and open source software project created by Primesense, Willow-Garage, Side-Kick, Asus and Appside in 2010. It is focused on the certification and the improvement in interoperability of natural user interfaces for Natural Interaction (NI) devices, such as Kinect Sensor. It also includes applications that use those devices and middleware that facilitates access and use of such devices.
PrimeSense, who was the company behind the technology used in the Kinect and founding member of OpenNI, shutdown the original OpenNI project when it was acquired by Apple on November, 2013. The OpenNI framework provides a set of open source APIs that are a standard for applications to access natural interaction devices. They support voice and voice command recognition, hand gestures but especially Body Motion Tracking.
OpenNI can be installed and used with ROS in easily way. NITE is a plug-in that works in OpenNI, developed by PrimeSense. It is proprietary and its sole function is to perform skeleton tracking.
The package OpenNI Tracker for Human Skeleton comes directly from OpenNI. It allows the tracking of a person's skeleton using a depth sensor. It also gives the positions, relative to the camera frame, of the person's head, neck, torso, shoulders, elbows, hands, hips, knees and feet. The OpenNI tracker broadcasts the OpenNI skeleton frames using *tf* transforms. For each body part, it will be considered its own coordinate frame, and the OpenNI tracker publishes the transformation necessary to convert a body part coordinate frame to the camera's coordinate frame.
After the user detection with automatic calibration, NITE or OpenNI tracker starts looking for its skeleton data (*tf* transforms of all joints of the body detected) with the user standing in front of the camera, showing to the sensor most of his body.

2.4 Overview of the Approach

The thesis has two main objectives: find the best approach to catch a ball and create a simulation of ball catching task in ROS where the simulator can be easily replaced by the real arm.

2.4.1 Development Phases

In order to pursue the main objectives of the work, two development phases were considered. In the first phase, the trade-off between the time allocated to perception and action will be addresses. The study and evaluation of different control strategies is based on MATLAB® simulations carried out on a powerful PC.

Predictive vs Prospective

The choice of predictive approach comes from previous works [45]. In that work, the comparison is between a least squares estimate and a prediction using Kalman Filter. The latter could be better in more trajectories but it is very difficult to initialize the matrix of the filter.
In this study, the first objective is finding a better selection of the landing point in the predicted trajectory, with the use of the least squares estimate. Thus, the comparison regards the choice of a landing point in a plane and with minimum distance from the end-effector. In this case, different robot's starting pose are used. The limitations come from the robot's joint velocities but also from the uncertainty given by the vision sensor. This work has to deal with this two limitations to find the best possible solution.
The idea of using Prospective Control comes to anticipate the robot movement that does not have to wait for a first admissible landing point. The difficulties encountered in tuning the parameters to setup this control are important, but something can be taken from prospective control. In fact, the way of acting is similar to the human behavior. In this context, where the work is related to entertainment games with interaction between human and robot, the first intention is to teach the same behavior of humans to the robot.

Simulation vs Reality

The second main objective of the thesis is implement the approach in ROS. The implementation is made in simulation, due to problems in the real robot available. Several nodes are created to accomplish the task. They subscribe and publish to topics that are used also by the real robot. In this way, it will be easy to reuse the nodes created for simulation with the real robot when it will be available. The simulation allows also to evaluate the study made with a model of the real robot, to confirm the solutions found in a simulated environment. With use of ROS visualization tools, it is possible to see the model of the real robot acting with a simulated ball, showing a ball thrown that is caught from the robot.

2.4.2 Assumptions

The important assumptions adopted for the proposed study are as follows:

- Air drag ignored;
- Orientation of the hand ignored;

- Catching of spherical objects (ball-like in which the center of mass coincides with the geometric center);
- Respecting physical constraints, such as joint physical limits, joint maximum velocities and accelerations.

2.4.3 Overall System's Architecture

In a second phase, the development of a testbed for robotic catching of a flying ball is based on the ROS platform and the RViz visualization tool. The overall system's architecture is illustrated in Figure 2.5.

The Laboratory 0.24 of IEETA in University of Aveiro includes the main components of this work that are:

- PC with Linux Ubuntu and ROS Kinetic
- Robotic arm: Cyton Gamma 1500 by Robai Corporation
- Vision system: Microsoft® Kinect Sensor

These three components together create the overall system that is represented in Figure 2.5.

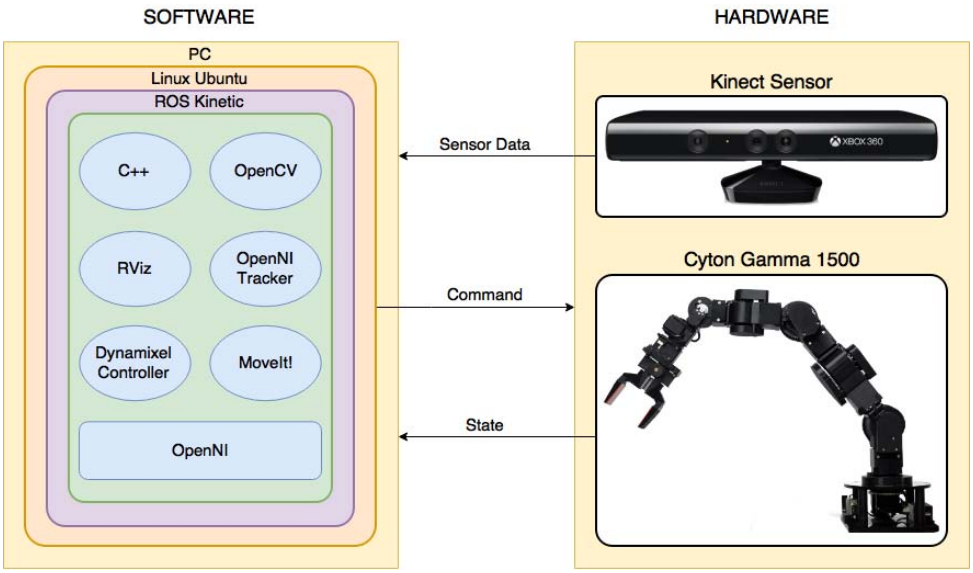


Figure 2.5: Overall System

The idea is to accomplish this task using compromise in performance and pricing. In fact, Cyton Gamma 1500 is available in University of Aveiro, Microsoft® Kinect Sensor is a depth sensor which holds a compromise among accuracy, sensing range and price and the PC is a requirement for each student. Software includes ROS in Linux Ubuntu, both available and downloadable for free. They come together with tools to code in C++ and with driver, like OpenCV for the vision sensor, OpenNI for the depth sensor to track people and RViz as visualization tools.

The ball catching task requires both software and hardware. It involves a powerful PC with software development tools to control the robotic arm, the actor of the task, and the vision system, that has to return information to guide the movement of the robot.

The testbed considers the overall system but with two main differences:

1. The Kinect data was first recorded and then played back, using ROSBag files (Section 2.4.3), for testing the algorithms;
2. A kinematic simulation is performed using Rviz, a 3D visualization tool (Section 2.4.3).

Thus, the computer simulations are used to validate the kinematic-control algorithms implemented, instead of using the physical Cyton arm and Kinect data in real-time.

PC with Linux Ubuntu

The first part of the project uses a PC, given by the University of Aveiro, with 2.67 GHz quad-core Intel Core i5 CPU 750 processor, 8 GB of RAM and Gallium 0.4 on NVA8 graphics. The OS is Linux Ubuntu 14.04 LTS (called also Trusty Tahr, released in April 2014) installed. The ROS version used in this PC is ROS Indigo Igloo, released in July 2014. This PC is the base of the project because it includes all the working packages for Kinect and Cyton.
The second part uses a laptop, Apple Macbook Pro, with 2.9 GHz dual-core Intel Core i5 processor, 8 GB of 2133 MHz LPDDR3 onboard memory of RAM and Intel Iris Graphics 550. The OS is Linux Ubuntu 16.04 LTS [25] (called also Xenial Xerus, released in April 2016) installed using Parallels Desktop® 12 for Mac software. The ROS version used in this PC is ROS Kinetic Kame [44], released in May 2016. These are the latest long term versions available at the moment of writing this thesis. This laptop is used to import the packages in the latest versions and to verify them with new codes and improvements in ball catching task. It is installed also MATLAB® version 2016*b* for simulation and evaluation of approaches.

MATLAB®

The MATLAB® platform [28] is optimized for solving engineering and scientific problems, using the matrix-based MATLAB® language that express computational mathematics, allowing matrix manipulations, plotting of functions and data, implementation of algorithms, in easy way. The language also provides features of traditional programming languages, including flow control, error handling, object-oriented programming, unit testing, and source control integration.
MATLAB® is used for its impeccable numerics. It supports both numeric and symbolic calculations, that makes it straightforward to capture the mathematics behind ideas, which means the code is easier to write, read, understand and maintain. It has a desktop environment tuned for iterative exploration, design, and problem-solving, with graphics for visualizing data and tools for creating custom plots. Integrated tools support simultaneous exploration of data and programs, letting evaluate more ideas in less time, while 2D and 3D plotting functions enable to visualize and understand data and communicate results.
It includes a vast library of prebuilt toolboxes with algorithms. These MATLAB® tools and capabilities are designed to work together, also with ROS. It is possible to interactively preview, select, and preprocess the data to import, while an extensive set of built-in math functions supports engineering and scientific analysis. With MATLAB®, it is possible to visualize how different algorithms work with data, and iterate until the results wanted are obtained.
This work uses MATLAB® version 2016*b* to test the strategies in the first part of the thesis, changing easily several parameters. In fact, MATLAB® is very useful to debug all the idea in a faster way, allowing to control the workspace and understand better and earlier the errors that can be made.

C++

All codes tested in MATLAB® are implemented in C++ programming language [5], one of the most important object-oriented programming languages. One of the main goals of this thesis is learning the basic skills to code using C++ programming language.
C++ was developed by Bjarne Stroustrup, a Danish computer scientist, since 1979 and released in 1985 when became the reference for the languages. It is an extension of the C language, adding it classes, to be efficient, flexible and provide high-level features for program organization. It is a compiled language, standardized by the International Organization for Standardization (ISO) with the latest version, named C++14. It maintains all aspects of the C language, while providing new features to programmers that make it easier to write useful and sophisticated programs.
A C++ program is a collection of commands, which tell the computer to do some action, usually called C++ source code or just code. Commands are either functions or keywords. Keywords are a basic building block of the language, while functions are, in fact, usually written in terms of simpler functions. Thankfully, C++ provides a great many common functions and keywords ready to use.
Every program in C++ has one function, named main, that is always called when the program first executes. From main, it is possible to call other functions whether they are written by the programmer or, as mentioned earlier, provided by the compiler. To access those standard functions that comes with the compiler, the *#include* directive includes the header. What this does is effectively take everything in the header and paste it into the program. C++ uses variables of different types that need to be declare in the program.
The compiler checks all the lines of the code and does not compile if there is an error. Debug requires a lot of time but it is needed to obtain useful results and to improve the code.

ROS

ROS contains different concepts. Among all, the most important are:

- **Packages**: the main unit for organizing software in ROS. A package may contain ROS runtime processes (nodes), a ROS-dependent library, datasets, configuration files, or anything else that is usefully organized together. Packages are the most granular thing to build and release.

- **Messages**: a message is simply a data structure, comprising typed fields. Standard primitive types (integer, floating point, boolean, etc.) are supported, as are arrays of primitive types. Messages can include arbitrarily nested structures and arrays. Message descriptions, stored in $my\_package/msg/MyMessageType.msg$, define the data structures for messages sent in ROS. Messages are routed via a transport system with publish/subscribe semantics.

- **Nodes**: processes that perform computation. Nodes communicate with each other by passing messages. ROS is designed to be modular; a robot control system usually comprises many nodes. For example, one node controls a laser range-finder, one node controls the wheel motors, one node performs localization, one node performs path planning, one node provides a graphical view of the system. A ROS node is written with the use of a ROS client library, such as *roscpp* in this project.

- **Topics**: a node sends out a message by publishing it to a given topic. The topic is a name that is used to identify the content of the message. A node that is interested in a certain kind of data will subscribe to the appropriate topic. There may be multiple concurrent publishers and subscribers for a single topic, and a single node may publish and/or subscribe to multiple topics. The idea is that publishers and subscribers are not aware of each other, to decouple the production of information from its consumption. Each bus has a name, and anyone can connect to the bus to send or receive messages, as long as they are the right type.

- **Bags**: a format for saving and playing back ROS message data. Bags are an important mechanism for storing data, such as sensor data, that can be difficult to collect but are necessary for developing and testing algorithms.

- **Distributions**: collections of versioned stacks that can be installed. Distributions play a similar role to Linux distributions: they make it easier to install a collection of software, and they also maintain consistent versions across a set of software.

- **Repositories**: ROS relies on a federated network of code repositories, where different institutions can develop and release their own robot software components.

- **ROS Wiki and Answers**: the main forum for documenting information about ROS. Anyone can sign up for an account and contribute their own documentation, provide corrections or updates, write tutorials. ROS Answers is a Q&A site for answering the ROS-related questions. Both are very useful to obtain help in low time.

RViz

RViz (ROS visualization) [46] is a powerful 3D visualization tool for ROS. It allows the user to view the simulated robot model, log sensor information from the robot's sensors, and replay the logged sensor information. By visualizing what the robot is seeing, thinking, and doing, the user can debug a robot application from sensor inputs to planned (or unplanned) actions. The window contains a list on the left with a check box for each item, where it is possible to show or hide any visual information instantly.
RViz displays both 3D sensor data from stereo cameras, lasers, Kinects, and other 3D devices in the form of point clouds or depth images, both 2D sensor data from web-cams, RGB cameras, and 2D laser range-finders.
If an actual robot is communicating with a workstation that is running RViz, RViz will display the robot's current configuration on the virtual robot model. ROS topics will be displayed as live representations based on the sensor data published by any cameras, infrared sensors, and laser scanners that are part of the robot's system. This can be useful to develop and debug robot systems and controllers. RViz provides a configurable Graphical User Interface (GUI) to allow the user to display only information that are useful.

Environment

The environment considered is the Laboratory 0.24 of University of Aveiro in IEETA (Institute of Electronics and Informatics Engineering of Aveiro). The robot stands

with a distance of $4 - 5\,\mathrm{m}$ from the person that throw the ball at almost the same height. A vision sensor, like Kinect, is above the robot with a clear and perfect view about what happen in front of it without obstacles. It can view the person and the ball.
An overview of the ball catching scenario with robotic arm, depth sensor and subject throwing the ball is depicted in Figure 2.6.

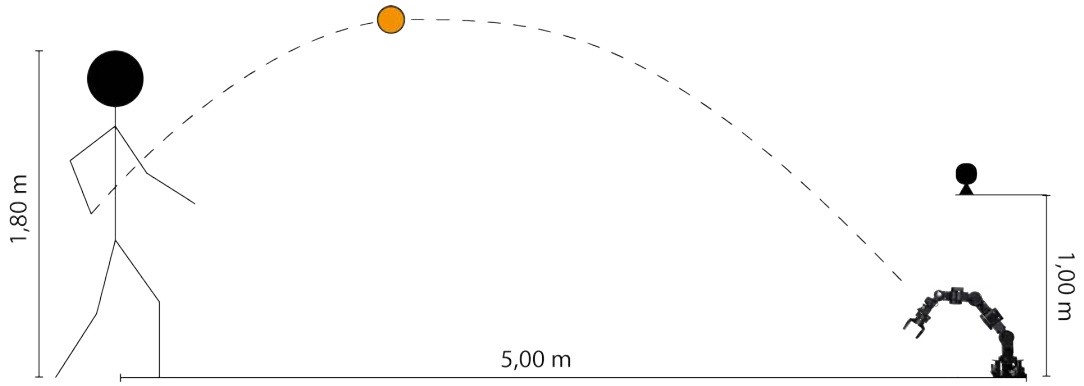


Figure 2.6: Ball Catching Scenario

Chapter 3

Robot Kinematics Analysis

This chapter includes the analysis of robot kinematics used in this work. It refers to the Cyton Gamma 1500, simplified to 3-DOF. It presents the study about forward and inverse kinematics, Jacobian, error correction, the singularity problem with some solutions and the manipulability measures.

3.1 3-DOF Cyton Gamma 1500

This work uses a Robai Cyton Gamma 1500, reduced to 3-DOF. From the previous works [31] [45], the Robai Cyton Gamma 1500 robotic arm with 7-DOF is simplified with the model in simulation using only 3-DOF. This is done because the simplified arm is controlled faster (e.g. 30 Hz against the 5 Hz of the 7-DOF version) and is more suitable for highly dynamic application like ball catching. Although it is less flexible, it is easier to implement and to compute the inverse kinematics.
The reduction selects 3 of the 7 joints available, precisely the Shoulder Roll (Joint 0 or q_1), Shoulder Pitch (Joint 1 or q_2) and Elbow Pitch (Joint 3 or q_3), from Table 2.3, maintaining the other joints at angle 0 ° and without updating either position or velocity.
The links' length of the 3-DOF are: $L_1 = 0.17735\,\text{m}$, $L_2 = 0.24121\,\text{m}$ and $L_3 = 0.34168\,\text{m}$.
From Table 2.3 it is possible to take the angle limits: the first and second joints have velocity limits of 75 deg/s, while the third has 65 deg/s.

3.2 Forward Kinematics

The forward kinematic analysis is used to find the position of the end-effector of the manipulator with respect to the base frame for the given set of joint parameters. The Denavit-Hartenberg convention [48], described in Appendix A.1, is applied to the Cyton Gamma 1500 arm with 3-DOF. The origin of Frame 0 is chosen at the intersection of z_0 with z_1 ($d_1 = 0$); further, z_1 and z_2 are parallel and the choice of axes x_1 and x_2 is made as for two link. Thus, since the revolute axes are all parallel, the simplest choice is made for all axes x_i along the direction of the relative links (the direction of x_0 is arbitrary) and all lying in the plane (x_0,y_0). In this way, all the parameters d_i are null and the angles between the axes x_i directly provide the joint variables.
The DH parameters of Cyton Gamma 1500 3.1 are specified in Table 3.1.
The homogeneous transformation matrices, defined in (A.5) are different for the first

Link	$\mathbf{a}_i$	α_i	$\mathbf{d}_i$	θ_i
1	0	$\pi/2$	L_1	$\theta_1 - \pi/2$
2	L_2	0	0	θ_2
3	L_3	0	0	θ_3

Table 3.1: Cyton Gamma 1500: Denavit-Hartenberg (DH) parameters

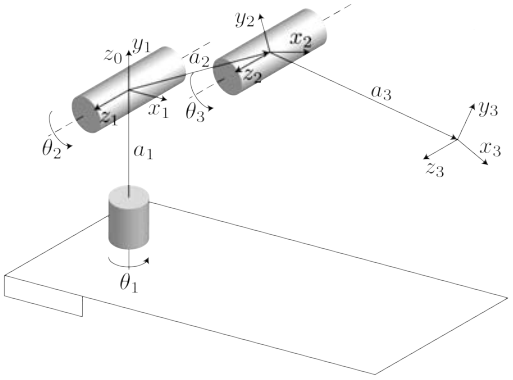


Figure 3.1: Cyton Gamma 1500 with 3-DOF

joint with respect to the second and the third:

$$A_1^0(\theta_1) = \begin{bmatrix} s_1 & 0 & -c_1 & 0 \\ -c_1 & 0 & -s_1 & 0 \\ 0 & 1 & 0 & L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

$$A_i^{i-1}(\theta_i) = \begin{bmatrix} c_i & -s_i & 0 & L_i c_i \\ s_i & c_i & 0 & L_i s_i \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad i = 2, 3 \tag{3.2}$$

where $c_i = \cos(q_i)$ represents the cosine while $s_i = \sin(q_i)$ the sine.
Computation of the direct kinematics function as in (A.1) yields:

$$T_3^0(\mathbf{q}) = A_1^0 A_2^1 A_3^2 = \begin{bmatrix} s_1 & 0 & -c_1 & 0 \\ -c_1 & 0 & -s_1 & 0 \\ 0 & 1 & 0 & L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_2 & -s_2 & 0 & L_2 c_2 \\ s_2 & c_2 & 0 & L_2 s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_3 & -s_3 & 0 & L_3 c_3 \\ s_3 & c_3 & 0 & L_3 s_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

$$= \begin{bmatrix} s_1 & 0 & -c_1 & 0 \\ -c_1 & 0 & -s_1 & 0 \\ 0 & 1 & 0 & L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_{23} & -s_{23} & 0 & L_2 c_2 + L_3 c_{23} \\ s_{23} & c_{23} & 0 & L_2 s_2 + L_3 s_{23} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

$$= \begin{bmatrix} s_1 c_{23} & -s_1 s_{23} & -c_1 & s_1(L_2 s_2 + L_3 s_{23}) \\ -c_1 c_{23} & c_1 s_{23} & -s_1 & -c_1(L_2 s_2 + L_3 s_{23}) \\ s_{23} & c_{23} & 0 & L_1 + L_2 c_2 + L_3 c_{23} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

where $\mathbf{q} = [\theta_1\, \theta_2\, \theta_3]^T$. Since z_3 is aligned with z_2, Frame 3 does not coincide with a possible end-effector frame, and needs a proper constant transformation.
It is desired to find the joint variables θ_1, θ_2 and θ_3 corresponding to a given end-effector position p_W. Notice that the direct kinematics for p_W is expressed by (3.5). Hence, it follows:

$$p_{w_x} = x_{ee} = s_1(L_2 s_2 + L_3 s_{23}) \tag{3.6}$$
$$p_{w_y} = y_{ee} = -c_1(L_2 s_2 + L_3 s_{23}) \tag{3.7}$$
$$p_{w_z} = z_{ee} - L_1 = L_2 c_2 + L_3 c_{23} \tag{3.8}$$

Thus, the direct kinematics for the 3-DOF robotic arm is very simple and calculated as follows:

$$x_{ee} = \sin q_1 \cdot [L_2 \cdot \sin q_2 + L_3 \cdot \sin(q_2 + q_3)] \tag{3.9a}$$
$$y_{ee} = \cos q_1 \cdot [L_2 \cdot \sin q_2 + L_3 \cdot \sin(q_2 + q_3)] \tag{3.9b}$$
$$z_{ee} = L_1 + L_2 \cdot \cos q_2 + L_3 \cdot \cos(q_2 + q_3) \tag{3.9c}$$

3.2.1 Jacobian

The analytical calculus about the Analytical Jacobian matrix J as first point using:

$$J = \begin{bmatrix} \frac{dx_{ee}}{dq_1} & \frac{dx_{ee}}{dq_2} & \frac{dx_{ee}}{dq_3} \\ \frac{dy_{ee}}{dq_1} & \frac{dy_{ee}}{dq_2} & \frac{dy_{ee}}{dq_3} \\ \frac{dz_{ee}}{dq_1} & \frac{dz_{ee}}{dq_2} & \frac{dz_{ee}}{dq_3} \end{bmatrix} \tag{3.10}$$

that, in the case of Cyton Gamma 1500 with 3-DOF, is:

$$J = \begin{bmatrix} c_1 \cdot (L_2 \cdot s_2 + L_3 \cdot s_{23}) & s_1 \cdot (L_2 \cdot c_2 + L_3 \cdot c_{23}) & L_3 \cdot s_1 \cdot c_{23} \\ s_1 \cdot (L_2 \cdot s_2 + L_3 \cdot s_{23}) & -c_1 \cdot (L_2 \cdot c_2 + L_3 \cdot c_{23}) & -L_3 \cdot c_1 \cdot c_{23} \\ 0 & -(L_2 \cdot s_2 + L_3 \cdot s_{23}) & -L_3 \cdot s_{23} \end{bmatrix} \tag{3.11}$$

where $c_1 = \cos(q_1)$ and $c_{23} = \cos(q_2 + q_3)$ represent the cosine with sum of angles. The same holds for the sine with $s_1 = \sin(q_1)$ and $s_{23} = \sin(q_2 + q_3)$.

3.2.2 Manipulability

The manipulability measures the robot posture in the workspace from the viewpoint of object manipulation. The manipulability index of given robot poses, introduced by Yoshikawa in 1985 [51], can be calculated as a quality value that gives information about how good an adjustment in workspace is possible. For one arm, Yoshikawa proposed, the manipulability index μ given by:

$$\mu = \mu(x, y) = |det(\mathbf{J}(\mathbf{q}))| \tag{3.12}$$

where $\mathbf{J}$ is Jacobian of the robot kinematics. The idea is to select a starting pose based on the highest determinant of the Jacobian, so in different words, is finding for which values of q_2 and q_3 the determinant is the maximum possible, considering a range of angles $(q_2, q_3) \in [-105°, 0°]$.
For a non redundant manipulator, the differential kinematics solution

$$\dot{\mathbf{q}} = \mathbf{J}^{-1}(\mathbf{q})\mathbf{v}_e \tag{3.13}$$

is used to derive $\dot{\mathbf{q}}^T\dot{\mathbf{q}} = 1$ that become:

$$\mathbf{v}_e^T(\mathbf{J}(\mathbf{q})\mathbf{J}^T(\mathbf{q}))^{-1}\mathbf{v}_e = 1 \tag{3.14}$$

In this case the points on the surface of the sphere in the joint velocity space are mapped into points on the surface of the ellipsoid in the end-effector velocity space. Along the direction of the major axis of the ellipsoid, the end-effector can move at large velocity, while along the direction of the minor axis small end-effector velocities are obtained. The closer the ellipsoid is to a sphere, the better the end-effector can move isotropically along all directions of the operational space. Hence, the ellipsoid is an index characterizing manipulation ability of the structure in terms of velocities. As can be recognized from (3.14), the shape and orientation of the ellipsoid are determined by the core of its quadratic form and then by the matrix $\mathbf{J}\mathbf{J}^T$ which is in general a function of the manipulator configuration. The directions of the principal axes of the ellipsoid are determined by the eigenvectors u_i, for $i = 1, ..., r$, of the matrix $\mathbf{J}\mathbf{J}^T$, while the dimensions of the axes are given by the singular values of $\mathbf{J}$, $\sigma_i = \sqrt{\lambda_i(\mathbf{J}\mathbf{J}^T)}$, for $i = 1, ..., r$ where $\lambda_i(\mathbf{J}\mathbf{J}^T)$ denotes the generic eigenvalue of $\mathbf{J}\mathbf{J}^T$. A global representative measure of manipulation ability can be obtained by considering the volume of the ellipsoid. This volume, in the case of a nonredundant manipulator ($r = n$), is proportional to the quantity:

$$w(\mathbf{q}) = |det(\mathbf{J}(\mathbf{q}))| \tag{3.15}$$

which is the manipulability measure. It is easy to recognize that it is always $w > 0$, except for a manipulator at a singular configuration when $w = 0$. For this reason, this measure is usually adopted as a distance of the manipulator from singular configurations. In more general cases when it is not easy to find a simple, meaningful index, one can consider the ratio between the minimum and maximum singular values of the Jacobian $\frac{\sigma_r}{\sigma_1}$ which is equivalent to the inverse of the condition number of matrix $\mathbf{J}$. This ratio gives not only a measure of the distance from a singularity ($\sigma_r = 0$), but also a direct measure of eccentricity of the ellipsoid. The disadvantage in utilizing this index is its computational complexity; it is practically impossible to compute it in symbolic form, i.e., as a function of the joint configuration, except for matrices of reduced dimension.
A zero-manipulability configuration (i.e. a singular configuration) expresses the impossibility for the robot end-effector to be moved in any direction of its working area, whereas a non-zero manipulability expresses the possibility for the robot end-effector to be moved in any direction of its working area with task velocities higher for given joint velocities as the manipulability criterion is high.
The manipulabity measure of the Cyton Gamma 1500 with 3-DOF is:

$$w = |det(\mathbf{J})| = L_2 \cdot L_3 \cdot |(L_2 \cdot s_2 + L_3 \cdot s_{23}) \cdot s_3| \tag{3.16}$$

The best posture for given L_2 and L_3 is obtained as follows. First, θ_1 is not related to w and can take any value. Second, deriving $\partial w/\partial\theta_2 = 0$:

$$\tan\theta_2 = \frac{L_2 + L_3 \cdot c_3}{L_3 \cdot s_3} \tag{3.17}$$

This means that the tip of the arm should be on the xy-plane, that is, at the same height as the second joint. This can further be interpreted as maximizing the contribution of the angular velocity of the first joint to the manipulability measure.

Substituting (3.17) into (3.16) yields:

$$w = L_2 \cdot L_3 \cdot \sqrt{L_2^2 + L_3^2 + 2 \cdot L_2 \cdot L_3 \cdot c_3} \cdot |s_3| \tag{3.18}$$

The value of θ_3 that maximizes w is:

$$\cos\theta_3 = \frac{\sqrt{(L_2^2 + L_3^2)^2 + 12 \cdot L_2^2 \cdot L_3^2} - (L_2^2 + L_3^2)}{6 \cdot L_2 \cdot L_3} \tag{3.19}$$

Substituting the real values of the Cyton Gamma 1500, it becomes:

$$\theta_2 = \arctan(\tan\theta_2) = 0.8274\,\text{rad} = 47.4067\,^\circ \tag{3.20}$$

$$\theta_3 = \arccos(\cos\theta_3) = 1.2415\,\text{rad} = 71.1339\,^\circ \tag{3.21}$$

The same can be computed considering:

$$\mu_1 = max[\mu(x,y)\,\forall x, y \in W] = |det(\mathbf{J}(\mathbf{q}))| \tag{3.22}$$

it is possible to compute the same calculus in MATLAB®, considering:

$$\mu_1 = max[\mu(x,y)\,\forall x, y \in W] = |det(\mathbf{J}(\mathbf{q}))| \tag{3.23}$$

The determinant of $\mathbf{J}$ is equal to 0.373 using $q_2 = -47.4067°$ and $q_3 = -71.1339°$, or in radiant $q_2 = -0.8274\,\text{rad}$ and $q_3 = -1.2415\,\text{rad}$. Obviously, q_1 is not needed in the determinant, therefore it is maintained equals to 0 (e.g. $q_1 = 0\,\text{rad}$).

3.3 Inverse Kinematics

The inverse kinematics problem, given the position of the end-effector of the manipulator, is to find the values of joint angles and displacements that can attain the specified position of the end-effector. The equations formulated to solve the inverse kinematic problem are nonlinear and it is very difficult to obtain closed form solutions. The problem may have multiple solutions or sometimes any solution does not exist. The solutions of the inverse kinematic problem for manipulators are helpful to define the workspace of manipulators. If solutions exist for a point then it is in the workspace of the manipulator, and if no solution exists then the point is not in the workspace.

3.3.1 Analytical Inverse Kinematics

The Analytical Inverse Kinematics study starts squaring and summing (3.6), (3.7) and (3.8), yielding:

$$p_{w_x}^2 + p_{w_y}^2 + p_{w_z}^2 = (L_2 s_2 + L_3 s_{23})^2 + (L_2 c_2 + L_3 c_{23})^2 \tag{3.24}$$
$$= L_2^2 s_2^2 + L_3^2 s_{23}^2 + 2L_2 L_3 s_2 s_{23} + L_2^2 c_2^2 + L_3^2 c_{23}^2 + 2L_2 L_3 c_2 c_{23} \tag{3.25}$$
$$= L_2^2 + L_3^2 + 2L_2 L_3 [s_2(s_2 c_3 + c_2 s_3) + c_2(c_2 c_3 - s_2 s_3)] \tag{3.26}$$
$$= L_2^2 + L_3^2 + 2L_2 L_3 [s_2^2 c_3 + c_2 s_2 s_3) + c_2^2 c_3 - c_2 s_2 s_3)] \tag{3.27}$$
$$= L_2^2 + L_3^2 + 2L_2 L_3 c_3 \tag{3.28}$$

from which:

$$c_3 = \frac{p_{w_x}^2 + p_{w_y}^2 + p_{w_z}^2 - L_2^2 - L_3^2}{2L_2 L_3} \tag{3.29}$$

where the admissibility of the solution obviously requires that $-1 \leq c_3 \leq 1$, or equivalently $|L_2 - L_3| \leq \sqrt{p_{w_x}^2 + p_{w_y}^2 + p_{w_z}^2} \leq |L_2 + L_3|$, otherwise the wrist point is outside the reachable workspace of the manipulator. Hence, it is:

$$s_3 = \pm\sqrt{1 - c_3^2} \tag{3.30}$$

and thus.

$$\theta_3 = atan2(s_3, c_3) \tag{3.31}$$

giving the two solutions, according to the sign of s_3:

$$\theta_{3,I} \in [-\pi, \pi] \tag{3.32}$$
$$\theta_{3,II} = -\theta_{3,I} \tag{3.33}$$

After determining θ_3, it is possible to compute θ_2 as follows. Squaring and summing (3.6) and (3.7) gives:

$$p_{w_x}^2 + p_{w_y}^2 = (L_2 s_2 + L_3 s_{23})^2 \tag{3.34}$$

from which:

$$\pm\sqrt{p_{w_x}^2 + p_{w_y}^2} = L_2 s_2 + L_3 s_{23} = (L_2 + L_3 c_3) s_2 + L_3 s_3 c_2 \tag{3.35}$$

while:

$$p_{w_z} = L_2 c_2 + L_3 c_{23} = (L_2 + L_3 c_3) c_2 - L_3 s_3 s_2 \tag{3.36}$$

The system of the two equations (3.35) and (3.36), admits the solutions:

$$s_2 = \frac{-L_3 s_3 p_{w_z} \pm (L_2 + L_3 c_3)\sqrt{p_{w_x}^2 + p_{w_y}^2}}{L_2^2 + L_3^2 + 2L_2 L_3 c_3} \tag{3.37}$$

$$c_2 = \frac{(L_2 + L_3 c_3)p_{w_z} \pm L_3 s_3 \sqrt{p_{w_x}^2 + p_{w_y}^2}}{L_2^2 + L_3^2 + 2L_2 L_3 c_3} \tag{3.38}$$

From (3.37) and (3.38) it follows:

$$\theta_2 = atan2(s_2, c_2) \tag{3.39}$$

which gives the four solutions for θ_2, according to the sign of s_3 in (3.30):

$$\theta_{2,I} = atan2(-L_3 s_3^+ p_{w_z} + (L_2 + L_3 c_3)\sqrt{p_{w_x}^2 + p_{w_y}^2}, (L_2 + L_3 c_3)p_{w_z} + L_3 s_3^+ \sqrt{p_{w_x}^2 + p_{w_y}^2}) \tag{3.40}$$

$$\theta_{2,II} = atan2(-L_3 s_3^+ p_{w_z} - (L_2 + L_3 c_3)\sqrt{p_{w_x}^2 + p_{w_y}^2}, (L_2 + L_3 c_3)p_{w_z} - L_3 s_3^+ \sqrt{p_{w_x}^2 + p_{w_y}^2}) \tag{3.41}$$

corresponding to $s_3^+ = +\sqrt{1 - c_3^2}$ and:

$$\theta_{2,III} = atan2(-L_3 s_3^- p_{w_z} + (L_2 + L_3 c_3)\sqrt{p_{w_x}^2 + p_{w_y}^2}, (L_2 + L_3 c_3)p_{w_z} + L_3 s_3^- \sqrt{p_{w_x}^2 + p_{w_y}^2}) \tag{3.42}$$

$$\theta_{2,IV} = atan2(-L_3 s_3^- p_{w_z} - (L_2 + L_3 c_3)\sqrt{p_{w_x}^2 + p_{w_y}^2}, (L_2 + L_3 c_3)p_{w_z} - L_3 s_3^- \sqrt{p_{w_x}^2 + p_{w_y}^2}) \tag{3.43}$$

corresponding to $s_3^- = -\sqrt{1 - c_3^2}$. Finally, to compute θ_1, it is sufficient to rewrite (3.6) and (3.7), using (3.35), as:

$$p_{w_x} = \pm s_1 \sqrt{p_{w_x}^2 + p_{w_y}^2} \tag{3.44}$$

$$p_{w_y} = \mp c_1 \sqrt{p_{w_x}^2 + p_{w_y}^2} \tag{3.45}$$

$$\tag{3.46}$$

which, once solved, gives the two solutions:

$$\theta_{1,I} = atan2(p_{w_x}, p_{w_y}) \tag{3.47}$$

$$\theta_{1,II} = atan2(-p_{w_x}, -p_{w_y}) \tag{3.48}$$

Consider that (3.48) gives:

$$\theta_{1,II} = \begin{cases} atan2(p_{w_x}, p_{w_y}) - \pi & p_{w_x} \geq 0 \\ atan2(p_{w_x}, p_{w_y}) + \pi & p_{w_x} < 0 \end{cases} \tag{3.49}$$

Remembering that $atan2(-y, -x) = -atan2(y, -x)$ and:

$$atan2(y, -x) = \begin{cases} \pi - atan2(y, x) & y \geq 0 \\ -\pi - atan2(y, x) & y < 0 \end{cases} \tag{3.50}$$

As can be recognized, there exist four solutions according to the values of θ_3 in (3.32), (3.33), θ_2 in (3.40), (3.41), (3.42), (3.43) and θ_1 in (3.47), (3.48):

$$(\theta_{1,I}, \theta_{2,I}, \theta_{3,I}) \qquad (\theta_{1,I}, \theta_{2,III}, \theta_{3,II}) \qquad (\theta_{1,II}, \theta_{2,II}, \theta_{3,I}) \qquad (\theta_{1,II}, \theta_{2,IV}, \theta_{3,II}) \tag{3.51}$$

It is possible to find the solutions only if at least

$$p_{w_x} \neq 0 \qquad \text{or} \qquad p_{w_y} \neq 0 \tag{3.52}$$

If $p_{w_x} = p_{w_y} = 0$, infinity solutions are obtained, since it is possible to determine the joint variables θ_2 and θ_3 independently of the value of θ_1; the arm in such configuration is kinematically singular.

3.3.2 Inverse Jacobian

To obtain the inverse kinematics, it is needed to compute the inverse of the Analytical Jacobian. To avoid losing time, it is better to compute it, that is:

$$J^{-1} = \begin{bmatrix} \frac{c_1}{L_2 \cdot s_2 + L_3 \cdot s_{23}} & \frac{s_1}{L_2 \cdot s_2 + L_3 \cdot s_{23}} & 0 \\ \frac{s_1 \cdot s_{23}}{L_2 \cdot s_3} & \frac{-c_1 \cdot s_{23}}{L_2 \cdot s_3} & \frac{c_{23}}{L_2 \cdot s_3} \\ \frac{-s_1 \cdot (L_2 \cdot s_2 + L_3 \cdot s_{23})}{L_2 \cdot L_3 \cdot s_3} & \frac{c_1 \cdot (L_2 \cdot s_2 + L_3 \cdot s_{23})}{L_2 \cdot L_3 \cdot s_3} & \frac{-(L_2 \cdot c_2 + L_3 \cdot c_{23})}{L_2 \cdot L_3 \cdot s_3} \end{bmatrix} \tag{3.53}$$

From this point, only Jacobian will be used instead of Analytical Jacobian.

3.4 Kinematics control

The procedure uses inverse Jacobian to compute joints velocities, represented by $\dot{\mathbf{q}}$, from end-effector velocity, represented by $\mathbf{v}_e$. Therefore, using the Euler integration method, it integrates joint velocities to obtain joint positions.
Once obtained end-effector velocity:

$$\mathbf{v}_e = \begin{bmatrix} \dot{x}_{ee} \\ \dot{y}_{ee} \\ \dot{z}_{ee} \end{bmatrix} = \frac{1}{\Delta t} \begin{bmatrix} \hat{x}_{ball} - x_{ee} \\ \hat{y}_{ball} - y_{ee} \\ \hat{z}_{ball} - z_{ee} \end{bmatrix} \tag{3.54}$$

It is easy with inverse Jacobian to obtain joint velocities, as follows:

$$\dot{\mathbf{q}} = \mathbf{J}^{-1}(\mathbf{q}) \times \mathbf{v}_e \Rightarrow \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = \mathbf{J}^{-1}(\mathbf{q}) \times \begin{bmatrix} \dot{x}_{ee} \\ \dot{y}_{ee} \\ \dot{z}_{ee} \end{bmatrix} \tag{3.55}$$

Then it compares $\dot{\mathbf{q}} = [\dot{q_1}, \dot{q_2}, \dot{q_3}]^T$ with the velocity limits of each joint (see Table 2.3), and, in case that they exceed the limits, it scales the velocities to be inside the bounds in the following way:

$$k = \frac{v_{max}}{\dot{q}_i} < 1 \Rightarrow \dot{\mathbf{q}} = \begin{cases} v_{max} & \text{for joint } i \\ k \cdot \dot{q}_j & \text{for joint } j \neq i \end{cases} \tag{3.56}$$

and so on for the other joints, if their velocity stills outside the bounds.
After, it uses the technique of the Euler integration method in discrete time:

$$\frac{d\mathbf{q}(t_i)}{dt} = \dot{\mathbf{q}}(t_i) = \frac{\mathbf{q}(t_{i+1}) - \mathbf{q}(t_i)}{\Delta t} \tag{3.57}$$

that given an integration interval or time step Δt and knowing joint positions and velocities at time t_i, it can compute the joint positions at time $t_{i+1} = t_i + \Delta t$ in the following way:

$$\begin{bmatrix} q_1(t_{i+1}) \\ q_2(t_{i+1}) \\ q_3(t_{i+1}) \end{bmatrix} = \begin{bmatrix} q_1(t_i) \\ q_2(t_i) \\ q_3(t_i) \end{bmatrix} + \Delta t \cdot \begin{bmatrix} \dot{q}_1(t_i) \\ \dot{q}_2(t_i) \\ \dot{q}_3(t_i) \end{bmatrix} \tag{3.58}$$

or more precisely:

$$\mathbf{q}(t_{i+1}) = \mathbf{q}(t_i) + \mathbf{J}^{-1}(\mathbf{q}(t_i)) \times \mathbf{v}_e(t_i) \cdot \Delta t \tag{3.59}$$

Algorithm 3.2 resumes the procedure.

Data: End-effector velocity
Result: Joint positions
for *each end-effector velocity received* **do**
    Obtain joints velocities using Inverse Jacobian;
    Compute Inverse Jacobian;
    Compare with velocity limits;
    Compute numerical integration (Euler method);
end

Algorithm 3.2: Velocity Control

Thus from (3.59) it is clear that the robot is actually controlled in position and not in velocity. It has to be highlighted that it is always necessary that the Jacobian be square and of full rank to avoid problems of singularity, as will be treated in Section 3.4.2.

3.4.1 Avoiding error

Using (3.59), the numerical integration does not satisfy the values of the continuous time because of drift phenomena in the solutions.
To avoid this error, it is used a simple scheme that considers the error between the desired and the actual end-effector position:

$$\mathbf{e} = \mathbf{x}_d - \mathbf{x}_e \tag{3.60}$$

and deriving it:

$$\dot{\mathbf{e}} = \dot{\mathbf{x}}_d - \dot{\mathbf{x}}_e \Rightarrow \dot{\mathbf{e}} = \dot{\mathbf{x}}_d - \mathbf{J}_A(\mathbf{q})\dot{\mathbf{q}} \tag{3.61}$$

where $\mathbf{J}_A$ is the analytical Jacobian, cause it is in the operational space, that it is assumed square and nonsingular. The goal is to find:

$$\dot{\mathbf{q}} = \dot{\mathbf{q}}(\mathbf{e}) : \mathbf{e} \to 0 \tag{3.62}$$

choosing:

$$\dot{\mathbf{q}} = \mathbf{J}_A^{-1}(\mathbf{q})(\dot{\mathbf{x}}_d + \mathbf{K}\mathbf{e}) \tag{3.63}$$

that brings to the linear system:

$$\dot{\mathbf{e}} + \mathbf{K}\mathbf{e} = 0 \tag{3.64}$$

with matrix $\mathbf{K}$ positive definite to have the system asymptotically stable. The eigenvalues of $\mathbf{K}$ influence the convergence rate of the error to zero; the smaller the eigenvalues, the slower the convergence. The Jacobian used in this scheme is called *Pseudo-inverse.*
Another method includes the use of the Jacobian transpose, to find $\dot{\mathbf{q}} = \dot{\mathbf{q}}(\mathbf{e})$ without linearizing error dynamics.
Using the Lyapunov method:

$$V(\mathbf{e}) = \frac{1}{2}\mathbf{e}^T\mathbf{K}\mathbf{e} \tag{3.65}$$

where:

$$V(\mathbf{e}) > 0 \quad \forall \mathbf{e} \neq 0 \text{ with } V(\mathbf{0}) = 0 \tag{3.66}$$

$$\dot{V}(\mathbf{e}) = \mathbf{e}^T\mathbf{K}\dot{\mathbf{x}}_d - \mathbf{e}^T\mathbf{K}\dot{\mathbf{x}} \tag{3.67}$$

$$= \mathbf{e}^T\mathbf{K}\dot{\mathbf{x}}_d - \mathbf{e}^T\mathbf{K}\mathbf{J}_A(\mathbf{q})\dot{\mathbf{q}} \tag{3.68}$$

choosing:

$$\dot{\mathbf{q}} = \mathbf{J}_A^T(\mathbf{q})\mathbf{K}\mathbf{e} \tag{3.69}$$

it leads to:

$$\dot{V}(\mathbf{e}) = \mathbf{e}^T\mathbf{K}\dot{\mathbf{x}}_d - \mathbf{e}^T\mathbf{K}\mathbf{J}_A(\mathbf{q})\mathbf{J}_A^T(\mathbf{q})\mathbf{K}\mathbf{e} \tag{3.70}$$

From (3.70), if $\dot{\mathbf{x}}_d = \mathbf{0}$, there is asymptotic stability because $\dot{V} < 0$ and $V > 0$; if $\dot{\mathbf{x}}_d \neq \mathbf{0}$, there is $\mathbf{e}(t)$ bounded and $\mathbf{e}(\infty) \to \mathbf{0}$.

3.4.2 Kinematic Singularities

It is remarkable that solution (3.55) can be computed only when the Jacobian has full rank. If it is not, the manipulator is at a singular configuration and the system $\mathbf{v}_e = \mathbf{J}\dot{\mathbf{q}}$ has linearly dependent equations.
Identifying manipulator singularities is important for several reasons, such as:

1. Singularities represent configurations from which certain directions of motion may be unattainable;

2. At singularities, bounded end-effector velocities may correspond to unbounded joint velocities;

3. At singularities, bounded end-effector forces and torques may correspond to unbounded joint torques;

4. Singularities usually, but not always, correspond to points on the boundary of the manipulator workspace, that is, to points of maximum reach of the manipulator;

5. Singularities correspond to points in the manipulator workspace that may be unreachable under small perturbations of the link parameters, such as length, offset;

6. Near singularities there will not exist a unique solution to the inverse kinematics problem. In such cases there may be no solution or there may be infinitely many solutions.

The Jacobian inversion is a real problem also in the neighborhood of a singularity. There are different possible solutions to avoid it. The singularities can happen when

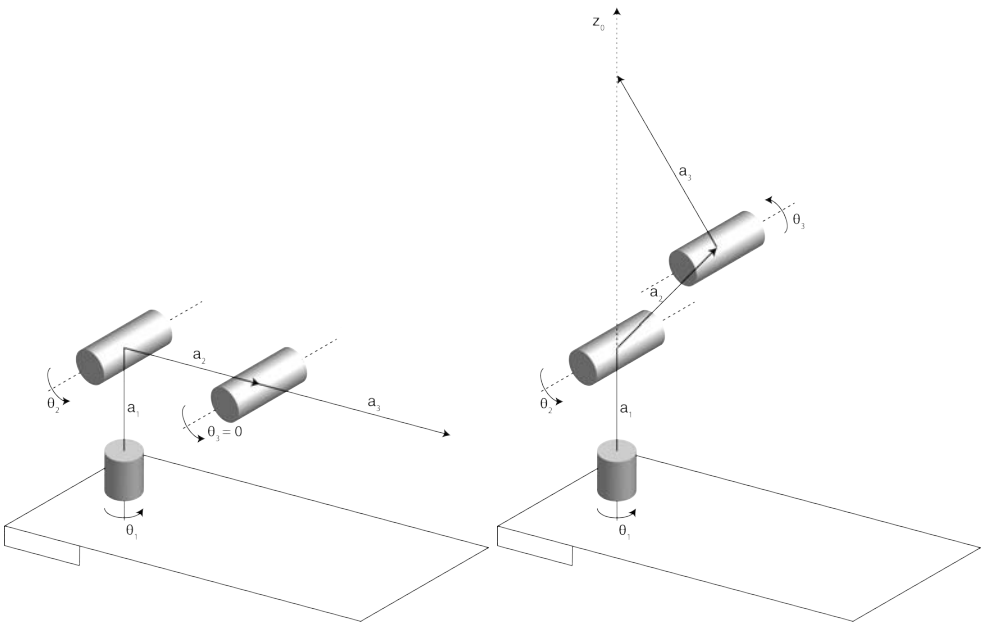
Jacobian has determinant equal to 0.
In this study, the determinant of the Jacobian for a Cyton with 3-DOF is:

$$\begin{aligned} det(J) &= L_2 \cdot L_3 \cdot (L_2 \cdot s_2 \cdot s_3 + L_3 \cdot c_2 - L_3 \cdot c_2 \cdot c_3^2 + L_3 \cdot c_3 \cdot s_2 \cdot s_3) \\ &= L_2 \cdot L_3 \cdot (L_2 \cdot s_2 \cdot s_3 + L_3 \cdot c_2 \cdot (1 - \cdot c_3^2) + L_3 \cdot c_3 \cdot s_2 \cdot s_3) \\ &= L_2 \cdot L_3 \cdot (L_2 \cdot s_2 \cdot s_3 + L_3 \cdot c_2 \cdot s_3^2 + L_3 \cdot c_3 \cdot s_2 \cdot s_3) \\ &= L_2 \cdot L_3 \cdot s_3 \cdot (L_2 \cdot s_2 + L_3 \cdot c_2 \cdot s_3 + L_3 \cdot c_3 \cdot s_2) \\ &= L_2 \cdot L_3 \cdot s_3 \cdot (L_2 \cdot s_2 + L_3 \cdot s_{23}) \end{aligned} \tag{3.71}$$

The determinant does not depend on the first joint variable q_1, while L_2, $L_3 \neq 0$. Determinant is equal to zero, so singularities can happen, when $s_3 = 0$ or when $(L_2 \cdot s_2 + L_3 \cdot s_{23}) = 0$. The first situation occurs when:

$$q_3 = 0 \qquad q_3 = \pi \tag{3.72}$$

This means that the joint represented by q_3 is totally extended (outstretched), as in Figure 3.3a, and is termed *elbow singularity*, while the second is termed *shoulder singularity* and occurs when $p_{w_x} = p_{w_y} = 0$, characterized by the end-effector point that is on axis z_0, as in Figure 3.3b. The rotation of q_1 does not cause any translation of the end-effector position and the kinematics equation admits infinite solutions.



(a) Cyton 3-DOF at an elbow singularity (b) Cyton 3-DOF at a shoulder singularity

Figure 3.3: Representation of singularities about Cyton 3-DOF

Damped least-squares inverse

One solution to avoid singularities is the *damped least-squares inverse*:

$$\mathbf{J}^{\star} = \mathbf{J}^T(\mathbf{J}\mathbf{J}^T + k^2\mathbf{I})^{-1} \tag{3.73}$$

where k is the damping factor to have a better inversion. The vector $\dot{\mathbf{q}}$ minimizes the cost function:

$$g''(\dot{\mathbf{q}}) = \|\mathbf{v}_e - \mathbf{J}\dot{\mathbf{q}}\|^2 + k^2\|\dot{\mathbf{q}}\|^2 \tag{3.74}$$

Avoid workspace's limits

The last and simplest implementation to avoid singularities regards the addition of a constraint to the end-effector: it cannot reach the boundary of the workspace and if it reaches it has to come back with the same velocity but opposite direction. The workspace limits are represented in Figure 3.4.

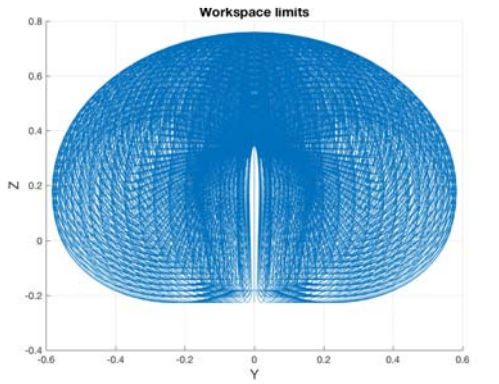


Figure 3.4: Workspace limits

Chapter 4

Control Strategies for Ball Catching

This chapter discusses the general approaches for robot ball catching: predictive and prospective control strategies. The key difference between them is the way to act. The first needs a prediction of the ball motion to know where the robot has to go. The second uses, at each time, information about robot and ball positions to know the velocity at which the robot has to move. The analysis and evaluations are made with various parameters and scenarios. In the beginning, it introduces the tools used: the assumptions, the ways to create different parabolas in a grid, the errors of the vision system and how are considered, the number of balls that it is possible to catch. The evaluation of predictive control is made in two ways, proving the variations between them and selecting the best solution. The evaluation of prospective control shows the problems of tuning and controlling using this technique.

4.1 Scope and assumptions

This section takes into account the assumptions made to reduce the number of parameters. Assumptions are among all about environment, frame-rate, errors, way to select how consider the ball caught.

4.1.1 Robot Implementation

The 3-DOF implementation robot is described in Section 3.1.
The workspace is bounded by the circumference of a sphere with radius $L_2 + L_3 = 0.5829m$ and origin in $O \in [0, 0, L_1] = [0, 0, 0.17735]$.

4.1.2 Time of Perception and Action

The approaches for ball catching normally split in perception and action. In ball catching tasks, the time is really important because it is very small.
A ball threw by a human that is $4 - 5\,\text{m}$ from the target with a common velocity of about $4 - 5\,\text{m/s}$ needs $1\,\text{s}$ to reach it. In this time, the vision sensor needs about $0.3\,\text{s}$ to predict the movement with acceptable results. Therefore, the robot has only about $0.7\,\text{s}$ to move from its starting pose to the final predicted pose. To do this, it is important both the vision and the robot: the vision to detect the ball and predict its behavior; the robot to move as fast as it can to the predicted catch point.

4.1.3 Frame-rate

The time is one of the most important parameter among all projects. In this work there are many ways to select it to have a good result in MATLAB®. ROS instead

needs a different selection.
The beginning of this work in MATLAB® starts with $\Delta t = 0.001s$ that brings very precise calculus to evaluate the system but that does not respect the real values available. Therefore, it selects two different values: $\Delta t = 1/30 = 0.0333\,\text{s}$ represents the Kinect Sensor's frame rate (30 fps); $\Delta t = 1/120 = 0.008333\,\text{s}$ represents a normal nowadays Stereo Camera's frame rate (120 fps).

4.1.4 Spatial uncertainty in perception

Hypothesis takes into account the use of Kinect, or another camera, as sensor to detect the ball. These sensors cannot detect the ball with a very high precision and there is an error in the point detected for each frame.
The errors often have distributions that are nearly the Normal or Gaussian distribution. A random variable X is distributed normally with mean μ and variance σ^2 (notation: $X \sim \mathcal{N}(\mu, \sigma^2)$). The probability density of the normal distribution is:

$$f(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

To simulate this error, the Gaussian error is added to the projectile motion in each axis, using in MATLAB® the function `randn()`, scaled to have a maximum error of about 10 cm (i.e. the Kinect sensor has a maximum error of about 5 cm but the idea is to take care of bigger errors in evaluation process). The error is in a range between 0 and 10 cm, to see how the error changes the performances of the strategies. To simplify, the possibles errors are five: 0 m, 0.01 m, 0.025 m, 0.05 m and 0.1 m.

4.1.5 Caught flag

In each throw, the robot can catch the ball or not. If it is caught, obviously, the distance between the ball and the end-effector is equal to 0. If it does not catch, the processor computes the minimum distance between the real ball and the end-effector to understand how much far and at which time the ball is away from the end-effector. This is used also to select some possible ball's trajectories that are very close to be intercepted by the robot if the distance is less than a threshold (e.g. $dist < 10\,\text{cm}$).

4.2 Tools to evaluate the best strategy

The evaluation of strategies needs a setup of different tools to help understand which is the best to use. This section describes the creation of various parabolas to test and the check of parameters.

4.2.1 Creation of projectile motions

The ball trajectory is considered as a parabolic motion, also known as a projectile or a ballistic motion. The model is simplified neglecting the air friction. Equations that describe the motion are:

$$x = x_0 + v_x t \tag{4.1a}$$

$$y = y_0 - v_y t \tag{4.1b}$$

$$z = z_0 + v_z t - \frac{1}{2} g t^2 \tag{4.1c}$$

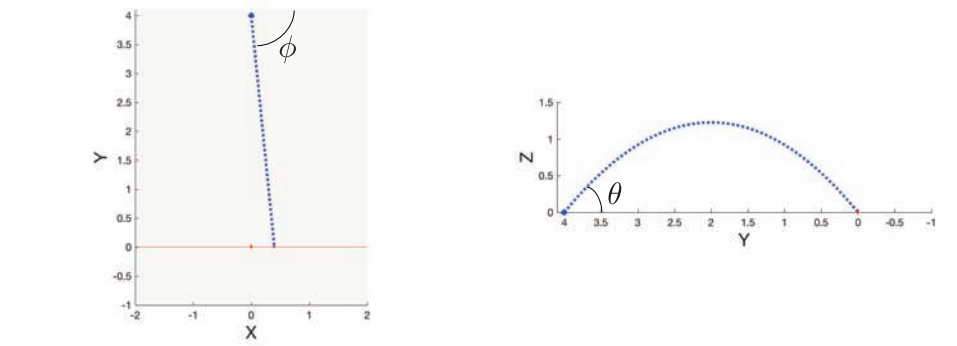
where $[x_0, y_0, z_0]^T$ are the coordinates of the points where the motion starts, $[v_x, v_y, v_z]^T$ is the vector representing the velocity in each direction, g is the gravitational constant of acceleration and t is the time.
Hence, the motion is linear through x and y-axes (horizontal and depth directions), while it has acceleration in z-axis (vertical direction). Equations (4.1) become:

$$x = x_0 + v_0 \cos(\theta) \cos(\phi) t \tag{4.2a}$$

$$y = y_0 - v_0 \cos(\theta) \sin(\phi) t \tag{4.2b}$$

$$z = z_0 + v_0 \sin(\theta) t - \frac{1}{2} g t^2 \tag{4.2c}$$

where v_0 is the absolute value of the ball's initial velocity (i.e. $v_0 = \sqrt{v_x^2 + v_y^2 + v_z^2}$) and ϕ and θ are the angles that denote the inclination of the motion. In simpler words, ϕ is the angle between the initial velocity and x-axis that means the direction of the projectile motion from the top (e.g. $\phi = 90°$ is a ball thrown towards to the center), as in Figure 4.1a and θ is the initial launch angle between the initial velocity and y-axis that means the direction of the projectile motion from the side, as in Figure 4.1b. The parabolic motion changes with these parameters.



(a) View with angle ϕ of plane xy (b) View with angle θ of plane xz

Figure 4.1: Parabola with angles

The tool creates several projectile motions. The model of the ball neglects the effect of air friction.
Using equations (4.2), described in Section 4.2.1, the tool creates different parabolas selecting the landing point on a grid ($x \in [-0.7, 0.7], y \in [-1, 1]$ or $y \in [-0.7, 1.3]$) using the same starting point ($[x_0, y_0, z_0]^T = [0, 4, 0]^T$) and same time ($t = 1$) for all the possible trajectories. Using these parameters, the values of initial velocity and the angles θ and ϕ obtained are:

$$v_x = v_0 \cos(\theta) \cos(\phi) = \frac{x - x_0}{t} \tag{4.3a}$$

$$v_y = v_0 \cos(\theta) \sin(\phi) = -\frac{y - y_0}{t} \tag{4.3b}$$

$$v_z = v_0 \sin(\theta) = \frac{z - z_0 + \frac{1}{2} g t^2}{t} \tag{4.3c}$$

From (4.3), using the arc tangent, the inverse of the tangent function that is in MATLAB®, the four quadrant arc tangent of the elements of X and Y such that

$-\pi \leq$ `atan2(Y,X)` $\leq \pi$, it becomes:

$$\phi = \arctan \frac{v_y}{v_x} \tag{4.4}$$

$$\theta = \arctan \frac{v_z \cdot sin(\phi)}{v_y} \tag{4.5}$$

The range of angles obtained is: $76.866° \leq \phi \leq 103.134°$, $44.1725° \leq \theta \leq 58.5492°$ and $5.7497\,\text{m/s} \leq \|v_0\| \leq 7.0391\,\text{m/s}$, where $\|v_0\| = \sqrt{v_x^2 + v_y^2 + v_z^2}$.
The landing grid points are represented in Figure 4.2 with one example of the 315 (15× 21) parabolas created.

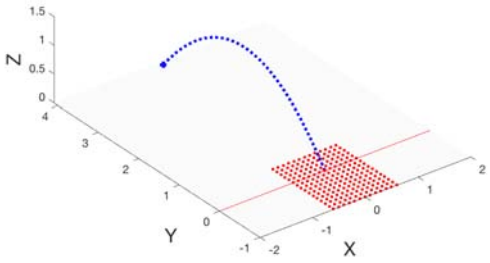


Figure 4.2: Parabolas in a grid

This is the simulated motion of the real ball, but with the addition of the Gaussian error to simulate vision sensor uncertainty.
The caught flag is one of the most important parameter in the cycle that use the different parabolas created, as in Section 4.2.1. The parameter `flag_caught` is equal to 1 if ball is caught or to 0 if not.
A parameter that represent the time of action, named `action_count`, reported in Figure 4.3, is also important in this work. It is a counter of the steps when the robot moves (circle in position one at that step). Therefore, if the predicted trajectory is outside the robot's workspace, the robot does not need to move to this point. Hence, it stops in its position for that step so the counter does not update (circle in position zero at that step).

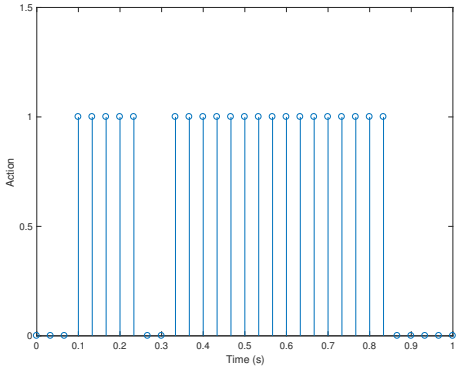


Figure 4.3: Example of time of action

4.3 Predictive Control

The predictive control is the first approach used to solve the catching problem, with block scheme in Figure 4.4.
The procedure takes into account all the assumptions (Section 4.1) and uses the tools to evaluate the study (Section 4.2).
In the following, the analysis considers two starting poses: one used in previous study [45] and one based on manipulability measures (Section 3.2.2).

Figure 4.4: Predictive control block scheme

Ball data

For each case, the cycle creates parabolas with landing point in a grid and different errors to obtain results, following the explanation of Section 4.2.1.
The cycle simulates ball data at two frame-rates: 30 Hz and 120 Hz. In real scenario, Kinect sensor gives the data at 30 Hz.

Model estimation

The estimation of the possible pose (position and orientation) of moving targets has a big importance in ball catching tasks. It includes several issues such as air-drag, angle of throw, initial velocity. The estimation method used among all in this work is the curve fitting method. Kalman was analyzed [45] but the problem of matrix initialization creates difficulties in the estimation, losing important time at the beginning of the throw.
Catching a flying ball implies to predict its trajectory ahead of time so that to determine the intersection point along the same trajectory. In this work, a well-known model of the dynamics of the motion was assumed by modeling the trajectory of a flying ball as a parabola (ballistic model). In addition to that, this study is tuned for spherical objects by estimating only the position of the ball. Whereas, other possible estimation parameters can be velocity and acceleration along with position. The polynomial approximation and estimation of the parameters are done recursively through least squares optimization.
The least squares method is one of the best-known approaches to solve the problem of finding the best polynomial approximation to the input samples. It was originally developed for statistical regression, but nowadays a general concept of least squares approximation is widely used for various applications beyond the statistics. In this work, the method of approximation is used to estimate the ball position from a few past samples using simple and quadratic linear regressions.
This choice is due to ball motion in different axis that follows both a first (simple linear regression) and a second order (quadratic linear regression). Algorithm 4.5 resumes the procedure. In Appendix A.2.1 and A.2.2 are reported the mathematical derivation of Linear and Quadratic Regressions. Model estimation is the process of picking the best kind and structure of model and finding the coefficients that enable a model (the kind and structure of which is already determined) to most closely reflect a particular known dataset.

```
for each frame received do
    if ball detected then
        Add ball in the array;
    end
    Compute linear regression for two axes;
    Compute quadratic regression for one axis;
    Update ball trajectory;
    Find possible landing point;
end
```

Algorithm 4.5: Ball estimation and prediction

The solution of Kalman Filter, studied in [45], has the problem of matrix initialization. Thus, the choice is the more simple and effective Polyfit function, discarding Kalman Filter.
The Polyfit function `P = polyfit(X,Y,N)` finds the coefficients of a polynomial $P(X)$ of degree N that fits the data Y best in a least-squares sense. P is a row vector of length $N+1$ containing the polynomial coefficients in descending powers, $P(1) \cdot X^N + P(2) \cdot X^{N-1} + \cdots + P(N) \cdot X + P(N+1)$. The order chosen is $N = 1$ for x and y directions and order $N = 2$ for z direction.

Prediction

The prediction using Polyfit function takes into account the Kinect data obtained for the ball.
One of the main problem of this approach is the time needed to have a good prediction of the trajectory (Figure 4.6). In the upper, the robot waits for a reliable landing point, based on the fact that after some frames (e.g. $6-8$ frames at 30 Hz) the predicted landing point does not change more than $1-3$ cm. After this time, it continues to move to the final position.
In the upper, the robot loses the initial time because it does not move until it has a prediction point that is inside its workspace area. Then, it starts moving and it continues to update the final position, based on new ball data received. Due to this, the robot fails to catch the ball in many situations, but it is more reliable to accomplish this task.

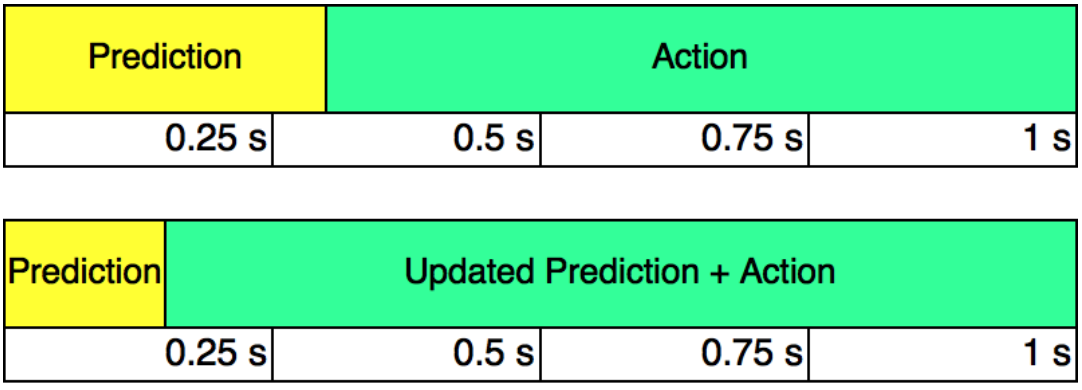


Figure 4.6: Different ways to use prediction

Time and Location

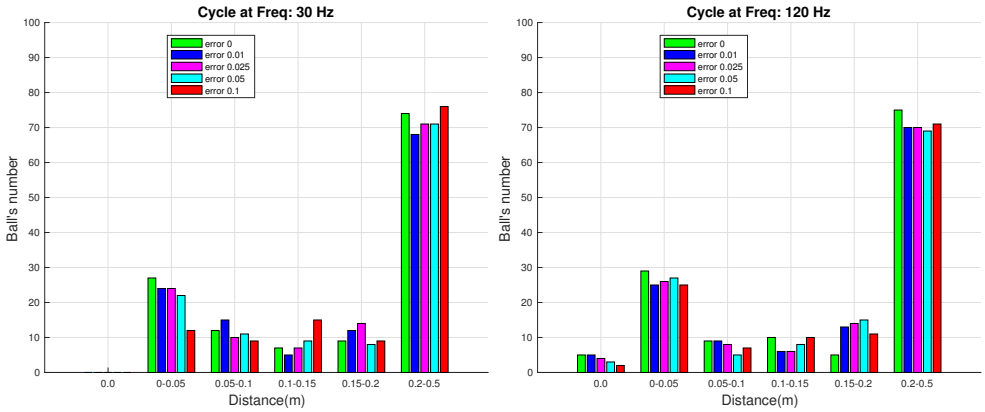
After the prediction, this study includes two different ways to select the time and the location.
The approach starts from the prediction at a given height, solutions used in previous study [45]. The results obtained suggest finding and verify other approaches. The second way selects as landing point the one in the ball's trajectory that has minimum distance from the end-effector. This allows the robot to move in three axes, taking advantage of the 3-DOF.

Robot Control

The robot uses velocity control, described in Section 3.4, with limits in velocity and position to take care of the real limits of the Cyton Gamma 1500. It updates its joint positions at each frame to reach the previous location selected that can be updated at each frame.

4.3.1 Prediction at a given height

The first way to use the prediction is to obtain the intersection point between the predicted ball's trajectory and the plane at a given height and send the command to the robot to move directly at that point.
In this case, height is equal to 17.73 cm to the ground that it is also equal to the best starting pose in z-axis. Therefore, the robot moves only along the x and y-axes and this is not a good solution because the robot does not use all the DOF, remembering that it is already simplified from 7-DOF to 3-DOF.
To confirm it, simulations predict the landing point at the height where the end-effector is. Histograms will report the number of ball caught or not caught, representing different errors and distance for the balls that robot does not catch. Each color represents a different error and each set of 5 columns represents the ball's distance from the end-effector. In Figure 4.7, it is possible to see why this type of prediction is not a good solution. At 30 Hz (Figure 4.7a) the balls cannot be caught from the end-effector. At 120 Hz (Figure 4.7b), the catching task is better but it stills have a low number of balls caught.



(a) Histogram at 30 Hz with new pose at a given height

(b) Histogram at 120 Hz with new pose at a given height

Figure 4.7: Histograms of predictive cycle with different errors and new pose at a given height

These results constrain to select and verify other strategies.

4.3.2 Prediction at the minimum distance from the end-effector

The second way to use the prediction is to find the point in the predicted ball's trajectory that has minimum distance from the end-effector and send the command to the robot to go directly at that point.
To do that, for each new ball detected the polyfit function (Section 4.3) returns the coefficients of each axis trajectory. Then, the procedure chooses the predicted points that are inside the robot's workspace. Among these points, after calculating the distance between end-effector and each point, it selects the one which is the closest point to the end-effector, so the one with minimum distance from the end-effector and sends this point to the robot.
To be more precise, the ball predicted points $(\hat{\mathbf{x}}_b, \hat{\mathbf{y}}_b, \hat{\mathbf{z}}_b)$ are inside the workspace when they are almost inside a sphere, given by the following equation:

$$\hat{x}_b^2 + \hat{y}_b^2 + (\hat{z}_b - L_1)^2 < (L_2 + L_3)^2 \tag{4.6}$$

Actually this is not exactly the real workspace but gives an idea of the space where the robot can moves. The real workspace limits was already represented in Figure 3.4.
The following equation compute the distance between all the predicted point and the position of the end-effector:

$$\|dist(ball, ee)\| = \sqrt{(\hat{\mathbf{x}}_b - x_{ee})^2 + (\hat{\mathbf{y}}_b - y_{ee})^2 + (\hat{\mathbf{z}}_b - z_{ee})^2} \tag{4.7}$$

The procedure takes the $\hat{x}_b, \hat{y}_b, \hat{z}_b$ that has the minimum distance:

$$\hat{x}_b, \hat{y}_b, \hat{z}_b \in \text{ Workspace s.t. } min(\|dist(ball, ee)\|) \tag{4.8}$$

If there are no predicted balls inside the workspace, the robot does not move and remain in its position waiting for a new possible point.
Once that it has a possible landing point, the difference between the landing point and the actual end-effector position gives the direction of the end-effector's velocity vector:

$$\mathbf{v}_{ee} = \begin{bmatrix} \hat{x}_b - x_{ee} \\ \hat{y}_b - y_{ee} \\ \hat{z}_b - z_{ee} \end{bmatrix} \tag{4.9}$$

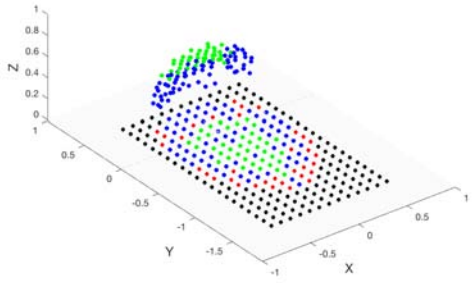
If the difference between the previous predicted ball landing point and the actual one is less than a threshold, the robot continues to move in the same previous direction.
If the ball is at a distance less than 1 cm from the end-effector, the procedure considers that the robot catches the ball.
Algorithm 4.8 resume the explanation.

```
Vector Initialization;
for i ← 2 to balls seen by camera do
    Creation of parabolic motion with error;
    Polyfit of ball's trajectory points received;
    if trajectory passes inside workspace then
        Find the closest point of the predicted trajectory to the end-effector;
        Compute Inverse Jacobian;
        Control joints limits;
        Compute numerical integration (Euler method);
        Update robot's position;
    end
    if distance ball-ee < 0.01 m then
        Ball caught;
    end
end
```

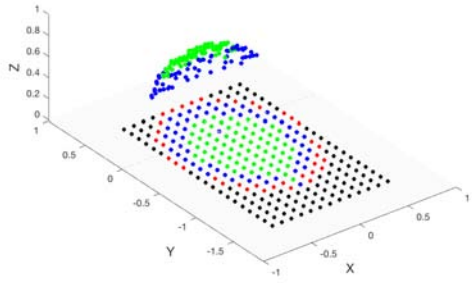
Algorithm 4.8: Predictive Control implementation

4.3.3 Performance Evaluation of Predictive Control

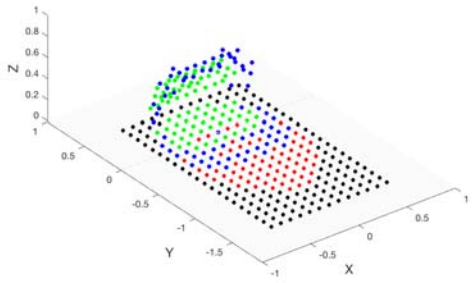
MATLAB® computes different simulations, always using the inverse Jacobian J^{-1} to planning the movement of the robot, trying errors of 0.0 m (Figure 4.9), 0.01 m (Figure 4.10), 0.025 m (Figure 4.11), 0.05 m (Figure 4.12) and 0.1 m (Figure 4.13) and frequencies of 30 Hz (Subfigure (a) and (c)) and 120 Hz (Subfigure (b) and (d)), using Algorithm 4.8 with two starting poses, one selected from past works, called standard pose, with $q_1 = 0\,\text{rad}$, $q_2 = -0.1745\,\text{rad} = -10\,^\circ$ and $q_3 = -0.5236\,\text{rad} = -30\,^\circ$ (Subfigure (a) and (b)) and the other one selected as described in 3.2.2 , called manipulability pose, with $q_1 = 0\,\text{rad}$, $q_2 = -0.8275\,\text{rad} = -47.41\,^\circ$ and $q_3 = -1.2415\,\text{rad} = -71.13\,^\circ$ (Subfigure (c) and (d)).
Figures represent scenarios with the balls caught and the color grid in the landing plane representing the trajectories landing points. The points are green if the robot catches the ball, blue if the ball pass at a distance of the end-effector less than a threshold (e.g. 0.1 m) or red if it is not caught and distant from the end-effector. The evaluation just from the view of the plot could be: the more green points are in the plot, the better is the proposed approach.
Figure 4.14 represents histograms of the ball's number caught or balls that have a distance from the end-effector between 0.0 m and 0.05 m and so on each 5 cm.
From Figure 4.14a, with error higher than 2.5 cm the robotic arm does not catch the ball, while in Figure 4.14b, it catches in all the cases but with a decreasing number inverse proportional to the growing error. Figure 4.15a refers to the same situation but without joint velocities limitations. This is done to understand better how much the velocity could cause a limit to accomplish the task. The velocity chosen is not unlimited, but can exceed the limits of the Cyton Gamma 1500, considered in this work. Therefore, without limitations the number of balls caught increases enormously. This is extremely true also for the higher frame rate case, in Figure 4.15b.
Then, the initial position of the robot changes to a new pose, related to the manipulability measures, because the catching task is dependent also to the robot's starting pose.
From Figure 4.16a it is possible to catch more balls with error $2.5cm$, but the robotic arm does not catch more balls than in the previous case. In Figure 4.16b, the higher



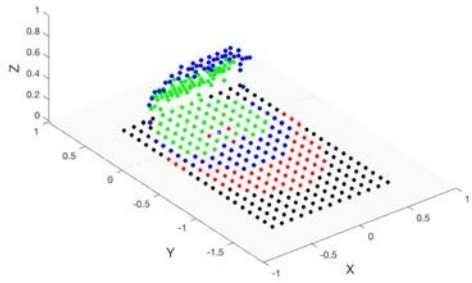
(a) Cycle at 30 Hz with standard pose



(b) Cycle at 120 Hz with standard pose

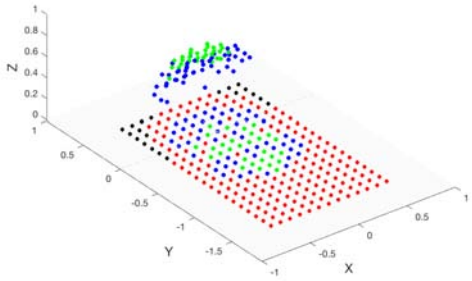


(c) Cycle at 30 Hz with manipulability pose

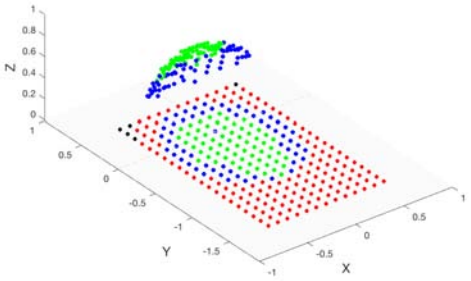


(d) Cycle at 120 Hz with manipulability pose

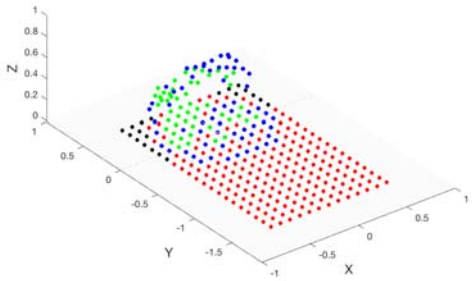
Figure 4.9: Catching predictive cycle with error 0.0 m



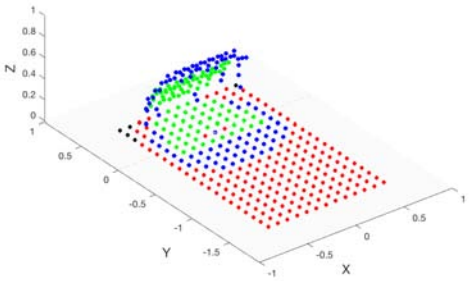
(a) Cycle at 30 Hz with standard pose



(b) Cycle at 120 Hz with standard pose

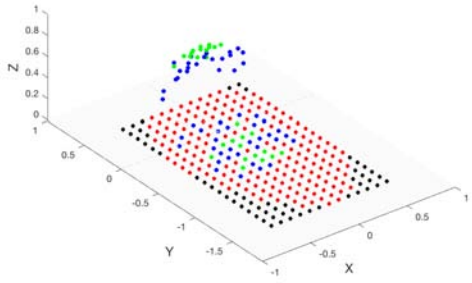


(c) Cycle at 30 Hz with manipulability pose

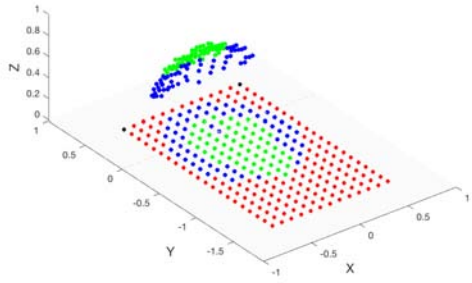


(d) Cycle at 120 Hz with manipulability pose

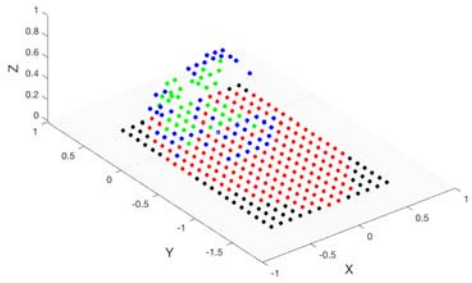
Figure 4.10: Catching predictive cycle with error 0.01 m



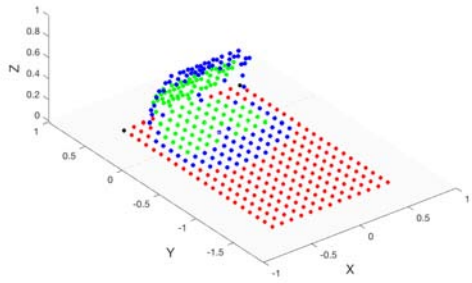
(a) Cycle at 30 Hz with standard pose



(b) Cycle at 120 Hz with standard pose

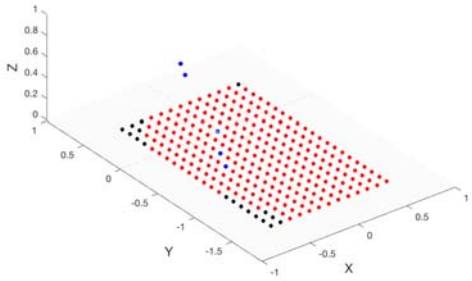


(c) Cycle at 30 Hz with manipulability pose

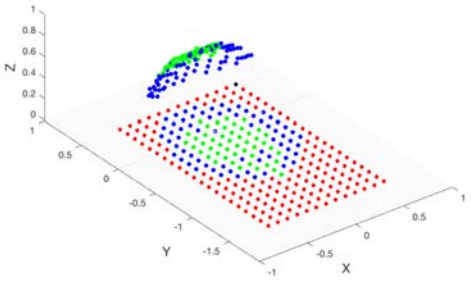


(d) Cycle at 120 Hz with manipulability pose

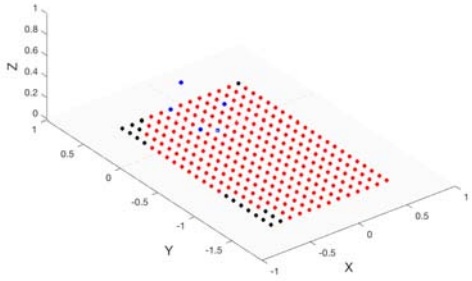
Figure 4.11: Catching predictive cycle with error 0.025 m



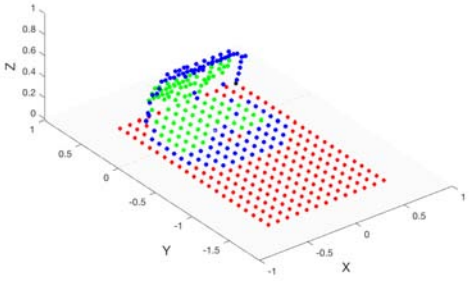
(a) Cycle at 30 Hz with standard pose



(b) Cycle at 120 Hz with standard pose

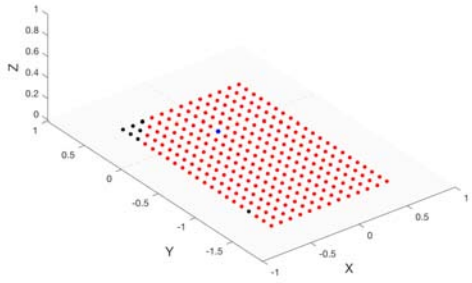


(c) Cycle at 30 Hz with manipulability pose

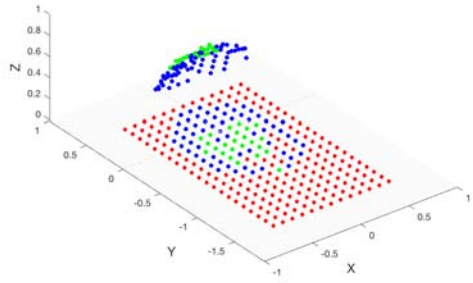


(d) Cycle at 120 Hz with manipulability pose

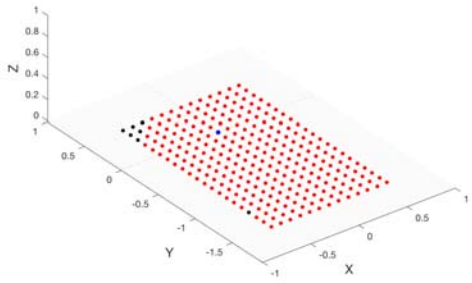
Figure 4.12: Catching predictive cycle with error 0.05 m



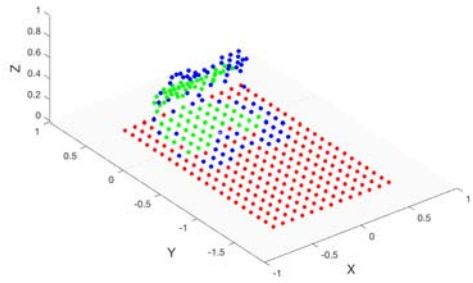
(a) Cycle at 30 Hz with standard pose



(b) Cycle at 120 Hz with standard pose

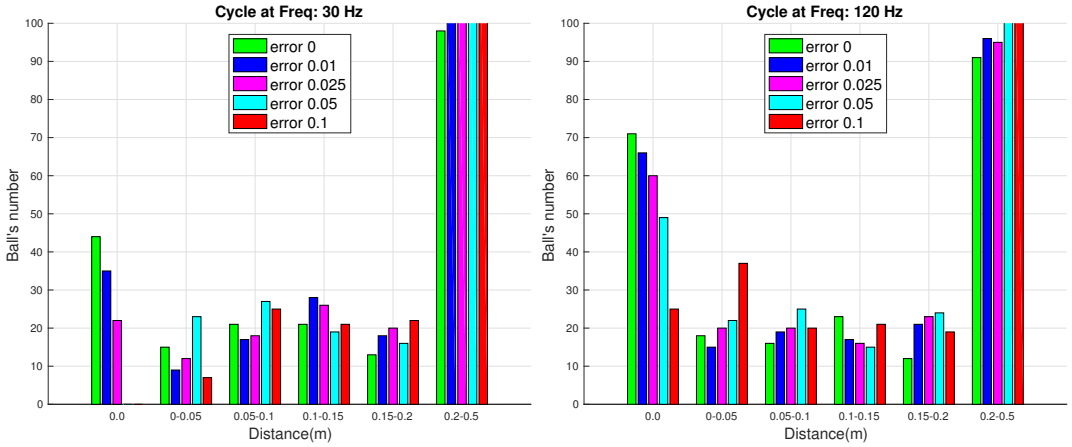


(c) Cycle at 30 Hz with manipulability pose



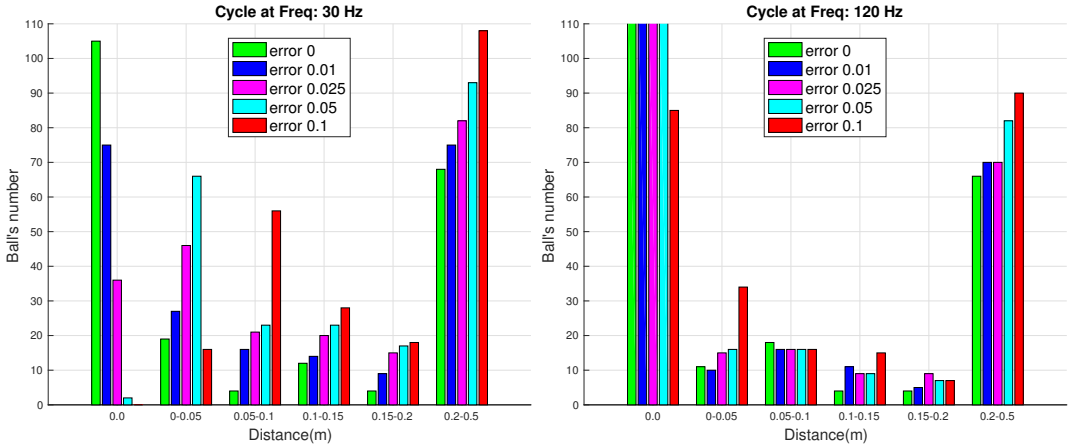
(d) Cycle at 120 Hz with manipulability pose

Figure 4.13: Catching predictive cycle with error 0.1 m



(a) Histogram at 30 Hz (b) Histogram at 120 Hz

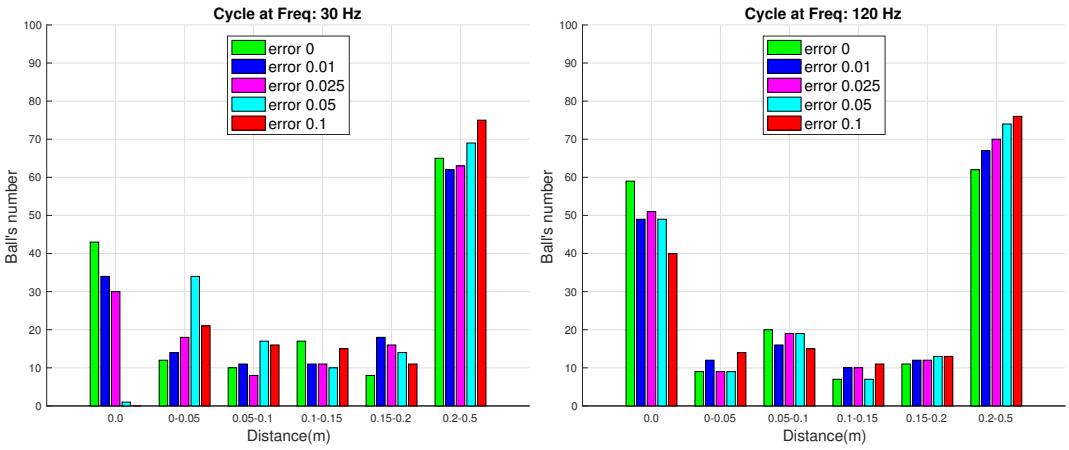
Figure 4.14: Histograms of predictive cycle with different errors and with limitations starting from standard pose



(a) Histogram at 30 Hz (b) Histogram at 120 Hz

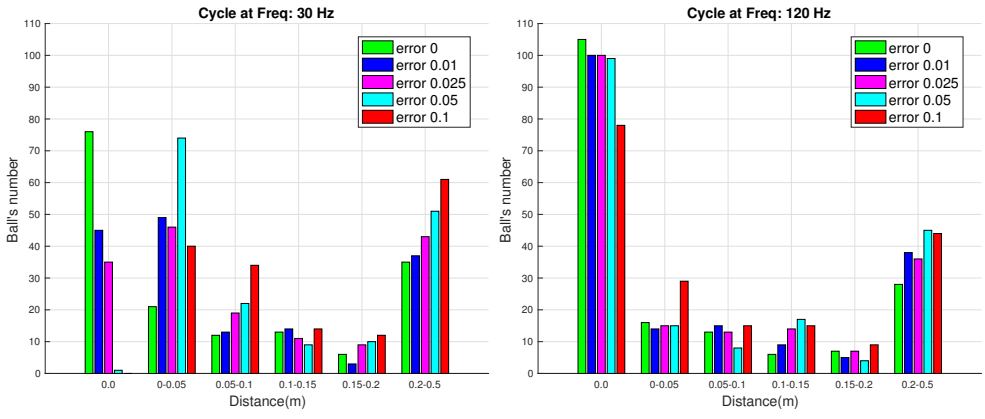
Figure 4.15: Histograms of predictive cycle with different errors without limitations starting from standard pose

frame rate causes an improvement with respect to the lower one, but it is worst in general for small error but better for higher error. Using this starting pose, there are more balls that are not caught.
The low number of balls caught using the first strategy (Section 4.3.1) forced to find a new one to evaluate (Section 4.3.2). The results confirm the effort and also with low frequency rate it is possible to have acceptable results and select it as approach to be used.
Figure 4.17a can be compared to Figure 4.16a. In this comparison, there is a big difference among the errors considered but the ratio remains similar in both cases. The same evaluation can be done with Figure 4.17b where the difference is in the highest number of balls caught due to the higher frame rate.



(a) Histogram at 30 Hz (b) Histogram at 120 Hz

Figure 4.16: Histograms of predictive cycle with different errors and with limitations starting from manipulability pose



(a) Histogram at 30 Hz (b) Histogram at 120 Hz

Figure 4.17: Histograms of predictive cycle with different errors without limitations starting from manipulability pose

4.4 Prospective Control

The prospective control is a totally new idea, completely different from the predictive control, just described in Subsection 4.3.2.
It tries to reduce the error between the ball and the end-effector each frame but takes into account also the time to contact.
The procedure starts from the model, described by Bootsma et al. [29] that controls the amount of acceleration produced on the basis of the difference between the required velocity and the current velocity of the end-effector. The required velocity at any instant is equal to the ratio of the current lateral distance between the end-effector and the ball to the current first-order time to contact between the ball and the end-effector movement axis. The first-order time-to-contact (TTC_1) at any instant is equal to the ratio of the current distance between the moving ball and the end-effector movement axis to the current speed of the moving ball.
One of the advantages of the required velocity model is that, it proposes an interesting concept to link these three variables. The distance traveled and the speed are not very important on their own; it is the relationship between the two that is relevant for movement control.

4.4.1 Time to contact

The time to contact is one of the most important aspect of this control. This function can be of first or second order. First order time to contact for linear motion is calculated as:

$$TTC_1(x) = \frac{x}{\dot{x}} \tag{4.10}$$

and for angular motion is:

$$TTC_1(\theta) = \frac{\theta}{\dot{\theta}} \tag{4.11}$$

Second order time to contact for linear motion is calculated as:

$$TTC_2(x) = \frac{-\dot{x} + \sqrt{\dot{x}^2 - 2 \cdot \ddot{x} \cdot x}}{\ddot{x}} \tag{4.12}$$

and for angular motion is:

$$TTC_2(\theta) = \frac{-\dot{\theta} + \sqrt{\dot{\theta}^2 - 2 \cdot \ddot{\theta} \cdot \theta}}{\ddot{\theta}} \tag{4.13}$$

where x is the distance between the ball and the end-effector, $\dot{x}$ is the instantaneous linear velocity, $\ddot{x}$ is the instantaneous linear acceleration; θ is the angle formed by the ball and the target with respect to the end-effector, $\dot{\theta}$ is the instantaneous angular velocity of the closure of this angle, and $\ddot{\theta}$ is the angular acceleration of the closure of this angle.
In this work, the time to contact chosen is the first order for linear motion of (4.10), as described in [4]. The angular motion is not used in this case because there is not a clear target point, while the second order time to contact is not necessary for this first view of the control.

4.4.2 Update law

The acceleration law is at time t_i:

$$\ddot{\mathbf{x}}_{ee}(t_i) = \alpha\dot{\mathbf{x}}_{ee,req}(t_i) - \beta\dot{\mathbf{x}}_{ee}(t_i) \tag{4.14}$$

with:

$$\dot{\mathbf{x}}_{ee,req}(t_i) = \frac{\mathbf{x}_{ee}(t_i) - \mathbf{x}_{ball}(t_i)}{TTC_1(x)} \tag{4.15}$$

where $\ddot{\mathbf{x}}_{ee}$ is the acceleration, $\dot{\mathbf{x}}_{ee,req}$ is the required end-effector velocity and $\dot{\mathbf{x}}_{ee}$ is the end-effector velocity. The parameters (gains) α and β are constant, while $\mathbf{x}_{ee}$ is the end-effector's position, $\mathbf{x}_{ball}$ is the ball's position and TTC_1 is the first order time to contact between the ball and the end-effector, computed in this way from (4.10):

$$TTC_1(x) = \left\| \frac{x_{ball}(t_{i-1}) - x_{ee}(t_{i-1})}{x_{ball}(t_{i-1}) - x_{ball}(t_{i-2})} \right\| \cdot \Delta t \tag{4.16}$$

.

4.4.3 Tuning of α and β

The main complication of the prospective control is to find the right values for α and β. It does not exist a rule to select it, thus the values are selected using a trial and error method cycle. This cycle combined values $\alpha = 10^{\{-2,-1,0,1,2\}}, 5^{\{-1,1\}}, 2^{\{-1,1\}}$ and $\beta = \{1,2,3,4,5,6,7,8\} \cdot \frac{\alpha}{4}$.
For a 2D-DOF robotic arm, the cycle creates $9{\cdot}9{\cdot}8{\cdot}8 = 5184$ different combinations for α_x, α_y, β_x and β_y, but only 381 are good enough to catch the ball. This cycle uses the same trajectory for all the possible combinations. After it, a successive cycle tries the 381 successful combinations with a different trajectory, where only 6 combinations of values leads to a ball caught by the robotic arm. These results confirm the enormous difficult to find right values for this law and suggest to look for other possible solutions.

4.4.4 Performance Evaluation of Prospective Control

Montagne et al. in [29] and [30] describe a movement in only 1 axis and it is easy to find good values of α and β. The application in 2D or 3D brings problems. The strategy considers the end-effector as a point that can move wherever in the axis without limitation. Adding the link length, joint positions and velocities limitations, the strategy becomes even harder to setup. This section discusses four trials: 1D, 2D with two planes xy and xz and 3D. Algorithm 4.18 is for 1D and 2D with plane xz, while 2D with plane xy and 3D use Algorithm 4.19.

1D Prospective Control

The 1D Scenario, in Figure 4.20, shows the movement of the point, that represents the possible end-effector in only one axis, the x-axis, and tries to catch the ball using Algorithm 4.18. The model is very simple but necessary for a first examination of the problem and a recognition of the importance of the tuning of α and β. This scenario selects $T_s = 1/1000\,\text{s}$, $\alpha_x = 50$ and $\beta_x = 25$. In the case of Figure 4.20, the ball starts from $x_{0\_ball} = 1.0\,\text{m}$, $y_{0\_ball} = 3.5\,\text{m}$ with velocity $v_{x\_0\_ball} = 4\,\text{m/s}$, $v_{y\_0\_ball} = 4\,\text{m/s}$ and angle $\theta = -100\frac{\pi}{180}\text{rad}$ that intersects the x-axis in $x_{0\_ball} - \frac{y_{0\_ball}}{\tan(\pi+\theta)}$, while the point (end-effector) starts in position $x_{0\_point} = 0.0\,\text{m}$, $y_{0\_point} = 0.0\,\text{m}$ with velocity $v_{x\_0\_point} = 0\,\text{m/s}$, $v_{y\_0\_point} = 0\,\text{m/s}$.

```
Vector Initialization;
Gain selection;
for i ← 2 to balls seen by camera do
    Creation of parabolic motion;
    Compute TTC for x and for x, z;
    Compute required end-effector velocity for x and for x, z;
    Compute required end-effector acceleration for x and for x, z;
    Compute numerical integration (Euler method);
    if distance ball-ee < 0.01m then
        Ball caught;
    end
end
```

Algorithm 4.18: Prospective Control in 1D and in 2D with plane xz

```
Vector Initialization;
Gains selection;
for i ← 2 to balls seen by camera do
    Creation of parabolic motion;
    Compute TTC for x, y and z;
    Compute required end-effector velocity for x, y and z;
    Compute required end-effector acceleration for x, y and z;
    Compute numerical integration (Euler method);
    if distance ball-ee < 0.2 m (2D xy) or distance ball-ee < 0.5 m (3D) then
        Change gains;
        Compute TTC for x, y and z;
        Compute required end-effector velocity for x, y and z;
        Compute required end-effector acceleration for x, y and z;
        Compute numerical integration (Euler method);
    end
    if distance ball-ee < 0.01 m then
        Ball caught;
    end
end
```

Algorithm 4.19: Prospective Control in 2D with plane xy and in 3D

Figure 4.21a shows in the top the position of the ball (blue line) and of the hand (red line) w.r.t. x-axis, while in the bottom the error between the two that this algorithm try to reduce. The ball is caught after about 0.887 s, time when the ball passes through the x-axis and unique step when it can be caught.
Figure 4.21b represents the required velocity and the effective velocity of the hand along the x-axis. After an initial time where the hand moves to the x-position of the ball the required velocity reach a constant value and when it equals the position it changes oscillating between two opposite values, until it catches the ball. This is due to the gains α and β that lead the hand to overshoot and bring the velocity to oscillate, but also tuning the gains did not give better results.

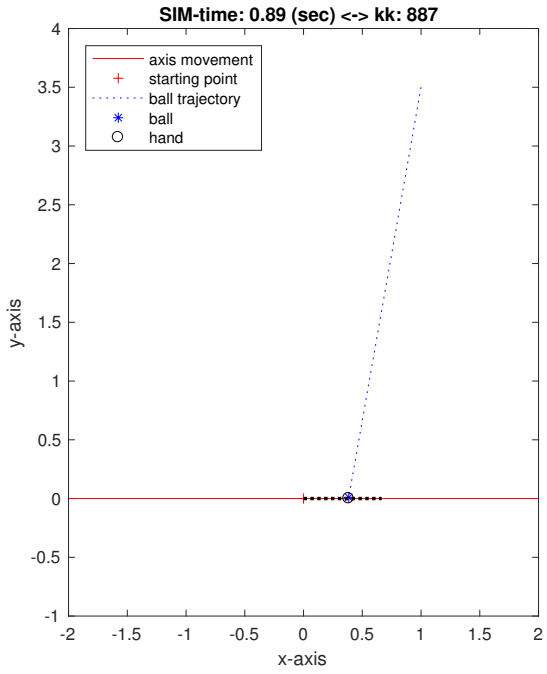
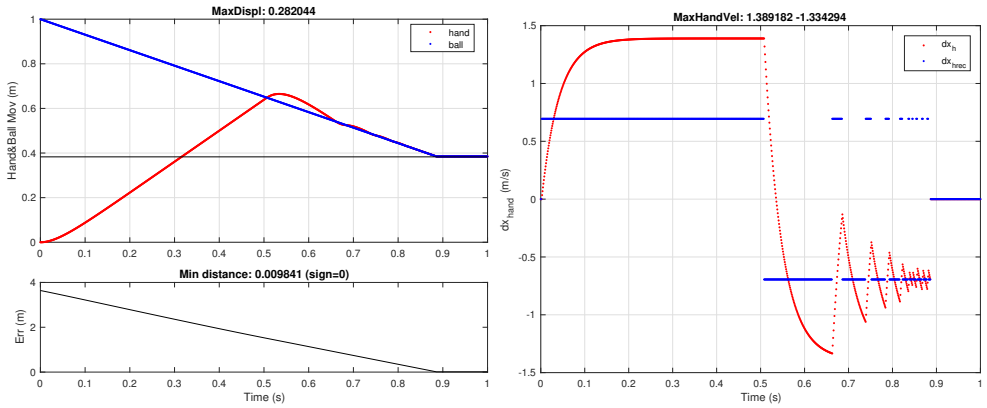


Figure 4.20: 1D Prospective Scenario



(a) Position of the hand and the ball in 1D (b) Velocity of the hand in 1D

Figure 4.21: Position and Velocity of the hand in 1D

2D Prospective Control in plane xy

The second implementation, in Figure 4.22, shows the movement of the point in two axes, x and y-axes, expressed by Algorithm 4.19. It maintains the same block of the 1D case (4.4.4) with $T_s = 1/1000\,\text{s}$, with the addition of the computation about the y-axis and the change of gains when the ball is less than 20 cm distant from the hand.

At the beginning, using the same gain for x and y-axes, it is possible to obtain a bigger velocity and therefore bigger movement, due to the high distance in y-axis between the ball and the hand. Thus, the constraint is changing gains, reflecting on the fact that in future works the idea is substituting the hand with the end-effector of the robot that has joint limits to be considered.
In this example, gains in the first phase are $\alpha_x = 100$ and $\beta_x = 45$, $\alpha_y = 1$ and $\beta_y = 1$ to limit the movement in y-axis. Then, with distance less than 20 cm, gains are equal

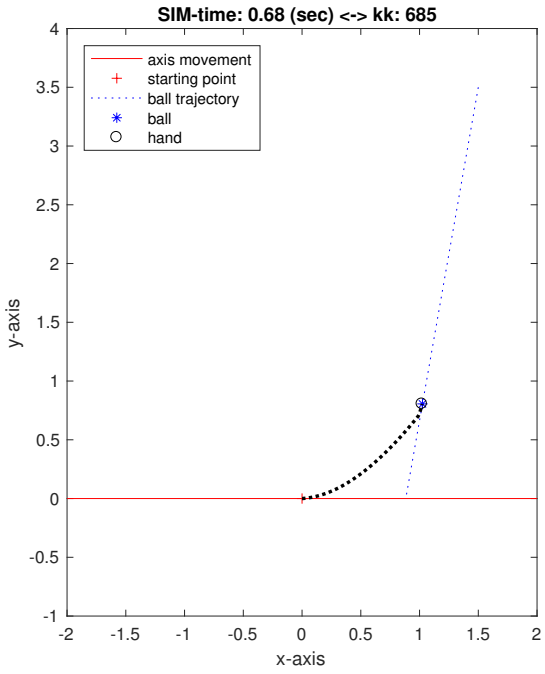
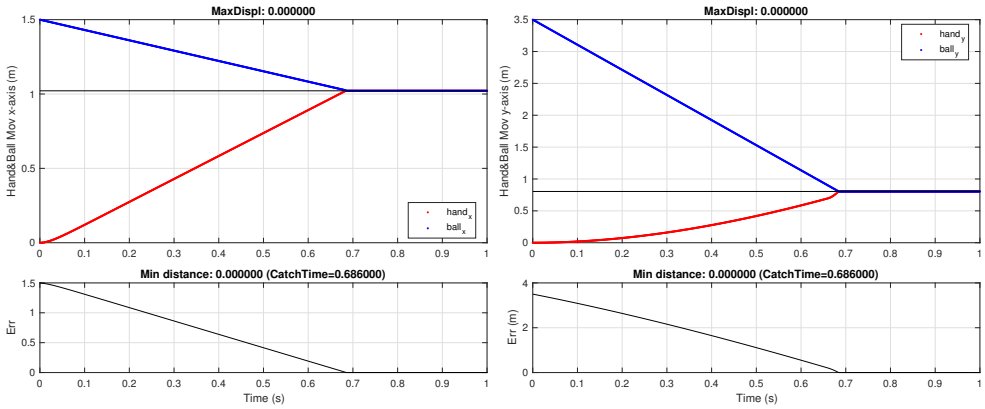


Figure 4.22: 2D Prospective Scenario

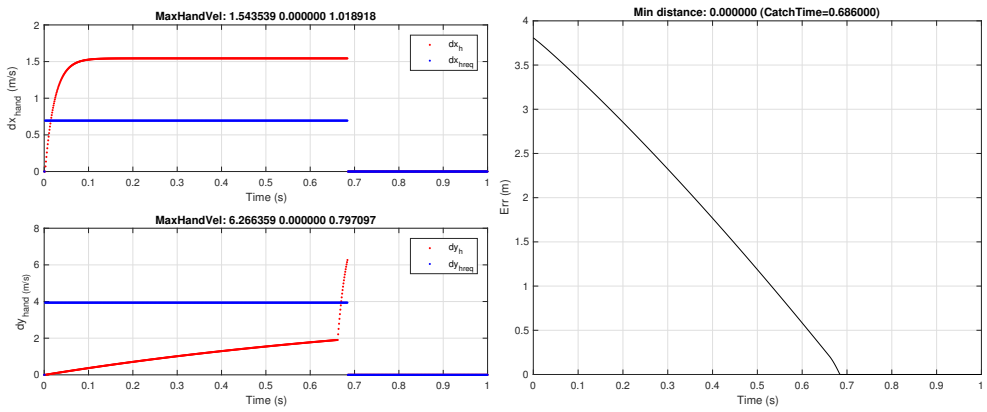
in both directions (e.g. $\alpha_{x,y} = 100$ and $\beta_{x,y} = 45$).
Figure 4.23 has in the left (Figure 4.23a) the position of the ball (blue line) and of the hand (red line) w.r.t. x-axis, while in the right (Figure 4.23b) is w.r.t. y-axis. In the bottom of each it is represented the error between ball and hand in their respective axis. The ball is caught at about 0.686 s, and this is represented by the error that is zero from that time and the two positions that still in the same line after that time.



(a) X-axis position of the hand in 2D (b) Y-axis position of the hand in 2D

Figure 4.23: Position and Velocity of the hand in 2D

Figure 4.24 is split in two subfigures: Figure 4.24a in the top shows the required (blue line) and real velocity (red line) in x-axis and in the bottom shows them in y-axis, where it is easy to recognize the change when the ball is close to the hand (20 cm).
Figure 4.24b represents the absolute error between ball and hand, confirming the linear trend to zero and the catching at about 0.686 s.



(a) Velocity of the hand and the ball in 2D (b) Distance between ball and hand in 2D

Figure 4.24: Velocity and Distance of the hand in 2D

2D Prospective Control in plane xz

The second implementation in 2D, in Figure 4.25, maintains the movement of the point in two axes, but in x and z-axes, expressed by the Algorithm 4.18. It maintains the same block of the 1D case (4.4.4) with $T_s = 1/1000\,\text{s}$, with addition of the computation about z-axis.
It uses also the parabolic motion instead of linear to control z-axis, with equations of Section 4.2.1, with starting point in $x_{0\_ball} = 1.5\,\text{m}$, $y_{0\_ball} = 3.5\,\text{m}$, $z_{0\_ball} = 0.0\,\text{m}$, starting velocity $|v_{0\_ball}| = 6.1\,\text{m/s}$, angles $\theta = 45\frac{\pi}{180}\text{rad}$ (formed by the motion with plane xy) and $\phi = 80\frac{\pi}{180}\text{rad}$ (formed by the motion with plane yz). In this example, gains are $\alpha_x = 90$, $\alpha_z = 50$ and $\beta_x = 50$, $\beta_z = 45$.

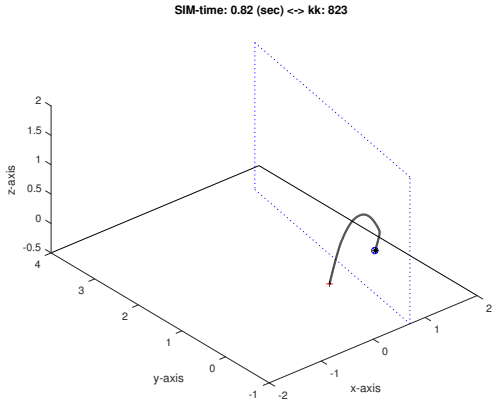
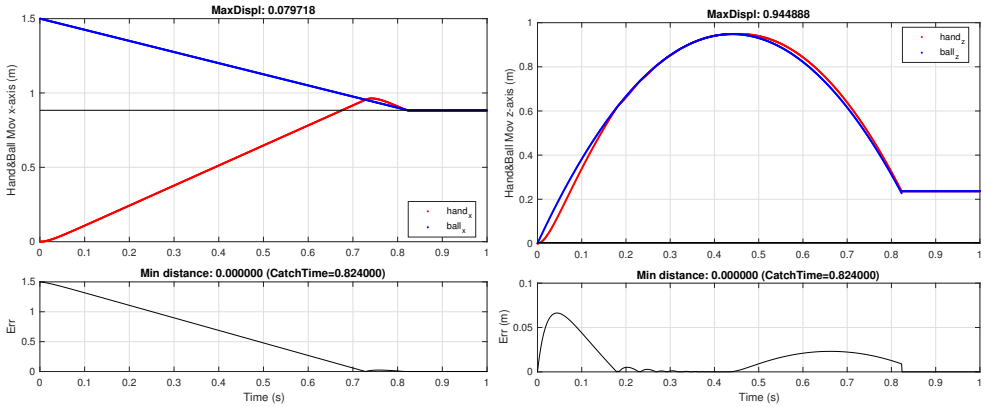


Figure 4.25: 2D Prospective Scenario in plane xz

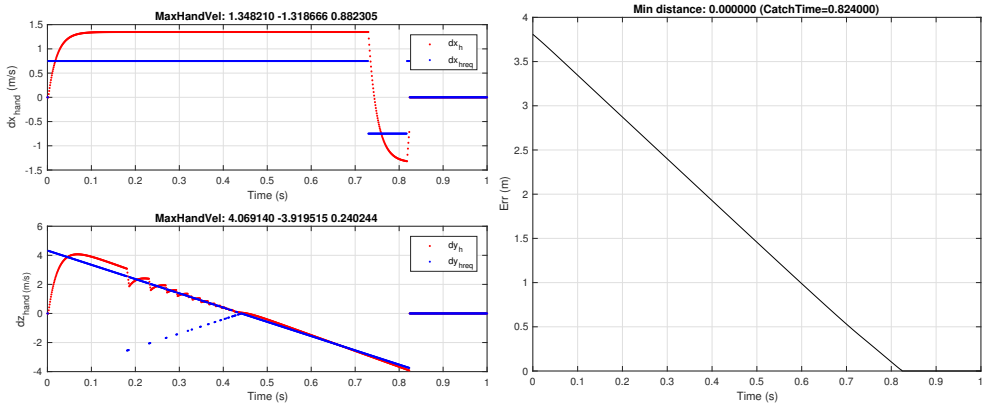
Figure 4.26 has in the left (Figure 4.26a) the position of the ball (blue line) and of the hand (red line) w.r.t. x-axis, while the right (Figure 4.26b) is w.r.t. z-axis. In the bottom of each there is the error between ball and hand in their respective axis, that goes linearly to zero in x and change its trend in z. This depends on the fact that in x case ball and hand start at a different point and the hand tries to reach the ball position, while in z case they start from 0 and the hand can follow the ball. The ball is caught at about 0.822 s (i.e. the time when the ball intersect the x-axis), represented by the error that is zero since that time and the two positions that still in the same line after that time.



(a) X-axis position of the hand in plane xz (b) Z-axis position of the hand in plane xz

Figure 4.26: Position and Velocity of the hand in 2D in plane xz

Figure 4.27a shows in the top the required (blue line) and real velocity (red line) in x-axis while the bottom shows them in z-axis, where there is a small oscillation due to the gains found. At the beginning gains was $\alpha_x = \alpha_z = 100$ and $\beta_x = \beta_z = 45$, but the oscillation was even bigger. A tuning session found these parameters that let the system to have low oscillation in z-axis and almost zero in x-axis.
Figure 4.27b represents the absolute error between ball and hand, confirming the catching at about 0.822 s and the linear trend to zero.



(a) Velocity of the hand and the ball in plane xz (b) Distance between ball and hand in plane xz

Figure 4.27: Velocity and Distance of the hand in 2D in plane xz

3D Prospective Control

The third implementation, expressed by the Algorithm 4.19, is in three dimensions, with the same addition and change of gains explained in Section 4.4.4 plus the z-axis. It uses, as in Section 4.4.4, the parabolic type of motion instead of linear, with the equations of Section 4.2.1 and the same initialization plus the y-axis.
The point that represents the hand can move in the three x, y and z-axes. Initial gains are not equal for the three axes (e.g. $\alpha_x = 100$ and $\beta_x = 45$, $\alpha_z = 100$ and $\beta_z = 45$ while $\alpha_y = 1$ and $\beta_y = 1$) and when between the ball and the hand there is a distance less than half a meter (e.g. distance(ball,hand) < 0.5 m), gains of y-axis

become equal to x and z-axes. In this way, the point can move initially to the right plane where the ball will fly and when it is almost in the plane, it can move to catch it with more success. This is one of the solutions tried to solve the problem of tuning α and β.

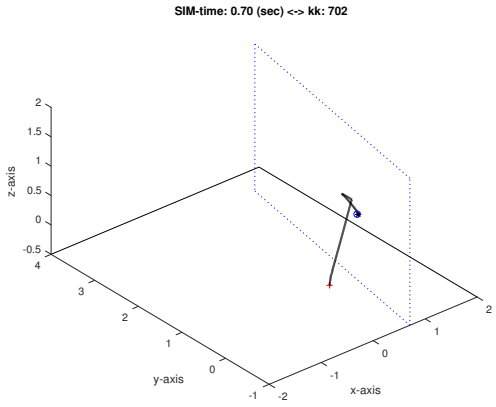
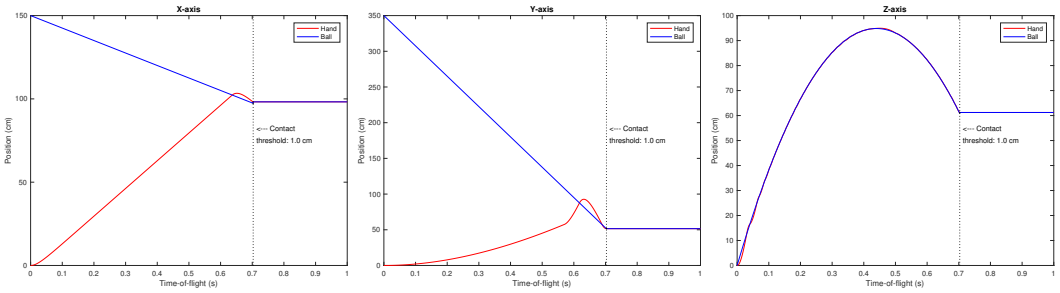


Figure 4.28: 3D Prospective Scenario

Figure 4.29 has in the left (Figure 4.29a) the position of the ball (blue line) and of the hand (red line) w.r.t. x-axis, in the center (Figure 4.29b) the same positions w.r.t. y-axis and in the right (Figure 4.29c) w.r.t. z-axis. In this case, the ball is caught at about 0.702 s and after that time ball and hand still in the same positions for each axis.

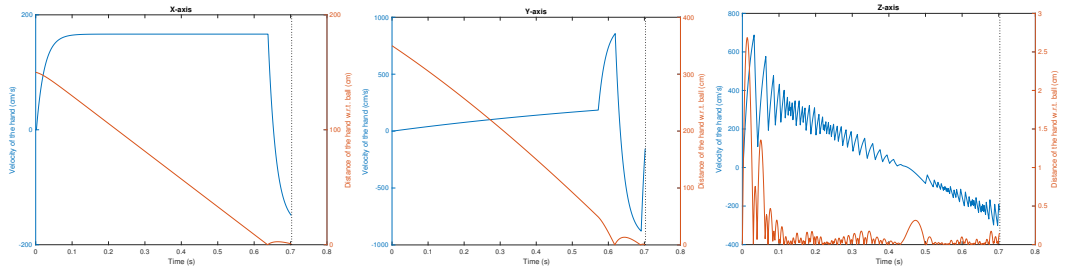


(a) X axis position of the hand and the ball in 3D (b) Y axis position of the hand and the ball in 3D (c) Z axis position of the hand and the ball in 3D

Figure 4.29: Position of the hand and the ball in 3D for each axis

Figure 4.30 is split in three figures. Each subfigure 4.24a shows the velocity of the hand with a cyan line and the distance of the hand w.r.t. ball for each axis, with Figure 4.30a for x-axis, Figure 4.30b for y-axis and Figure 4.30c for z-axis. From that, it is possible to recognize that the error becomes zero at 0.702 s, confirming the catch point and time. Figure 4.31 represents the absolute value of the velocity of the hand and of the distance between ball and hand where the hand comes very close to ball at about 0.62 s. It was not possible to find the right tuning of α and β to catch it at that time.

This evaluation shows the severe problem of tuning. It has to be highlight that frame rate used is very high (e.g. 1000 fps) for this evaluation. This value is impossible to have in real time application, but it is necessary to show acceptable results that cannot be reached using a Kinect with only 30 fps. Thus, it is problematic to use only Prospective control, but it can be used in a hybrid system with Predictive, to move



(a) X-axis velocity of the hand and distance in 3D (b) Y-axis velocity of the hand and distance in 3D (c) Z-axis velocity of the hand and distance in 3D

Figure 4.30: Velocity of the hand and distance in 3D for each axis

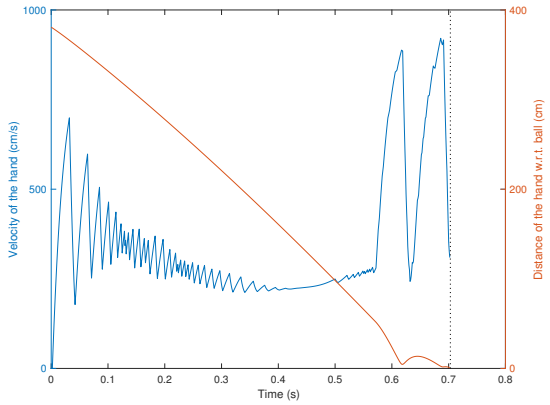


Figure 4.31: Velocity of the hand and distance between hand and ball in 3D Prospective

from the first time to the possible traveling plane of the ball and then correct the landing point using Predictive.

4.5 Discussion about results

The problem of tuning the parameters of the Prospective control demonstrates that this is not a suitable control for ball catching task. Indeed, to use it is needed a very high frame-rate that it is not possible to have with camera available in shop at an accessible price. The other problem is the tuning of the parameters that needs to adapt to each situation that can occur. The impossibility of finding a general solution using the prospective control leads to other strategies to find out. If Prospective Control can be considered a successful solution, it has to work with other plans of action to find a solution that can work in various situations, that it is not achievable only with prospective. The idea will be to improve this control and try a hybrid version with predictive to obtain the desirable control to perform this task.
Therefore, this chapter validates the predictive control with minimum distance from the end-effector as the best possible strategy for this work. It is clear that this solution is better than the one used in previous works, that intersected the predicted trajectory with a plane, forcing the robot to move only to this plane without using all the 3D area. Due to this, the minimum distance predictive control is the choice implemented in ROS using C++ functions, that will be explained in Chapter 5, creating a simulation with only this effective control. Nothing of the other idea need to be discarded at all,

because this work is a starting point for other possible successful idea, using new control strategies.

Chapter 5

Implementation in ROS

This chapter describes the implementation in ROS of the control strategies studied in the previous Chapter 4. This is the transposition from MATLAB® code to C++ programming language in ROS that is necessary to use with a real robot in real time. Learning ROS is one of the objective of the master thesis and this chapter describes mainly what has been learned about this fundamental tool.
The implementation takes into account both real and simulated balls. Using the vision part, described in Section 5.1, ROS with C++ code detects the real ball and save a matrix including all the points seen by Kinect of the ball thrown. Instead, for the simulated ball, using the projectile motion creation (Section 4.2.1), it considers directly a matrix including all the points created by MATLAB®. RViz visualizes the ball and the robot while ROS publishes and subscribes to them, performing a complete simulation of ball catching task. All the nodes that will be presented in this chapter are ready to be used also by the real robot in future work.

5.1 Improvements on the Visual Perception

This section describes the vision algorithms used and developed within this work for ball, body and hand detection. It starts from the type of ball trajectory, the methods considered for detecting the ball, considerations about the body and hand detection and the methods used to predict the ball.
At the beginning, following previous works [45], there are two solutions to detect the ball: one based on voxelization grid [10], one based on color [33]. The challenge is to develop a reliable 3D ball tracking algorithm in order to provide a good trajectory estimation, because the more ball's points the system has, the better the trajectory is estimated.

5.1.1 Ball Detection

Computer vision includes four phases: object detection, segmentation, tracking, and estimation. Robotic arms use them to recognize and manipulate objects. The main problems for these algorithms are the changes in illumination, occlusion, scales, background and dynamics. In this case, the necessity for a quick estimation of the trajectory given the time constraints of ball catching make the process even more challenging, because of these factors, mixed with computational performances and sensor errors.
Kinect sensor has limits in depth working range (Section 2.3.2) that can be overtaken suffering more errors. The thesis explores two different methods of ball detection and tracking: the first method uses OpenCV with color processing in the images to detect the ball; the second uses point cloud data of the environment acquired from Kinect's

depth sensor. The latter suffers more with errors at a longer distance from the sensor, while the first depends more on illumination.

Point Cloud data with CGrid 3D

A point cloud is a set of data points in a 3D coordinate system. These points represent the external surface of an object and are usually defined by x, y, and z coordinates. A voxel represents a 3D volume on a regular grid in 3D space. The name is originated from *vo* representing volume and *el* representing element. A voxel is considered as a box of certain dimension, A set of voxel creates a mask and a voxel grid is a group containing masks. To detect the ball, given the properties of a flying ball for ball catching task, the work space is voxelized in an occupancy grid space rather than considering the whole cloud of points. The workspace is limited to maximize the algorithm execution speed.
The flying object [10], using this method, is an object that occupies a given number of voxels with a minimum number of points and whose surrounding voxels are empty. The grid and mask size depend on the ball's size or radius. The grid size must be large enough to allow that a real flying ball, when voxelized, does not become smaller than the space between any two planes. This is a big issue if the ball is at further distances. The grid mask must be large enough to accommodate a volume larger than the ball, since a ball with its high speeds achieved can occupy more than its normal volume. If the ball is on the ground, it is possible to detect it considering only the upper half of isolated cloud. This is done in cooperation with plane segmentation. A flying object is defined as an object that occupies a given number of voxels with a minimum number of points and whose surrounding voxels are empty. It can be seen as a 3D mask inside voxels non-empty (containing a minimum number of points) and the outside voxels empty. Figure 5.2 represents an example of 3D mask.
Number of voxels in the grid of specific volume depends on ball radius: larger the ball radius, less the number of voxels. This method works fairly well and it avoids most of false positive.
Implemented algorithm is resumed in Algorithm 5.1 using only point cloud data given by depth sensor.

```
for each point cloud do
    Voxelization of grid;
    Detection of flying ball using mask;
    Computation of ball's 3D coordinates;
end
```

Algorithm 5.1: Point cloud ball detection

First, the point cloud data are acquired using OpenNI and subscribing in ROS the depth sensor. Point cloud data are used to voxelized the workspace creating a grid, according to ball size.
Then the algorithm look for flying object in the voxelized workspace using a mask of a given size.
Third, if the flying ball is found, ball's 3D coordinates are computed and published for further processing. RViz shows the ball and the confirmation is done visually.
This method is implemented in ROS platform [45]. The topic `/camera/depth/points` is used to get cloud of points. At a later stage, point cloud are voxelized knowing the volume. Mask are used in the voxelized grid and, if it found any isolated set of

points, it computes the center. Other topics of interest are published by `vision_node` for further use. The topic `/ballCord` is subscribed by `/estimation_node` to estimate ball's landing point, while a marker (`/ball3D`) is published to RViz for visualization in real-time.
The Kinect works at 30 Hz but is slower because give the large amount of data to process point cloud data. The performance is not affected by light and background but with the ball in hand it loses a lot of points, because the algorithm can not find the ball in a voxel space if there is the hand that occupies the voxel with the ball. This is useless for a real-time work with a limited frame rate, due to Kinect sensor. Hence, the idea in this work is to use color detection with some limitations to avoid errors in detection.

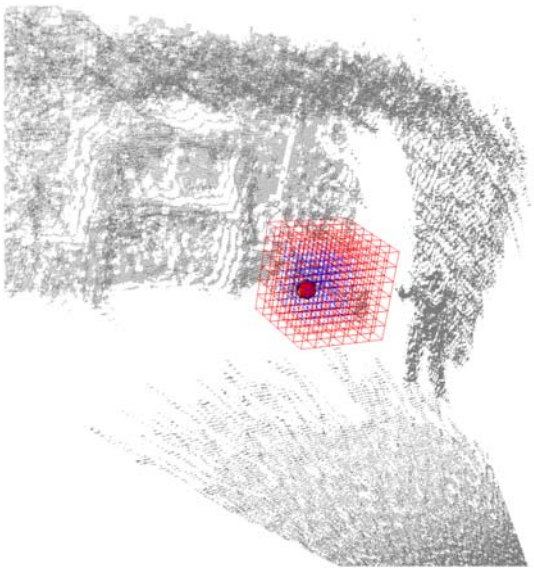


Figure 5.2: Example of 3D Mask used for flying object identification. [10]

Color detection

The choice of coming back to color detection is done with limitations in the environment about lights. Controlling lights in the laboratory and using a glove in the hand of the person who throw the ball it is possible to obtain good object identification.
The object segmentation depends on light conditions, because it is important the values of the color but also the color difference between the object and the background. Color selection were tested using implementations based on OpenCV with HSV and RGB.
HSV stands for Hue, Saturation and Value (i.e. Brightness or Lightness) and is a representation of points in a RGB color model. Based on CIECAM02 that is the color appearance model (abbreviated CAM), a mathematical model that seeks to describe the perceptual aspects of human color vision, published in 2002 by the International Commission on Illumination (CIE) Technical Committee 8-01, Hue is defined technically as *the degree to which a stimulus can be described as similar to or different from stimuli that are described as red, green, blue, and yellow.* Saturation is the *colorfulness of an area judged in proportion to its brightness*, which in effect is the perceived freedom from whiteness of the light coming from the area. Value is the *attribute of a visual sensation according to which an area appears to emit more or less light (radiating or reflecting light)*, or also a representation of variation in the perception of a

color or color space's brightness.
The RGB (Red, Green and Blue)color model is an additive color model in which red, green and blue light are added together in various ways to reproduce a broad array of colors.
The procedure works as described in [45]. OpenCV functions are widely used in this part. The RGB color image is converted to HSV color-space. In this space the image can be easily threshold to extract the desired object through color. The algorithm obtains first the object's center in the color image and, after, the 3D coordinates of the detected object in 3D scene, since depth and color image are registered in Kinect sensor.

HSV image The user chooses the color with a track-bar, representing ranges of values in the three specifications of HSV. In this way, the selection needs to give to the ball a white color, while the background is black. After the threshold, the image stills have some small white points. These points can be noise or other object with the same color of the ball. Normally, due to light conditions and reflection, the image could have some false positive objects.

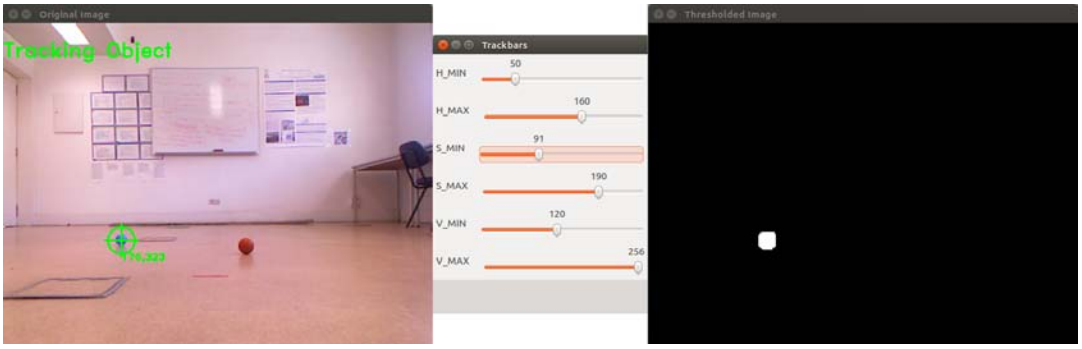


Figure 5.3: Blue ball seen with Kinect sensor

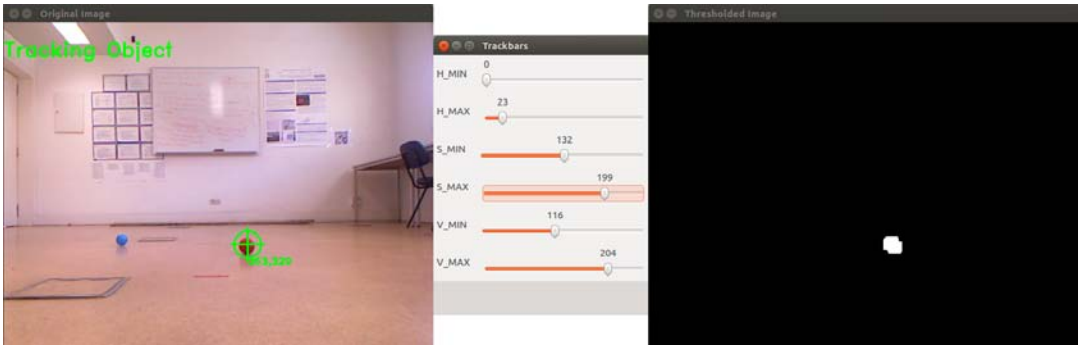


Figure 5.4: Orange ball seen with Kinect sensor

Mathematical morphology, as its name implies, is concerned with the form or shape of objects in the image.
A common use of morphological opening is to remove noise after a thresholding operation. The intention is to remove the white pixels that do not belong to the ball and to fill in the black holes in the ball. With a symmetric circular structuring element, the procedure applies a closing operation to fill the holes in the ball. After that, the noise pixels have grown to be small circles and some have agglomerated. An opening

operation eliminates that and cleans up the image. If operations are applied in the inverse order, opening then closing, the results are much poorer. Although the opening removes the isolated noise pixels, it removes large chunks of the targets which cannot be restored.
Morphological opening is done with image erosion followed by a dilation, while the closing with image dilation followed by an erosion.
Figure 5.3 represents on the left the color image by Kinect with the tracked ball, selected using the HSV values in the central track-bar. On the right there is the threshold image to evaluate and tune the values in the image where the white represents the color area selected using values $H \in [50; 160], S \in [91; 190], V \in [120; 256]$.
Figure 5.4 shows same windows with different values to select the orange ball that is on the right of the blue ball using values $H \in [0; 23], S \in [132; 199], V \in [116; 204]$.

Blob detection The detection of the ball's center uses moments.
Moments are a rich and computationally cheap class of image features which can describe region size and location as well as shape. The moment of an image I is a scalar:

$$m_{pq} = \sum_{(u,v)\in I} u^p v^q I[u, v] \tag{5.1}$$

where $(p + q)$ is the order of the moment. The zero-th moment $p = q = 0$ is:

$$m_{pq} = \sum_{(u,v)\in I} I[u, v] \tag{5.2}$$

and for a binary image where the background pixels are zero this is simply the number of non-zero (white) pixels.
Moments can be given a physical interpretation by regarding the image function as a mass distribution. Consider the region as being made out of thin metal plate where each pixel has one unit of area and one unit of mass. The total mass of the region is m_{00} and the center of mass or centroid of the region is

$$u_c = \frac{m_{10}}{m_{00}} \qquad v_c = \frac{m_{01}}{m_{00}} \tag{5.3}$$

where m_{10} and m_{01} are the first-order moments.
The central moments are computed with respect to the centroid:

$$\mu_{pq} = \sum_{(u,v)\in I} (u - u_c)^p (v - v_c)^q I[u, v] \tag{5.4}$$

and are invariant to the position of the region. They are related to the moments m_{pq} by:

$$\mu_{10} = 0, \qquad \mu_{01} = 0 \tag{5.5}$$

$$\mu_{20} = m_{20} - \frac{m_{10}^2}{m_{00}}, \quad \mu_{02} = m_{02} - \frac{m_{01}^2}{m_{00}}, \quad \mu_{11} = m_{11} - \frac{m_{10} m_{01}}{m_{00}} \tag{5.6}$$

Using the thin plate analogy again, the inertia matrix of the region is:

$$J = \begin{bmatrix} \mu_{20} & \mu_{11} \\ \mu_{11} & \mu_{02} \end{bmatrix} \tag{5.7}$$

about axes parallel to the u- and v-axes and intersecting at the centroid of the region. The central second moments μ_{20}, μ_{02} are the moments of inertia and μ_{11} is the product of inertia. The product of inertia is non-zero if the shape is asymmetric with respect to the region's axes.
The equivalent ellipse is the ellipse that has the same inertia matrix as the region. The eigenvalues (denoted as λ_1 and λ_2) and eigenvectors (denoted, respectively, as v_1 and v_2) of J are related to the radii of the ellipse and the orientation of its major and minor axes. The maximum and minimum radii of the equivalent ellipse are:

$$a = 2\sqrt{\frac{\lambda_2}{m_{00}}}, \qquad b = 2\sqrt{\frac{\lambda_1}{m_{00}}} \tag{5.8}$$

respectively where $\lambda_2 \geq \lambda_1$. The ratio b/a is the aspect ratio of the region and is a useful measure to characterize the shape of a region that is scale and rotation invariant. The eigenvectors of J are the principal axes of the ellipse, namely, the directions of its major and minor axes. The major, or principal, axis is the eigenvector corresponding to the maximum eigenvalue; according to the notation this eigenvector is v_2. The orientation of the ellipse is computed using the angle of v_2 with the respect to the horizontal axis, namely,

$$\theta = \tan^{-1} \frac{v_{2,y}}{v_{2,x}} \tag{5.9}$$

where $v_{2,x}$ and $v_{2,y}$ are, respectively, the x and y components of v_2.
The orientation and aspect ratio of the equivalent ellipse is a useful indicator of the region's shape and orientation.
To summarize, the procedure creates an image containing a spatially contiguous set of pixels corresponding to one of the objects in the scene that is segmented from the original color image. It determines its area, a box that entirely contains it, the location of its centroid, the equivalent ellipse and its orientation.
In order to recognize particular objects, it needs some measure of shape that is invariant to the relative pose of the camera and the object. This section concerns only with planar objects that are fronto-parallel to the camera and which are subject to translation, rotation and scale change. The shape of an object can be described very simply by the aspect ratio, the ratio of major to minor ellipse axis lengths a/b, and this is invariant to translation, rotation and scale. Another commonly used and intuitive shape feature is circularity which is defined as:

$$\rho = \frac{4\pi m_{00}}{p^2} \tag{5.10}$$

where p is the region's perimeter length. Circularity has a maximum value of $\rho = 1$ for a circle, is $\rho = \pi/4$ for a square and zero for an infinitely long line. Circularity is also invariant to translation, rotation and scale. More complex ratios of moments can be used to form invariants for recognition of planar objects irrespective of position, orientation and scale, see for instance Hu moments which are invariant to translation, scale and orientation within a plane.
In this thesis, the centroid is the parameter extracted with OpenCV. The equation (5.4) computes the moments of the object.

ROS implementation for ball detection The ball detection and tracking algorithm is implemented in ROS using Kinect's depth sensor, OpenNI, and OpenCV.
The message format of ROS for images is `sensor_msgs/Image`, but this format is not compatible with OpenCV. The ROS image needs to be converted to be used in

OpenCV. ROS library provides CvBridge, an interface between ROS and OpenCV. CvBridge defines a CvImage type containing an OpenCV image, its encoding and a ROS header and has exactly the same information of `sensor_msgs/Image`. After conversion to OpenCV format, it can be processed to detect objects. The moments method to detect the center gives it in pixels. To get ball position in 3D space it is necessary to use point cloud data from Kinect's depth sensor.
The procedure subscribes two topics, the RGB image and point-cloud topics with synchronization in time, to get ball position with time. The point cloud can be accessed in real-time to get 3D coordinate of a particular pixel-index. Further details about the camera calibration process are shown in Appendix B, establish the relation between 2D pixel index and 3D point.
Algorithm 5.5 resumes the procedure:

for *each frame received* **do**
    Conversion to OpenCV format;
    Conversion from RGB image to HSV format;
    Image threshold;
    Application of morphological operations;
    Computation of moments;
    Computation of 3D coordinate of the detected object;
end

Algorithm 5.5: Color ball detection in ROS

The vision node is written in C++ language in ROS and subscribes to two topics from depth sensor through OpenNI: `/camera/rgb/image_color` and `/camera/depth/points`. The topic `/camera/rgb/image_color` publishes the RGB image in ROS format that is later converted using CvBridge to OpenCV format inside the vision node. The ball is detected in this image, from which can be calculated the centroid using moments. The node subscribes the topic `/camera/depth/points` and gets the 3D coordinates from the depth camera points. The estimation uses the 3D ball point obtained, that it is also published for visualization in RViz.
The point cloud image of the laboratory used during this work is in Figure 5.6. This is saved directly from RViz, the ROS visualization tool that subscribe to point cloud data seen by Kinect.

Mixture of color and depth information

The third idea is not implemented in this work but it is by theory [33] the best one. In this solution, color information to detect blobs of the ball color is mixed with the depth information to filter the image in order to remove objects that are found but that are not the real ball thrown.
After performing a color segmentation on the input image, a filter is applied based on depth information in order to remove the color classification of objects that are not balls.
In [33], it is proved by experimental results that it is possible to have a fast detection of aerial objects in clustered environments by merging color and depth information. This algorithm can be used successfully in real-time applications.

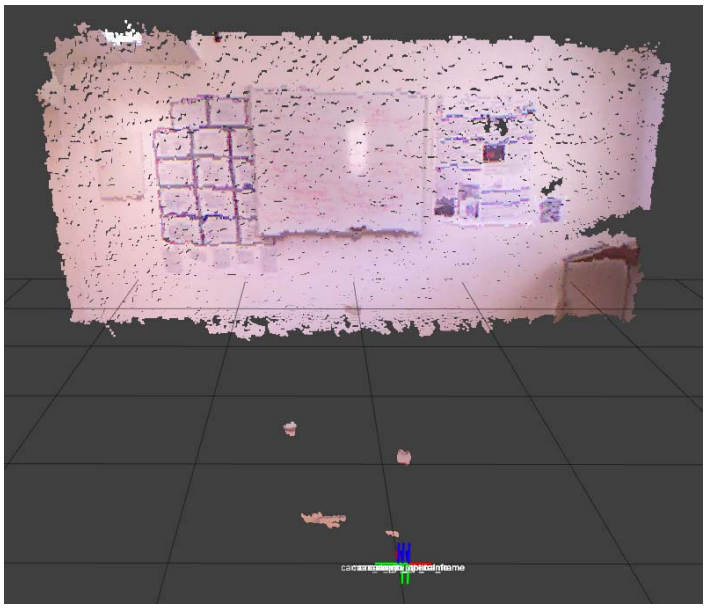


Figure 5.6: Point Cloud of the laboratory

5.1.2 Hand Detection

The hand detection is considered in this work to anticipate the ball tracking using the movement of the person who throw the ball. Better, the movement of the hand where the ball is almost the same of the ball itself and that information could be very useful to understand the resulting trajectory of the ball and start earlier the prediction to obtain faster the point where the robot has to catch the ball.
The procedure starts from the whole body detection using the openni_tracker package. In Figure 5.7 it is shown the detection of a body while it is moving in the environment of the laboratory.
OpenNI tracker can detect 15 body joints such as:

- neck joint
- head joint
- torso joint
- left and right shoulder joints
- left and right elbow joints
- left and right hand joints
- left and right hip joints
- left and right knee joints
- left and right foot joints

and can return the (x, y, z) position of each joint.
The aim of body's tracking is to receive information about the position of the hand while the person is throwing the ball with the objective to anticipate the motion of the ball before it enter in ballistic motion gaining some time. With this, it is possible to receive information about the starting velocity and the plane where the ball will

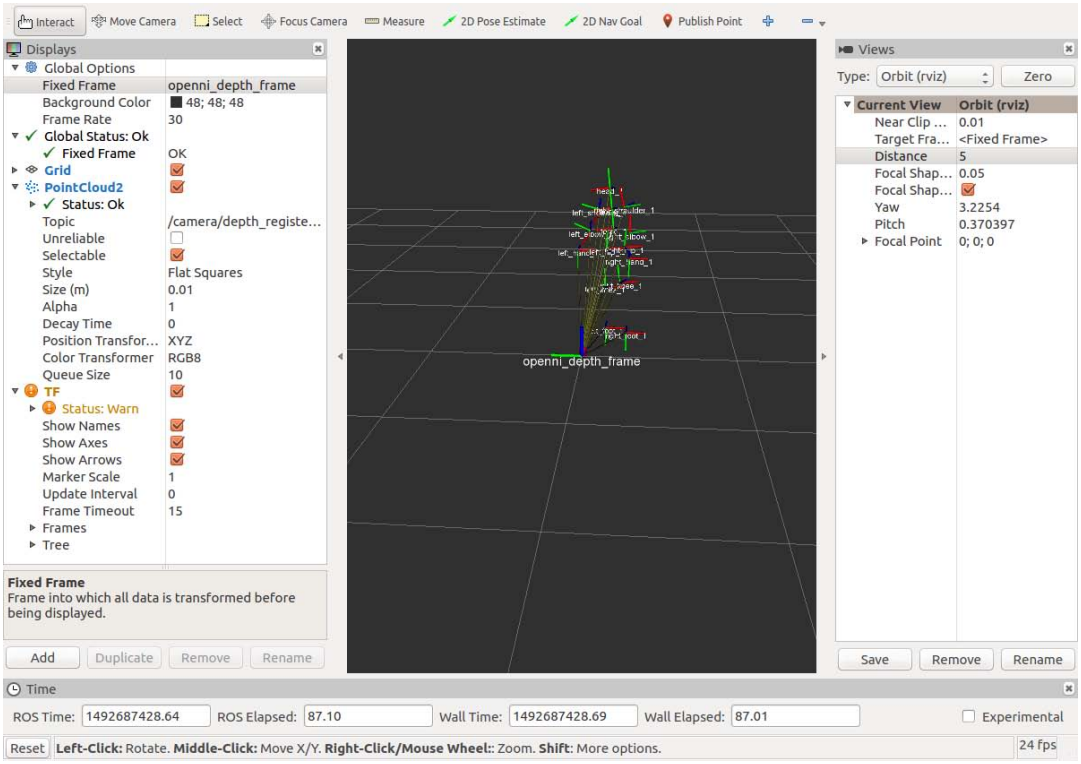


Figure 5.7: RViz screen with openni_tracker

pass. Then, the procedure can publish the direction where the robot has to move and then it can correct this information with the predictive or the prospective control. To use body detection in ROS, the setup needs to have:

- a ROS master running (roscore);
- the `openni_tracker` node running, which will be publishing skeleton positions acquired by the Kinect on a specific topic;
- a subscriber that will connect to the topic to get the skeleton positions and make them accessible.

The following command launches the `openni_tracker` node:

```
rosrun openni_tracker openni_tracker
```

and if everything is OK, nothing should be outputted until someone is detected in front of the Kinect and it outputs something similar to:

```
[ INFO] [time]: New User 1
```

The user adopts the Psi pose, reported in Figure 5.8, to calibrate the tracker that computes skeleton data from the Kinect. With the transform listener (`tf::TransformListener listener;`) in ROS, it is possible to compute the transform from each joint frame to the `openni_depth frame`, considered as the reference frame. The Algorithm 5.9 resumes the procedure used. This tracking can bring a lot of little problem to adjust. The subject's wrist rotation cannot be taken in consideration and so if he/she does not have a perfect throw and launch the ball in the same direction of his hand, only bad and useless information are received. Another possibility is that the ball can hide

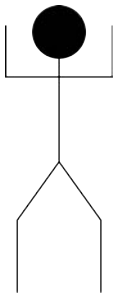


Figure 5.8: Psi Pose

```
for each joint do
    Get transform from joint frame to reference frame;
    Get pose of each joint;
    Publish pose;
end
```

Algorithm 5.9: Body detection

the hand and information can be lost.
The other problem to take care is the difficult to switch from the hand detection to the ball detection instantly. The use of two laptops and two Kinect sensors can solve this problem in future works.

5.2 ROS-Based Software Architecture

This section shows the software architecture in ROS, with different type of files, from packages to messages and nodes to complete a simulation of the robot.

5.2.1 Packages

The packages are stored in the `src` folder of the `catkin_ws`, the workspace used by ROS. The workspace contains two different packages for the two approaches. The packages released in ROS allow an easily robot's implementation.

Roslaunch files

Each package contains multiple launch files. The command `roslaunch` starts nodes together as defined in a launch file, allowing time saving. It includes options to automatically re-spawn processes that have already died and to set parameters. Roslaunch files are as reusable as possible. In this case, moving from the robot to a simulator can be done with only a few changes or without changing the launch files at all.

Cyton Gamma 1500 Description

The package `cyton_gamma_1500_description` contains RViz configuration files (with joint names and limits), meshes of each joint robot in *.dae* format (Digital Asset Exchange (.dae), the filename extension used by COLLADA Collaborative Design Activity), *.urdf* (Unified Robot Description Format (URDF), which is an XML format for representing a robot model) file that connect all the joints. The filename *.dae* stands for Digital Asset Exchange, an extension used by COLLADA (Collaborative

Design Activity), while URDF stands for Unified Robot Description Format, which is an XML format for representing a robot model. XML (Extensible Markup Language) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.
The package includes also a *.xacro* (XML Macros), a XML macro language and the launch files to visualize the Cyton Gamma 1500 in RViz. Xacro allows the construction of shorter and more readable XML files by using macros that expand to larger XML expressions.
Figure 5.10 represents joints description from this package.
Figure 5.11 shows the model of Cyton Gamma 1500 in two singularity positions.

Transform tf

Package `tf2` is the second generation of the transform library, which lets the user keep track of multiple coordinate frames over time. The package maintains the relationship between coordinate frames in a tree structure buffered in time, and lets the user transform points or vectors between any two coordinate frames at any desired point in time.
With `tf` it is possible to create both listener and broadcaster. Listener is used in this thesis to compute skeleton data for a possible hand detection and tracking.
This package uses `geometry_msgs/TransformStamped` messages, that expresses a transform from coordinate frame header to the coordinate frame child, including the transform between two coordinate frames in free space.

Robot and joint state publisher

The `robot_state_publisher` is a tool that publish robot's relevant frames all to `tf`. In practice, it helps to broadcast the state of the robot to the tf transform library. The robot state publisher internally has a kinematic model of the robot; so given the joint positions of the robot, the robot state publisher can compute and broadcast the 3D pose of each link in the robot.
Two things are needed to run the robot state publisher as a node:

1. A urdf xml robot description loaded
2. A source that publishes the joint positions as a `sensor_msgs/JointState`

The `robot_state_publisher` is used in conjunction with the `joint_state_publisher` . This package contains a tool for setting and publishing joint state values for a given URDF and publishes `sensor_msgs/JointState` messages for a robot. It reads the `robot_description` parameter, finds all the non-fixed joints and publishes a JointState message with all those joints defined.

MoveIT!

Using the configuration wizard of MoveIT!, it is created a package to control robot and take advantage of its features. MoveIT! includes a tool, called IKfast, that creates a C++ file to have a very fast inverse kinematics algorithm based on the model of the robot, to solve it without taking care of the time. Although this, it is better to reduce the degrees of freedom from 7 to 3 with an inverse kinematics algorithm easy to implement and faster than the IKfast. Indeed, the joints that are ignored does not help with a faster movement for ball catching task. Instead, the control of more degrees of freedom slows down the action.

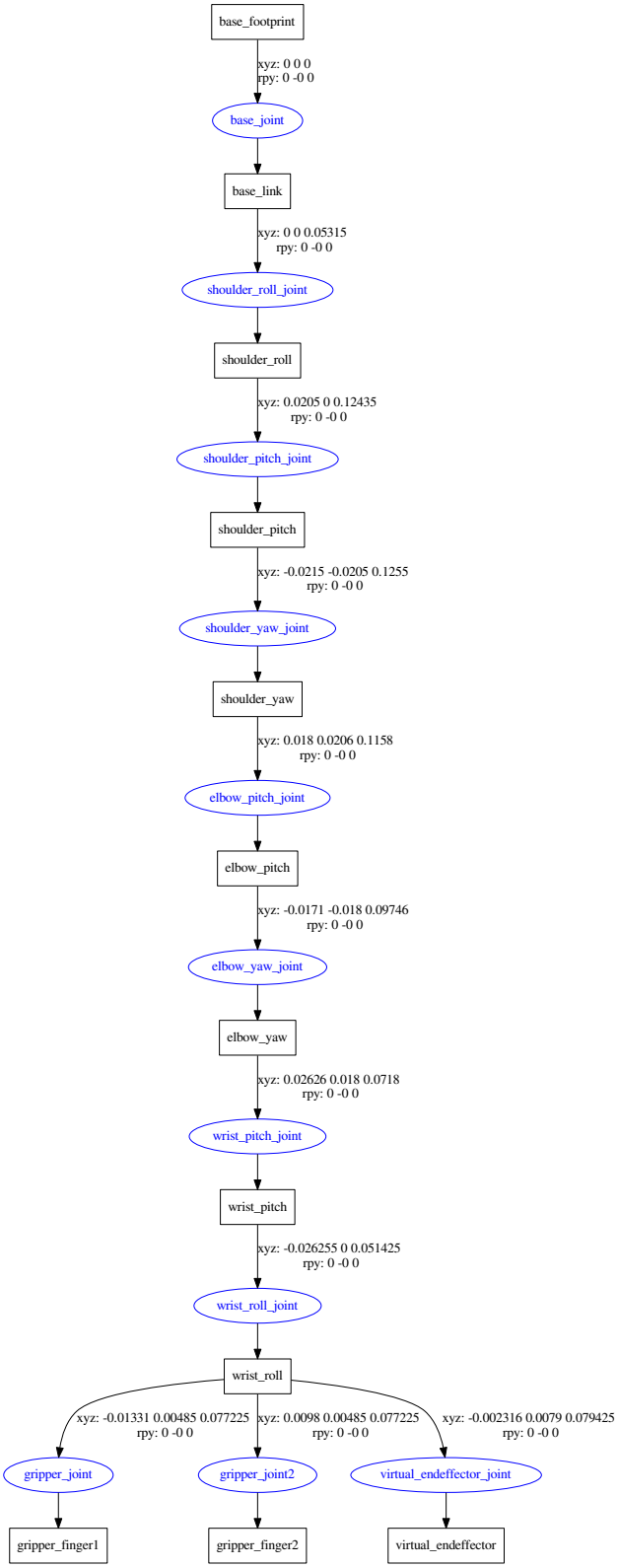
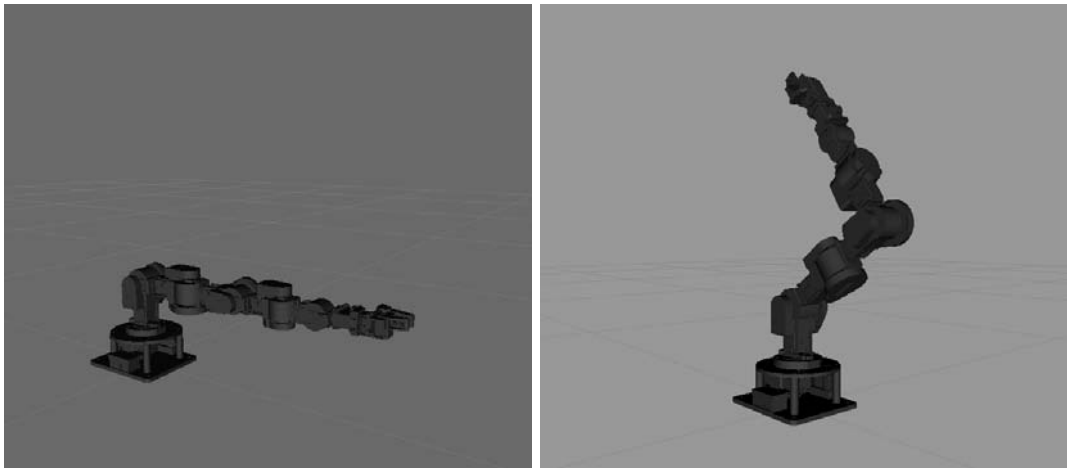


Figure 5.10: Cyton Gamma 1500 package joints description



(a) Cyton Gamma 1500 at an elbow singularity (b) Cyton Gamma 1500 at a shoulder singularity

Figure 5.11: Representation of singularities about Cyton Gamma 1500 in Rviz

5.2.2 Messages

Messages are an important part of the publisher and subscriber nodes creation. In this packages, the messages available to use are:

- `standard_msgs - Float32MultiArray`: carries the vector with the three final joint angle The multiarray declares a generic multi-dimensional array of a particular data type. Dimensions are ordered from outer most to inner most. It is composed of:
  `std_msgs/MultiArrayLayout layout`
  `float32[] data`

- `geometry_msgs`

  - `PoseArray`: carries the joint angles to perform the trajectory until the three final angles, it is an array of poses with a header for global reference. It is assembled of:
    * `std_msgs/Header header`
    * `geometry_msgs/Pose[] poses`
  - `PoseStamped`: carries the possible landing point, represent a Pose with reference coordinate frame and timestamp. It is made up of:
    * `std_msgs/Header header`
    * `geometry_msgs/Pose pose`
  - `Pose`: is a representation of pose in free space, composed of position and orientation, with Point for position and Quaternion for orientation.

- `sensor_msgs - JointState`: includes the joints position This is a message that holds data to describe the state of a set of torque controlled joints. The state of each joint (revolute or prismatic) is defined by:

  - the position of the joint (rad or m),
  - the velocity of the joint (rad/s or m/s) and
  - the effort that is applied in the joint (Nm or N).

Each joint is uniquely identified by its name. The header specifies the time at which the joint states were recorded. All the joint states in one message have to be recorded at the same time. This message consists of a multiple arrays, one for each part of the joint state. The goal is to make each of the fields optional. All arrays in this message should have the same size, or be empty. This is the only way to uniquely associate the joint name with the correct states. It is composed of:

  - `Header header`
  - `string[] name`
  - `float64[] position`
  - `float64[] velocity`
  - `float64[] effort`

- `visualization_msgs - Marker`: includes the different markers representing predicted and real ball and the landing point. It is made up of:

  - `Header header`
  - `string ns`
  - `int32 id`
  - `geometry_msgs/Pose pose`
  - `geometry_msgs/Vector3 scale`
  - `std_msgs/ColorRGBA color`
  - `duration lifetime`

ROS has builtin time and duration primitive types, described respectively by `ros::Time` and `ros::Duration` classes. A Time is a specific moment whereas a Duration is a period of time. The system time in ROS, using `ros::time::now()`, follows the Unix or POSIX time standard. POSIX time is defined as the time that has elapsed since 00:00:00 Coordinated Universal Time (UTC), 1 January 1970, not counting leap seconds. It stores the time past from the beginning. It also exists the `ros::Rate` class which makes a best effort at maintaining a particular rate for a loop (e.g. `ros::Rate(30)` means a rate of 30 Hz).

5.3 Simulation of the Ball Catching Task

The tool simulates in RViz the real environment that can be used in the laboratory. A sphere represents the ball. This simulation requires different nodes and topics to publish and subscribe. In this way, the future use of the real robot will be easier. Thus, it will replace only some nodes of the simulation with the ones of the real robot that have to subscribe and publish to the same topics. Practically, the only things to do is to plug the real robot and substitutes the simulation with it. The topic list published and subscribed is obtained typing in the terminal `rostopic list`. With the commands `rostopic info nameofthetopic` or `rostopic echo nameofthetopic` it is possible to obtain info about the topic and the things that are published for each topic, respectively.
Several nodes execute tasks that can be resumed in:

- Vision node
- Inverse Kinematics node
- Planning node

5.3.1 Vision Node

The Vision node regards the initial part of the task. It subscribes to ball information each time that a new ball point is available. It subscribes also to actual joint positions and computes the forward kinematics to compare the distance between the ball and the end-effector. It estimates the possible landing point with polyfit function, implemented by OpenCV: in the first implementation, it selects the target point where the ball will land considering the intersection between the predicted trajectory and the plane at a given height; in the second, it selects the point with minimum distance from the end-effector. It publishes the ball point as a sphere in the simulated environment, the predicted trajectory as an array of little spheres with a green sphere for the target point.

5.3.2 Inverse Kinematics Node

The second node computes the Inverse Kinematics algorithm. It subscribes the expected target point from the previous node and the actual joint positions. It computes the Jacobian considering as end-effector velocity the difference between the landing point and the end-effector position. If the target point is outside the workspace or if the joints angle computed are not real number, it does not compute any velocity. Finally, it publishes the joint velocities obtained with inverse Jacobian and end-effector velocity.

5.3.3 Planning Node

The Planning node concerns the update of robot position. It subscribes the joints velocity that the robot needs to have and updates the positions using Euler formula from the previous and actual velocity needed. It takes into account joint limits in velocity and position. To update the position it publishes all the joints angle of the robot.
In another version, the planning is a trapezoidal planning, as described in Section 5.3.3. This node takes care about the time of execution, the maximum velocity and acceleration that can be reach by each joint. The prospective strategy does not use

the trapezoid because it updates the velocity and therefore the position step by step. In Figure 5.12 and 5.13 are reported the two graph with all the nodes and topics active to perform the predictive and prospective strategies.

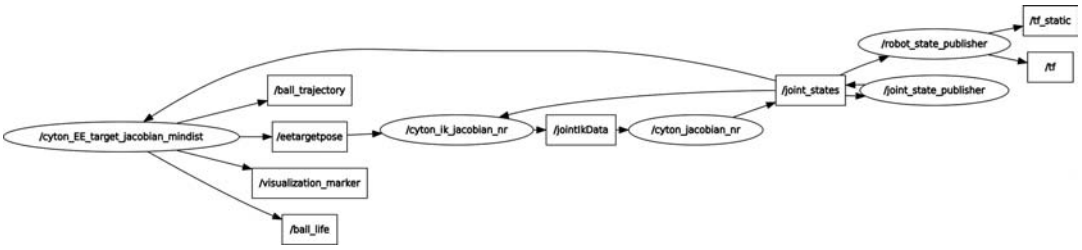


Figure 5.12: Nodes and topics of predictive strategy

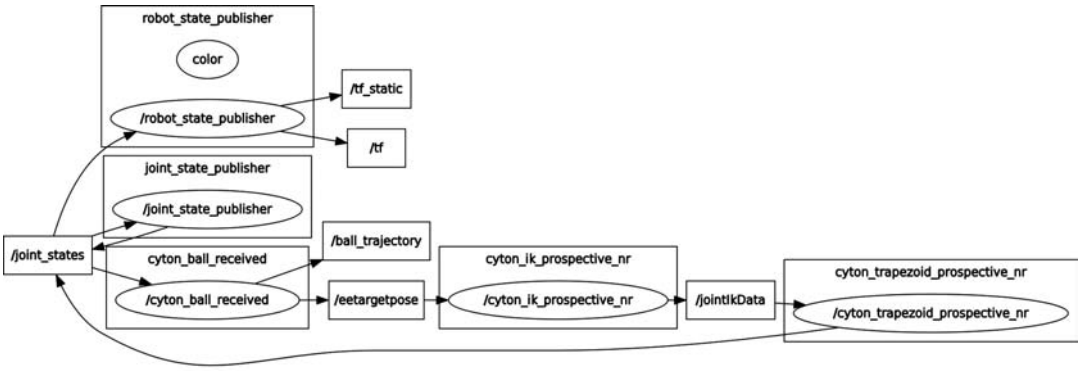


Figure 5.13: Nodes and topics of prospective strategy

Trajectory planning

At the beginning, the controller implementation was different to the one used in MATLAB®. The inverse kinematics algorithm does not use the Jacobian, controlling the robot directly in position, planning different types of trajectories.

Linear Trajectory Planning The linear trajectory planning is a simple linear interpolation between the initial and final joint position with a Δt time used for calculate the joint position at each iteration. It follows from:

$$s(t) = s_0 + (t - t_0)\left(\frac{s_1 - s_0}{t_1 - t_0}\right) \tag{5.11}$$

where s_1 is the final angle, s_0 is the initial angle, t_1 is the final time and t_0 the initial time. s is the actual angle during the trajectory at the actual time t. The latter formula (5.11) is rewritten to be implemented in C++ in the following way:

$$s(t) = a \cdot t + b \tag{5.12}$$

where:

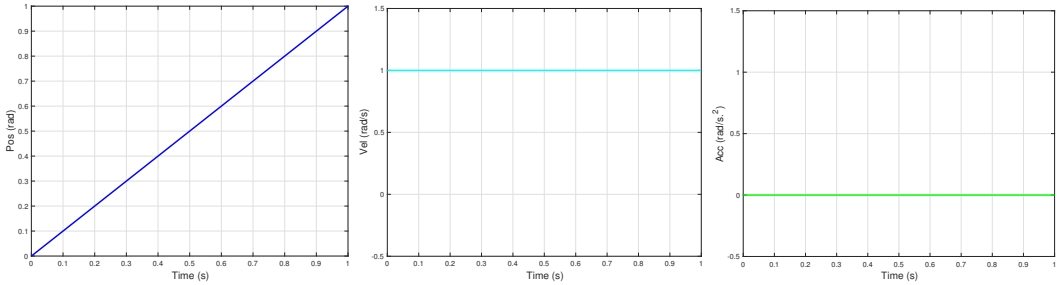
$$a = \frac{s_1 - s_0}{t_1 - t_0} \tag{5.13a}$$

$$b = -at_0 + s_0 \tag{5.13b}$$

Therefore, the velocity is constant:

$$\dot{s}(t) = a = \frac{s_1 - s_0}{t_1 - t_0} \tag{5.14}$$

and the acceleration is equal to zero ($\ddot{s}(t) = 0$). In the beginning phase of the thesis is chosen the linear interpolation to have easily a working system that, after, has been improved. Figure 5.14 represents the linear trajectory (Figure 5.14a), constant velocity (Figure 5.14b) and zero acceleration (Figure 5.14c).
These Figures and all the following about trajectory planning consider $s(0) = 0\,\text{rad}$ and $s(t) = 1\,\text{rad}$ with time $t_0 = 0\,\text{s}$ and $t_f = 1\,\text{s}$.



(a) Position trajectory (b) Velocity trajectory (c) Acceleration trajectory

Figure 5.14: Linear Interpolation Trajectory

The linear interpolation is a simple solution that does not take into account acceleration that it is always equal to zero and a linear velocity that it is not possible in practice because the robotic arm starts from a standing position. Thus, other solutions need to be found.

Fifth-Order Polynomial Trajectory Planning The Fifth-Order Polynomial Trajectory interpolates two points, the initial and the final:

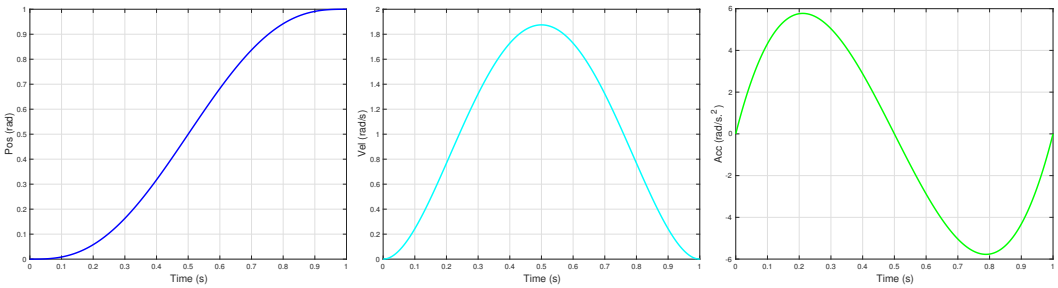
$$s(t) = At^5 + Bt^4 + Ct^3 + Dt^2 + Et + F \tag{5.15a}$$
$$\dot{s}(t) = 5At^4 + 4Bt^3 + 3Ct^2 + 2Dt + E \tag{5.15b}$$
$$\ddot{s}(t) = 20At^3 + 12Bt^2 + 6Ct + 2D \tag{5.15c}$$

with $t \in [0, T]$. It imposes the initial and final values to solve the linear system:

$$\begin{bmatrix} s(0) \\ s(T) \\ \dot{s}(0) \\ \dot{s}(T) \\ \ddot{s}(0) \\ \ddot{s}(T) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ T^5 & T^4 & T^3 & T^2 & T & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 5T^4 & 4T^3 & 3T^2 & 2T & 1 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 20T^3 & 12T^2 & 6T & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \\ E \\ F \end{bmatrix} \tag{5.16}$$

Once the parameters A, B, C, D, E and F are found computing the inverse, they can be substitute in (5.15) to obtain the complete trajectory. The problem of this trajectory is that it does not use the maximum velocity for a long time but it reaches only for a little time. Sometimes the velocity limits are below the maximum velocity point reached using this trajectory type and this is not applicable in real motion.
An example of Fifth-Order Polynomial Trajectory, including position, velocity and acceleration is in Figure 5.15. In this case, the choice of initial and final values in

matrix computation is: $s(0) = 0\,\text{rad}$, $s(T) = 1\,\text{rad}$, $\dot{s}(0) = \dot{s}(T) = 0\,\text{rad/s}$ and $\ddot{s}(0) = \ddot{s}(T) = 0\,\text{rad/s}^2$ (e.g. $[s(0)\ s(T)\ \dot{s}(0)\ \dot{s}(T)\ \ddot{s}(0)\ \ddot{s}(T)]^T = [0\ 1\ 0\ 0\ 0\ 0]^T$). Therefore, the parameters D, E and F are equal to zero (e.g. $D = E = F = 0$) while the other are different from zero.



(a) Position trajectory (b) Velocity trajectory (c) Acceleration trajectory

Figure 5.15: Fifth-Order Polynomial Trajectory

Trapezoidal Trajectory Planning The Trapezoidal Trajectory wants to improve both the linear and the fifth-order: starting from the first it is important to include acceleration and deceleration phases; the shape of the second takes into account these phases but needs to have maximum velocity for a longer time. Thus, trapezoidal name comes from the shape of the velocity using this trajectory planning that is a trapezoid. This profile splits the time in three intervals:

$$I_1 = \{t : t_0 \leq t < t_1\} = [t_0, t_a) \tag{5.17a}$$
$$I_2 = \{t : t_a \leq t \leq t_d\} = [t_a, t_d] \tag{5.17b}$$
$$I_3 = \{t : t_d < t \leq t_f\} = (t_d, t_f] \tag{5.17c}$$

The first phase with $t \in I_1$ (5.17a) and the third phase with $t \in I_3$ (5.17c) have same equations with different parameters, a and c respectively:

$$s(t) = a_2 \cdot t^2 + a_1 \cdot t + a_0 \tag{5.18a}$$
$$\dot{s}(t) = 2 \cdot a_2 \cdot t + a_1 \tag{5.18b}$$
$$\ddot{s}(t) = 2 \cdot a_2 \tag{5.18c}$$

The second phase with $t \in I_2$ (5.17b) is:

$$s(t) = b_1 \cdot t + b_0 \tag{5.19a}$$
$$\dot{s}(t) = b_1 \tag{5.19b}$$
$$\ddot{s}(t) = 0 \tag{5.19c}$$

The assumption is $\dot{s}(t_0) = 0 = a_1$, thus $s(t_0) = a_0 = q_0$, whereas $\dot{s}(t_f) = 0$, where t_f is the trajectory's final time. The first phase becomes:

$$s(t) = s(t_0) + \frac{\dot{s}_{max}}{2 \cdot t_a} \cdot t^2 = q_0 + \frac{\alpha}{2} \cdot t^2 \tag{5.20a}$$
$$\dot{s}(t) = \frac{\dot{s}_{max}}{t_a} \cdot t = \alpha \cdot t \tag{5.20b}$$
$$\ddot{s}(t) = \frac{\dot{s}_{max}}{t_a} = \alpha \tag{5.20c}$$

At time t_a the trajectory needs to be continue, therefore:

$$s(t_0) + \frac{\dot{s}_{max}}{2 \cdot t_a} \cdot t_a^2 = b_0 + \dot{s}_{max} \cdot t_a \Rightarrow b_0 = s(t_0) - \frac{\dot{s}_{max}}{2} \cdot t_a \tag{5.21}$$

Thus, the second phase, with $b_1 = \dot{s}_{max}$ changes into:

$$s(t) = s(t_0) - \frac{\dot{s}_{max}}{2} \cdot t_a + \dot{s}_{max} \cdot t \tag{5.22a}$$

$$\dot{s}(t) = \dot{s}_{max} \tag{5.22b}$$

$$\ddot{s}(t) = 0 \tag{5.22c}$$

The third phase is opposite to the first one and turns into:

$$s(t) = s(t_f) - \frac{\dot{s}_{max}}{2t_a}(t_f - t)^2 = q_F - \frac{\alpha}{2}(t_f - t)^2 \tag{5.23a}$$

$$\dot{s}(t) = \frac{\dot{s}_{max}}{t_a}(t_f - t) = \alpha(t_f - t) \tag{5.23b}$$

$$\ddot{s}(t) = -\frac{\dot{s}_{max}}{t_a} = -\alpha \tag{5.23c}$$

The second phase (5.22) has by symmetry:

$$q\left(\frac{t_f}{2}\right) = \frac{q_0 + q_F}{2} = b_0 + V \cdot \frac{t_f}{2} \Rightarrow b_0 = \frac{q_0 + q_F - V \cdot t_f}{2} \tag{5.24}$$

where $V = \dot{s}_{max}$. From (5.21) and (5.24), it is possible to obtain t_a and t_f:

$$t_a = \frac{q_0 - q_F + V \cdot t_f}{V} \tag{5.25}$$

$$t_f = \frac{q_F - q_0 + V \cdot t_a}{V} \tag{5.26}$$

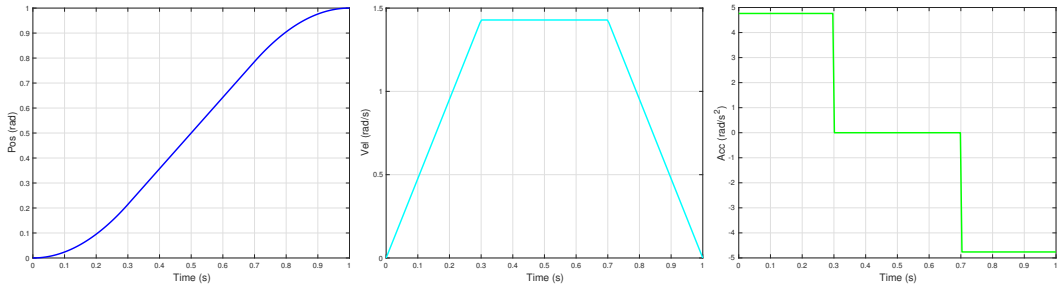
It is known that $0 < t_a \leq \frac{t_f}{2}$ and, using (5.26), it holds if:

$$\frac{q_F - q_0}{t_f} < V \leq 2 \cdot \frac{q_F - q_0}{t_f} \tag{5.27}$$

The final trajectory representing only the position angle in three phases is:

$$s(t) = q(t) = \begin{cases} q_0 + \frac{V}{2 \cdot t_a} \cdot t^2 & t \in I_1 \\ q_F - \frac{V}{2 \cdot t_a} \cdot (t_f - t)^2 & t \in I_3 \end{cases} \tag{5.28}$$

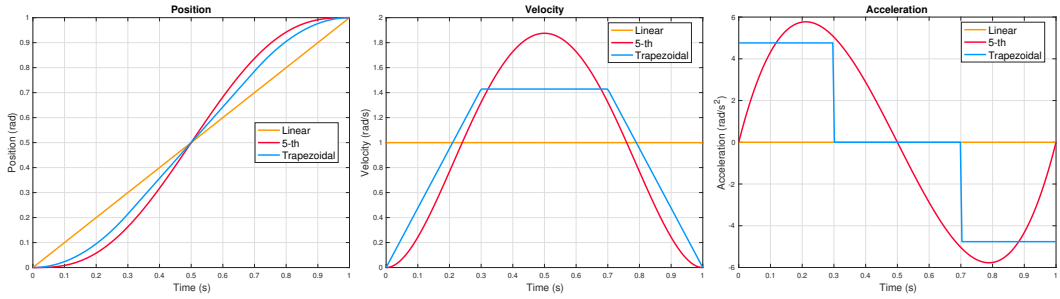
The profiles of position, velocity and acceleration are in Figure 5.16.
It is possible to distinguish the three phases: Figure 5.16c depict the acceleration with constant values in all the phases, in the second is equal to zero while in first and third has opposite values; Figure 5.16b shows the velocity that is at maximum value for the majority of the time with a linear trend in first and third, due to constant acceleration; in Figure 5.16a, first and third phases follow a quadratic trend, while the second has linear trend. These figures confirm the movement using trapezoidal motion.
The parameters chosen in the example figures are: $q_0 = 0\,\text{rad}$, $q_f = 1\,\text{rad}$ and $t_a = 0.3\,\text{s}$. From these three parameters all the other can be derived easily.
Finally, Figure 5.17 compares position, velocity and acceleration for three trajectories:



(a) Position trajectory (b) Velocity trajectory (c) Acceleration trajectory

Figure 5.16: Trapezoidal Trajectory

position (Figure 5.17a), velocity (Figure 5.17b) that shows trapezoidal planning with its trend linear-constant-linear and acceleration (Figure 5.17c) shapes. All the trajectories use the same values of initial and final position angle, velocity and acceleration but the shape of the movement is different for each.



(a) Positions (b) Velocities (c) Accelerations

Figure 5.17: Linear interpolation, 5-th order and Trapezoidal Trajectories

Using planning node with trajectory planning, the choice implemented in ROS is the trapezoidal motion planning. It is one of the easier to implement in C++ code and to compute during a real-time task. Using this trajectory, it is easy to take into account joint velocities and accelerations limits. It is possible to see that using trapezoidal planning, the shape is almost equal to the fifth-order but includes limitation that are fundamental for the real execution of the task.

5.4 Performance Evaluation

This chapter shows that the simulator created is a good alternative to the real robot to test the algorithm. The cost of a PC as simulator environment is clearly less than a real robot, but, moreover, it is also easier to understand the behavior of the robot without report big damages to the robot itself.
The environment created using ROS and C++ is easy to use for the user and with future works it can implement other functions. At this time, the simulation of the ball trajectory and the robot movement to catch it is the ideal work that wanted to be done in this work, due to the impossibility of repairing the real robot, damaged some weeks before the beginning of this thesis, that could not be repaired before the end of the dissertation.
The time to catch and the kinematics of the robot are perfectly reproduced. The results using Kinect data are quite different from the ones obtained in simulation

using MATLAB® where the data used was obtained mathematically with a gaussian noise added artificially to the data. Also, the polyfit function in C++ is implemented in a different way from the function available in MATLAB®.
Although these little differences, the result is achieved. The simulation works properly and all the nodes are perfectly set with ROS messages and ROS topics that can be published and subscribed either from the real and the simulated robot, with an easily swap of some lines of code.

Chapter 6

Conclusion

The latest chapter of the thesis regards the conclusion about the work done with the summary of the results and the possibles works that can be done in the future.

6.1 Result

The starting aim of the thesis was the optimization of the trade-off between the time allocated to perception and action, that it is required by ball catching task for successful completion and to maximize task performance.
The frame rate of the Kinect sensor was certainly a constraint that could cause an unsuccessful performance. Therefore, the evaluation in simulation using different frame-rate was necessary to be done. The other point to act is the time used for the prediction of the point where the ball will be caught. The choice of moving the arm, as soon as a point in the manipulability area is available, is the right direction but the results obtained do not satisfy at all with the frame-rate of the Kinect. The simulation in MATLAB® includes the error of the camera recognizing the ball. It is obvious that it is needed a camera with a frame-rate higher than 30 fps and an error lower than 0.5 m to accomplish perfectly the task.
This dissertation explored approaches by which the robot can anticipate the intentions of the human partner and/or learn how and when to act.
Anticipation refers to taking prior actions on the basis of information about their effects, including during the period in which he/she is preparing to throw the ball towards the robot. The use of skeleton data was taken into account to include the anticipation but this information was not useful as expected. Indeed, the person who throws the ball needs to respect a behavior that is not normal, and with the skeleton tracking it is almost impossible to track the rotation of the wrist of the person that, at the moment when the ball leaves the hand, could extremely change the trajectory of the ball. The aim is to find a solution that is more general as possible and the skeleton data added problems instead of simplifying the problem.
The how and when to act was developed with the evaluation of two different strategies: the Predictive Control and the Prospective Control, an essential task to evaluate the overall system's performance with different approaches. In this task, the knowledge obtained from past experiences [31] [45], including that obtained from failed trials, was useful. The Predictive Control was improved changing the selection of the landing point, using both plane intersection and closest point to the end-effector, while the Prospective strategy was a totally new area to evaluate.
The Predictive control was confirmed as one of the best method to solve this task. With a good prediction, that can be done using a simple least squares method (linear and quadratic regression), the point where the ball will be is predicted easily. Then it remains to move the robot in time to that point.
The Prospective control was a new area to explore. It considers an online approach

with time to contact, trying to reduce the error between the ball and the end-effector. The tuning of the parameters α and β is certainly the big difficult of this control. It is almost impossible to find two parameters that are suitable for all different throws. This control works very well in only one dimension and very bad in 3D. The best idea is to use a hybrid system, both in 2D moving in the first part of the throw the robot in only 2 axes and then at the end correct it with the third axis, both mixing prospective with predictive control.
Another aim of this work was to create tools to simulate the ball catching task and try new possibles strategies in a simulated environment before applying it to the real, avoiding possible error in effort that can reduce the potential of the real robotic arm. This was necessary because of a failure of the real robot that happened just before the start of this master thesis. The evaluation was done in MATLAB®, the easiest way to study algorithms, and after the evaluation, the solutions were implemented in C++.
Therefore, the second main objective was the implementation of a simulated arm system in ROS, based on a real model, to have all the topics and nodes ready to be used with the real physical arm when it will be available (Cyton Gamma 1500 or another industrial manipulator). In this way, topics can be easily swap between simulated and real robot. This was a big change done during the work, due to problems with the real Cyton Gamma 1500 available in the laboratory in University of Aveiro, that cannot be fixed before the deadline of Erasmus+ Programme.
This constraint leaves the possibility, as future works, to confirm the study made in this thesis, using the real robot and not only the simulated one. Other robots were considered but, because of time, there was no possibility to move to a new one that requires some adjustments in topics but especially new study about the model and its kinematics.

6.2 Future Work

First of all, the future work will be to implement all the effort done in this dissertation with MATLAB® and ROS with C++ in a real robot and confirm, in a real environment, what was obtained in the simulation.
According to the previous results, future works includes the idea to replace the current Kinect sensor and the Cyton manipulator arm by a new camera and an industrial manipulator, because the time and problems with real manipulator available in the university laboratory did not permit the change. The Kinect sensor must be replaced with a new sensor with a higher frame-rate or by a stereo camera. Less error and higher frame-rate allow a better and faster estimation that is fundamental to accomplish the task in less than a one-second throw.
A faster manipulator is the other requirement, also if it has fewer degrees of freedom, that in this thesis puts only bigger errors in kinematics than improvements in velocity. A higher joint velocity is required, considering that the robot has to move in almost a one-meter cube in less than one second.
With these improvements, it is possible to apply the predictive strategy analyzed in this thesis that will allow the best performance in a real environment.

Appendix A

Mathematical Derivations

The Appendix A regards the mathematical derivations computed during the work.

A.1 Denavit-Hartenberg convention

The Denavit-Hartenberg convention is presented to compute the robot's kinematics [48]. It considers an open-chain manipulator constituted by $n+1$ links connected by n joints, where Link 0 is conventionally fixed to the ground. It is assumed that each joint provides the mechanical structure with a single DOF, corresponding to the joint variable.
Using the typical open kinematic chain of the manipulator structure, it derives the construction of an operating procedure for the computation of direct kinematics. In fact, since each joint connects two consecutive links, it considers the description of kinematic relationship between consecutive links and then obtain the overall description of manipulator kinematics in a recursive fashion. To this purpose, it is worth defining a coordinate frame attached to each link, from Link 0 to Link n. Then, the coordinate transformation describing the position and orientation of Frame n with respect to Frame 0 (Figure A.1) is given by:

$$T_n^0(q) = A_1^0(q_1)A_2^1(q_2)\dots A_n^{n-1}(q_n) \tag{A.1}$$

The recursive computation of the direct kinematics function is obtained in a system-

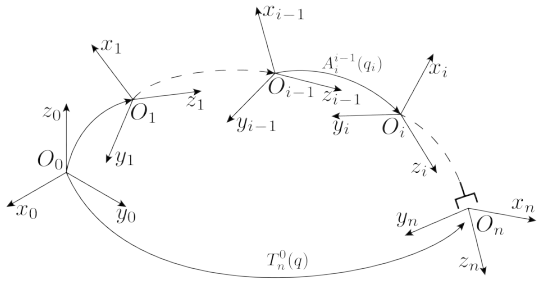


Figure A.1: Coordinate transformations in an open kinematic chain

atic way by simple products of the homogeneous transformation matrices $A_i^{i-1}(q_i)$(for $i = 1,\dots,n$), each of which is a function of a single joint variable. The actual coordinate transformation, describing the position and orientation of the end-effector frame with respect to the base frame, can be obtained as:

$$T_e^b(q) = T_0^b T_n^0(q) T_e^n \tag{A.2}$$

where T_0^b and T_e^n are two constant homogeneous transformations describing the position and orientation of Frame 0 with respect to the base frame, and of the end-effector frame with respect to Frame n, respectively.
In order to compute the direct kinematics equation for an open-chain manipulator according to the recursive expression (A.1), a general method is to be derived to define the relative position and orientation of two consecutive links. The problem is that to determine two frames attached to the two links and compute the coordinate transformations between them. It is convenient to set some rules for the definition of the link frames.

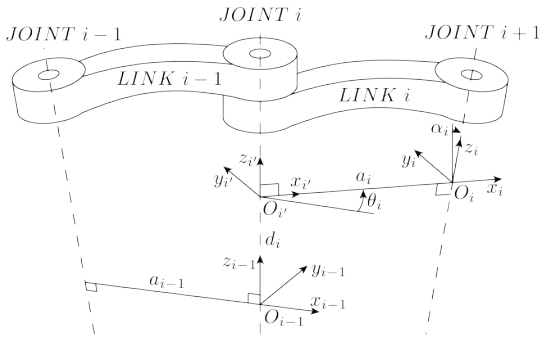


Figure A.2: Denavit-Hartenberg kinematic parameters

With reference to Figure A.2, let Axis i denote the axis of the joint connecting Link $i-1$ to Link i; the so-called Denavit-Hartenberg convention (DH) is adopted to define link Frame i:

- Choose axis z_i along the axis of Joint $i+1$;
- Locate the origin O_i at the intersection of axis z_i with the common normal to axes z_{i-1} and z_i Also, locate $O_{i'}$, at the intersection of the common normal with axis z_{i-1};
- Choose axis x_i along the common normal to axes z_{i-1} and z_i with direction from Joint i to Joint $i+1$;
- Choose axis y_i so as to complete a right-handed frame.

The Denavit-Hartenberg convention gives a non-unique definition of the link frame in the following cases:

- For Frame 0, only the direction of axis z_0 is specified; then O_0 and x_0 can be arbitrarily chosen;
- For Frame n, since there is no Joint $n+1$, z_n is not uniquely defined while x_n has to be normal to axis z_{n-1}. Typically, Joint n is revolute, and thus z_n is to be aligned with the direction of z_{n-1};
- When two consecutive axes are parallel, the common normal between them is not uniquely defined;
- When two consecutive axes intersect, the direction of x_i is arbitrary;
- When Joint i is prismatic, the direction of z_{i-1} is arbitrary.

In all such cases, the procedure can be simplified making parallel the axis of consecutive frames.
Once the link frames have been established, the position and orientation of Frame i with respect to Frame $i-1$ are completely specified by the following parameters:

- a_i: distance between O_i and $O_{i'}$;
- d_i: coordinate of $O_{i'}$ along z_{i-1};
- α_i: angle between axes z_{i-1} and z_i about axis x_i to be taken positive when rotation is made counter-clockwise;
- θ_i: angle between axes x_{i-1} and x_i about axis z_{i-1} to be taken positive when rotation is made counter-clockwise.

Two of the four parameters (a_i and α_i) are always constant and depend only on the geometry of connection between consecutive joints established by Link i. Of the remaining two parameters, only one is variable depending on the type of joint that connects Link $i-1$ to Link i. In particular:

- if Joint i is revolute the variable is θ_i;
- if Joint i is prismatic the variable is d_i.

At this point, it is possible to express the coordinate transformation between Frame i and Frame $i-1$ according to the following steps:

- Choose a frame aligned with Frame $i-1$;
- Translate the chosen frame by d_i along axis z_{i-1} and rotate it by θ_i about axis z_{i-1}; this sequence aligns the current frame with Frame i' and is described by the homogeneous transformation matrix:

$$A_{i'}^{i-1} = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i} & 0 & 0 \\ s_{\theta_i} & -c_{\theta_i} & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.3}$$

  ;
- Translate the frame aligned with Frame i' by a_i along axis $x_{i'}$ and rotate it by α_i about axis $x_{i'}$; this sequence aligns the current frame with Frame i and is described by the homogeneous transformation matrix:

$$A_i^{i'} = \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & c_{\alpha_i} & -s_{\alpha_i} & 0 \\ 0 & s_{\alpha_i} & c_{\alpha_i} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.4}$$

  ;
- The resulting coordinate transformation is obtained by postmultiplication of the single transformations as:

$$A_i^{i-1}(q_i) = A_{i'}^{i-1} A_i^{i'} = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i} c_{\alpha_i} & s_{\theta_i} s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i} c_{\alpha_i} & -c_{\theta_i} s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.5}$$

  .

Notice that the transformation matrix from Frame i to Frame $i-1$ is a function only of the joint variable q_i, that is, θ_i for a revolute joint or d_i for a prismatic joint. To summarize, the Denavit-Hartenberg convention allows the construction of the direct kinematics function by composition of the individual coordinate transformations expressed by (A.5) into one homogeneous transformation matrix as in (A.1).
The procedure operates in the following form:

1. Find and number consecutively the joint axes; set the directions of axes $z_0, \ldots, z_{n-1}$;
2. Choose Frame 0 by locating the origin on axis z_0; axes x_0 and y_0 are chosen so as to obtain a right-handed frame. If feasible, it is worth choosing Frame 0 to coincide with the base frame;
3. Locate the origin O_i at the intersection of z_i with the common normal to axes z_{i-1} and z_i. If axes z_{i-1} and z_i are parallel and Joint i is revolute, then locate a_i so that $d_i = 0$; if Joint i is prismatic, locate O_i at a reference position for the joint range, e.g., a mechanical limit. Compute this step for $i = 1, \ldots, n-1$;
4. Choose axis x_i along the common normal to axes z_{i-1} and z_i with direction from Joint i to Joint $i+1$. Compute this step for $i = 1, \ldots, n-1$;
5. Choose axis y_i so as to obtain a right-handed frame. Compute this step for $i = 1, \ldots, n-1$;
6. Choose Frame n; if Joint n is revolute, then align z_n with z_{n-1}, otherwise, if Joint n is prismatic, then choose z_n arbitrarily. Axis x_n is set according to step 4;
7. Form the table of parameters a_i, α_i, d_i, θ_i for $i = 1, \ldots, n$;
8. On the basis of the parameters in 7, compute the homogeneous transformation matrices $A_i^{i-1}(q_i)$ for $i = 1, \ldots, n$;
9. Compute the homogeneous transformation $T_n^0(q) = A_1^0 \ldots A_n^{n-1}$ that yields the position and orientation of Frame n with respect to Frame 0;
10. Given T_0^b and T_e^n, compute the direct kinematics function as $T_e^b(q) = T_0^b T_n^0 T_e^n$ that yields the position and orientation of the end-effector frame with respect to the base frame.

A.2 Regression

A.2.1 Simple Linear Regression

A simple linear regression considers the relationship between variables y, called response, and x, called predictor. The linear model is:

$$y = \beta_1 x + \beta_0 + \epsilon \tag{A.6}$$

where ϵ is a random disturbance or error, β_1 and β_0 are the parameters, also known as the intercept (the predicted value of y when $x = 0$) and the slope of the line (the change in y for unit change in x), respectively.
From (A.6), each observation can be written as

$$y_i = \beta_1 x_i + \beta_0 + \epsilon_i, \quad i = 1, 2, \ldots, n \tag{A.7}$$

where i represents the i-th value of the response, the predictor and the error variables. The estimation of the parameters is equivalent to finding the straight line that gives the best fit of the points. This is done using least squares method, which gives the line that minimizes the sum of squares of the vertical distances from each point to the line. The vertical distances represent the errors in the response variable. These errors can be obtained by (A.7) as:

$$\epsilon_i = y_i - \beta_1 x_i - \beta_0, \quad i = 1, 2, \ldots, n \tag{A.8}$$

The sum of squares of these distances is:

$$S(\beta_0, \beta_1) = \sum_{i=1}^{n} \epsilon_i^2 = \sum_{i=1}^{n} (y_i - \beta_1 x_i - \beta_0)^2 \tag{A.9}$$

The condition for $S(\beta_0, \beta_1)$ to be a minimum is that:

$$\frac{\delta S(\beta_0, \beta_1)}{\delta \beta_i} = 0, \quad i = 0, 1 \tag{A.10}$$

that in this case become for β_0 and β_1:

$$\frac{\delta S(\beta_0, \beta_1)}{\delta \beta_0} = -2 \sum_{i=1}^{n} (y_i - \beta_1 x_i - \beta_0) = 0 \tag{A.11}$$

$$\frac{\delta S(\beta_0, \beta_1)}{\delta \beta_1} = -2 \sum_{i=1}^{n} (y_i - \beta_1 x_i - \beta_0) x_i = 0 \tag{A.12}$$

These lead to the equations:

$$n\beta_0 + \beta_1 \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i \tag{A.13}$$

$$\beta_0 \sum_{i=1}^{n} x_i + \beta_1 \sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} x_i y_i \tag{A.14}$$

Remember that:

$$\bar{y} = \frac{\sum_{i=1}^{n} y_i}{n}, \qquad \bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} \tag{A.15}$$

is the way to compute the mean of each variable. The values of $\hat{\beta}_0$ and $\hat{\beta}_1$ that minimize $S(\beta_0, \beta_1)$ are given by:

$$\hat{\beta}_1 = \frac{\sum (y_i - \bar{y})(x_i - \bar{x})}{\sum (x_i - \bar{x})^2} \tag{A.16}$$

and:

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x} \tag{A.17}$$

The estimates $\hat{\beta}_0$ and $\hat{\beta}_1$ are called the least squares estimates of β_0 and β_1 because they are the solution to the least squares method. The least squares regression line is given by:

$$\hat{y} = \hat{\beta}_1 x + \hat{\beta}_0 \tag{A.18}$$

The fitted regression equation can be used for prediction of the value of the response variable y which corresponds to any chosen value, x_0, of the predictor variable. The

predicted value $\hat{y}_0$ is:

$$\hat{y}_0 = \hat{\beta}_1 x_0 + \hat{\beta}_0 \tag{A.19}$$

A.2.2 Quadratic Regression

In quadratic regression, the data consist of n observations on a dependent or response variable y and a predictor variable x. The relationship is a quadratic model:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \epsilon \tag{A.20}$$

where $\beta_0, \beta_1, \beta_2$ are constants referred to regression coefficients and ϵ is a random disturbance of error.
The model is linear from the point of view of estimation, since the regression function is linear in terms of the unknown parameters. The technique for solving the problem is the same of multiple linear regression, treating x and x^2 as distinct independent variables.
It is assumed that for any set of fixed values of x, x^2, that fall within the range of the data, the linear equation (A.20) provides an acceptable approximation of the true relationship between y and the two x. From that, equation (A.20), for each observation, can be rewritten as:

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \epsilon_i, \quad i = 1, 2, \ldots, n \tag{A.21}$$

where i represents the i-th value of the variables and the error. It is an extension of the simple linear regression of Section A.2.1.
The least squares method minimize the sum of squares of the errors, rewritten as:

$$\epsilon_i = y_i - \beta_0 - \beta_1 x_1 - \beta_2 x_i^2, \quad i = 1, 2, \ldots, n \tag{A.22}$$

The sum of squares of these errors is:

$$S(\beta_0, \beta_1, \beta_2) = \sum_{i=1}^{n} \epsilon_i^2 = \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i - \beta_2 x_i^2)^2 \tag{A.23}$$

The least squares estimates $\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2$ which minimize $S(\beta_0, \beta_1, \beta_2)$ are given by the solution of linear equations, that is assumed solvable and with unique solution. The solution will be discussed later in the section. Using the estimated coefficients, the least squares regression equation is:

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x + \hat{\beta}_2 x^2 \tag{A.24}$$

For each observation:

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i + \hat{\beta}_2 x_i^2, \quad i = 1, 2, \ldots, n \tag{A.25}$$

To solve it, it is needed to define the following matrices:

$$\mathbf{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \mathbf{X} = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 \end{bmatrix}, \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix}, \boldsymbol{\epsilon} = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}, \tag{A.26}$$

The linear model in (A.20) can be expressed in terms of the above matrices as:

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} \tag{A.27}$$

The assumptions made about $\boldsymbol{\epsilon}$ for least squares estimation are:

$$E(\boldsymbol{\epsilon}) = 0, \quad Var(\boldsymbol{\epsilon}) = E(\boldsymbol{\epsilon}\boldsymbol{\epsilon}^T) = \sigma^2 I_n \tag{A.28}$$

where $E(\boldsymbol{\epsilon})$ is the expected value (mean) of $\boldsymbol{\epsilon}$, $\mathbf{I}_n$ is the identity matrix of order n, and $\boldsymbol{\epsilon}^T$ is the transpose of $\boldsymbol{\epsilon}$. Accordingly, ϵ_i are independent and have zero mean and constant variance. This implies that:

$$E(\mathbf{Y}) = \mathbf{X}\boldsymbol{\beta} \tag{A.29}$$

The least squares estimator $\hat{\boldsymbol{\beta}}$ of $\boldsymbol{\beta}$ is obtained by minimizing the sum of squared deviations of the observations from their expected values. Hence, the least squares estimators are obtained by minimizing $S(\boldsymbol{\beta})$, where

$$S(\boldsymbol{\beta}) = \boldsymbol{\epsilon}^T\boldsymbol{\epsilon} = (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}) \tag{A.30}$$

Minimization of $S(\boldsymbol{\beta})$ leads to the system of equations:

$$(\mathbf{X}^T\mathbf{X})\hat{\boldsymbol{\beta}} = \mathbf{X}^T\mathbf{Y} \tag{A.31}$$

Assuming that $(\mathbf{X}^T\mathbf{X})$ has an inverse, the least squares estimates $\hat{\boldsymbol{\beta}}$ can be written explicitly as:

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y} \tag{A.32}$$

from which it can be seen that is a linear function of $\mathbf{Y}$. The vector of fitted values $\mathbf{Y}$ corresponding to the observed $\mathbf{Y}$ is:

$$\hat{\mathbf{Y}} = \mathbf{X}\hat{\boldsymbol{\beta}} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y} = \mathbf{P}\mathbf{Y} \tag{A.33}$$

where $\mathbf{P}$ is the hat or projection matrix. The vector of residuals is given by:

$$\mathbf{e} = \mathbf{Y} - \hat{\mathbf{Y}} = \mathbf{Y} - \mathbf{P}\mathbf{Y} = (\mathbf{I}_n - \mathbf{P})\mathbf{Y} \tag{A.34}$$

Appendix B

Tutorials

The Appendix reports all the tutorial to follow for installing ROS in Linux Ubuntu and setup the Kinect in a PC.

B.1 Installing ROS

To install ROS, it is needed a PC with Linux Ubuntu 16.04 LTS [25], the latest long-term support version available. In this study, Linux Ubuntu is installed through Parallels Desktop 12 for Mac.
To do that, the installation guide in the official site of ROS is followed [44], precisely in http://wiki.ros.org/kinetic/Installation/Ubuntu.

- **Setup sources list**: setup the computer to accept software from [39], packages.ros.org.

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu$(
    lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.
    list'
```

- **Setup keys**:

```
$ sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net
    :80--recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
```

- **Installation**: first, make sure that Debian package index is up-to-date:

```
$ sudo apt-get update
```

  Install the Desktop-Full install that includes: ROS, rqt, rviz, robot-generic libraries, 2D/3D simulators, navigation and 2D/3D perception.

```
$ sudo apt-get install ros-kinetic-desktop-full
```

  To find available packages, use:

```
$ apt-cache search ros-kinetic
```

- **Initialize rosdep**: before use ROS, it is needed to initialize rosdep that enables to easily install system dependencies for source that is wanted to compile and is required to run some core components in ROS.

```
$ sudo rosdep init
$ rosdep update
```

- **Environment setup**: it is convenient if the ROS environment variables are automatically added to the bash session every time a new shell is launched:

```
$ echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
$ source ~/.bashrc
```

and

```
$ echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
```

- **Getting rosinstall**: rosinstall is a frequently used command-line tool in ROS that is distributed separately. It enables to easily download many source trees for ROS packages with one command. To install this tool on Ubuntu, run:

```
$ sudo apt-get install python-rosinstall
```

ROS at this point is installed.
The following tutorial allows the user to understand how it works and create a working ROS workspace:

- **Create a catkin workspace**:

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/src
$ catkin_init_workspace}
```

- **Build a catkin workspace**: even though the workspace is empty (there are no packages in the `src` folder, but just a single `CMakeLists.txt` link) it can still build the workspace:

```
$ cd ~/catkin_ws/
$ catkin_make
```

The `catkin_make` command is a convenience tool for working with catkin workspaces. In current directory there should be a `build` and `devel` folder. Inside the `devel` folder there are now several `setup.*sh` files. Sourcing any of these files will overlay this workspace on top of the environment.

- **Source the setup file**: before continuing source the new `setup.*sh` file:

```
$ source devel/setup.bash
```

- **Control workspace**: to make sure the workspace is properly overlayed by the setup script, make sure `ROS_PACKAGE_PATH` environment variable includes the current directory.

```
$ echo $ROS_PACKAGE_PATH
/home/youruser/catkin_ws/src:/opt/ros/kinetic/share
```

The environment is now setup.

B.2 Installing Cyton packages for ROS

To install the Robai Cyton Gamma 1500, it has to be downloaded the git clone of Gert Kanter [8]. To use Gazebo, already installed, with ROS use:

```
$ sudo apt-get install ros-kinetic-gazebo-ros-control
```

To install MoveIt in Kinetic use:

```
$ sudo apt-get install ros-kinetic-moveit
$ source /opt/ros/kinetic/setup.bash
```

The last command install and setup the environment.

B.3 Install Kinect

To install the Kinect, the tutorial made by 20paper [20] has to be followed. Different packages are downloaded to use Kinect in Linux environment using `openni_launch` [16]. At the beginning, it is possible to have problems due to incompatibility.
The tutorial installs packages to use Microsoft Kinect in Linux Ubuntu using OpenNI 1.5.4 and NITE 1.5.2 [37]. This tutorial does not work with OpenNI 2. To talk to the Kinect, there are two basic parts: OpenNI itself, and a Sensor module that is actually responsible for communicating with the hardware. Then, if needed, there is NITE, which is another module for OpenNI that does skeletal tracking [50], gestures, and stuff.

- **Prerequisites**: the readme file included with OpenNI lists all the packages to install

  ```
  $ sudo apt-get install git build-essential python libusb-1.0-0-
      dev freeglut3-dev default-jdk doxygen
  ```

  There are also some optional packages

  ```
  $ sudo apt-get install graphviz mono-complete
  ```

- **OpenNI 1.5.4**: OpenNI is a framework for working with what they are calling natural interaction devices. Anyway, this is how it is installed:
  - **Check out from Git**: OpenNI is hosted on Github, so checking it out is simple:

    ```
    $ mkdir -p ~/Kinect
    $ cd Kinect
    $ git clone https://github.com/OpenNI/OpenNI.git
    ```

  - **Checkout Unstable 1.5.4**: the first thing to do is checkout the Unstable 1.5.4 tag. If it is not, then the SensorKinect library won't compile in next step. From there, change into the `Platform/Linux-x86/CreateRedist` directory, and run the RedistMaker script. Note that even though the directory is named x86, this same directory builds 64 bit versions just fine.

```
$ cd OpenNI
$ git checkout Unstable-1.5.4.0
$ cd Platform/Linux/CreateRedist
$ chmod +x RedistMaker
$ ./RedistMaker
```

– **Install**: the RedistMaker script will compile everything. Then change into the Redist directory and run the install script to install the software on the system.

```
$ cd ../Redist/OpenNI-Bin-Dev-Linux-[xxx]
(where [xxx] is architecture and this particular OpenNI
    release)
```

In this study, the command is:

```
$ cd ../Redist/OpenNI-Bin-Dev-Linux-x64-v1.5.4.0/
$ sudo ./install.sh
```

- **Kinect Sensor Module**: OpenNI does not actually provide anything for talking to the hardware, it is more just a framework for working with different sensors and devices. It is needed to install a Sensor module for actually doing the hardware interfacing. Think of an OpenNI sensor module as a device driver for the hardware. The sensor module needed is also on GitHub, but from a different user. Check out the code looking for the Kinect branch, not master.

```
$ git clone https://github.com/avin2/SensorKinect
$ cd SensorKinect
```

The install process for the sensor is pretty much the same as for OpenNI itself:

```
$ cd Platform/Linux/CreateRedist
$ chmod +x RedistMaker
$ ./RedistMaker
$ cd ../Redist/Sensor-Bin-Linux-[xxx] (where [xxx] is
    architecture and this particular OpenNI release)
```

In this thesis, the command is:

```
$ cd ../Redist/Sensor-Bin-Linux-x64-v5.1.2.1/
$ chmod +x install.sh
$ sudo ./install.sh
```

On Ubuntu, regular users are only given read permission to unknown USB devices. The install script puts in some udev rules to fix this, but if find that none of the samples work unless run them as root, try unplugging and plugging the Kinect back in again, to make the new rules apply.

- **Test the OpenNI Samples**: at this point, enough is installed to get data from the Kinect. The easiest way to verify this is to run one of the OpenNI samples.

```
$ cd OpenNI/Platform/Linux-x86/Bin/Release
```

In this dissertation, the command is:

```
$ cd Kinect/OpenNI/Platform/Linux/Bin/x64-Release/
$ ./Sample-NiSimpleViewer
```

There should be a yellow-black depth image. At this point, it is an option installing the higher level NITE module.

- **Install NITE 1.5 (optional)**: To obtain NITE 1.5.2, download NITE 1.5.2 for the platform at the link: http://www.openni.org/openni-sdk/openni-sdk-history-2/ [37] Extract the archive, and run the installer:

```
$ sudo ./install.sh
```

At some point, it asks for a license key that can be found just about anywhere on the Internet. Just copy that license into the console, including the equals sign at the end, and NITE will install just fine.

After following these steps, it is possible to write programs that use the Microsoft Kinect through OpenNI and NITE middleware.

B.3.1 Second Install Kinect

To install only `openni_camera`:

```
$ sudo apt-get install ros-kinetic-openni-camera
```

where *kinetic* is the ROS distribution. It is also recommended to install `openni_launch` :

```
$ sudo apt-get install ros-kinetic-openni-launch
```

After that there will be some errors like:

```
[ INFO] [1484047391.416383018]: Number devices connected: 2
[ INFO] [1484047391.522740482]: 1. device on bus 001:04 is a SensorKinect
    (2ae) from PrimeSense (45e) with serial id '0'
[ INFO] [1484047391.522801766]: 2. device on bus 001:04 is a SensorV2 (2ae
    ) from PrimeSense (45e) with serial id 'A00364A10508108A'
[ INFO] [1484047391.523781988]: Searching for device with index = 1
[camera/camera_nodelet_manager-2] process has died [pid 4522, exit code
    -11, cmd /opt/ros/kinetic/lib/nodelet/nodelet manager __name:=
    camera_nodelet_manager __log:=/home/ubuntunick/.ros/log/294eb2c6-d727
    -11e6-b4de-001c42e1662f/camera-camera_nodelet_manager-2.log].
log file: /home/ubuntunick/.ros/log/294eb2c6-d727-11e6-b4de-001c42e1662f/
    camera-camera_nodelet_manager-2*.log
```

So uninstall SensorKinect:

```
$ cd Kinect/SensorKinect/Platform/Linux/Redist/Sensor-Bin-Linux-x64-v5
    .1.2.1/
$ sudo ./install.sh -u
```

To use `openni_tracker`, downloaded from GitHub [38], install the NiTE v1.5.2.23, available in [36]. Installing OpenNI SDK on Linux Extract the tarball to a directory, go into this directory and run the install script:

```
$ ./install.sh
```

The installation creates udev rules which will allow usage of OpenNI-compliant USB devices without root privileges. And then follow the guide in [16].

B.3.2 Calibration

To calibrate the Kinect, there is a tutorial and are needed other packages found on the internet. It is useful to understand how to do the calibration: it is needed a check board 10×7 that is 9×6 to put as command to make the calibration that is detected by the software using the Kinect camera.
Two different calibration are made, one for RGB camera and one for IR depth camera.

B.3.3 Skeleton tracking

To track the skeleton it is necessary the use of `openni_tracker` package [50] that, using Kinect, can extract the information of all the joint of the skeleton and publishing their in RViz through a tf transform topic. The skeleton data are reported in MATLAB® as an animation using the *.csv* file exported using tf listener. The code in C++ to obtain data in a *.csv* file, uses a tf listener to get the position in 3 axis of all the joint of the skeleton with a frame rate that is a little bit lower than 30 Hz. Kinect is acquiring information at 30 Hz so `openni_tracker` cannot compute all the information at the same frame rate but a bit lower.

Bibliography

[1] B. Bäuml, T. Wimböck, and G. Hirzinger. "Kinematically optimal catching a flying ball with a hand-arm-system". In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2010, pp. 2592–2599.

[2] B. Bäuml et al. "Catching flying balls with a mobile humanoid: System overview and design considerations". In: *2011 11th IEEE-RAS International Conference on Humanoid Robots*. 2011, pp. 513–520.

[3] R. Baurès et al. "Intercepting free falling objects: Better use Occam's razor than internalize Newton's law". In: *Vision Research* 47.23 (2007), pp. 2982 –2991.

[4] N. Benguigui, H. Ripoll, and M. P. Broderick. "Time-to-contact estimations of accelerated stimuli is based on first-order information". In: *Journal of Experimental Psychology: Human Perception and Performance* 29.6 (2003), pp. 1083–1101.

[5] *C++*. `http://www.cplusplus.com`. (Visited on 06/14/2017).

[6] B. Cesqui et al. "Gaze Behavior in One-Handed Catching and Its Relation with Interceptive Performance: What the Eyes Can't Tell". In: *PLOS ONE* 10.3 (Mar. 2015), pp. 1–39.

[7] P. Cigliano et al. "Robotic Ball Catching with an Eye-in-Hand Single-Camera System". In: *IEEE Transactions on Control Systems Technology* 23.5 (2015), pp. 1657–1671.

[8] *Cyton Gamma 1500 Description*. `wiki.ros.org/cyton_gamma_1500_description`. (Visited on 06/14/2017).

[9] J. Dessing et al. "How Position, Velocity, and Temporal Information Combine in the Prospective Control of Catching: Data and Model". In: 17 (May 2005), pp. 668–86.

[10] P. Dias et al. "Detection of Aerial Balls Using a Kinect Sensor". In: *Lecture Notes in Artificial Intelligence* 8992 (Jan. 2015), pp. 537–548.

[11] A. A. Faisal and D. M. Wolpert. "Near Optimal Combination of Sensory and Motor Uncertainty in Time During a Naturalistic Perception-Action Task". In: *Journal of Neurophysiology* 101.4 (2009), pp. 1901–1912.

[12] U. Frese et al. "Off-the-shelf vision for a robotic ball catcher". In: *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium*. Vol. 3. 2001, pp. 1623–1629.

[13] A. Hauck et al. "What can be learned from human reach-to-grasp movements for the design of robotic hand-eye systems?" In: *Proceedings 1999 IEEE International Conference on Robotics and Automation*. Vol. 4. 1999, pp. 2521–2526.

[14] W. Hong and J. J. E. Slotine. "Experiments in hand-eye coordination using active vision". In: *Experimental Robotics IV*. Ed. by O. Khatib and J. K. Salisbury. Berlin, Heidelberg: Springer, 1997, pp. 130–139.

[15] B. Hove and J. J. E. Slotine. "Experiments in Robotic Catching". In: *1991 American Control Conference*. 1991, pp. 380–386.

[16] *How to run OpenNI.* `https://github.com/jstnhuang/cse481c_tutorials/wiki/How-to-run-openni_tracker`. (Visited on 06/14/2017).

[17] H. Katsumata and D. M. Russell. "Prospective versus predictive control in timing of hitting a falling ball". In: *Experimental Brain Research* 216.4 (2012), pp. 499–514.

[18] S. Kim, E. Gribovskaya, and A. Billard. "Learning motion dynamics to catch a moving object". In: *2010 10th IEEE-RAS International Conference on Humanoid Robots*. Nov. 2010, pp. 106–111.

[19] S. Kim, A. Shukla, and A. Billard. "Catching Objects in Flight". In: *IEEE Transactions on Robotics* 30.5 (2014), pp. 1049–1065.

[20] *Kinect on Ubuntu with OpenNI.* `http://www.20papercups.net/programming/kinect-on-ubuntu-with-openni/`. (Visited on 06/14/2017).

[21] *Kinect Sensor.* `https://developer.microsoft.com/en-us/windows/kinect/hardware`. (Visited on 06/14/2017).

[22] R. Lampariello et al. "Trajectory planning for optimal robot catching in real-time". In: *2011 IEEE International Conference on Robotics and Automation*. May 2011, pp. 3719–3726.

[23] M. F. Land and P. McLeod. "From eye movements to actions: how batsmen hit the ball". In: *Nature Neuroscience* 3 (Dec. 2000), p. 1340.

[24] D. Lee. "Guiding Movement by Coupling Taus". In: 10 (Sept. 1998), pp. 221–250.

[25] *Linux Ubuntu.* `https://www.ubuntu.com`. (Visited on 06/14/2017).

[26] J. López-Moliner and E. Brenner. "Flexible timing of eye movements when catching a ball". In: *Journal of Vision* 16.5 (2016), p. 13.

[27] C. de la Malla and J. Lopez-Moliner. "Predictive Plus Online Visual Information Optimizes Temporal Precision in Interception". In: *Journal of experimental psychology. Human perception and performance* 41 (June 2015).

[28] *MATLAB.* `https://www.mathworks.com/products/matlab.html`. (Visited on 06/14/2017).

[29] G. Montagne et al. "Movement reversals in ball catching". In: *Experimental Brain Research* 129.1 (1999), pp. 87–92.

[30] G. Montagne et al. "Perception-action coupling in an interceptive task: First-order time-to-contact as an input variable". In: *Human Movement Science* 19.1 (2000), pp. 59 –72.

[31] T. Moura. "Development of a Dual-Arm Robotic System for Gesture Imitation". MA thesis. DETI-UA, 2015.

[32] A. Namiki and M. Ishikawa. "Robotic catching using a direct mapping from visual information to motor command". In: *2003 IEEE International Conference on Robotics and Automation*. Vol. 2. 2003, pp. 2400–2405.

[33] A. J. R. Neves et al. "Detection of Aerial Balls in Robotic Soccer Using a Mixture of Color and Depth Information". In: *2015 IEEE International Conference on Autonomous Robot Systems and Competitions*. 2015, pp. 227–232.

[34] K. Nishiwaki et al. “The humanoid Saika that catches a thrown ball”. In: *Robot and Human Communication, 1997. RO-MAN ’97. Proceedings., 6th IEEE International Workshop on*. 1997, pp. 94–99.

[35] *OpenCV*. `http://opencv.org`. (Visited on 06/14/2017).

[36] *OpenNI*. `http://openni.ru`. (Visited on 06/14/2017).

[37] *OpenNI SDK*. `http://www.openni.ru/openni-sdk/openni-sdk-history-2/`. (Visited on 06/14/2017).

[38] *OpenNI tracker drivers*. `https://github.com/ros-drivers/openni_tracker`. (Visited on 06/14/2017).

[39] *Packages for ROS*. `packages.ros.org`. (Visited on 06/21/2017).

[40] G. R. Park et al. “Human-like catching motion of humanoid using Evolutionary Algorithm (EA) -based imitation learning”. In: *RO-MAN 2009 - The 18th IEEE International Symposium on Robot and Human Interactive Communication*. 2009, pp. 809–815.

[41] C. L. Peper et al. “Catching Balls: How to Get the Hand to the Right Place at the Right Time”. In: *Journal of experimental psychology. Human perception and performance* 20 (July 1994), pp. 591–612.

[42] M. Riley and C. G. Atkeson. “Robot Catching: Towards Engaging Human-Humanoid Interaction”. In: *Autonomous Robots* 12.1 (2002), pp. 119–128.

[43] *Robai Cyton Gamma 1500*. `http://www.robai.com/document/cyton-gamma-1500-spray-painting/`. (Visited on 06/14/2017).

[44] *ROS*. `http://www.ros.org`. (Visited on 06/14/2017).

[45] P. Routray. “Entertainment Robot for Catching a Flying Ball”. MA thesis. DETI-UA, 2016.

[46] *RViz*. `http://wiki.ros.org/rviz`. (Visited on 06/14/2017).

[47] S. S. M. Salehian, M. Khoramshahi, and A. Billard. “A Dynamical System Approach for Softly Catching a Flying Object: Theory and Experiment”. In: *IEEE Transactions on Robotics* 32.2 (2016), pp. 462–471.

[48] B. Siciliano et al. *Robotics: Modelling, Planning and Control*. 1st ed. Springer Publishing Company, Incorporated, 2008.

[49] J. A. Stone et al. “Emergent perception–action couplings regulate postural adjustments during performance of externally-timed dynamic interceptive actions”. In: *Psychological Research* 79.5 (2015), pp. 829–843.

[50] *Tutorial Skeleton for Kinect*. `https://flowers.inria.fr/tutorial-how-to-get-skeleton-from-kinect-openni-through-ros/`. (Visited on 06/14/2017).

[51] T. Yoshikawa. “Manipulability of Robotic Mechanisms”. In: *The International Journal of Robotics Research* 4 (June 1985), pp. 3–9.

[52] M. Zago et al. “Internal models and prediction of visual gravitational motion”. In: *Vision Research* 48.14 (2008), pp. 1532 –1538.

[53] M. Zhang and M. Buehler. “Sensor-based online trajectory generation for smoothly grasping moving objects”. In: *Proceedings of 1994 9th IEEE International Symposium on Intelligent Control*. 1994, pp. 141–146.

[54] H. Zhao and W. H. Warren. “On-line and model-based approaches to the visual control of action”. In: *Vision Research* 110 (2015). On-line Visual Control of Action, pp. 190 –202.