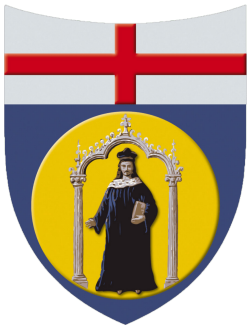
Sensor-based Dynamic Control for a Ball Catching Robot



Francesco Grella

DIBRIS - Department of Computer Science, Bioengineering, Robotics and System Engineering

University of Genova

Supervisors

Prof. Ing. Giorgio Cannata, Dott. Ing. Alessandro Albini

In partial fulfillment of the requirements for the degree of

Master of Science in Robotics Engineering

October 29, 2019

Dedicated to my family, to my girlfriend, to my friends, and to myself.

Acknowledgements

I would like to acknowledge all those who supported me along the development of my project. First of all I what to thank my supervisor, Prof. Ing. Giorgio Cannata and my co-supervisor, Dott. Ing. Alessandro Albini and Dott. Ing. Si Hao Wang for all the precious advices and technical support which made this entire work possible.
I would like to thank my parents Giorgio and Nicoletta for having patiently supported me through my entire studies, as well as my girlfriend Lorenza which always told me to keep up even in front of the toughest challenges.
I'd like to thank my room mates Marco and Laura for having brightened my day many times, as well as my course mates Luca G., Luca M. and Andrea because without them there would never have been the 'Avengers'.
Last but not least, a huge 'thank you' to my friends Giovanni, Edoardo, Gabriele Z., Lorenzo, Gabriele C., Federico, Gabriele M., Milosz and Gabriele R. for the countless laughters during breakfasts, lunches and coffee breaks.
Hoping that I haven't forgot anyone, thank you all again.

Francesco Grella

Abstract

The increasing complexity in robotic systems is leading to several challenges spread on different domains, spacing from robot control techniques to sensing architectures and software design.
The main purpose of a robot is the interaction with an external environment of any kind, which obeys to physical laws described by continuous mathematical models: a robotic application first of all must comply to these external rules in order to complete the task.
The well-known technique for controlling a robot inside a dynamic environment is sensor-based control, where one or more sensors provide awareness on the external changes, which is used to act accordingly.
The presented work consists in the complete development of a sensor based control framework for a robotic manipulator, starting from the latency assessment of the physical components as basis for the design of software components.
The control architecture is then implemented for performing a ball catching problem, identified as valuable benchmark for demonstrating the capabilities of the framework as well as its practical limitations.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

The concept of *sensor-based control* can assume different levels of complexity with respect to the target context and of course different approaches both on software and hardware sides can be required.
Indeed the complexity of the problem and the computational work required, as well as the electrical and mechanical efforts necessary for accomplishing a dynamic task increase according with the presence of spatial and timing constraints.
For instance a rapidly evolving environment is more challenging for a mobile robot which moves inside it: an autonomous driving car moving at 80 km/h in an highway surrounded by other moving vehicles has to respond efficiently to rapid variations of its surroundings to avoid accidents.
On the other hand the robot could be fixed, and the environment around it presents fast dynamic characteristics, for instance objects moving on a conveyor belt in an industrial scenario: here many of the movements are scripted ad-hoc, but feedback data from sensors are eventually required in order to discard eventual defective products.
In any case when the environment presents some considerable dynamics is mandatory to provide the fastest response possible, or at least drive a motion which is constrained to be executed under strict timing constraints.
In this kind of situations measurements must be handled in an efficient way too, in order to compute an efficient response of the system to the environmental variations.
Sensor-based software architectures are involved in a large amount of automation (*supply chains, chemical process control*) and robotics (*high performance motion control, drone navigation, obstacle avoidance*) applications.
The realization of a sensor-based software control architecture is not trivial, since its goal is to drive the behavior of a physical system in order to achieve high performances in motion

and safeness and interface it with asynchronous external events.
The whole process can be resumed in three main phases: *Design, Implementation and Benchmarking*:

- The design phase is the basis for the entire further development, and is basically voted to point out the limitations and capabilities of the physical components which will be used for acquiring perceptual data under time constraints.
  This becomes even more important in case of distributed control systems in which more processing units are involved, since communications between workstations and/or agents generate further time delays which sum up to those given by sensors.
  Moreover during this investigation process, to build a reliable timing model, is necessary to identify the maximum frequency of the physical plant to be controlled (in case of a robot its control loop frequency), because the whole control system must be synchronized to it.

- The definition of a reliable timing model based on practical delays and characteristic frequencies of sensors and plant is mandatory because the implementation phase must be driven by those constraints, in order to choose the proper software tools and programming techniques.

- As last part of the whole process, the architecture must be tested on a challenging to benchmark the control architecture.
  The main goals of this phase are to assess the capabilities of the control system and identify its limitations and its weak points.

1.1 Objectives

This work presents a full implementation of a sensor-based control framework for robot manipulators based on vision by following a well defined software design process.
To match the prefixed goals the architecture will be capable of handling image acquisition and processing from a camera, extract reliable three-dimensional spatial measurements and send them to the machine voted to generate real-time control signals for the robot.
In particular the workstation connected to the robot will acquire measurements, perform trajectory planning and then send commands to the robot controller.
After the implementation, the benchmark of the architecture will be performed over a non-prehensile ball catching control problem: the choice of such an exercise for the manipulator

was because of its timing and spatial constraints related to the projectile motion model.

1.2 Dissertation Structure

This dissertation is divided along six chapters: this introduction is the first one, while in the second one a review on the state-of-the-art in vision-based robot control and robot ball catching problems is presented.
Along the third chapter the implementation of the control architecture is discussed through the description of the image processing pipeline, the estimation algorithms and the control laws adopted for the different phases of the ball catching task.
The fourth chapter contains an in-depth technical description of the experimental setup and software technologies chosen for the implementation of the architecture.
In the fifth chapter the experimental results are presented for the design phase and each testing situation: the tuning and validation of the estimation algorithms, the motion law benchmark performed on the robot under time constraints and finally the prehensile and non-prehensile ball catching tasks.
Finally the conclusions and future work are depicted in the sixth chapter.

Chapter 2

State of the art

2.1 Vision-based Robot Control

Since from the early '80s, when the first vision systems with a frequency resembling the human perception (50/60 Hz) were built, the problem of controlling a robot through visual information is claimed to be a challenge for engineers.
Early applications of high-speed machine vision for closed-loop position control, or *visual servoing*, of a robot manipulator have been presented in [1], where the control problem through feature extraction with an eye-in-hand camera is performed, and many of the issues which are still consistent nowadays are pointed out for the first time.

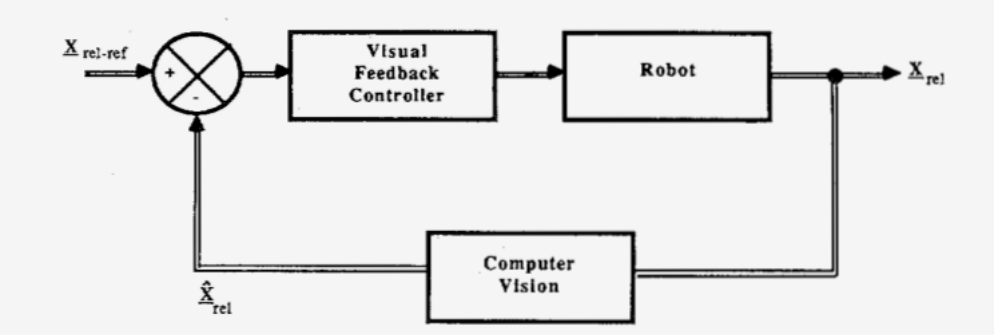


Figure 2.1 General vision-based control system shown in [1].

A hierarchical sensor-based control architecture is defined through a stack of observers which handle both propioceptive and exteroceptive sensors, but the most important point is that the work highlights the problem of *time delays* given by image acquisition and processing.

The visual servoing technique requires the camera to be mounted on the end effector in order to track a target: this 'eye-in-hand' configuration is still nowadays a state-of-the-art technique for this kind of applications.
The article presents control techniques by taking into account the image acquisition delay, which is depicted as a physical unavoidable phenomena with which must be deal.
This is done by including the estimated delay in the feedback measurements so that the error fed into the controller will be computed by keeping into account the shift between the two signals.

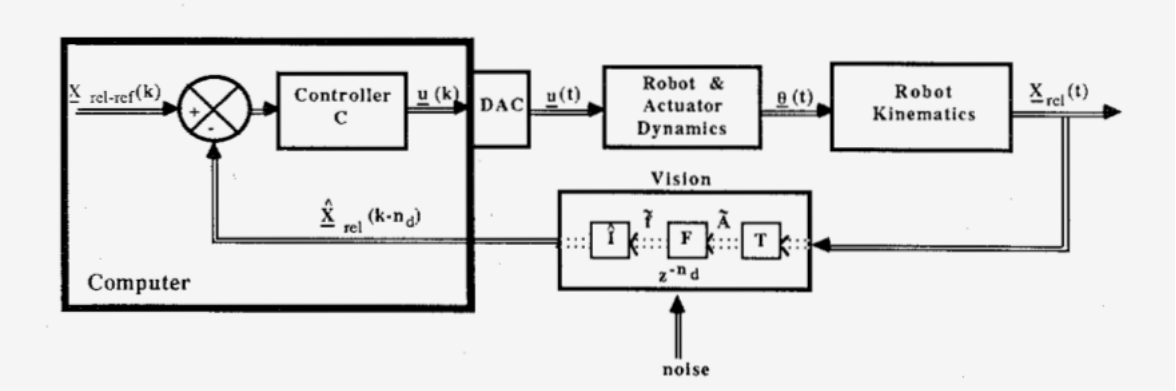


Figure 2.2 Detail of the vision-based control system with feedback time delays [1].

As pointed out in the control scheme, the measurement system represented by one or more vision sensors is affected by noise but most importantly introduces a certain amount of delay modeled as z^{-n_d}.
This quantity can be modeled as static, for instance as result of a statistic analysis on many values assumed by n_d, or even estimated online, and can be used in the design of the controller.
Moreover is analyzed an approach known as Image-Based Visual Servoing (IBVS) which has been broadly adopted in further years and completely exposed in [9].
This technique involves using information from the image to directly control the degrees of freedom of the robot,

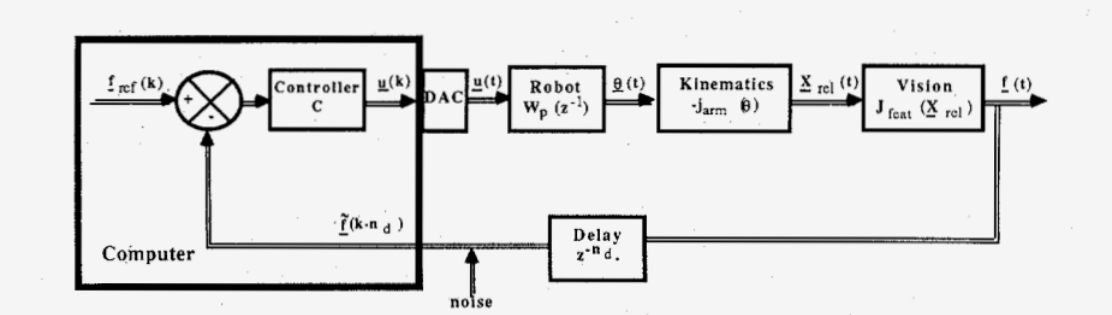


Figure 2.3 Detail of the IVBS control scheme [1].

The work depicted in [2] shows that in sensor-based control the performance is dependent on the sensor sampling rate, on delays in signal processing, and on the robot dynamics, and describes an approach in which control is inherently stable as long as the time instant of sensing is known, independently of delays.
Sensor data are needed to provide a refined target for positional control, and experiments are performed on a contour following task, exploiting a slow but very precise and neat motion of the robot.

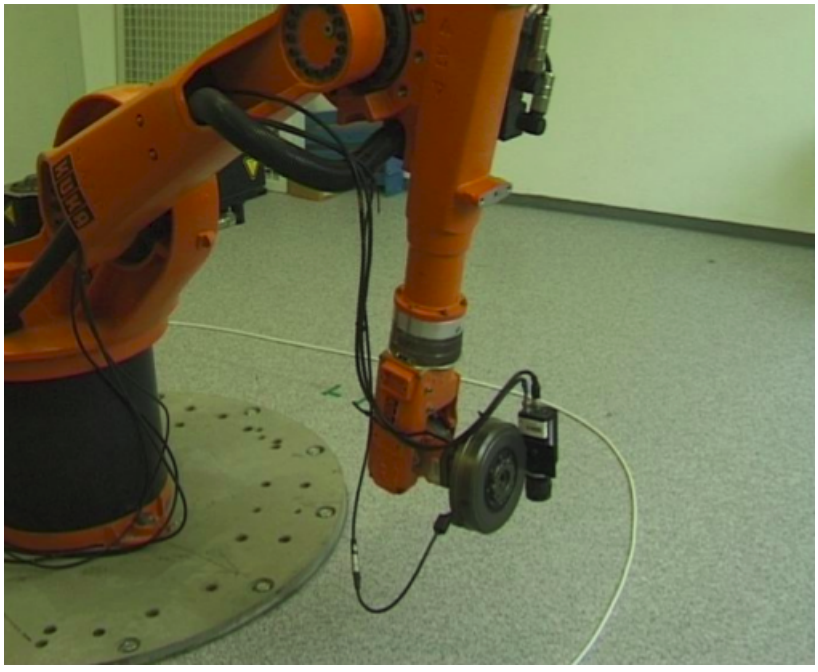


Figure 2.4 Cable following experiment depicted in [2].

The work in [3] addresses the development of a control scheme for visual servoing that explicitly takes into account the delay introduced by image acquisition and processing.

An estimator that predicts several samples ahead of time is properly included in the scheme and the system is analyzed in terms of dynamics and steady-state errors.

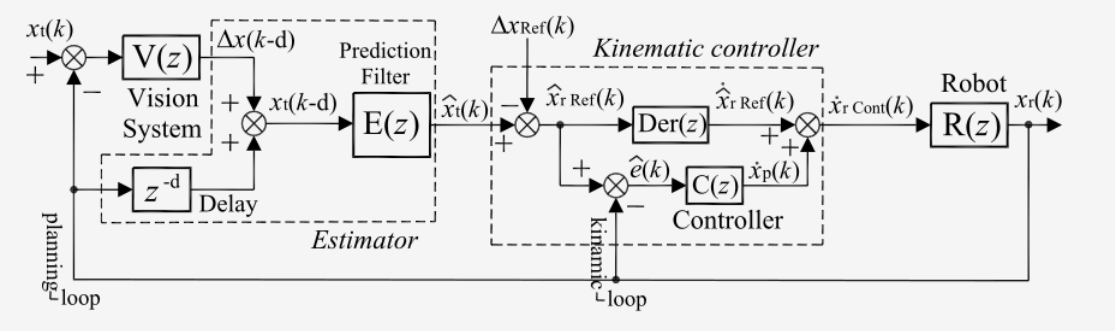


Figure 2.5 The delay-compensating control scheme shown and analyzed in [3].

The valuable contribution of the work is in fact the definition of a control architecture which explicitly models the time delay in the system: its benefits and limitations have been comparatively illustrated by simulations and experiments using a 3 dof Cartesian robot, but no further analyses have been performed on arm manipulators.
As shown before, visual servoing control presents indeed several practical problems common for any vision application, but the results are more oriented on precision and smoothness of motion rather than execution speed.
To achieve high-speed results the motion control must be intended in a different way, because in order to reach an objective acquired through sampled images within a time deadline is not always possible to have a continuous visual feedback, but instead some significant informations at a given time.

2.2 Robotic Ball Catching

Endow a robotic manipulator with the capability of catching moving objects, with or without a gripper, is still nowadays a challenge which requires finely tuned estimation algorithms and sensor-based control architectures with high performances.
Before diving more deeply through the most recent and valuable projects related to this subject, it's important to remind a pioneering work which certainly have been the starting point for this kind of applications.
In [4] is described how a small 3-DOF manipulator can catch a ping-pong plastic ball: this is

the proof that the value of such a robotic task, to be used as benchmark and/or demonstration, is well conceived since before the 00's.

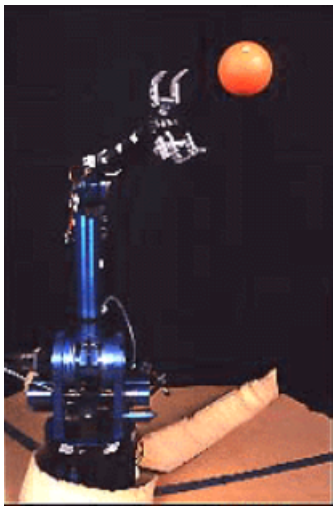


Figure 2.6 First robot ball-catching task in literature, by [4].

By taking into account more recent works, [5] investigates the possibilities of real-time control of manipulators by the means of a ping-pong ball juggling system that is able to catch a ball thrown by a human and to keep the ball bouncing.
Two industrial cameras operating at 60 Hz have been used as sensors for the measurement part of the control loop, a non-linear physical model which takes into account friction forces has been used for trajectory prediction jointly with a Kalman Filter, and most importantly three different controllers have been used for vertical reflection, maximum impulse and deviation zeroing through PID action.
The work, even if interesting from the control design point of view, lacks a proper section related to timing aspects and how they have influenced the control design.

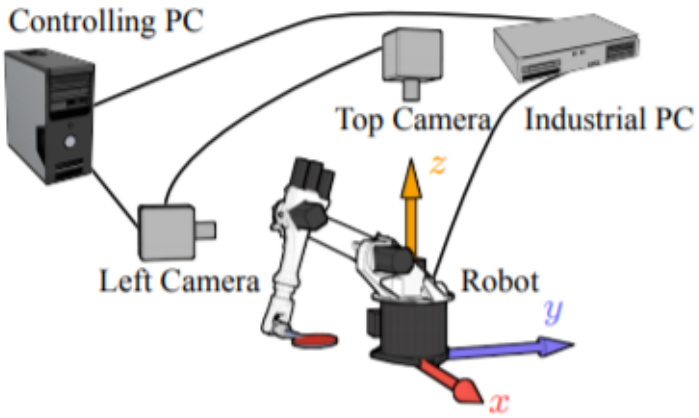


Figure 2.7 Ball Juggling Robot Architecture as depicted in [5].

The work of [6] shows a non-prehensile ball catching task: a basket ball is caught by a plate attached to the end effector of a 6-dof arm.
The situation of catching after an initial bounce on the plate is taken into account to depict the concept of *Dynamic Manipulability* and the fact that considering the dynamics of the ball is mandatory for this kind of control tasks.
Similarly to [5] the trajectory of the ball is estimated through a vision algorithm and a least-squares predictor, and the control has been implemented with Matlab/Simulink-based software tools running on an RTAI Linux environment.

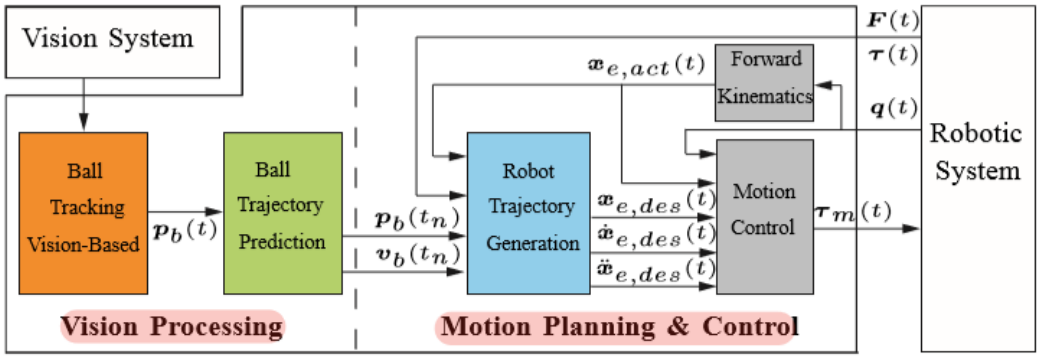


Figure 2.8 Complete control architecture depicted in [6].

Another work related to object catching using a vision-based control framework is presented in [7], where an high-payload industrial robot and a monocular camera are used for an

eye-in-hand ball catching application.
In this work the control algorithm is divided in two main phases: the first one is a vision-based ball interception and the second can be described as an estimation-based catching.
First, when the ball is thrown by the pitcher, the robot is guided to follow the movements of the ball through the visual information provided by the camera mounted in eye-in-hand configuration: this is a particularity of this project, since the vision system is not fixed but is rigidly attached to the flange of the robot, so relative motion had to be kept into account.
In particular the image processing is kept fast and reliable through a dynamic windowing technique voted to reduce the portion of the image that has to be elaborated, in order to reduce the computational complexity of the image processing task.
A color-based clustering in Hue, Lightness, and Saturation Color Space (HSL) has been used for ball detection, and an extended Kalman filter (EKF) was employed in non-linear trajectory refinement.

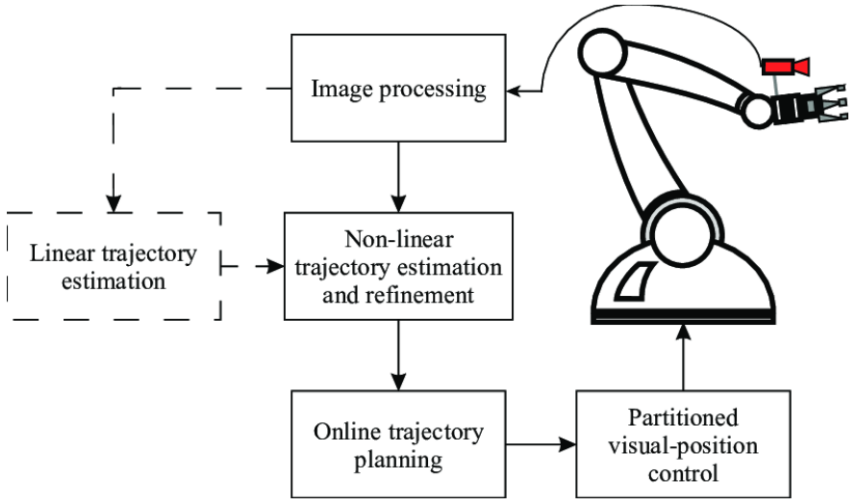


Figure 2.9 Monocular Ball Catching: block diagram of the system depicted in [7].

The robot performs a continuous tracking of the ball in order to keep it inside the view of the camera and for bringing the gripper in an advantage position for the catch: when the ball is far the end-effector is moved in a suitable direction, and while the ball approaches the trajectory estimation is refined at each step until the stop moment.
In detail, the motion control is performed by computing references in the 3D Cartesian space through a fifth-order polynomial: with initial conditions from the current robot state (position, velocity and acceleration), the coefficients are computed to reach the estimated catching position, with null velocity and acceleration, at the estimated catching time.

When a new estimation is available, starting from the current position of the robot gripper, the coefficients can be tuned again in order to reach the new estimated catching position in the new estimated catching time.

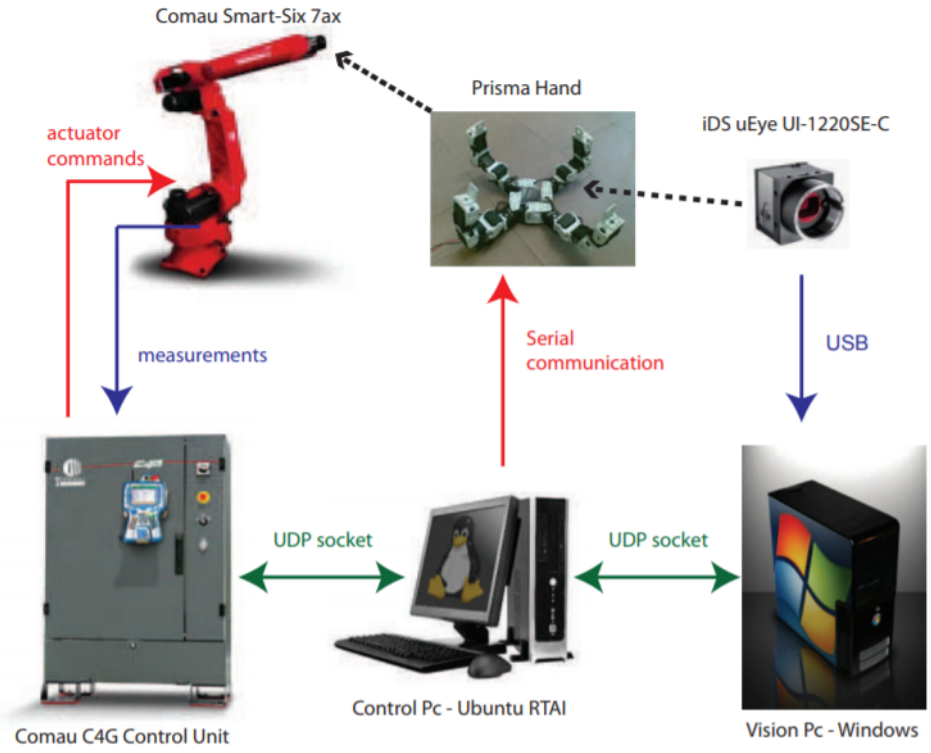


Figure 2.10 Monocular Ball Catching experimental setup adopted in [7].

The control architecture relies two PCs, one voted to image processing and another one, running RTAI Linux, which is used for sending references to the robot controller, showing that real-time software is strongly recommended for performing this kind of tasks.

Chapter 3

Control Architecture Design

In this chapter is presented the full design methodology of the control architecture for sensor-based robot control.
The design is driven by the goal of controlling a manipulator through vision sensors in order to intercept and catch flying objects: the conceptual architecture is composed by three main parts: an object detection pipeline, an estimation algorithm to deal with noisy measurements and a control module capable of computing control signals for a quick motion.
The ball detection problem is introduced, then the object tracking problem is presented and the Kalman filter estimation algorithm related to moving objects in 3D space is described.
Then the following section describes the techniques adopted to generate control signals.

3.1 Visual Sensing Module

The starting point in the design of a sensor-based control architecture is indeed the software module which handles the perceptual data and prepare them in an efficient way to be used for control purposes.
In this project a vision system has been employed for tracking mid-air traveling objects directed towards robot's workspace.
A pipeline of modules which implement computer vision techniques is used to merge these requirements with a reliable and efficient recognition and abstraction of object's coordinates to compute suitable trajectories for controlling the robot motion.
In particular a straightforward sequence processes images acquired from the camera and through steps which are repeated at each new frame extracts quantitative data on the position of the object in space.

The problem of ***object segmentation and tracking*** is a well-known issue in engineering and computer science [10]: the proposed module covers the steps performed in most of the works available in literature like [6], [7] and in general many of the classical computer vision applications.

3.1.1 Gaussian Smoothing

The very first step in the pipeline is an early processing operation on the raw image voted to reduce noise and is called *Smoothing*.
In statistics and image processing, to smooth a data set is to create an approximating function that attempts to capture important patterns in the data, while leaving out noise or other fine-scale structures/rapid phenomena. In smoothing, the data points of a signal are modified so individual points (presumably because of noise) are reduced, and points that are lower than the adjacent points are increased leading to a smoother signal [11].

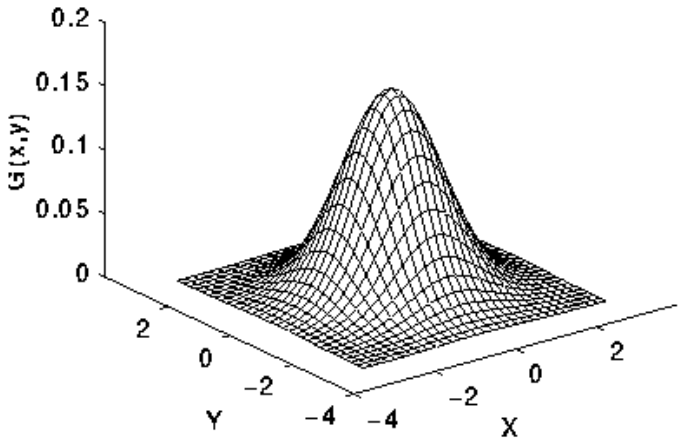


Figure 3.1 Sample shape of a Gaussian kernel.

The smoothing operation is performed by convolution with a Gaussian kernel: this operation allows to compute for each pixel a weighted sum of its surroundings, considering more neighbor pixels if the size of the kernel is higher.

HSV Color Space Transformation

After the application of a noise reduction filter, is necessary to change the color space of the image in order to have a sharp subdivision of the regions of the image.
A common practice in image processing and object recognition is to use the HSV (Hue, Saturation, Value) color space, even known as HSB (hue, Saturation, Brightness), which is an alternative representation of the RGB color model, designed in the 1970s by computer graphics researchers to more closely align with the way human vision perceives color-making attributes [12], [13].
In these models, colors of each hue are arranged in a radial slice, around a central axis of neutral colors which ranges from black at the bottom to white at the top.
The HSV representation models the way paints of different colors mix together, with the *saturation* dimension resembling various tints of brightly colored paint, and the *value* dimension resembling the mixture of those paints with varying amounts of black or white paint.

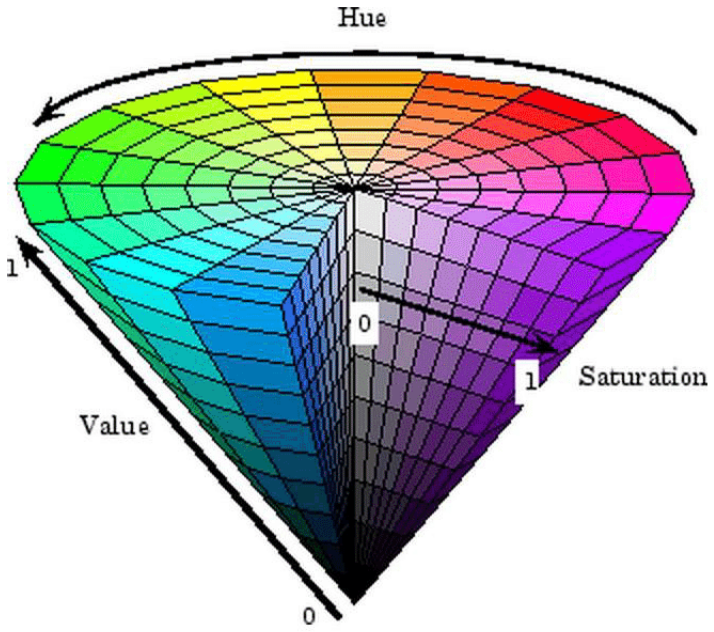


Figure 3.2 The HSV cone representation.

This color space has been chosen because it separates color information from intensity or lighting: this allows to threshold an image regardless of lighting changes in the value channel.
Even by singling out only the hue one can achieve a very meaningful representation of the

base color that will likely work much better than RGB. The end result is a more robust color thresholding over simpler parameters.

Image Thresholding and Binarization

After performing a conversion to a more convenient color space, a threshold on the image can be finally applied.
The thresholding operation is based on proper HSV values which correspond to RGB counterparts of specific colors, and is necessary to get to a new representation of the scene called *binary image*.
Basically each pixel's triplet of values is compared with a minimum threshold and the correspondent pixel of the output image is black ('0') if the value is under threshold or white ('1') if above.
The result is a black and white image where the only visible region is the segmented 'blob' corresponding to the object, from where is possible to estimate the centroid in pixel coordinates from inside the image itself [14].

Binary Image Morphing

After the binarization of the image some noise can still be present, especially in case that the camera has a quite large field of view and so many objects and details can be perceived: it can happen that other elements different from the object designed for segmentation have HSV similar values and are detected as positive results.
To avoid this and help the system to recognize only the requested object a very useful step in the pipeline is the morphing of the binary image.
Basically is a sequence of *erode* and *dilate* steps, which consist respectively in deleting and enhancing the edges of all the white regions which correspond to positive detections in the scene [15], [16].
Depending on how many 'false positives' appear in the scene, a suitable number of erosions must be performed in order to make them disappear; the drawback of this operation is that at the end of the morphing even the main blob, which corresponds to the correct object, has a quite reduced size, so at this point the edges must be dilated by converting to white all the adjacent pixels of the blob.
After this operation there are no more objects in the scene except for the segmented target,

which in the presented case is the yellow ball.

Depth Measurements

The computation of the 3D position in space is the final step of the image processing pipeline, after the computation of a single pixel corresponding to the centroid of the segmented object. The problem can be resumed in a change of coordinates, in particular from 2D pixel coordinates to 3D world coordinates: this operation is automatically executed by the device, since any RGB-D camera already provides depth map and point cloud measurements based on its intrinsic parameters.

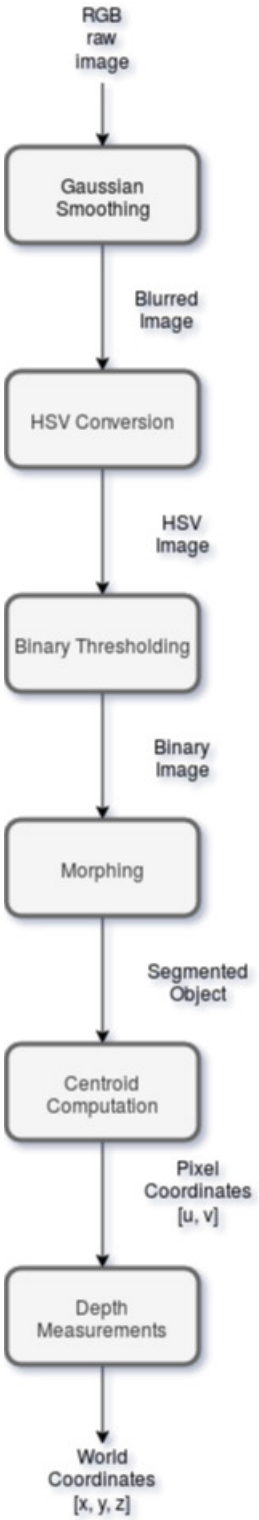


Figure 3.3 The scheme of the Image Processing pipeline.

3.1.2 Ball Tracking

The goal of a sensing module in a robot control architecture is to provide informations about the surrounding environment, whether is this rapidly changing or static.
For performing fast and/or precise control actions on the environment raw measurements are not enough, except for cases in which involved sensors provide almost perfect measurements or the situation requires measuring static or quasi-static quantities.
In fact for acting on a moving object a well defined practice in control engineering is to perform *tracking* [17]: this technique allows the system to get informations on the motion of an object by filtering the raw measurements of the sensors: in fact after retrieving 3D coordinates of the object through the image processing pipeline, the resulting data must be further processed in order to reduce noise and get to a more regular trajectory.
This process is known in signal processing and control engineering as *Filtering* and is performed through recursive algorithms.

3.1.3 Kalman Filter

Kalman filtering, also known as linear quadratic estimation (LQE), is an algorithm that uses a series of measurements observed over time, containing statistical noise and other inaccuracies, and produces estimates of unknown variables that tend to be more accurate than those based on a single measurement alone, by estimating a joint probability distribution over the variables for each time frame [18]. The algorithm works in a two-step process: in the prediction step, the Kalman filter produces estimates of the current state variables, along with their uncertainties.
Once the outcome of the next measurement (corrupted with some amount of error, including random noise) is observed, these estimates are updated using a weighted average, with more weight being given to estimates with higher certainty. The algorithm is recursive and can run in real time, using only the present input measurements, the previously calculated state and its uncertainty matrix [19].
The Kalman filter produces an estimate of the state of the system as an average of the system's predicted state and of the new measurement using a weighted average: the purpose of the weights is that values with smaller estimated uncertainty are "trusted" more.
The weights are computed from the covariance, a measure of the estimated uncertainty of the prediction of the system's state.
The result of the weighted average is a new state estimate that lies between the predicted and

measured state, and has a better estimated uncertainty than either alone.
This process is repeated at every time step, with the new estimate and its covariance informing the prediction used in the following iteration: this means that Kalman filter works recursively and requires only the last "best guess", rather than the entire history, of a system's state to calculate a new state [20].

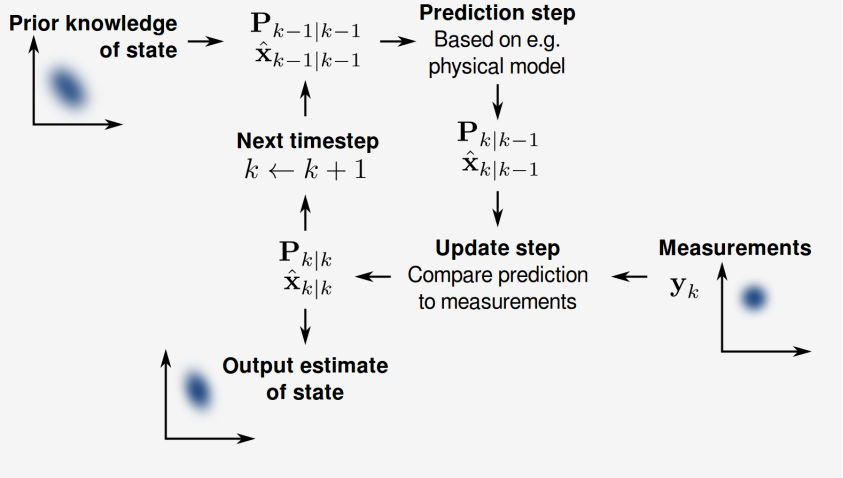


Figure 3.4 Graphical explaination of the Kalman Filter Algorithm.

Kalman filters are based on linear dynamical systems discretized in the time domain: they are modeled on a Markov chain built on linear operators perturbed by errors that may include Gaussian noise.
The state of the system is represented as a vector of real numbers, and at each discrete time increment, a linear operator, called **Transition Matrix** is applied to the state to generate the new state, with some noise mixed in, and optionally some information from the controls on the system if they are known.
Then, another linear operator mixed with more noise generates the observed outputs from the true ("hidden") state.

In order to use the Kalman filter to estimate the internal state of a process given only a sequence of noisy observations, one must model the process in accordance with the framework of the Kalman filter. This means specifying the following matrices:

- $\mathbf{F}_k$, the state-transition model
- $\mathbf{H}_k$, the observation model

- $\mathbf{Q}_k$, the covariance of the process noise
- $\mathbf{R}_k$, the covariance of the observation noise
- $\mathbf{B}_k$, the control-input model, which is not mandatory for the filter to work, and if the system has no inputs has to be left unspecified

The Kalman filter model assumes the true state at time k is evolved from the state at (k - 1) according to the following law:

$$\mathbf{x}_k = \mathbf{F}_k \mathbf{x}_{k-1} + \mathbf{B}_k \mathbf{u}_k + \mathbf{w}_k \tag{3.1}$$

where:

- $\mathbf{F}_k$ is the state transition model which is applied to the previous state $\mathrm{x}_k - 1$;
- $\mathbf{B}_k$ is the control-input model which is applied to the control vector $\mathbf{u}_k$;
- $\mathbf{w}_k$ is the process noise which is assumed to be drawn from a zero mean multivariate normal distribution.

then an observation (or measurement) z_k of the true state x_k is made according to the equation:

$$\mathbf{z}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{v}_k \tag{3.2}$$

where:

- $\mathbf{H}_k$ is the observation model which maps the true state space into the observed space;
- $\mathbf{v}_k$ is the observation noise which is assumed to be zero mean Gaussian white noise with covariance $\mathbf{R}_k$.

Predict and Update

The Kalman filter is conceptualized as two distinct phases: ***Predict*** and ***Update***.
The predict phase uses the state estimate from the previous time step to produce an estimate of the state at the current step: this predicted state estimate is also known as the a priori state estimate because, although it is an estimate of the state at the current step, it does not include current observation information.
In the update phase, the current a priori prediction is combined with current observation information to refine the state estimate, which is termed the a posteriori state estimate.
Typically, the two phases alternate, with the prediction advancing the state until the next scheduled observation, and the update incorporating the observation.
However, if an observation is unavailable for some reason, the update may be skipped and multiple prediction steps performed.

The prediction phase includes the following steps:

- Predicted (a priori) state estimate

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_k \hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_k \mathbf{u}_{k-1} \tag{3.3}$$

- Predicted (a priori) error covariance

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^{\top} + \mathbf{Q}_k \tag{3.4}$$

In the update phase the following operations are performed:

- Innovation or measurement pre-fit residual

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_{k|k-1} \tag{3.5}$$

- Innovation (or pre-fit residual) covariance

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^{\top} + \mathbf{R}_k \tag{3.6}$$

- Optimal Kalman gain

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^{\top} \mathbf{S}_k^{-1} \tag{3.7}$$

- Updated (a posteriori) state estimate

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k \tag{3.8}$$

- Updated (a posteriori) estimate covariance

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k)\, \mathbf{P}_{k|k-1} \tag{3.9}$$

- Measurement post-fit residual

$$\tilde{\mathbf{y}}_{k|k} = \mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_{k|k} \tag{3.10}$$

3.1.4 Ball Trajectory Prediction

In order to provide an efficient response of the robot under strict timing conditions is necessary to predict and anticipate the behavior of the target object.
The motion of the robot must be time-driven, so that the end-effector can reach desired position and orientation within a specified time interval related to the motion of the flying object.
The Kalman filter provides an overall good estimation of the parabolic motion and velocities of the object, moreover the acceleration is constant and is assumed as known, so it's possible to compute a prediction of the projectile motion at each time step.
In particular at each step the filter provides estimated space positions and velocities, so a feasible projectile motion can be initialized with those values.
The adopted equations resemble the 3D projectile motion:

$$x(t) = x_0 + v_{0x}t \tag{3.11}$$

$$y(t) = y_0 + v_{0y}t \tag{3.12}$$

$$z(t) = z_0 + v_{0z}t - \frac{1}{2}gt^2 \tag{3.13}$$

It's clear that the motion can be decomposed in two parts: a constant velocity motion on the (x,y) plane and a constant acceleration vertical motion on the z axis of the world frame.

3.2 Control and Trajectory Planning Module

In the design and implementation of a distributed and sensor based control architecture the same importance must be given on the sensors and on the robot side.
Indeed having an accurately tuned software block which handles the sensors is mandatory especially when the goal is a quite fast control action: as seen before the unique usage of raw measures even if provided by a very efficient device is definitely not enough.
On the other side it's clear that once suitable and reliable references can be computed and sent to the system, an efficient software model must ensure that the plant tracks them and computes proper signal to perform control by respecting the specifications.

3.2.1 Inverse Kinematics

This well-known robot control technique specifies allows to computes the associated joint angles by specifying the end-effector location [9]: in case of serial manipulators this is achieved by solving a set of polynomials obtained from the kinematics equations.
The time derivative of the kinematics equations yields the Jacobian of the robot, which relates the joint rates to the linear and angular velocity of the end-effector.
Moreover the Jacobian also provides a relationship between joint torques and the resultant force and torque applied by the end-effector, a property which is very important when performing dynamic control [21].

For which regards the tracking of a generic velocity profile derived by a trajectory expressed in Cartesian space $\mathbf{x}(t)$, the following equation yields the Inverse Kinematics problem:

$$\dot{\mathbf{x}}(t) = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}}(t) \tag{3.14}$$

Since $\dot{\mathbf{x}}(t)$ is arbitrary and the Jacobian $\mathbf{J}(\mathbf{q})$ is known at each instant through the robot controller, the required $\dot{\mathbf{q}}(t)$ can be computed by inverting the matrix like follows:

$$\dot{\mathbf{q}}(t) = \mathbf{J}^{\dagger}(\mathbf{q})\dot{\mathbf{x}}(t) \tag{3.15}$$

where $\mathbf{J}^{\dagger}(\mathbf{q})$ is the *pseudoinverse* of the Jacobian matrix.
The pseudo-inverse is usually adopted to find a "best fit" (least squares) solution to a system of linear equations that lacks a unique solution. [22], [9]

3.2.2 Control and Trajectory Planning Task

This module covers the main role of this part of the architecture, which is the generation of reference signals to be sent to the robot controller, voted to traduce them in motor commands to the joints.
Even if it's structured as a sequential pipeline, it can be conceptually divided into two sections with respect to which values of the end-effector pose have to be controlled, assuming that the pose is

$$\mathbf{x} = [x, y, z, \alpha, \beta, \gamma] \tag{3.16}$$

where α represents the angle of a rotation about the x axis, β about the y axis and γ about the z axis.

Orientation Control

The orientation error is computed and expressed through the ***Angle-Axis*** representation, which is intended to parameterize a rotation in a three-dimensional Euclidean space by two quantities: a unit vector $\hat{\mathbf{e}}$ indicates the direction of an axis of rotation, and an angle θ describing the magnitude of the rotation about the axis.[9]
This leads to the expression of ω which is the rotational velocity about the axis.
From ω a suitable proportional control law for the orientation of the end-effector is computed as follows, with γ set as constant scalar gain:

$$\omega = -\gamma \cdot \mathbf{e} \cdot sin(\theta) \tag{3.17}$$

which in the Inverse Kinematics module is mapped to joints:

$$\dot{\mathbf{q}}_{\omega}(t) = \mathbf{J}^{\dagger}(\mathbf{q})\omega(t) \tag{3.18}$$

where $\dot{\mathbf{q}}_{\omega}(t)$ points out the joint velocity vector related to Cartesian angular velocities of the end-effector.
Since the orientation control goal is a convergence to a desired orientation through the minimization of an angular error, adopting the sine of θ ensures stability when the angle becomes small.
A more detailed description of the algorithm can be found in appendix A.

Trajectory Planning

In an application which requires reaching a given position in space at a given time, the control of the end-effector's position assumes a primary role.
Moreover for the addressed problem is important that the end-effector reaches the catch point at the same estimated velocity of the ball in order to avoid bounces on the plate.
The most known robot control technique for achieving this result is the *Trajectory Planning* through *Polynomial Interpolation*, an analytical technique that allows to compute the interpolation of a given data set by the polynomial of lowest possible degree that passes through all the points of the dataset. [9], [7]

By assuming that the interpolation polynomial is in the form

$$\mathbf{p}(\mathbf{x}) = \mathbf{a_n x^n} + \mathbf{a_{n-1} x^{n-1}} + \cdots + \mathbf{a_2 x^2} + \mathbf{a_1 x} + \mathbf{a_0} \tag{3.19}$$

The following statement says that p interpolates the data points means that for each i from 1 to n:

$$p(x_i) = y_i \tag{3.20}$$

By substituting equation (5.34) in here a system of linear equations in the coefficients a_k can be found.
The system in matrix-vector form reads the following multiplication:

$$\begin{bmatrix} x_0^n & x_0^{n-1} & x_0^{n-2} & \dots & x_0 & 1 \\ x_1^n & x_1^{n-1} & x_1^{n-2} & \dots & x_1 & 1 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ x_n^n & x_n^{n-1} & x_n^{n-2} & \dots & x_n & 1 \end{bmatrix} \begin{bmatrix} a_n \\ a_{n-1} \\ \vdots \\ a_0 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix} \tag{3.21}$$

Solve the system for *ak* will construct the interpolating function $p(x)$.

After the solution of the system the following reference signals can be defined

$$\mathbf{x}^*(t) = \mathbf{p}(t) \tag{3.22}$$

$$\dot{\mathbf{x}}^*(t) = \dot{\mathbf{p}}(t) \tag{3.23}$$

so the Cartesian velocity control law is built by composing the feedforward signal $\dot{\mathbf{x}}^*(t)$ plus a differential position signal for ensuring the tracking of the third order trajectory, defined as follows:

$$\mathbf{e}(t) = \mathbf{x}(t) - \mathbf{x}^*(t) \tag{3.24}$$

so assuming a positive arbitrary gain γ, the Cartesian control law is

$$\dot{\mathbf{x}}_c(t) = -\gamma \cdot \mathbf{e}(t) + \dot{\mathbf{p}}(t) \tag{3.25}$$

The suitable joint velocities are then computed through the inverse kinematics

$$\dot{\mathbf{q}}_\mathbf{c}(t) = \mathbf{J}^\dagger(\mathbf{q})\dot{\mathbf{x}}_c(t) \tag{3.26}$$

Chapter 4

Computational Model

In this chapter the implementation of the control architecture is discussed.
In the previous chapter the sensor-based control architecture has been presented on the modeling level, by keeping into account all the theoretical and mathematical aspects of each component.
Along this chapter the main components are presented under an implementation perspective: in fact after the definition of the general models of the blocks it must be specified how they are realized and merged to build the software architecture depending on the devices and technologies at disposal.

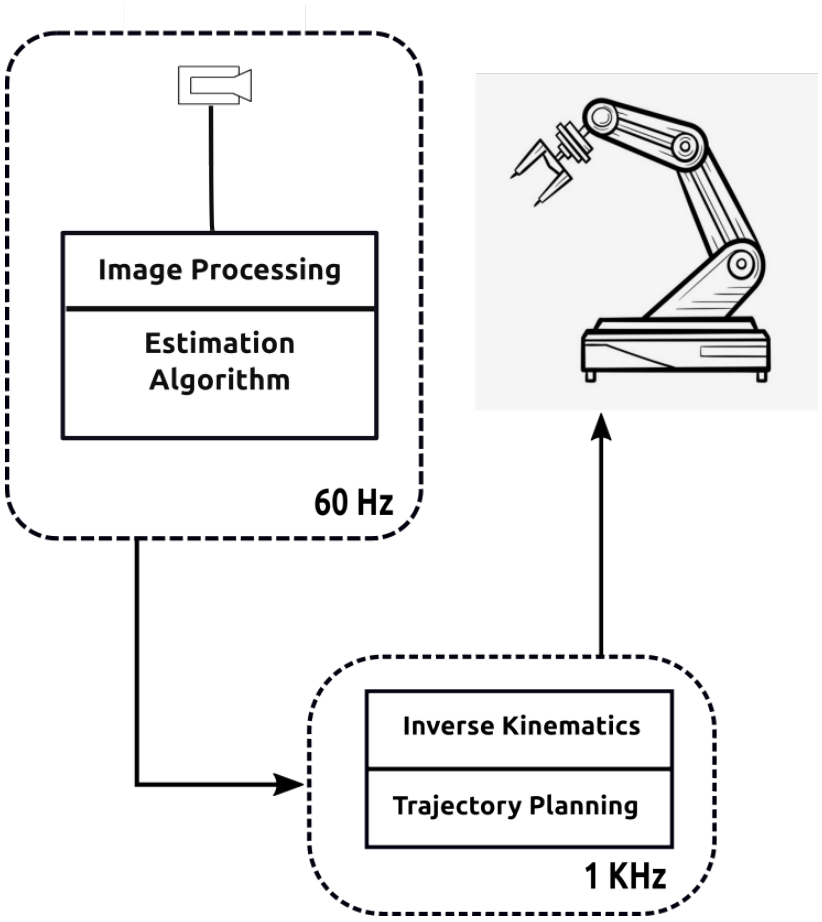


Figure 4.1 Schematic representation of the complete architecture.

4.1 Visual Sensing module

The implementation of the sensor handling software module has been driven by the vision system employed for tracking mid-air traveling objects directed towards robot's workspace: a minimum requirement on the frequency of acquisition around 60 Hz is mandatory for having an acceptable amount of samples of the object and in the meantime a resolution of $720p$ or above is required for a proper tracking in the scene.
The overall structure of the pipeline, as well as its modeling, have been discussed in the previous chapter, so the following sections will only report issues related to implementation aspects for the components.

4.1.1 Image Processing Pipeline

The Image Processing Pipeline is the starting point of each iteration of the proposed architecture, it's implemented as a 'software assembly line' which processes raw material (the images) and after a sequence of steps presents the final product which are the extracted informations, in this case spatial coordinates of an object.

Gaussian Smoothing

The smoothing operation in the pipeline has been implemented by convolution with a 3x3 gaussian matrix: due to early processing the acquisition noise is already handled by the camera itself, so the raw image already has a reduced level of noise.
An higher dimension smoothing kernel allows to achieve better noise reduction, but slows down the algorithm in a proportional way to the size of the matrix.

Figure 4.2 Smoothed scene at 720p resolution).

The algorithm implemented in OpenCV performs a convolution, which is a linear and time-invariant operation, of the image with the so-called 'gaussian kernel', a well known geometric structure for signal smoothing: In two dimensions, it is the product of two such

Gaussian functions, one in each dimension:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{4.1}$$

where x is the distance from the origin in the horizontal axis, y is the distance from the origin in the vertical axis, and σ is the standard deviation of the Gaussian distribution [11]. The adopted matrix is the discrete 'pixel-wise' version of the (5.3), and the convolution is performed by translating the kernel for each pixel and computing the linear combination of all the overlapped pixels.

Image Thresholding and Binarization

Due to the presence of many objects in the scene, a simple GUI has been developed in order to adjust on-line the thresholds for binarization while observing in real-time the video stream: this method allowed a simple and robust adjustment of the thresholds until the best combination under any light condition can be quickly found to perform experiments.
For performing object detection experiment a yellow ball has been chosen, due to the fact that is a color easy to isolate in HSV space: the following table reports the minimum and maximum values chosen for the segmentation of the ball region.

Table 4.1 Table of ball segmentation thresholds

Component	**Minimum value**	**Maximum value**
H	24	45
S	135	256
V	92	217

A more detailed description of the thresholding process can be found in Appendix D.

Depth Measurements

Is important that the RGB and depth retrieved images have the same resolution in order to have a 1:1 correspondence when retrieving the measurements: in the proposed pipeline the 2D pixel coordinates are fed into a function which returns the correspondent triplet of

coordinates taken by the point cloud measurements in meters (or other unit specified before the acquisition loop).

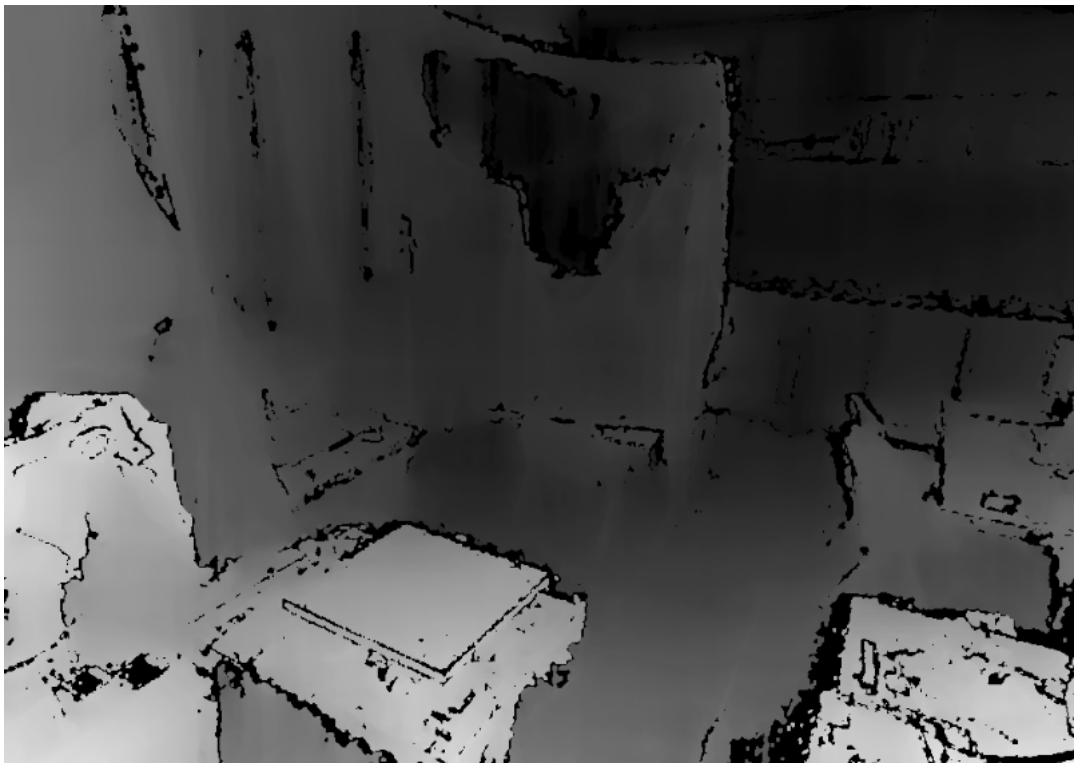


Figure 4.3 Depth map of the scene. The throw region is almost totally free from occlusions

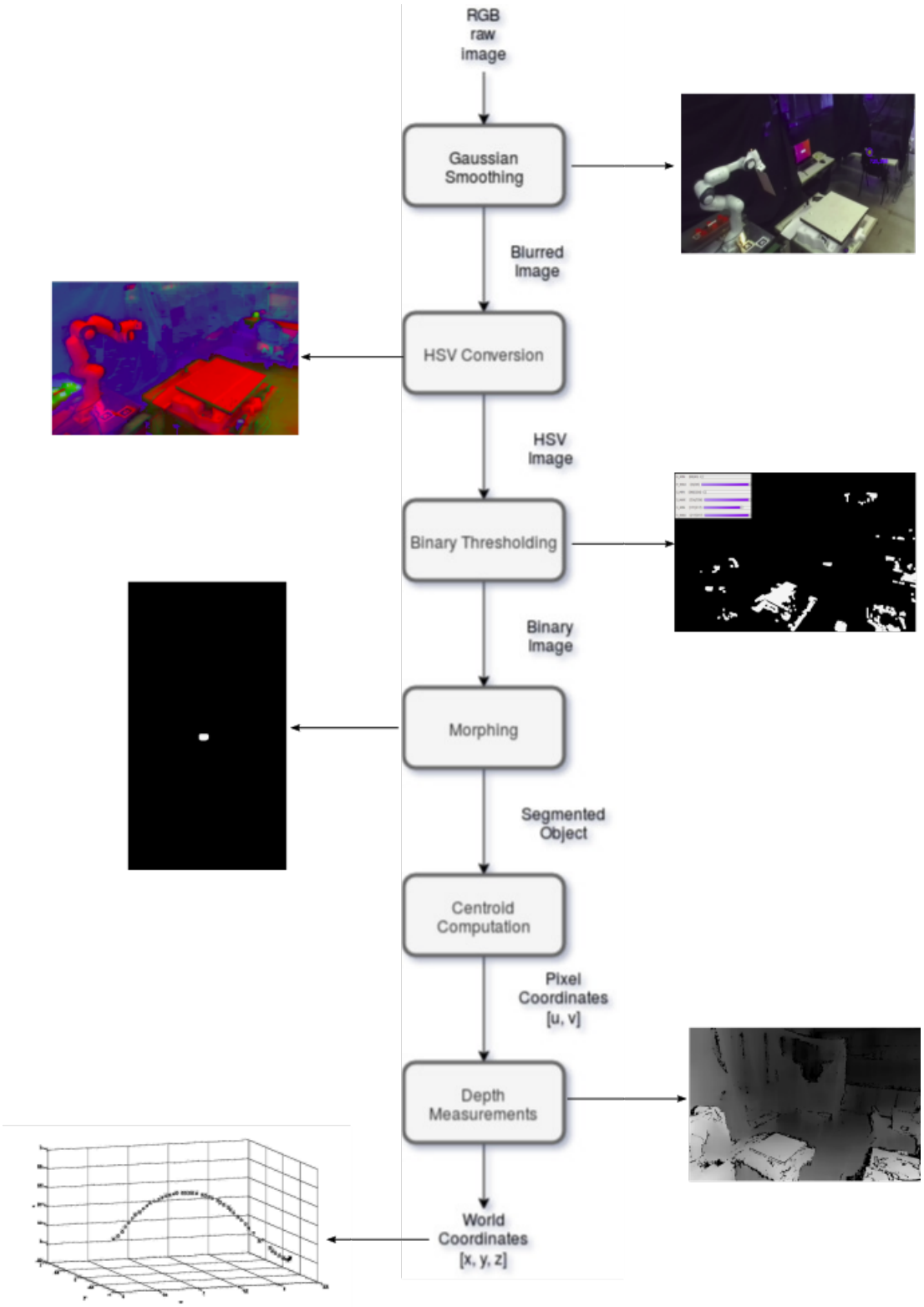


Figure 4.4 The Image Processing pipeline: results after the implementation.

4.1.2 Ball Tracking

The following figures represent the samples in 3D space of a typical trajectory registered during a throw of the ball.
It can be seen that parallax errors provide more noise on the Z axis of the optical camera frame, centered in the shutter.

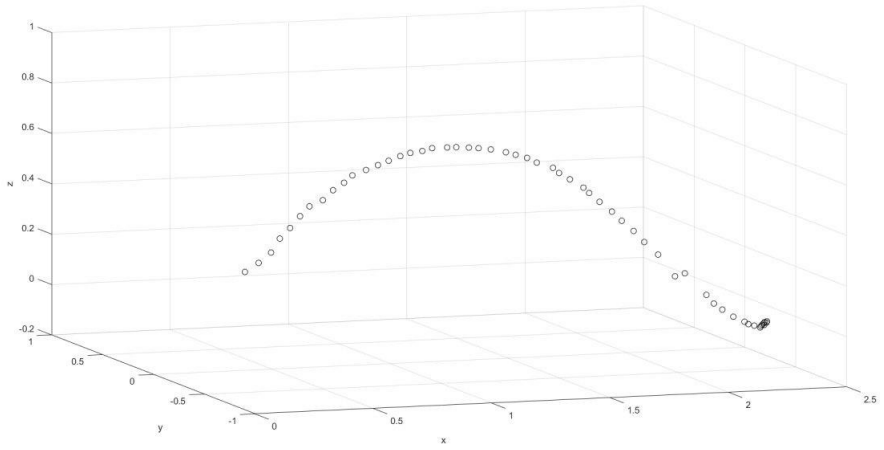


Figure 4.5 Raw measurements of thrown ball as result of the pipeline. 3D view.

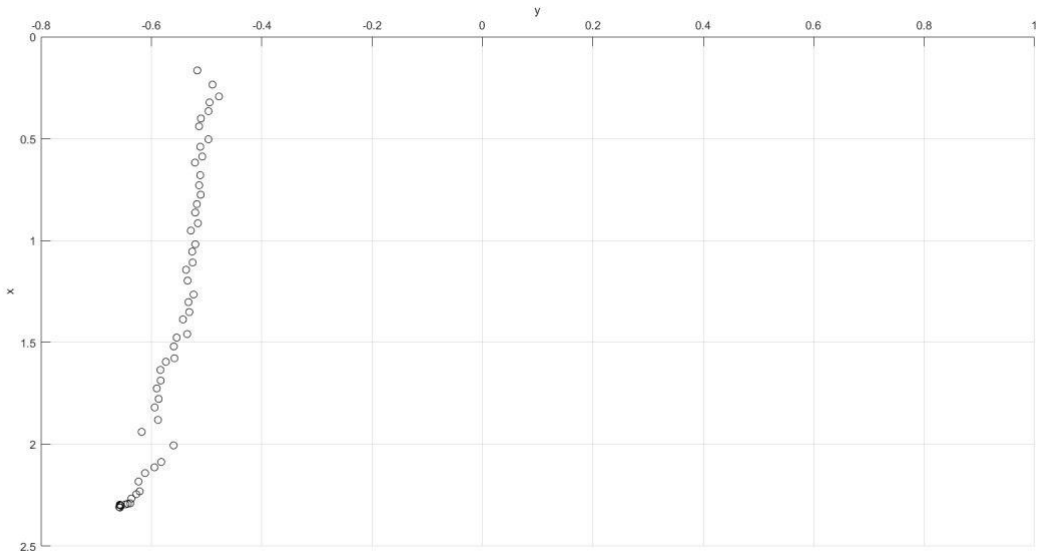


Figure 4.6 Raw measurements of thrown ball as result of the pipeline. XY plane view.

4.1.3 Constant Acceleration Kalman Filter

To work properly, the Kalman Filter must be implemented in order to properly fit the context for which its application is required.
For instance in localization problems it's common to have a nonlinear model of the system and so it's necessary to implement an Extended Kalman Filter whose matrices are computed as Jacobians of the model itself. [23]
Anyway the classical linear Kalman filters can be very useful in a big range of applications: for an object which is traveling in mid-air after a throw a *Constant Acceleration model* is the best choice since allows to keep into account the *gravity acceleration* constantly acting on the target. [24]
Assuming a generic Cartesian reference frame $[x, y, z]$, the state of the adopted filter is

$$[x, y, z, \dot{x}, \dot{y}, \dot{z}, \ddot{x}, \ddot{y}, \ddot{z}]^T \tag{4.2}$$

Since the acquisition/processing time is not fixed but has an average value of 16.4 ms with a jitter of 0.3 ms, the transition matrix $\mathbf{F}$ has to be considered as time-varying, so the notation $\Delta\mathbf{t}_k$ stands for the sample time at each acquisition loop iteration.
For this model the time-varying transition matrix is

$$\mathbf{F_k} = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 & \frac{1}{2}\Delta t^2 & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 & 0 & \frac{1}{2}\Delta t^2 & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t & 0 & 0 & \frac{1}{2}\Delta t^2 \\ 0 & 0 & 0 & 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.3}$$

while the measurement matrix does not changes over time

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.4}$$

By assuming that the measurement noise v_k is also normally distributed, with mean 0 and standard deviations σ_x,σ_y,σ_z, the matrix R can be set as follows:

$$\mathbf{R} = \begin{bmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_z^2 \end{bmatrix} \tag{4.5}$$

The described Kalman Filter has been implemented in the sensor processing node of the architecture by using the OpenCV Kalman module, a library of the framework which contains an object-oriented implementation of the filter.
The library hides the internal processes of the filter and presents an interface which allows the user to specify the matrices, read any kind of information (e.g. the Kalman Gains at each instant) and most importantly to choose when to perform the prediction and update steps within the data processing loop.
By using this implementation the filtering equations are hidden and the programmer/engineer's effort can be totally focused on the correct design of the model as well as in finding suitable covariance matrices.

4.1.4 Ball Trajectory Prediction

On the purpose of computing the predictions two similar modules have been implemented with different working modes: a module is for a better visual assessment of the results and computes a complete parabolic motion at each filter step, forwardly projected of 500 ms; the other module is the one adopted for control, and computes the predicted arrival time from the $x(t)$ expression by giving a fixed coordinate of the catch point.
Then the estimated time is imposed in the $y(t)$ and $z(t)$ equations and the catch point is computed within a time step.
A cubic virtual workspace has been defined around the manipulator for computing the catch point and most importantly for defining spatial bounds and avoiding to reach catch points which even if aren't outside robot's effective workspace are too far to be reached within a

small amount of time.
By defining t_c as the expression of the predicted catching time, the following equations are used to compute the coordinates of the catching point given a fixed value of the x axis component named x_c.

$$t_c = \frac{x_k - x_c}{v_{xk}} \tag{4.6}$$

After the computation of t_c, the other coordinates are computed as follows

$$y_c = y_k + v_{yk} t_c \tag{4.7}$$

$$z_c = z_k + v_{zk} t_c - \frac{1}{2} g t_c^2 \tag{4.8}$$

where k is the filter iteration.

The other important information needed for controlling robot's motion in the catch phase is the estimated orientation of the ball at the catch point, which in fact is the heading of its velocity vector computed at those coordinates.
In order to retrieve this information another catch point prediction is computed in the same way explained before, but the catch time is anticipated of a quite small arbitrary quantity, for instance 50 ms, so that two consecutive trajectory points are computed.
The resulting vector from the difference of the two points expressed in robot's frame divided by its module is the estimated orientation vector of ball's trajectory.

4.1.5 Design Choices on Communication

A brief confrontation on the communication architectures adopted for sensor-based robot control can be found in appendix E: the described analysis led to the following considerations.
If the Publish/Subscribe model works well under the scalability aspect, on the high-speed and time-constrained point of view is not the most affordable choice since implement a pub/sub communication outside the aforementioned frameworks is more complex than a simple client/server.
Moreover in applications with few components, like in the simplest case of a robot and a sensor, the advantages given by pub/sub are totally neglectable, given the reduced number of communicating workstations.
In such a simple network which already contains an important bottleneck like the working

frequency of the camera is not recommendable to introduce more uncertainty elements such as the latencies/delays which could derive by the usage of the ROS framework or any other application using the publish/subscribe architecture.
As further support to the adopted design choices is mandatory to remark that, for the specific application, the Franka Panda supports a ROS interface for its libraries, but is not intended for any kind of real-time behavior.
Even if the communication between camera and robot workstations is not properly real-time and anyway the size of data is too low to produce any kind of transmission delay, using a real-time control loop through *libfranka* side by side with a *franka_ros* node is not possible, so the choice of a client/server model turned out to be the best one for the purpose.

4.1.6 Catch Points Server

The ultimate goal of the sensing module is to transmit to the other workstation the informations extracted from the vision pipeline and processed in the estimation and prediction phases.
In order to minimize the computational effort of the machine voted to generate robot's motion, all and just the required informations for the control law must be sent, so that no more processing on flight data will be required.
In particular the following data must be stored into an array and sent to the other workstation:

- The ***predicted catch point*** which is expressed in its $[x, y, z]$ coordinates in space with respect to robot's base frame.
- The ***predicted catch time***, a number extracted from the projectile motion equations which gives a measure on the instant between the detection of the catch point and the instant at which the real ball will intersect the workspace at those coordinates.
- The ***predicted ball heading*** which are the coordinates of the unit vector representing the orientation of the velocity vector of the ball at the catch point. Also this information is expressed in robot's base link frame.
- The ***predicted ball velocities*** as terminal conditions for trajectory planning.

The implementation of this communication has been done by using UNIX socket libraries: a client/server model has been defined.

The server runs on the camera workstation and is executed on a separate thread, sending data at acquisition loop frequency of approximately $60Hz$.
In fact the server could not be implemented directly at the end of the pipeline since it would have been introducing undesired latency in the acquisition loop: instead by running on the detached thread it only sends the results of the vision and tracking pipeline to robot's workstation, where the correspondent client is also running on a dedicated thread.

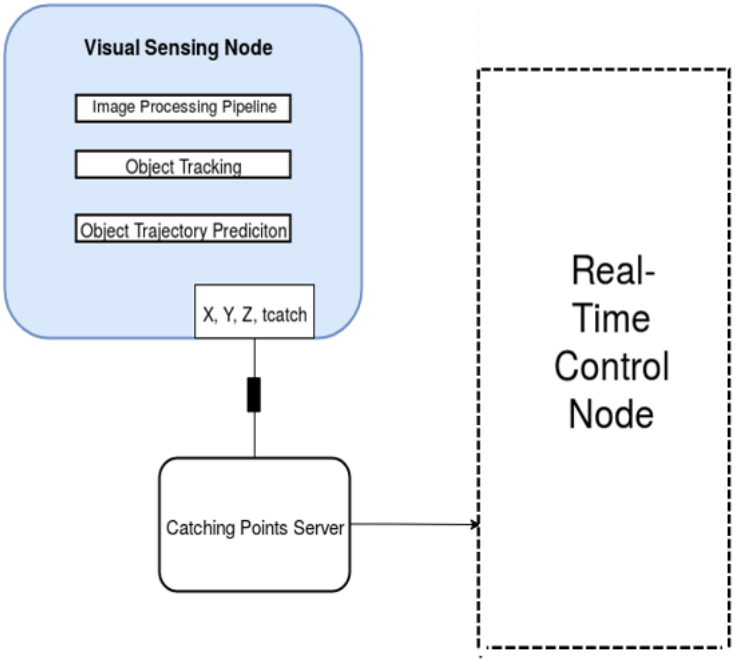


Figure 4.7 Scheme of the communication architecture.

4.2 Control and Trajectory Planning Module

The control module of the architecture has been designed in a *Master-Slave* fashion, a very common pattern in embedded systems and cyber-physical systems software design.
The rationale behind this choice is given by the fact that the control framework is centered on specific functions which allow to establish a real-time loop between the workstation and the robot controller.
The usage of those functions is mandatory for performing customized motion control and implementing complex motion-driving laws: they basically run a hard-realtime control loop which exploits a cabled Ethernet communication with the controller and sends commands

whose type is specified at the beginning.
Typically a control framework allows the following types of control:

- Cartesian Space Position commands
- Cartesian Space Velocity commands
- Joint Space Position commands
- Joint Space Velocity commands
- Joint Space Torque commands

The module has been conceived as a modular and thread-based block linked to the main control loop through its own time.
Since the real-time control loop has an exact duration of 1 ms, this time measure has been used to control the execution of all the other threads which run beside it in a concurrent way. This allowed to implement independent a basic synchronization mechanism as well as deadlines for the executions of the tasks which are bounded to run at multiple values of 1 ms, which in fact is the *heartbeat* of the system.
For each task a duration period in milliseconds can be specified: the implemented timing mechanism ensures that at each iteration, after all the instructions have been executed, the task waits until the end of the period and produces its output one instant before the deadline. This design choice increases the predictability in the execution of the architecture, since the system knows at what time the results of the computations will be available, and most importantly allows to stop the execution whenever a task requires more time than what specified in the period.

4.2.1 Inverse Kinematics Task

The necessity of having a task which computes the inverse kinematics is related to the specific requirements of the robot.
Early tests performed on robot's motion control showed that for tasks in which quite fast movements are required the robot controller does not allow to perform Cartesian control loops.
In particular the adopted motion trajectories violate the constraints imposed by the framework on Cartesian accelerations and cause an immediate stop of robot's motion.

The choice adopted to overcome this issue is to perform motion control loops directly in the joint space, in particular by sending velocity commands obtained through inverse kinematics algorithm.
Within this task the pseudo-inverse of the Jacobian matrix of the manipulator is computed at each instant through the technique of the *Singular Value Decomposition* or SVD [9].

4.2.2 Control and Trajectory Planning Task

For the kind of applications which require the meeting of strict timing constraints in which an action must be performed, some specific control techniques must be adopted, anyway in the described application a first design choice which allows to reduce efforts is the decoupling of orientation and position control, intended to be independent.

Orientation Control

The orientation control is not designed to be explicitly driven by the time constraint, anyway is conceived to be as fast as possible, since the orientation of the end-effector must be compliant with the trajectory of the ball at the reach of the catching point.
The data provided to the system by the client contain the coordinates of the unit vector representing the heading of the ball trajectory at the catch point, so the algorithm starts from this information.
The coordinates of the trajectory orientation are expressed in the base frame, and are used as control reference for driving the orientation of another unit vector, arbitrary chosen as the x component of the end-effector frame, also expressed in base link frame.
The choice of the axis of end-effector frame is due to the fact that is perpendicular to the catch plate, which sufficiently simplifies the problem.

Cartesian Velocity Control

In the implementation of this control part of the architecture the Cartesian space has been partitioned over the three axes of the base link frame: this leads to build three independent polynomial trajectories in space.
The solution of the problem requires the end-effector motion to meet a temporal constraint along with four spatial constraints, specified as initial Cartesian position and velocity and

final Cartesian position and velocity along each axis.

When the catching point is sampled and received by the client, the following system of equations is solved by using the *Householder* algorithm with pivoting both on rows and columns provided by the Eigen library, chosen because it's the more precise and for small matrices has a slight speed boost.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & t_f & t_f^2 & t_f^3 \\ 0 & 1 & 2t_f & 3t_f^2 \end{bmatrix} \begin{bmatrix} \mathbf{a}_0 \\ \mathbf{a}_1 \\ \mathbf{a}_2 \\ \mathbf{a}_3 \end{bmatrix} = \begin{bmatrix} \mathbf{x}_i \\ \mathbf{x}_f \\ \mathbf{x}_i \\ \dot{\mathbf{x}}_f \end{bmatrix} \tag{4.9}$$

In the system of equations t_f is intended to be the predicted catch time, the vectors $\mathbf{a_i}$ are the coefficients of the polynomials for the three axes and the known terms column contains the vectors of the initial and final position and velocities.

Given the coefficients computed through the solution algorithm, the following polynomial trajectory in time can be defined:

$$\mathbf{p}(t) = \mathbf{a_0} + \mathbf{a_1}t + \mathbf{a_2}t^2 + \mathbf{a_3}t^3 \tag{4.10}$$

This represents the Cartesian position profile of motion which the end-effector must follow. Since the control is performed through velocity commands, the feedforward signal which will be part of the control law is the time derivative of the third order polynomial:

$$\dot{\mathbf{p}}(t) = \mathbf{a_1} + 2\mathbf{a_2}t + 3\mathbf{a_3}t^2 \tag{4.11}$$

Moreover for time-constrained motions a good practice is to assess the Cartesian space accelerations, to evaluate if the robot is working too close to the boundaries defined by the vendor.
The accelerations along the three axes are simply computed as the second derivative of the polynomial:

$$\ddot{\mathbf{p}}(t) = 2\mathbf{a_2} + 6\mathbf{a_3}t \tag{4.12}$$

4.2.3 Ball Data Client

This task is asynchronous with respect to the execution of the control loop, so its behavior is not driven by the heartbeat of the system.
It's a client which at the beginning of its execution sends a request to the server running on the camera workstation, in order to receive the aforementioned data which completely describe the flight of the ball for control purposes.
The implementation is done by adopting the UNIX socket libraries just like for the server.
This task runs independently from the others and receives the data from the other workstation presenting them to the rest of the software modules through an interface; in this way new data are planned to be acquired from the other tasks every 16 ms, which is the nominal duration of an iteration of the camera loop.

4.2.4 Visualization and Data Storing Task

Another asynchronous task involved in the control architecture is not mandatory for the correct execution of the action, but it becomes useful for storing data and visualizing the results.
As any other task it can access all the important data involved in robot control and target tracking: for instance it can read the robot state whenever it's updated or can store the flight data received by the client.
The described software block provides two visualization services:

- It runs a ***ROS*** node which displays on *RViz* frames related to important elements and events, for instance the base link frame or the end-effector frame of the robot.
- It stores data in **.csv** format for plotting.

In fact this is the only task of the architecture which requires tools of the ROS framework, and due to the integration of the FCI all the robot informations can be easily visualized even while the robot is moving.

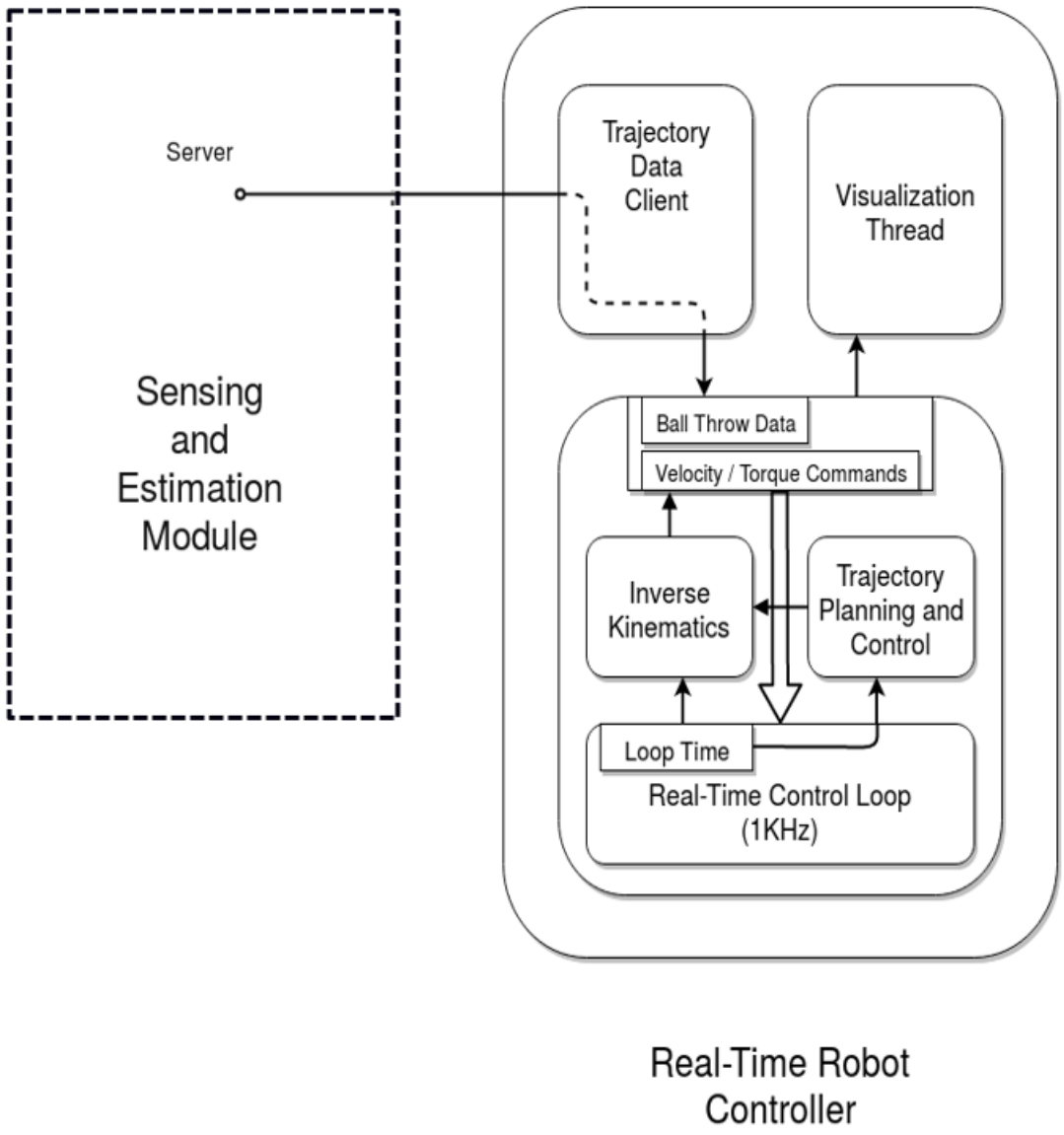


Figure 4.8 Scheme of the control module of the architecture

Chapter 5

Experimental Setup

Along this section a detailed description of the hardware and software technologies and devices used for building the experimental setup of the non-prehensile ball catching problem. Particular attention is given to technical details which are significant in control applications, like timing characteristics or software technologies adopted for handling the devices.
The robot and the camera will be analyzed by taking into account the hardware technologies and specifications as well as their software interfaces for building applications.
Later on the specifications of the two computing systems will be presented as well as details on the operating systems and real-time aspects, and a complete scheme of the setup will be provided.
A discussion on the design and prototyping of an articulated support for the camera will be carried on by taking into account the mechanical design software and the 3D printing technology.
Finally the extrinsic parameters calibration of the camera will be addressed and will be outlined its importance in a robot control application.

5.1 Franka Emika Panda Manipulator

The robot adopted for performing the non-prehensile ball catching task is the *Panda* manipulator by German manufacturer *Franka Emika*®.
The robot is intended to be quite versatile, and to serve mass production as well as low volume high-mix production, which is a modern and flexible production layout for small enterprises.
It's important to remark the concept of *Soft-robot performance*, which is achieved by a refined low-level control and allows production processes that require precision, force application and sensitive handling [25].
Panda can work in tightly constrained environments due to its capability of moving the end-effector very close to its base and its own joints.

5.1.1 Arm Technical Details

The following bullets report the technical details related to the robot arm.

- *Degrees of Freedom:* 7
- *Payload:* 3 kg
- *Force/Torque Sensing:* link-side torque sensors in all 7 axes
- *Joint Position Limits:*
  Joints 1, 3, 5, 7: [-166 deg / 166 deg]
  Joint 2: [-101 deg / 101 deg]
  Joint 4: [-176 deg / -4 deg]
  Joint 6: [-1 deg / -215 deg]

The robot is subject to the following constraints while performing *Joint Space trajectories*:

- Necessary Conditions

$$q_{min} < q_c < q_{max} \tag{5.1}$$

$$-\dot{q}_{max} < \dot{q}_c < \dot{q}_{max} \tag{5.2}$$

$$-\ddot{q}_{max} < \ddot{q}_c < \ddot{q}_{max} \tag{5.3}$$

$$-\dddot{q}_{max} < \dddot{q}_c < \dddot{q}_{max} \tag{5.4}$$

- Recommended Conditions

$$-\tau_{jmax} < \tau_{jd} < \tau_{jmax} \tag{5.5}$$

$$-\dot{\tau}_{jmax} < \dot{\tau}_{jd} < \dot{\tau}_{jmax} \tag{5.6}$$

At the beginning of the trajectory the following conditions should be satisfied:

$$q = q_c \tag{5.7}$$

$$\dot{q}_c = 0 \tag{5.8}$$

$$\ddot{q}_c = 0 \tag{5.9}$$

At the end of the trajectory the following conditions should be satisfied:

$$\dot{q}_c = 0 \tag{5.10}$$

$$\ddot{q}_c = 0 \tag{5.11}$$

Furthermore, the following constraints must be kept into account while performing *Cartesian Space trajectories*, where the symbol p stands for Cartesian position and T is a valid transformation matrix:

- Necessary Conditions

$$-\dot{p}_{max} < \dot{p}_c < \dot{p}_{max} \tag{5.12}$$

$$-\ddot{p}_{max} < \ddot{p}_c < \ddot{p}_{max} \tag{5.13}$$

$$-\dddot{p}_{max} < \dddot{p}_c < \dddot{p}_{max} \tag{5.14}$$

Conditions derived from Inverse Kinematics:

$$q_{min} < q_c < q_{max} \tag{5.15}$$

$$-\dot{q}_{max} < \dot{q}_c < \dot{q}_{max} \tag{5.16}$$

$$-\ddot{q}_{max} < \ddot{q}_c < \ddot{q}_{max} \tag{5.17}$$

- Recommended Conditions

$$-\tau_{jmax} < \tau_{jd} < \tau_{jmax} \tag{5.18}$$

$$-\dot{\tau_{jmax}} < \dot{\tau_{jd}} < \dot{\tau_{jmax}} \tag{5.19}$$

At the beginning of the trajectory the following conditions should be satisfied:

$$^{O}T_{EE} = ^{O}T_{EEc} \tag{5.20}$$

$$\dot{p_c} = 0 \tag{5.21}$$

$$\ddot{p_c} = 0 \tag{5.22}$$

At the end of the trajectory the following conditions should be satisfied:

$$\dot{p_c} = 0 \tag{5.23}$$

$$\ddot{p_c} = 0 \tag{5.24}$$

The following tables report all the constants presented in the constraints:

Table 5.1 Joint Space Limits

Name	**Translation**	**Rotation**	**Elbow**
$\dot{p_{max}}$	1.7000 m/s	2.5000 rad/s	2.1750 rad/s
$\ddot{p_{max}}$	13.0000 m/s^2	25.0000 rad/s^2	10.0000 rad/s^2
$\dddot{p_{max}}$	1.7000 m/s	2.5000 rad/s	2.1750 rad/s

Table 5.2 Cartesian Space Limits

Name	**Joint 1**	**Joint 2**	**Joint 3**	**Joint 4**	**Joint 5**	**Joint 6**	**Joint 7**	**Unit**
q_{max}	2.8973	1.7628	2.8973	-0.0698	2.8973	3.7525	2.8973	rad
q_{min}	-2.8973	-1.7628	-2.8973	-3.0718	-2.8973	-0.0175	-2.8973	rad
$\dot{q_{max}}$	2.1750	2.1750	2.1750	2.1750	2.6100	2.6100	2.6100	rad/s
$\ddot{q_{max}}$	15	7.5	10	12.5	15	20	20	rad/s^2
$\dddot{q_{max}}$	7500	3750	5000	6250	7500	10000	10000	rad/s^3
tau_{jmax}	87	87	87	87	12	12	12	Nm
$\det tau_{jmax}$	1000	1000	1000	1000	1000	1000	1000	Nm/s

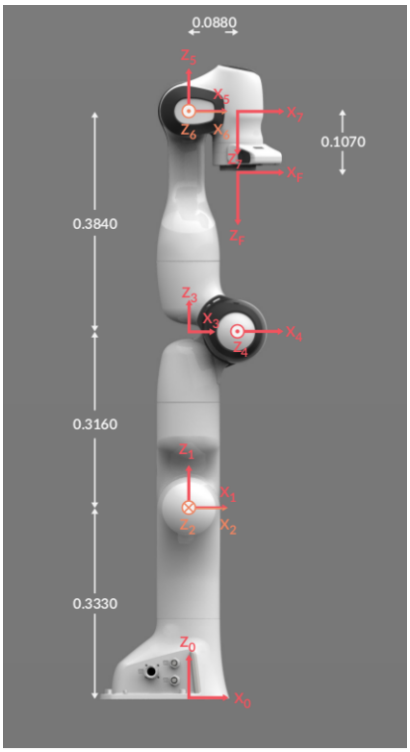


Figure 5.1 The Panda manipulator's kinematic chain.

Table 5.3 Denavitt Hartemberg Parameters

Joint	***a* (m)**	***d* (m)**	**α (rad)**	**θ (rad)**
Joint 1	0	0.333	0	θ_1
Joint 2	0	0	-π/2	θ_2
Joint 3	0	0.316	π/2	θ_3
Joint 4	0.0825	0	π/2	θ_4
Joint 5	-0.0825	0.384	-π/2	θ_5
Joint 6	0	0	π/2	θ_6
Joint 7	0.088	0	π/2	θ_7
Flange	0	0.107	0	0

5.1.2 Controller and Software Framework

A real-time controller is provided along the robotic arm, providing an interface between user's workstation and the Panda itself.
After the network configurations, the controller allows the user to interface with the robot in two ways: through an high-level browser-based GUI for programming simple tasks by connecting building blocks and a more complex framework called *Franka Control Interface (FCI)*.

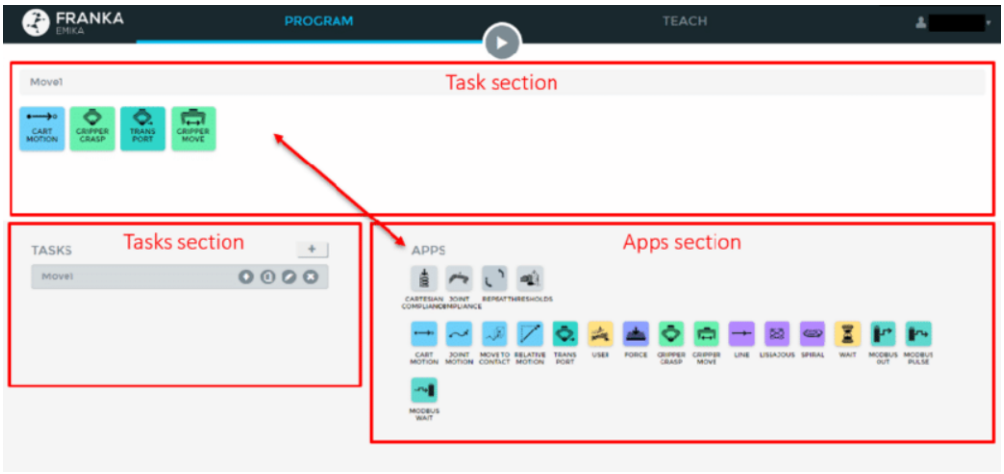


Figure 5.2 The Franka Desk Interface, where the user can sequence different blocks for programming simple tasks.

Franka Control Interface is a software framework which runs on the controller itself: the user can access to its functionalities by programming with the *libfranka* library, which is the C++ implementation of the client side of FCI [26].
The *libfranka* has been widely used during the project, because is capable to handle the network communication with the controller and provides interfaces to execute non-realtime commands to control the Hand and configure Arm parameters, execute realtime commands to run 1 kHz control loops and read the robot state to get sensor data at 1 kHz.
Moreover the user can access the model library to compute kinematic and dynamic parameters.
The usage of *libfranka* allows a noticeable flexibility and modularity in motion programming and it gives the capability of designing different control laws, based on position, velocity, force or even hybrid laws.
To facilitate the control of the robot under non-ideal network connections, *libfranka* includes

signal processing functions that will modify the user-commanded values to make them conform with the limits of the interface.
Two optional functions are included in all real-time control loops: a first-order low-pass filter to smooth the user-commanded signal and a rate limiter, that saturates the time derivatives of the user-commanded values.

When performing real-time control tasks it must be kept into account that the Panda Gripper is not intended to work under real-time conditions, so is handled by the FCI as a non-realtime component, and can perform motions of opening/closure with limited speed and reactivity: this is one of the motivations which have driven the project towards a non-prehensile object catching as benchmark, in order to free the task from the usage of the gripper.

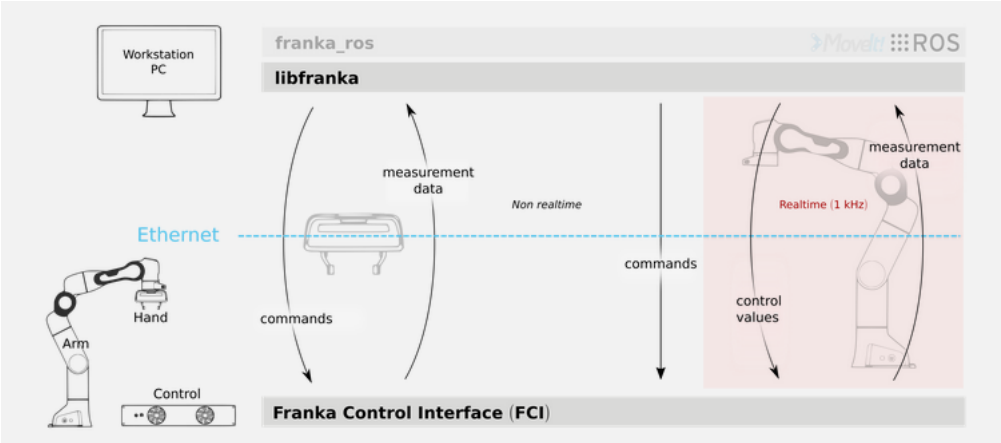


Figure 5.3 The schematic overview of the robot control architecture: the difference between the non-realtime and realtime layers is underlined, as well as the Client-Server UDP model of communication between *FCI* and *libfranka*.

5.2 Stereolabs ZED Camera

The *ZED* camera manufactured by *Stereolabs* is an RGB-D camera which adopts Stereo Vision for measuring depth.
According to the manufacturer ZED is the world's fastest depth camera: it can capture $1080p$ HD video at 30 fps or WVGA at 100 fps with a clear depth image [27].
Moreover it ensures a wide field of view with $16:9$ native sensors and ultra sharp 6 element all glass lenses that allow 110 wide-angle video and depth.

Figure 5.4 The ZED Mini (above) and the ZED (below) cameras: the small version is optimized for mobile applications and provides an IMU sensor.

The following features are supported by ZED technologies:

- **Stereo Capture**: due to its dual lenses it captures high-definition 3D video with a wide field of view and outputs two synchronized left and right video streams in side-by-side format on USB 3.0.

- **Depth Perception**: the camera can determine distances between objects and has 3D perception: using stereo vision, the ZED is the first universal depth sensor which can capture 3D informations at longer ranges, up to 20 m (Extended to 30 m in ULTRA mode).
  The camera can work indoors and outdoors, contrary to active sensors such as structured-light or time of flight [27].

- **Positional Tracking**: using computer vision and stereo SLAM technology, the ZED can also compute its position and orientation in space, offering full 6 DOF positional tracking.
  In particular, considering mobile robotics applications, the user can determine robot's position, orientation, and velocity and navigate autonomously to the coordinates of your choice on a map.

5.2.1 Technical Details

The ZED camera allows the user to grab depth frames at the same resolution selected for the RGB video, encoded in 32-bits format.
The depth range goes from 0.5 m to 20 m in standard mode, with a field of view of 90°(H) x 60°(V) x 110°(D).
The optical sensors have 4M pixels each one, and the pixel width is 2-micron with native 16:9 format; the shutter is an Electronic Synchronized Rolling Shutter.
The following table shows the video recording capabilities of the camera:

Table 5.4 Video Acquisition Characteristics

Video Mode	**Frames per second**	**Output Resolution**
2.2K	15	4416x1242
1080p	30	3840x1080
720p	60	2560x720
WVGA	100	1344x376

On the software side the ZED camera has its own SDK provided by the vendor, named *ZED SDK*, which provides several C++ and Python classes and functions to adjust offline and online parameters, grab frames and build image processing pipelines by using the Stereolabs image format.
The SDK also comprehends ready-to-use tools for recording and saving videos, export them in different formats and of course calibrating the intrinsic parameters of the camera.
To work properly, the ZED SDK requires the support of CUDA libraries (version 9.1 or higher, with respect to the version of the SDK): this is because to exploit the working speed of the ZED camera a Nvidia GPU is required on the connected workstation.

In fact the ZED needs the GPU to perform depth processing in an optimized way, and it's still usable without it but with the limitations of the UVC (Universal Video Camera) format. Moreover the ZED SDK provides interfaces and plugins for several modern software frameworks and technologies such as Matlab integrations, ROS (and recently ROS2) modules, an OpenCV bridge for converting acquired images from the Stereolabs format to the well-known *CV Matrix*.
Apart from Robotics the camera has a wide range of applications in Augmented and Virtual reality technologies, due to its third-party support for Unity and Unreal Engine.
Anyway regarding real-time robotic applications the key feature of the ZED camera is its speed in video and depth acquisition and processing, so that the programmer can deal the trade-off between speed and accuracy due to the timing constraints of the selected applications without excessively penalize the quality of the measurements.

5.2.2 Software Technologies

For the design and implementation of the computer vision pipeline, several software technologies and libraries have been adopted in order to achieve a fast processing as well as a good exploitation of the hardware resources at disposal.

- The ***OpenCV*** package is the most known open-source collection of libraries adopted world-wide for building computer vision applications.
  The OpenCV project was initially an Intel Research initiative to advance CPU-intensive applications, part of a series of projects including real-time ray tracing and 3D display walls. In the early days of OpenCV, the main goal of the project was to advance vision research by providing not only open but also optimized code for basic vision infrastructure, so that programmers wouldn't have to start from the beginning anymore and could have efficient building blocks already implemented and tested.
  The framework supports C++ and Python languages, and provides optimized libraries for performing 2D and 3D feature recognition, egomotion estimation, facial and gesture recognition, mobile robotics, object segmentation and motion tracking.
  OpenCV also includes a statistical machine learning library that provides decision tree learning, Naive Bayes classifier, Artificial Neural Networks and many other algorithms.[28]

- The ***ZED SDK*** provided by Stereolabs is a framework which provides a complete abstraction of the device in C++ and Python, giving access to all its features.
  Moreover it provides full support for integration with all the most common robotics, computer vision, machine learning and data analysis frameworks such as ROS, Tensorflow, Matlab and most importantly OpenCV.
  In fact the ZED SDK has its own proprietary data structure in which all captures are stored, so to perform further processing with the OpenCV features the framework provides a bridge capable of switching between the Stereolabs Matrix format and the OpenCV Matrix format.
  The framework allows to define a well-structured image processing pipeline suitable for several applications like SLAM, object tracking, feature recognition and so on: ad-hoc functions allow the user to set a large spectrum of parameters before the acquisition loop like resolution, optical frame of the camera for positional tracking, depth resolution or to load a pre-recorded video instead of acquiring frames.
  The ZED SDK has its own video format named *.svo*, which stands for *Stereolabs Video Object* and allows to store not only video informations but even the complete depth map for off-line depth sensing and processing.

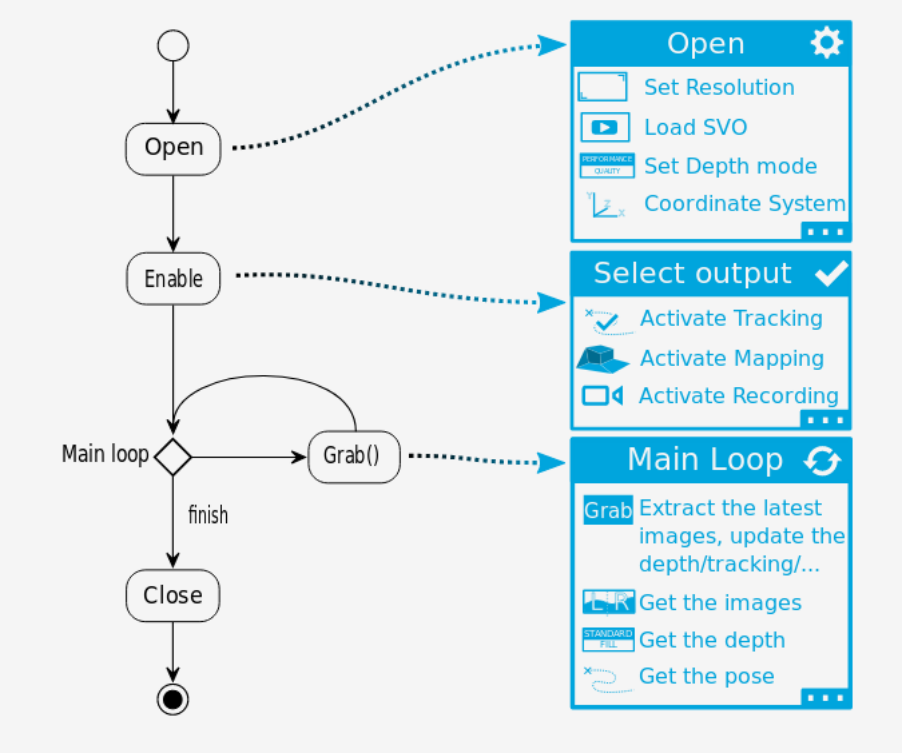


Figure 5.5 ZED SDK image processing pipeline.

- ***CUDA*** (Compute Unified Device Architecture) is a parallel computing platform and application programming interface (API) model created by Nvidia.
  It allows software developers and software engineers to use a CUDA-enabled graphics processing unit (GPU) for general purpose processing, an approach termed GPGPU (General-Purpose computing on Graphics Processing Units).
  The CUDA platform is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements, for the execution of compute kernels.
  A typical processing flow can be divided in 4 steps:

  1. Data are copied from main memory to GPU memory
  2. CPU initiates the GPU compute kernel
  3. GPU's CUDA cores execute the kernel in parallel
  4. The resulting data are then copied from GPU memory to main memory

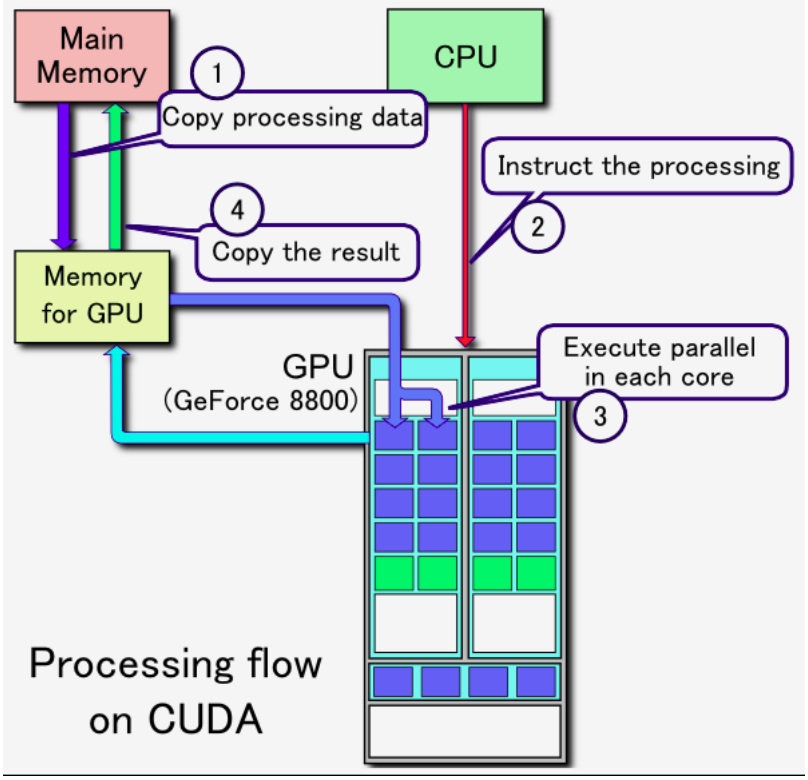


Figure 5.6 CUDA execution pipeline.

Due to this execution paradigm, the usage of CUDA-based algorithms is recommended when a large amount of data must be processed under timing constraints or in case of high performance graphical applications, for instance involving video-game physics engines which require intensive processing in real-time or even for reducing the training time of machine learning models.
In fact when processing relatively small quantities of data the achievable gain in terms of performance is minimum due to the overhead time in transferring operations from CPU to GPU.
To use the ZED SDK a CUDA-enabled GPU is anyway required, since stereo image processing is performed by taking advantage of GPU parallel computing: the ZED camera is anyway adoptable without disposing of a graphic processor but in that case the user can only access it as UVC (USB Video Class) camera, so is not possible to use stereo vision features like depth sensing.

5.3 Camera Support Device

In order to achieve a stable positioning of the ZED camera for the experimental setup a camera support device has been designed.
In tasks involving robots and vision systems is mandatory to have a solid support for the sensors in order to achieve a good coverage of the area, which in case of catching problems could be quite wide since the thrown object must be tracked for the entire trajectory.
Moreover the support must be stable and rigid enough to not degrade the reliability of the extrinsic parameters obtained with calibration.
In case of articulated supports another parameter to keep into account is the robustness of the links to undesired motions which, again, can compromise the accuracy of the calibration along time until another calibration is necessary.
The designed and implemented support is composed by three parts: *base, joint and slot*, all of them have been modeled with *Solidworks* and then printed with a WASP 3000 printer by using red PLA(Poli-Lattic Acid) wires.
The detailed list of components is described in appendix B.

5.4 Modular end-effector for ball-catching

In order to perform experiments with the robot, the gripper has been replaced with a modular 3D printed end-effector.
A full description of the components can be found in appendix B.

5.5 Camera Calibration

An important and mandatory step in any experimental setup which involves robots(manipulators, humanoids, mobile etc...) and one or more visual sensors is the *Extrinsic Parameters Calibration*, which aims to retrieve a geometrical transformation between the two systems of coordinates of the robot and the sensor(s).
The systems of coordinates are the well-known Cartesian Coordinate Systems, expressed as triplets of orthogonal axes $< x, y, z >$.
To understand the motivation of this procedure is sufficient to think about how visual measurements are retrieved: a single RGB-D camera acquires visual and spatial informations

with its sensors and sends raw data to be processed by its own firmware and then by software on the workstation.

The information encoded in data represents, for instance, the position of an object in space, but relatively to the camera itself: so a camera has its own system of coordinates in which the measurements are expressed.
A modern stereo-based camera like ZED has more than one reference frame, but it's common that depth measurements are expressed with respect to the *Left Optical Frame*: performing an *Extrinsic Parameters Calibration* means to express this frame in an arbitrarily chosen frame useful for the specific application.
In a manipulation task like the presented one, a smart choice for an absolute reference frame is the *Base Link Frame* of the manipulator, since the position of any other link and of the end effector is expressed with respect to it by the control software of the robot.

When a geometrical transformation is retrieved between camera frame and robot base frame it means that any perceptual data acquired by the camera can be immediately expressed in the system of coordinates of the manipulator.
The steps followed during the calibration process are described in appendix C.

Chapter 6

Experiments and Benchmarking

Along the following chapter it can be found the complete description of the experimental phases which have been performed along the development of the project.
The evaluation of the *ZED SDK* working modes and operating frequencies will be analyzed under control aspects, by describing what kind of trade-off can be applied and what are the most important parameters to be tuned, both online and offline.
Two methods of camera latency assessment are presented, by remarking the importance of experiments for validation of the results.
It follows a brief analysis on communication latencies derived by connection of two workstations, with motivations of the design choices adopted for addressing the problem and a description of possible issues in digital control systems design.
A technical description of timing constraints and possible issues related to delays in robot control is given, mostly relying on manufacturer's datasheets of the manipulator and controller's real-time requirements.
Finally a method for assessing the overall delay which affects a distributed control system based on vision sensors is described and quantitative results are mentioned.
The non-prehensile catching control problem is presented as valuable benchmark for assessing real-time characteristics as well as limitation of the control architecture.
The experimental results of the ball catching task are provided with a detailed description of the experiments and their outcomes.

6.1 Timing Analysis

6.1.1 Camera Acquisition Frequency Evaluation

The first step to be taken into account in the evaluation of a sensor-based control systems is certainly the frequency (or range of frequencies) at which the sensors can work and with which accuracy.
Since the sensor measurements are a fundamental part of a control system, it's mandatory to point out the bottleneck of the sensing system and define a trade-off between speed and accuracy of measurements.
The ZED camera allows the user to use several different resolutions, and each one corresponds to a maximum allowed frequency of frames: for a real-time control application in which an object moving quite fast is involved is discouraged to use less than 60 fps, anyway it always depends on how fast the object is moving and on how many samples does the user need to compute an efficient response of the robot.
On the other hand the maximum working frequency of the ZED camera is 100 fps, but is achievable only at its minimum available resolution which is the standard VGA resolution.
It's important to understand that in a control application which requires detection and tracking of objects the resolution is a critical parameter, since too small or too distant objects cannot be detected at all if the size is too low, or anyway the noise would not allow a clear and stable tracking.
Moreover in an RBG-D camera depth measurements must be taken into account, and their accuracy is again related to timing performances: the ZED camera SDK gives access to depth informations with different quality policies, which are *PERFORMANCE*, *MEDIUM*, *QUALITY*, *ULTRA*.
Different depth sensing settings relate low detail level with less computations and vice versa, so an increase in accuracy of measurements (e.g. using ULTRA mode) will bring an increase of computations time in the algorithm.

- The ULTRA mode offers the highest depth range and better preserves Z-accuracy along the sensing range.
- The QUALITY and MEDIUM modes are a compromise between quality and speed.
- The PERFORMANCE mode disables some filtering operations and operates at a lower resolution to improve performance and frame rate.

The vendor recommends using the ULTRA mode for both desktop and embedded applications unless in case of limited resources.
For tracking a moving object is also important to guarantee that no depth map occlusions occur or affect the computations for the entire trajectory.
Occlusions are points (or regions) of the depth map in which for some reason camera's algorithms cannot compute depth values, so they can be treated as 'blind points' of depth perception: the ZED SDK provides a feature which is called *FILL MODE* which fills this gaps when required by the application by using interpolation algorithms.

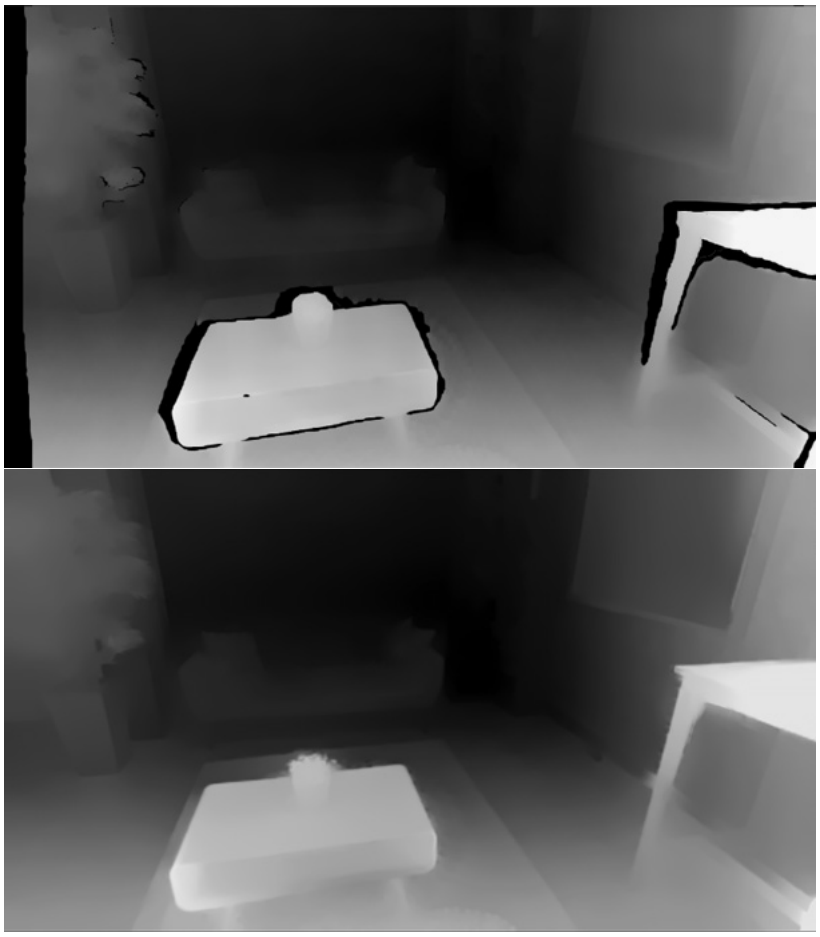


Figure 6.1 Depth mode STANDARD (top) and FILL (bottom).

The following table reports the ranges of values of the expected latencies according to the vendor website [27].

Table 6.1 Table of expected latencies

FPS	Expected Latency
100	20 ms - 30 ms
60	30 ms - 45 ms
30	66 ms - 100 ms
15	133 ms - 200 ms

The tests described in the Appendix E are voted to assess the correct amount of latency.

6.1.2 Architecture Delay Assessment

An early experiment performed after the implementation of the architecture is an analysis of the overall latency during an execution step, i.e. from the instant in which an image is acquired to the time in which the control loop is aware of the processed information and sends a command to the robot.
For this assessment a simple cyclic test was carried on by using exactly the same software modules which take part to the ball catching task, where the starting and end point is the workstation which sends commands to the controller.
In particular from the real-time control loop which runs at 1 KHz frequency a command with a specified period is sent to the parallel port with the goal of blinking an LED and the loop time is stored.
The device is placed in a suitable position in order to be seen from the camera, so through the vision pipeline the blinking is registered and isolated in the binary image.
The binary brightness level of a correspondent fixed pixel is then observed and basically used as a flag which signals the blinking; then the flag is stored in the first cell of an array of the same size used for sending data to the client on the robot workstation.
It must be said that since some computations like the estimation of the blob centroid and the 3D pose conversion are performed only when the object is detected, the ball must be kept in sight during the experiment to avoid any kind of confound on the latency values.
As last step of the test, when the raised flag is detected in the message by the main loop, the time is sampled again and the difference, which corresponds to the measured latency of the architecture, is computed as

$$t_d = T_f - T_s \tag{6.1}$$

By running the test several time an average value has been computed as mean of the values in the following table.

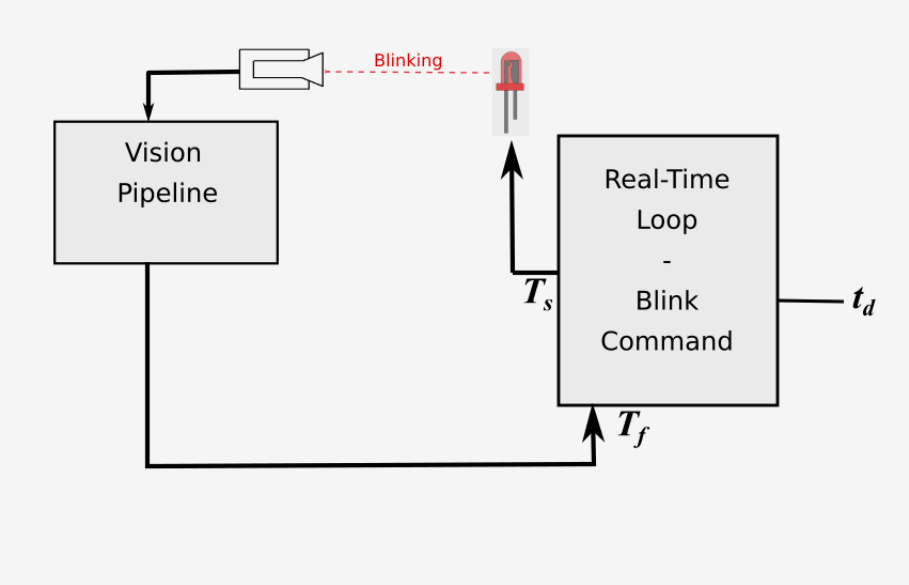


Figure 6.2 LED test scheme for architecture latency.

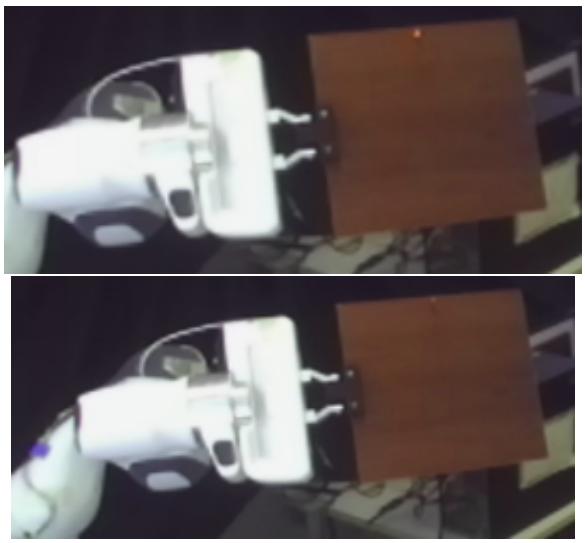


Figure 6.3 ON and OFF states of the LED during the complete test. The robot has been used as support for getting the device close to the camera.

6.2 Catch Point Estimation

An important experimental part during the implementation and testing of the architecture has been voted to assess the correctness and accuracy of the catch point estimate.
It's very important to analyze every single aspect of the architecture in an isolated way before putting all together, in order to understand eventual problems and know how any part will behave.
The first part to be analyzed is the predictive part of the perceptual module of the architecture, whose structures (Kalman filter, predictor) have been simulated and implemented within a continuous improvement process by using real data from the flight of the ball.
Then the modules have been tested in real-time by throwing the ball in the recorded workspace.
The main goal of this analysis is to tune the parameters of the module in order to provide the best estimate possible with the amount of samples at disposal.
In particular the most challenging part of this analysis was to understand if a feasible catch point was computable within the ascending part of the parabolic motion, possibly before the peak, otherwise the time remained to the robot for performing the motion would not be enough.

6.2.1 Preliminary Assumptions

To perform the analysis and drive further refinements of the architecture some initial assumptions have been done regarding the motion of the ball and are constrained to the positions of robot and camera, fixed in order to preserve the rigid transformation required for expressing ball motion data in robot's frame.

- The ball is supposed to be thrown in front of the robot, at approximately 2.2 m on the x axis of its base link frame.
  This choice is conceived to help the camera to better recognize the ball, since at more than 2.8 m the perceived blob region becomes too small to be kept from the erode/dilate sequence.
- In order to increase dexterity during motion and even to comply with the surrounding environment, the initial position of the throw has an offset on the left with respect to the center of robot's base link.

This because throwing the ball towards the body of the manipulator would not allow any damping of the motion without having self-collision of its links, which causes the immediate stop of its motion due to internal safety protocols.

- The throw must be done avoiding to impress too much initial speed to the ball, so that the performed motion resembles a projectile trajectory since from the first samples: this helps to improve the estimation of the ball motion.
- The trajectory must not go beyond 1.5 m on the vertical axis due to the configuration of the experimental setup and issues related to depth sensing: since there are windows behind the throwing zone which are in the field of view of the camera and the ball passes between the camera and them, a huge distortion can occur due to the external light which can completely corrupt the depth measurements.

6.2.2 Kalman Filter Tuning

Efforts have been spent for tuning the Kalman filter in order to achieve a good estimate of the ball position and velocity in space.
As described along the previous section, the Kalman filter is a recursive algorithm based on statistics and linear systems theory which is broadly used in control applications: in this case it's used to refine the position measurements retrieved by the vision pipeline and to provide an estimation of velocity and acceleration of the ball.
The initial state is not known a priori, even because the parabolic motion of the ball effectively starts after the throw.
Apart from the acceleration components, a common practice is to choose an initial position which is near to the launching point used during experiments, while for the velocities the initialization strongly depends on the reference coordinates system.
In experimental sessions this initial state has been used according with robot's reference frame:

$$\hat{\mathbf{x}}_{0|0} = \begin{bmatrix} 2.2 & -0.6 & 0.1 & -3.5 & 0 & 3.5 & 0 & 0 & -9.81 \end{bmatrix}^T \tag{6.2}$$

Since the initial state is not the real one but only a conceivable guess, the *Error Covariance matrix* is initialized as follows:

$$\mathbf{P}_{0|0} = \begin{bmatrix} 100 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 100 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 100 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 100 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 100 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 100 \end{bmatrix} \tag{6.3}$$

The tuning of the Kalman filter is performed through its matrices **P**, **Q** and **R**, which represent the statistical model of the uncertainties and noise acting on the system.
The elements of the matrices have been initially tuned and tested on a version of the filter implemented in *Matlab*, with a dataset obtained by corrupting with noise an ideal projectile motion and then with real flight data acquired from the camera.
The generated dataset allows to perform simulations on worst-case conditions like noise values higher than those encountered in real measurements and gives the capability to increase or reduce the amount of samples to assess how many of them are needed for complete convergence of the state of the filter.

Simulated Trajectory

The very first analysis has been performed on a simulation of the parabolic motion of the ball, by generating a pure trajectory through the projectile motion equations in space and corrupting it with Gaussian noise with $\mu = 0$ and $\sigma = 0.1$.
By observing the raw data to be processed, it's clear that this noise configuration is already good for tuning the parameters of the filter, since the real measurements acquired with ZED are less noisy.
An advantage of using a simulation is that allows to perform an assessment of how many steps are required to reach a good estimation of the values from the measurements by observing by the state covariance matrix **P** at the end of the iterative process.
The simulations have been performed on 20, 60, 100 and 150 samples in order to understand

how much the amount of measurements impacts on the performance of the filter, while the variance on measurements has been kept fixed.
A first result given by the simulations is that, as expected, increasing the samples amount leads to a better convergence of the filter's state to the effective position, velocity and acceleration of the ball: in particular the covariance matrix, which starts from a diagonal shape with all values set to 100, assumes uniform values related to the three estimated quantities, independently from the axis.

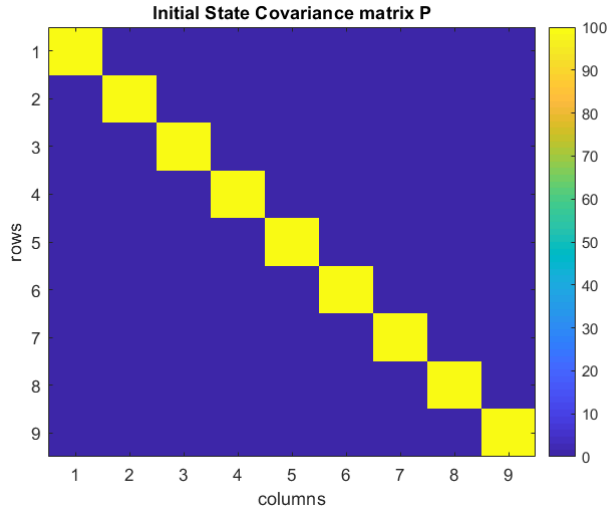


Figure 6.4 Error covariance matrix at initialization of the filter.

A measure of the convergence of the Kalman filter is given by the values of the matrix after the last iteration: in any case they are expected to reduce if the filter works properly, but the results of the tests have confirmed that more samples, so more iterations of the algorithm, lead to lower values of the parameters.

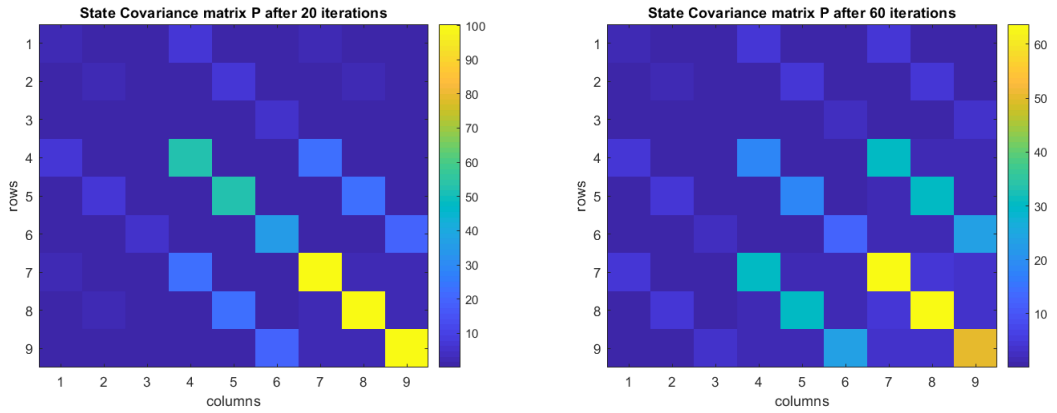


Figure 6.5 Error covariance matrix P after 20 and 60 iterations: the first case is compatible with the addressed problem, while the second resembles a complete trajectory caught on 60 fps camera. In both cases the amount of steps is enough to have a good estimate of the position and acceptable values for velocities, but the acceleration are still very uncertain.

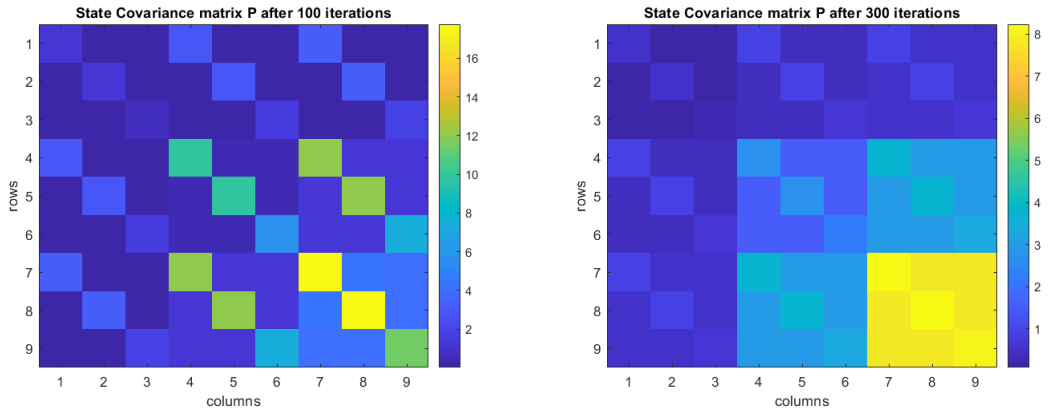


Figure 6.6 Error covariance matrix P after 100 and 300 iterations: it's clear to see that with a complete trajectory acquisition with an high performance camera (100 fps) or an industrial camera (up to 1000 fps) even the estimates of acceleration reach very low values with respect to initial ones.

For what concerns the matrix Q, which is the *process noise covariance* matrix, the tuning has been performed by considering the time-variance of the system given by the jitter in the sampling time: in particular by considering an acceleration process noise variance $\sigma_a^2 = 0.1$, Q has been built as follows:

$$\mathbf{G} = \begin{bmatrix} 0.5\Delta t^2 & 0.5\Delta t^2 & 0.5\Delta t^2 & \Delta t & \Delta t & \Delta t & 1.0 & 1.0 & 1.0 \end{bmatrix}^T \tag{6.4}$$

$$\mathbf{Q} = \mathbf{G} \cdot \mathbf{G}^{\mathbf{T}} \cdot \sigma_a^2 \tag{6.5}$$

So for a generic *dt* the **Q** matrix of the 3D constant acceleration model of the Kalman filter is

$$\mathbf{Q} = \begin{bmatrix} 0.25\Delta t^4 & 0.25\Delta t^4 & 0.25\Delta t^4 & 0.5\Delta t^3 & 0.5\Delta t^3 & ;0.5\Delta t^3 & 0.5\Delta t^2 & 0.5\Delta t^2 & 0.5\Delta t^2 \\ 0.25\Delta t^4 & 0.25\Delta t^4 & 0.25\Delta t^4 & 0.5\Delta t^3 & 0.5\Delta t^3 & 0.5\Delta t^3 & 0.5\Delta t^2 & 0.5\Delta t^2 & 0.5\Delta t^2 \\ 0.25\Delta t^4 & 0.25\Delta t^4 & 0.25\Delta t^4 & 0.5\Delta t^3 & 0.5\Delta t^3 & 0.5\Delta t^3 & 0.5\Delta t^2 & 0.5\Delta t^2 & 0.5\Delta t^2 \\ 0.5\Delta t^3 & 0.5\Delta t^3 & 0.5\Delta t^3 & \Delta t^2 & \Delta t^2 & \Delta t^2 & 1.0\Delta t & 1.0\Delta t & 1.0\Delta t \\ 0.5\Delta t^3 & 0.5\Delta t^3 & 0.5\Delta t^3 & \Delta t^2 & \Delta t^2 & \Delta t^2 & 1.0\Delta t & 1.0\Delta t & 1.0\Delta t \\ 0.5\Delta t^3 & 0.5\Delta t^3 & 0.5\Delta t^3 & \Delta t^2 & \Delta t^2 & \Delta t^2 & 1.0\Delta t & 1.0\Delta t & 1.0\Delta t \\ 0.5\Delta t^2 & 0.5\Delta t^2 & 0.5\Delta t^2 & 1.0\Delta t & 1.0\Delta t & 1.0\Delta t & 1.0 & 1.0 & 1.0 \\ 0.5\Delta t^2 & 0.5\Delta t^2 & 0.5\Delta t^2 & 1.0\Delta t & 1.0\Delta t & 1.0\Delta t & 1.0 & 1.0 & 1.0 \\ 0.5\Delta t^2 & 0.5\Delta t^2 & 0.5\Delta t^2 & 1.0\Delta t & 1.0\Delta t & 1.0\Delta t & 1.0 & 1.0 & 1.0 \end{bmatrix} \tag{6.6}$$

The process noise matrix is then updated at each iteration of the filter with the exact Δt value.

The following plots show a confrontation of the Kalman filter state variables obtained by simulating the two extreme cases of 20 and 150 samples: it's shown that 20 samples are enough for a good position estimation, even if the error covariance on acceleration is not reduced too much.

So in such scenario, very close to the real one, the velocities and accelerations of the ball never vary too much from the values with which the filter is initialize: it's very important to tune the initial state of the filter with some a priori knowledge when a limited amount of samples is available.

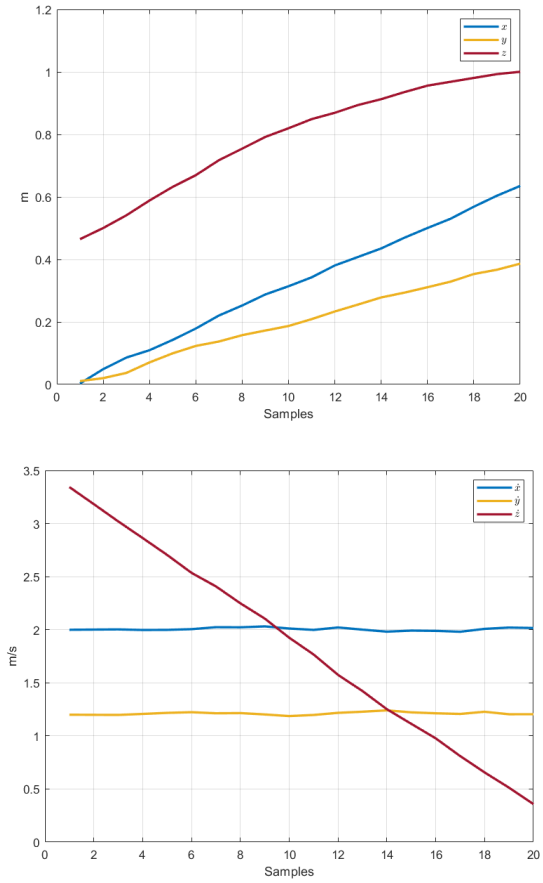


Figure 6.7 Position and velocity estimates computed after 20 filter iterations: as expected from the projectile motion model, the components on x and y axes are almost totally constant, while the z component decreases linearly.

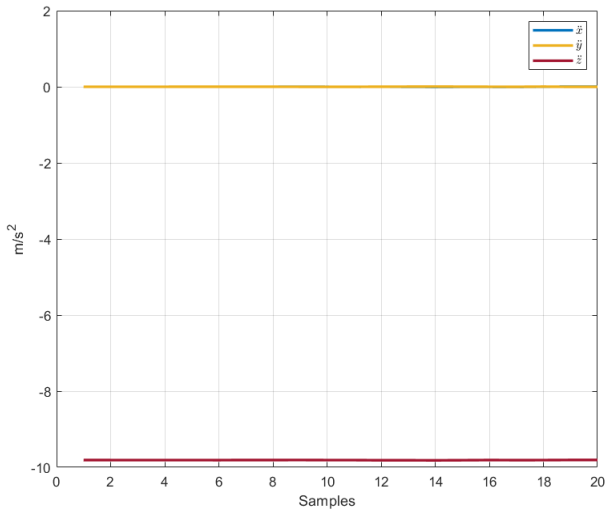


Figure 6.8 Acceleration estimates computed after 20 filter iterations: the covariance is still too high so the filter 'trusts' only the initial conditions.

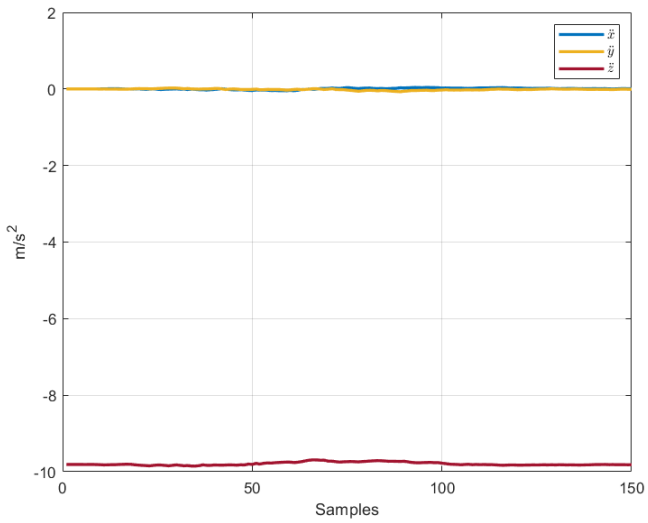


Figure 6.9 Acceleration estimates computed after 150 filter iterations: the covariance reduces significantly after 45/50 steps, the state of the filter starts to approaching to real values which are constant as expected.

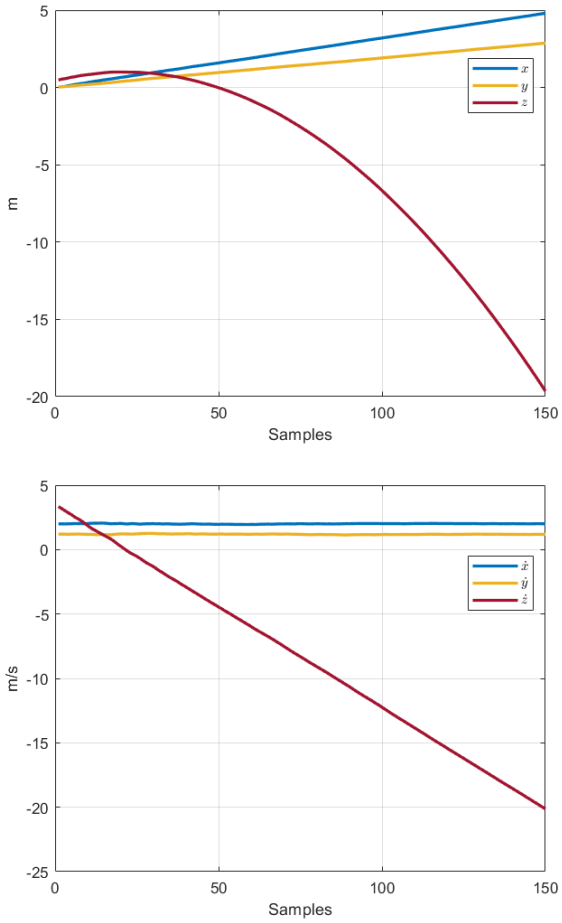


Figure 6.10 Position and velocity estimates computed after 150 filter iterations: again, the components on x and y axes assume and keep constant values, while the z component decreases linearly.

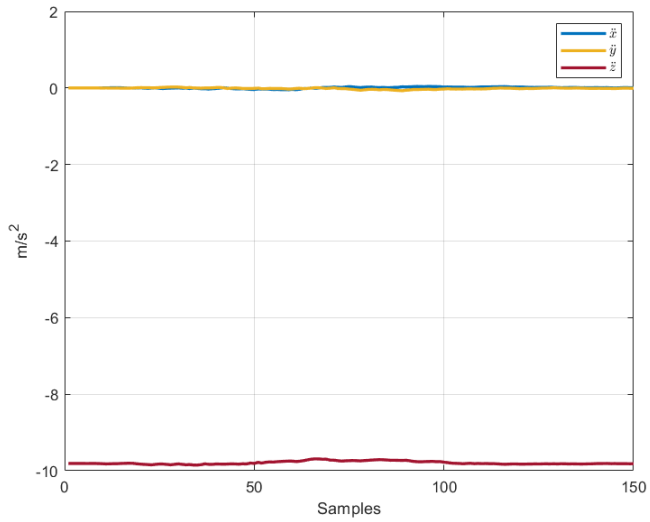


Figure 6.11 Acceleration estimates computed after 150 filter iterations: the covariance reduces significantly after 45/50 steps, the state of the filter starts to approaching to real values which are constant as expected.

Qualitative results on the estimation can be visualized by confronting raw and estimated trajectories for different amounts of samples:

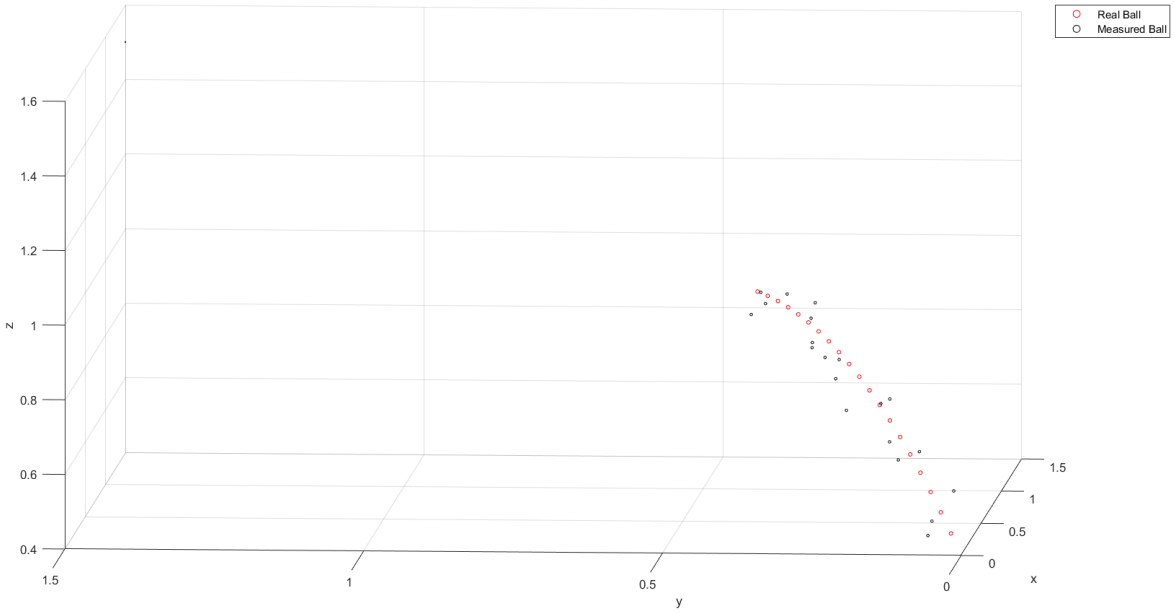


Figure 6.12 Raw simulated trajectory for 20 samples.

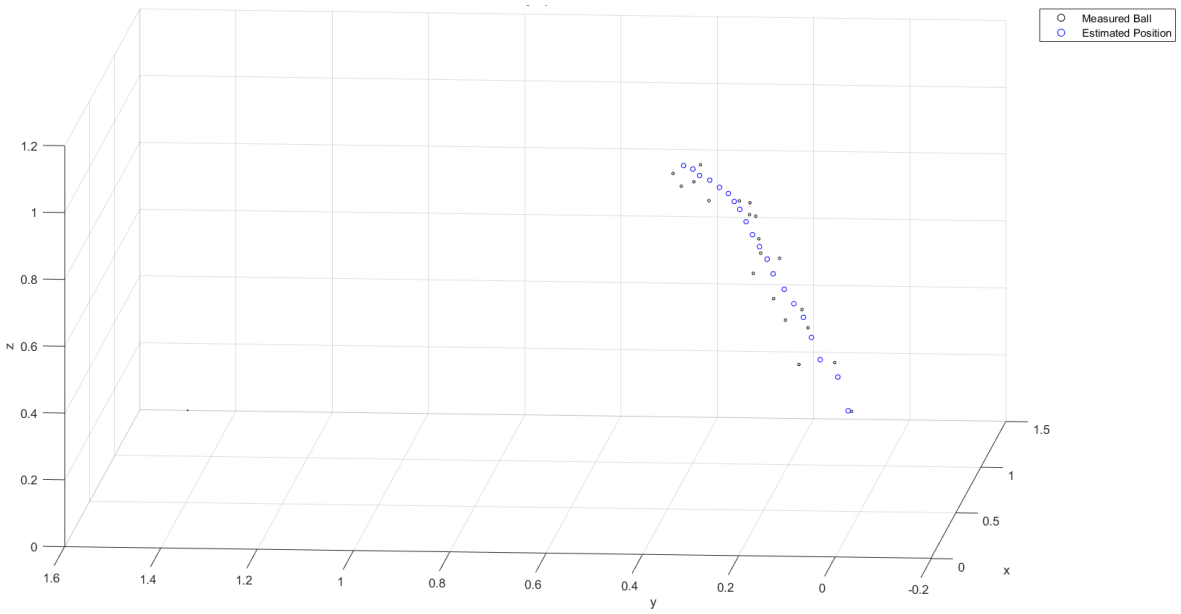


Figure 6.13 Filtered simulated trajectory for 20 samples.

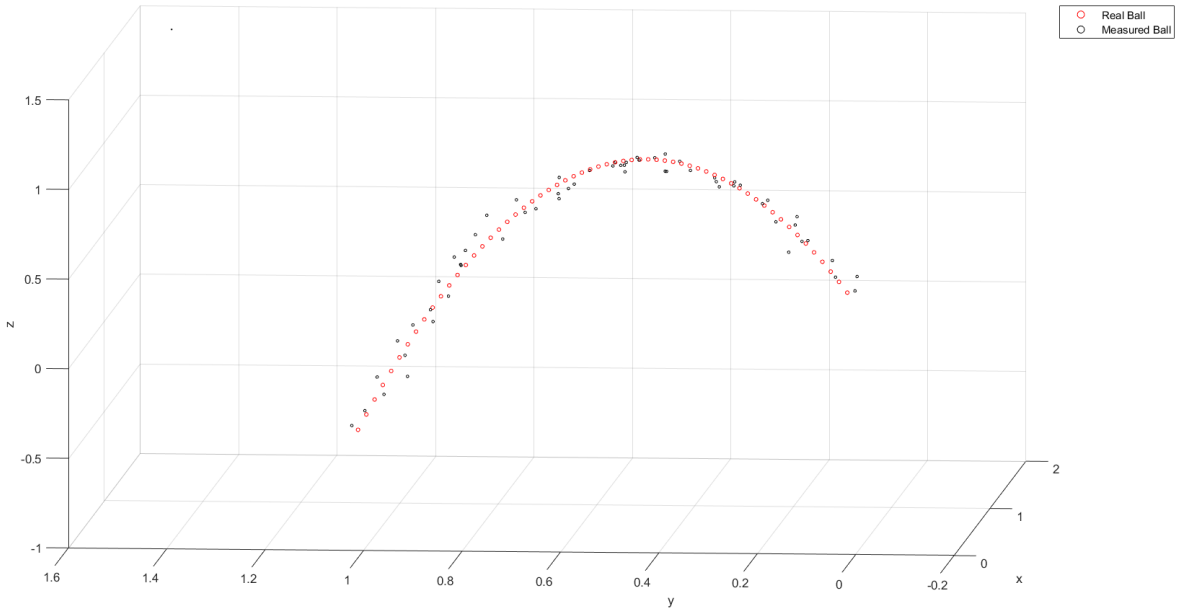


Figure 6.14 Raw simulated trajectory for 60 samples.

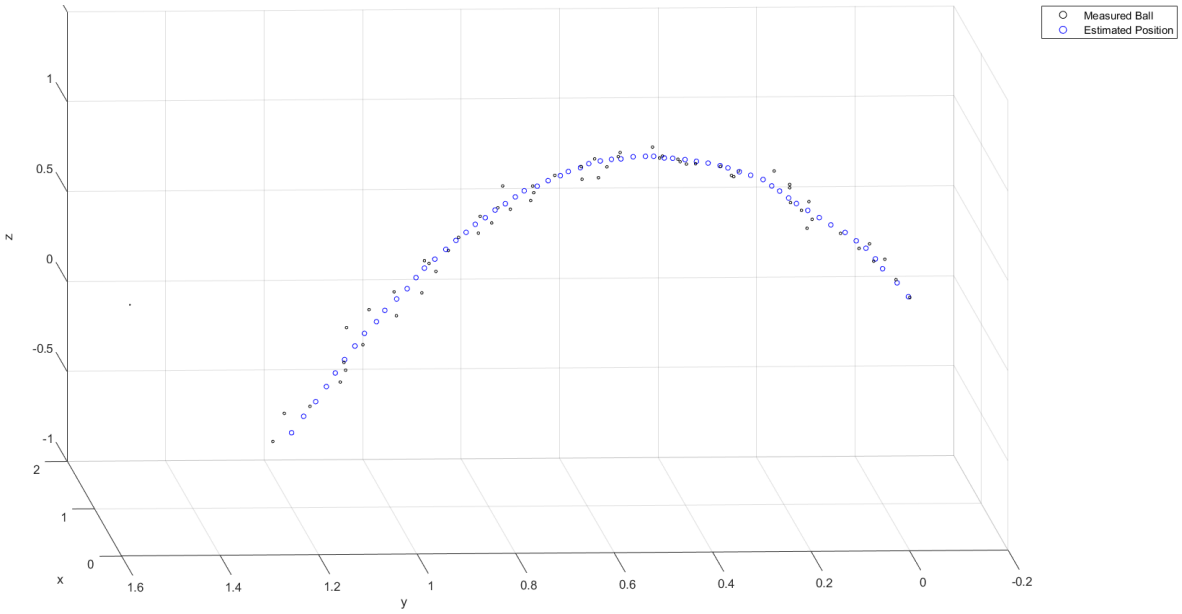


Figure 6.15 Filtered simulated trajectory for 60 samples.

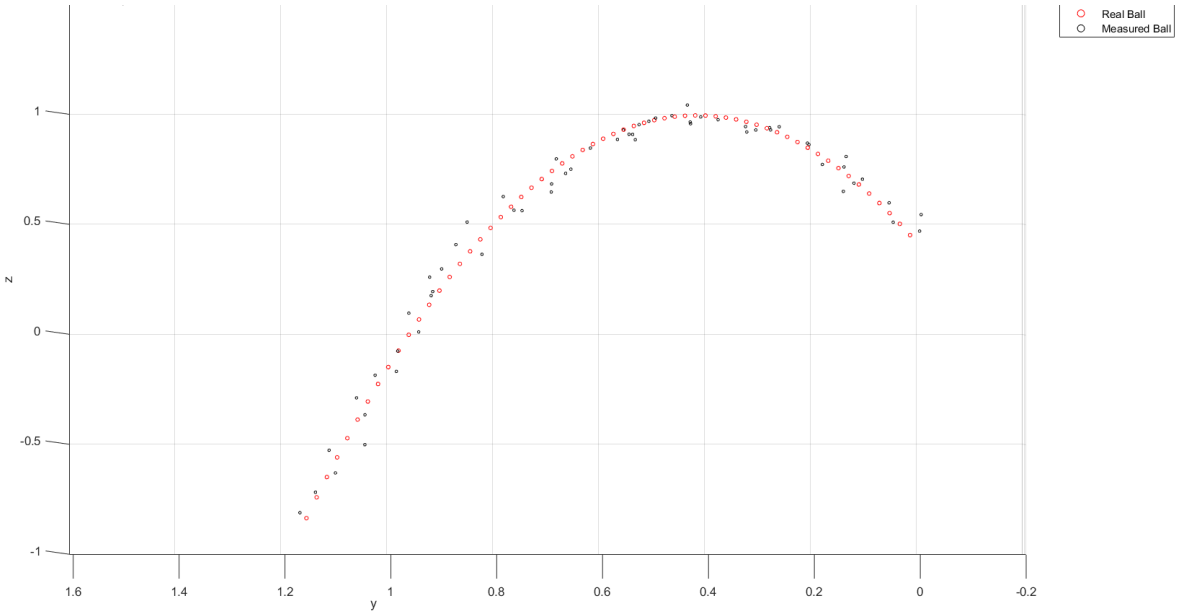


Figure 6.16 Raw simulated trajectory for 60 samples on *yz* plane.

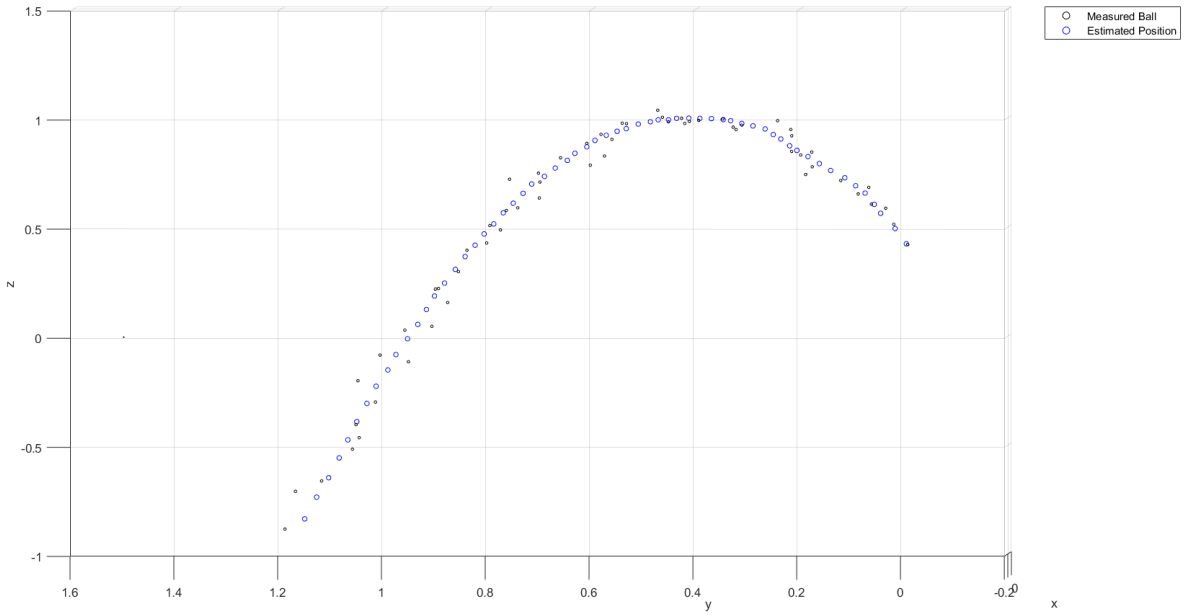


Figure 6.17 Filtetred simulated trajectory for 60 samples on *yz* plane.

Simulations with real measurements

Real position measurements from several throws of the ball have been recorded and used as datasets for verifying the previous assumptions and completing the tuning of the parameters. In particular the measurement noise covariance matrix **R** has been tuned by considering some static observations of the ball in fixed and known positions: due to the accuracy of the camera the measured position never differed from the real position of the ball for more than 2.5 cm. Moreover the point cloud measurements provide two additional sources of uncertainty: firstly during its motion the ball travels in front of a non-uniform background which can lead to a slight distortion along the Z axis of the camera which points out of the lenses; secondly the point cloud is a surface-based measure, so the radius of the ball must be considered in the uncertainty as an attempt in computing the real center of the ball.
Due to these considerations plus some empirical results, the R matrix is defined as follows

$$\mathbf{R} = \begin{bmatrix} 8 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 0 & 12 \end{bmatrix} \tag{6.7}$$

The simulations have given results which are very close to those obtained with the artificial measurements, so the configuration of the parameters obtained through the tuning has

been chosen as valid for experiments.

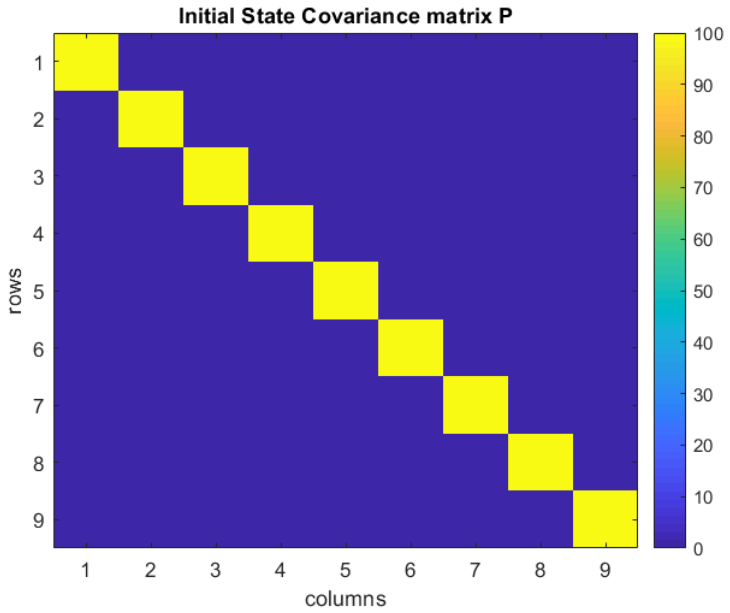


Figure 6.18 Error covariance matrix at initialization.

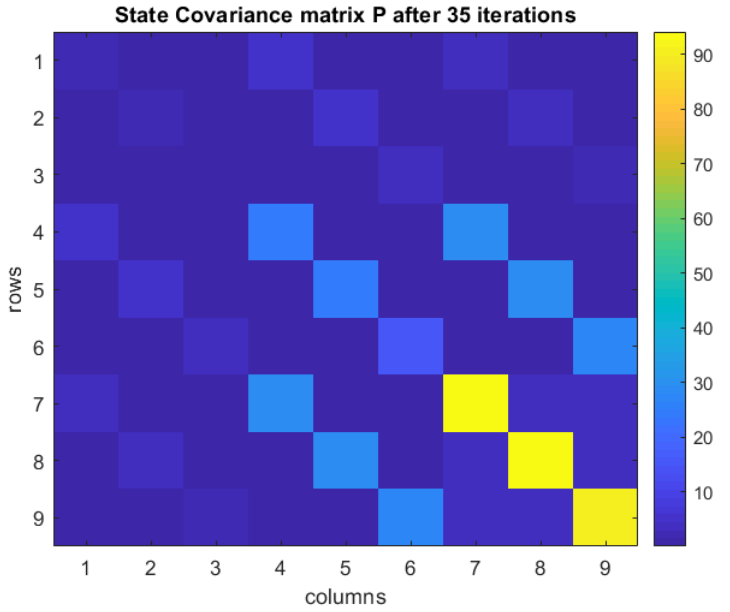


Figure 6.19 Error covariance matrix after 35 iterations on real measurements. The values on acceleration are still very high, so it would not be possible to have an accurate estimate, the initialization is crucial.

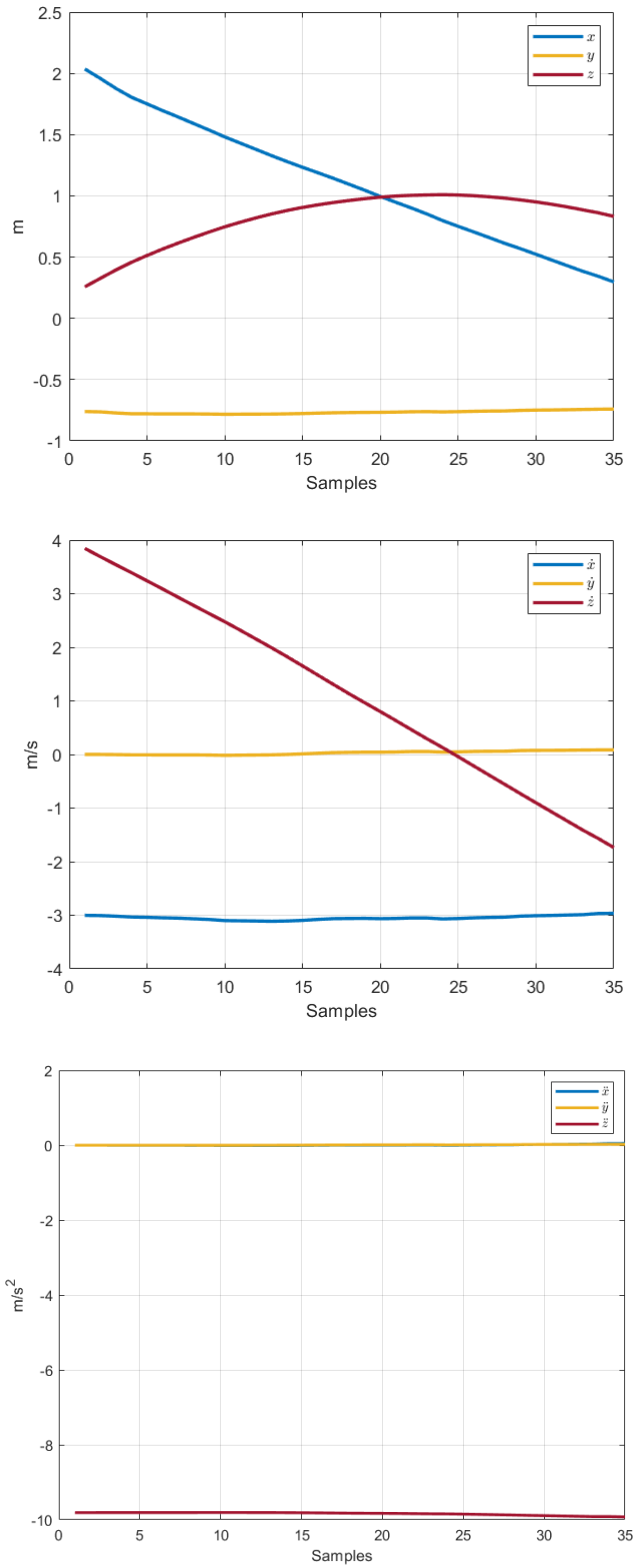


Figure 6.20 Position, velocity and acceleration estimates from real measurements. It can be seen that for each of the three variables the results are consistent with those obtained with the simulated measurements.

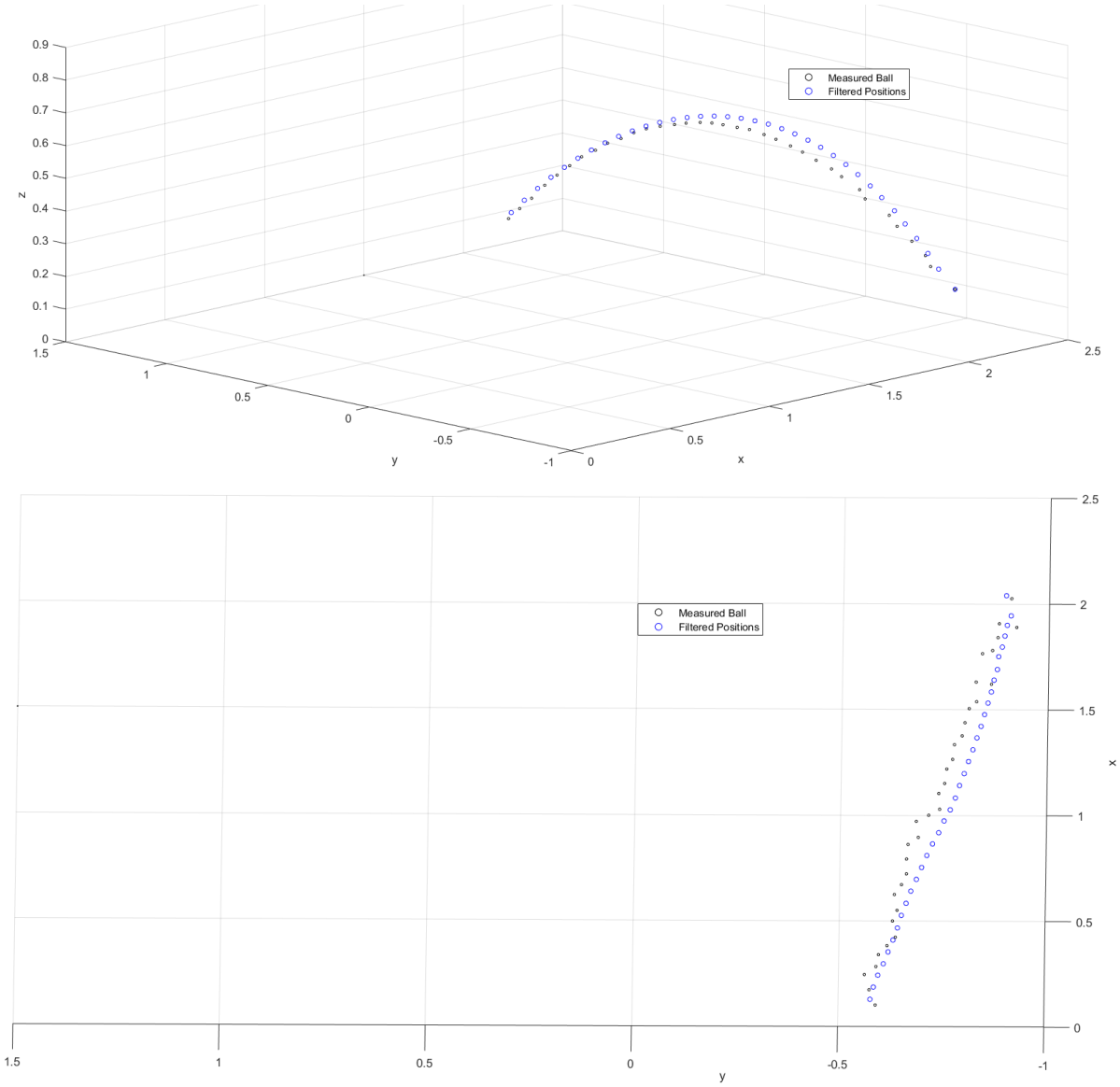


Figure 6.21 Ball trajectory after Kalman filter processing: 3D view and *xy* plane view. The tuned parameters are validated on real measurements.

Ball Throws

When performing an estimation of the position and velocity of an object flying in mid-air by adopting a linear model, the practical issue of *filter initialization* must be addressed.
An initial state for the Kalman filter must be defined in order to be compatible with the effective motion of the tracked object: the main purpose of a correct initialization is to help the algorithm to minimize the convergence error as much as possible within the execution steps.

In fact by considering the simple case of a linear planar motion with a fixed initial velocity, it can be shown that initializing the filter with a null velocity leads to a slower convergence of the state variables.
The initialization of position do not cause any issue since the initial spatial coordinates are sampled when the parabolic motion is recognized through the distance criteria previously described, and the same can be assumed for the acceleration which is initialized as a constant vector $[0 \quad 0 \quad -9.81]^T m/s^2$.
The assumption on an exact numeric value of gravity acceleration is feasible since the base frame of the robot is sufficiently leveled on the horizontal plane.
The issues related to the Kalman filter initialization in fact regard the velocity of the tracked ball, since is very difficult to give exact values for each throw without measuring it.
A feasible solution of this problem, or at least an approximation of it, has been found through the analysis of many throws already constrained to the spatial assumption described in the beginning of the section.
By acquiring data during the execution of the visual sensing node of the architecture it has been possible to identify a range of feasible values which correspond to the ball flight within the 2 meters in front of the robot.

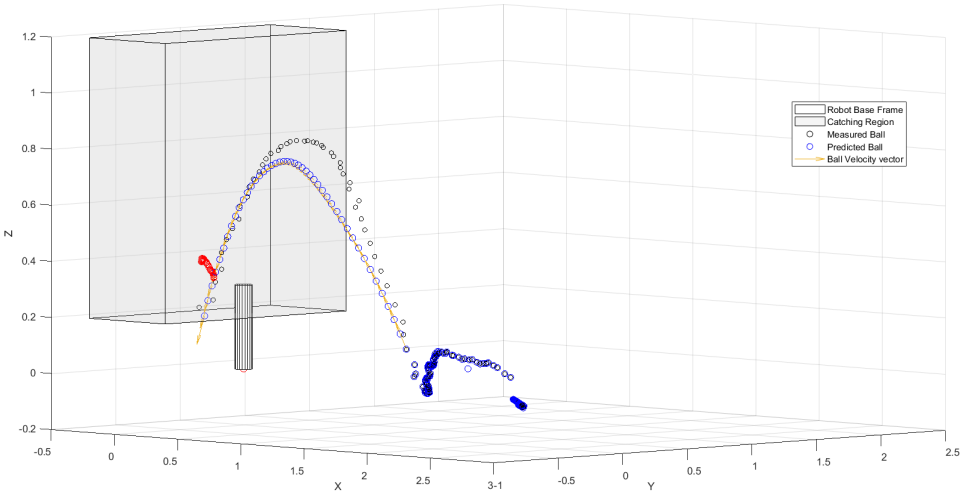


Figure 6.22 Visualization of a recorded dataset.

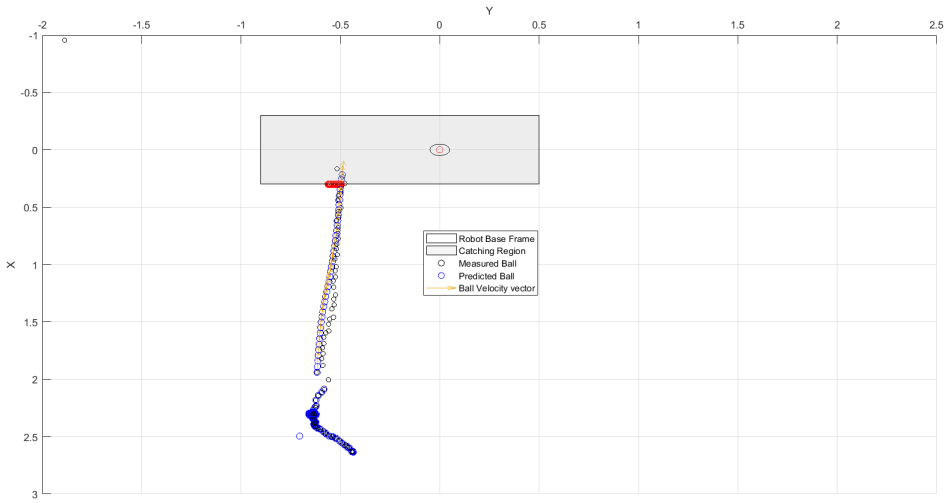


Figure 6.23 Visualization of a recorded dataset on the *xy* plane: the described geometry of the problem is shown.

The correctness of the final values of the filter state has been assessed through covariance matrix values and the state variables themselves: wrong initialization values bring to a weird behavior of the filter and/or an excessively slow convergence.
Moreover by considering the geometry of the parabolic motions some heuristic assumptions based on the flight time have been used in order to determine the initial conditions of the filter: for instance by considering that the distance between the throwing zone and the robot is approximately 2.5 m and the complete flight time without considering the catch point is about 0.8 s assumptions on the x and z components can be done. By considering the previous assumptions v_x is initialized as -2.8 m/s, while v_z has been used with different values within the range $[2.7, 3]$ m/s because the precise value inside the range has been found not so relevant for the results of the predictions.

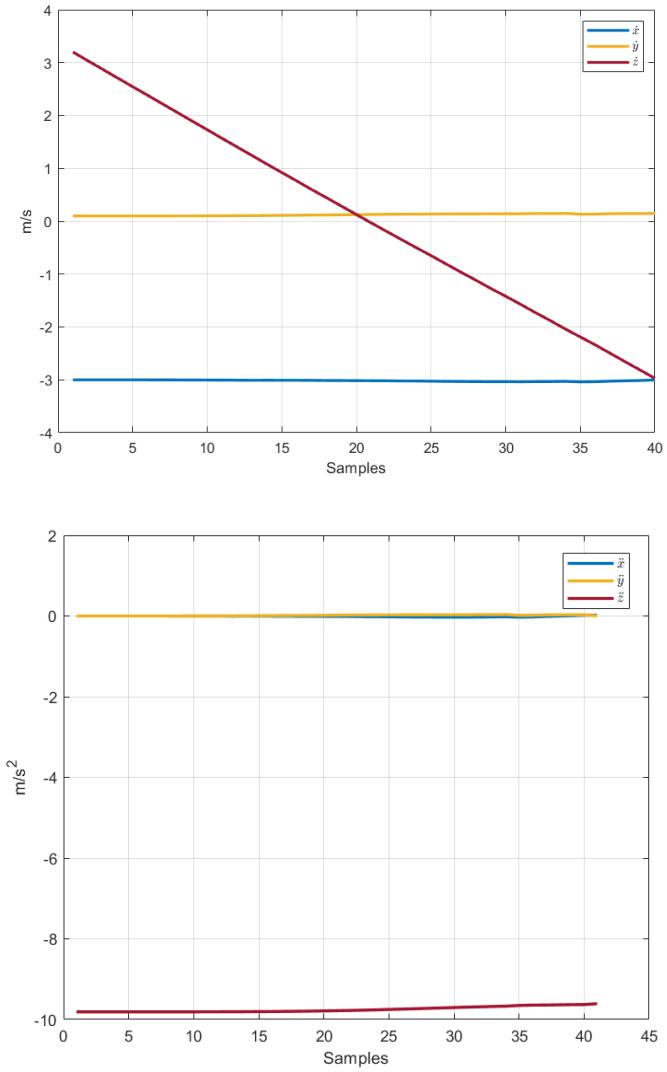


Figure 6.24 Velocity and acceleration estimates of a real throw, computed online by the vision processing node. The results match those obtained in simulations, both with fake and real measurements. The tuning of the filter is validated for the real application.

The most delicate component of the initial velocity of the vector is v_y: the experiments have revealed that errors in the initialization of the y component of the velocity vector can lead to difficulties in the convergence of the velocity and most importantly on the estimation of the catch point.

This importance is given by the fact that for x and z components the throw has a fixed direction: there never could be a positive velocity on x because this would mean throwing the ball in the opposite direction; at the same time a negative velocity component on z would not allow any projectile motion but will make the ball immediately reach the floor.

The y component instead is responsible of diagonal motions of the ball and defines the angle between the parabolic motion plane and the x axis of the robot frame, so it can assume both positive or negative direction dependently on the thrower's wrist movement, but in case of a non-zero initialization and a throw which points, even slightly, towards the opposite direction will cause a slower convergence and total misleading in the initial predictions.
These considerations led to the strong assumption of throwing the ball right in front of the robot with a parabolic motion parallel to its x axis, so v_y is initialized as 0 m/s in order to adapt more rapidly in case of very small positive or negative values.

6.2.3 Prediction Assessment

As previously remarked, the ultimate goal of the vision module is to predict the point at which the ball will be intercepting a fixed volume surrounding the robot, with an estimation of velocity, orientation and time.
This estimate is entirely based on the state of the Kalman filter and is updated at each step, and along this section the results of this stage are presented.
It's important to remark that the catch point selection criteria is based on time, which constrains the robot motion and so has a strict lower bound to be satisfied: for this reason having a complete convergence of the catch point to the correspondent predicted position of the ball is virtually possible but it would be too late to perform start any Cartesian trajectory.

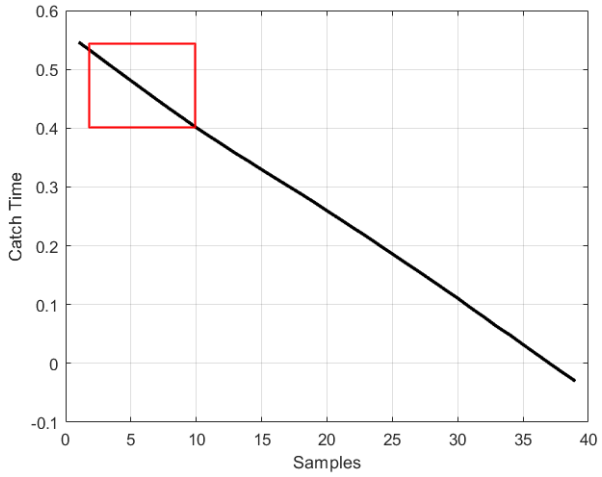


Figure 6.25 The predicted catch time follows a linear decreasing profile. The region for sampling the catch point is highlighted by the rectangle.

What is possible to do is to minimize the error between the sampled point, chosen before the time limit, and the prediction at that time instant projected on the plane described by the fixed x coordinate of the catch point.
It's important to mention that the spatial estimation is important as well for the computation of the catch point, since its selection must meet both time and spatial requirements: for instance a negative z coordinate will cause an immediate discard of the point even if the other spatial constraints are met; on the other hand if the spatial constraints are met after the lower time bound are discarded too.

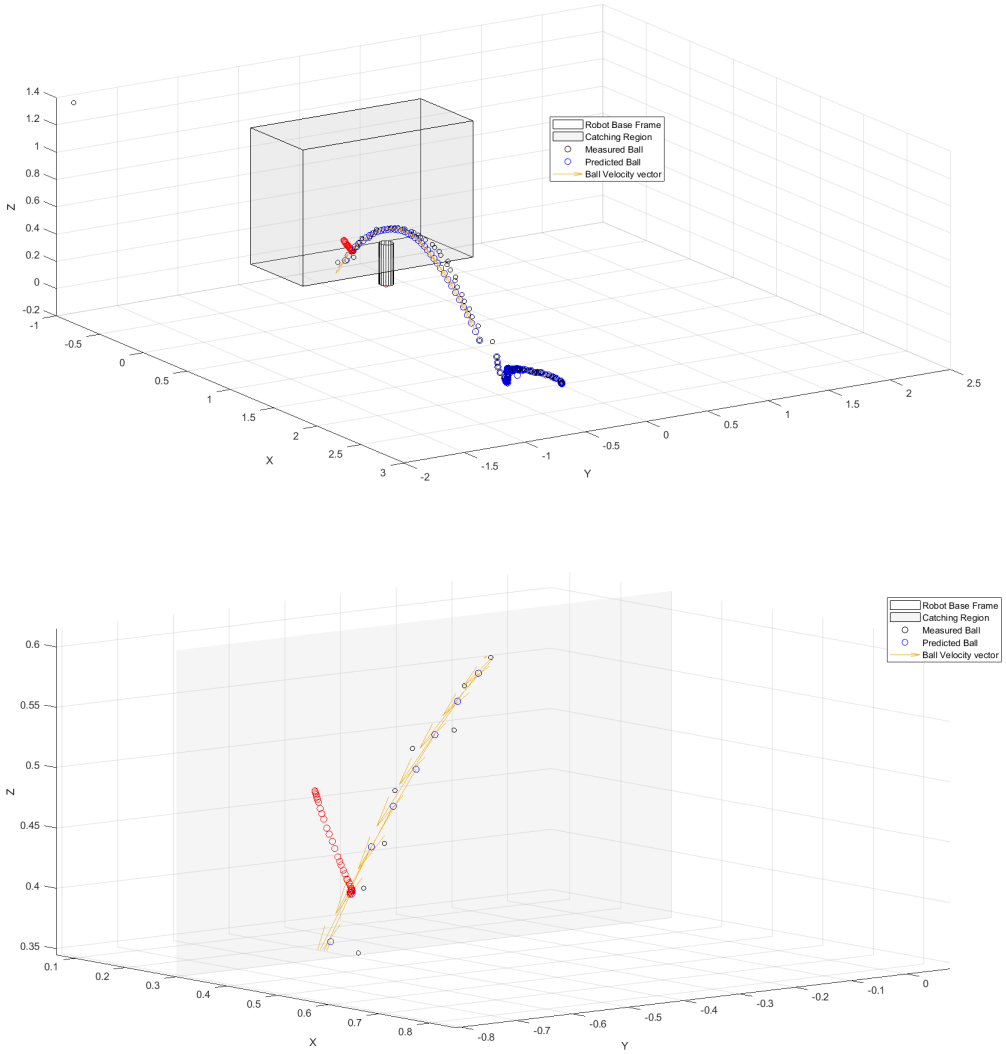


Figure 6.26 3D motion with detail on the catch points, which converge to the trajectory at the workspace intersection (experiment n. 1).

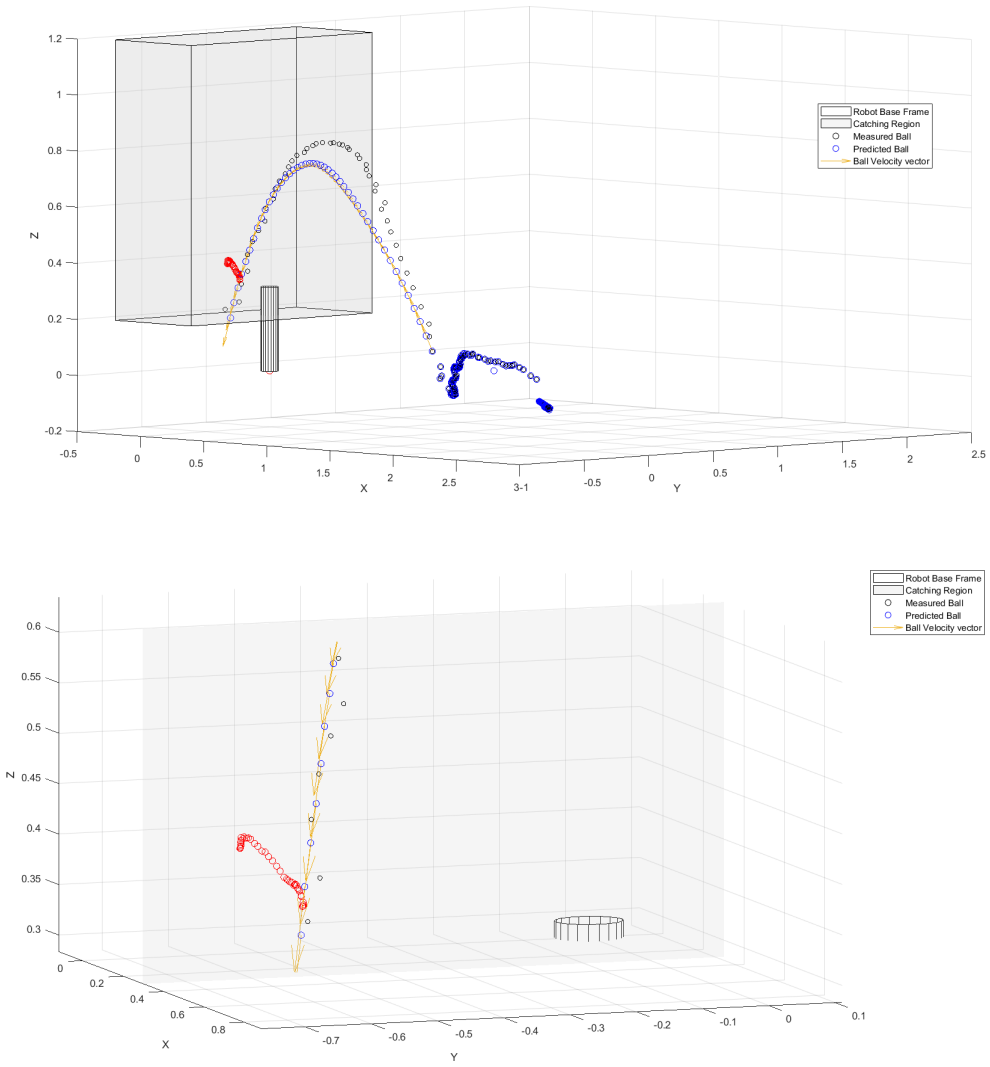


Figure 6.27 3D motion with detail on the catch points, which converge to the trajectory at the workspace intersection (experiment n. 2).

6.3 Robot Motion Benchmark

The motion towards the catch point is strictly driven by time, in particular the Cartesian trajectory from which the velocity control signals are retrieved are computed on the value of the catch time.

Due to this constraint a complete benchmark of robot's motion under time and spatial constraints has been performed with simulated catch points, with the goal of understanding and assess the limits of the manipulator's performance when performing high-speed tasks.

From the defined throw area the estimated catch time is between 0.4 s and 0.7 s depending on the angle of the projectile trajectory.
The analysis has been performed by keeping into account both open and closed loop control laws in order to analyze the benefits of the closed loop trajectory tracking and most importantly different catch time values within the range mentioned before.
Moreover the robot's motion has been tested on both tasks of prehensile and non-prehensile ball catching in order to point out the differences and possible issues, especially for the second case.

6.3.1 Prehensile Ball Catching

The motion control of the robot has been firstly tested for prehensile ball catching task, so the Cartesian polynomial trajectory is computed for reaching a specified position in space within a fixed amount of time with null final velocities.
The aim of this task is to reach the predicted catch point within the catch time in a point-to-point motion and with the predicted orientation, in order to catch the ball with a small basket mounted on the end-effector.
Here follow the results of the time-based evaluation both in Cartesian and joint space.

Neutral Initial Position

Along this first analysis no initial motion of the robot towards the catch region has been considered.
The manipulator starts its motion from a default position specified in joint space, with the cartiesian pose of the end-effector in $\begin{bmatrix} 0.15 & 0 & 0.5 \end{bmatrix}^T m$.

In order to assess how the robot motion changes when the speed requirements raise, some initial tests have been performed by computing a polynomial trajectory for a point to point motion with $t_f = 2.5$ s.
As shown in the following figures, the tracking of the polynomial is very precise and the tracking errors are almost null along any direction.

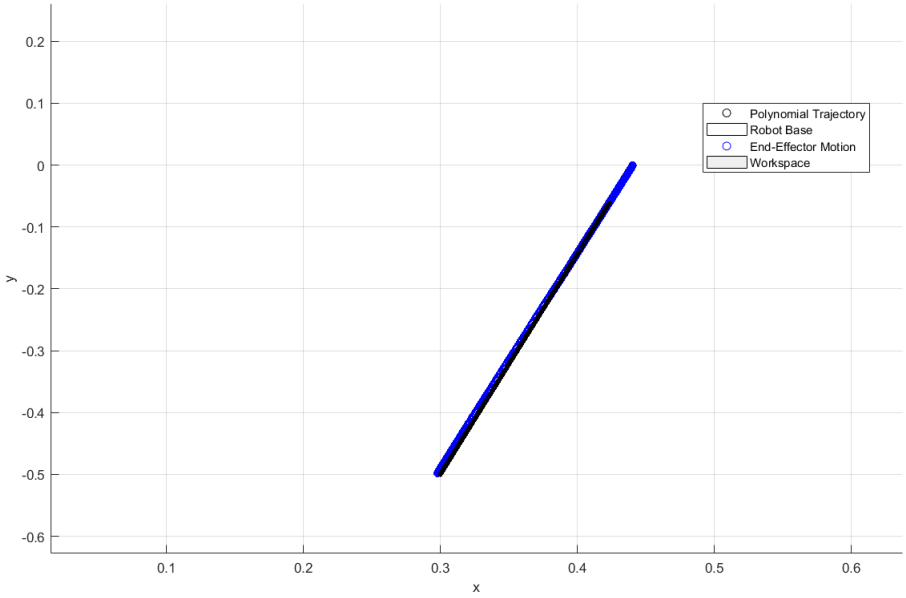


Figure 6.28 End-effector motion on the plane *xy* for a 2.5 s trajectory.

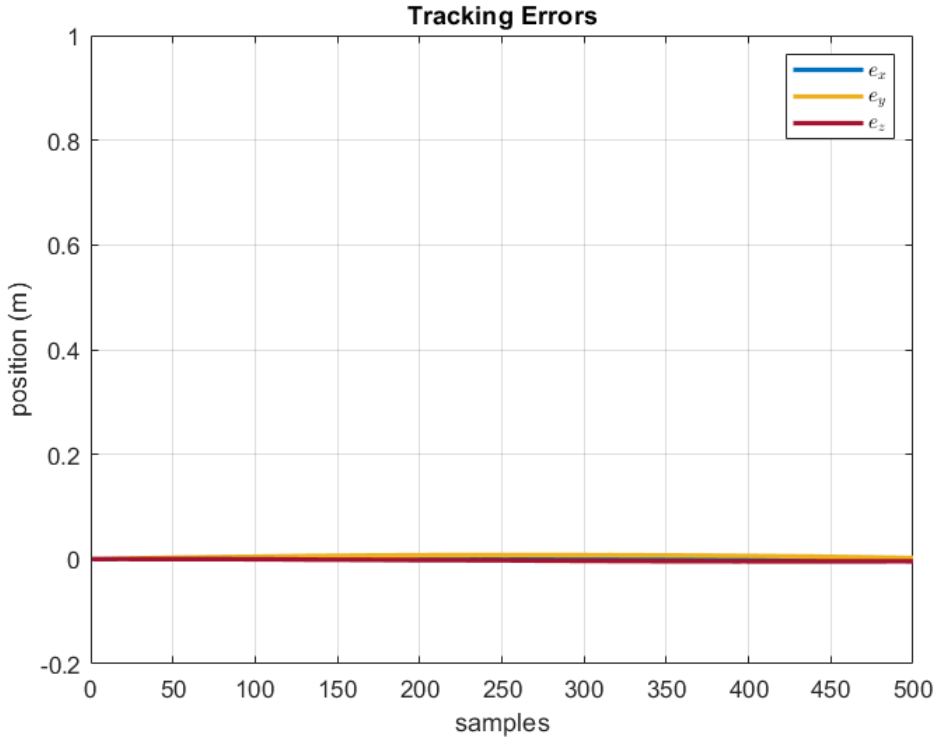


Figure 6.29 Position errors for a 2.5 s trajectory.

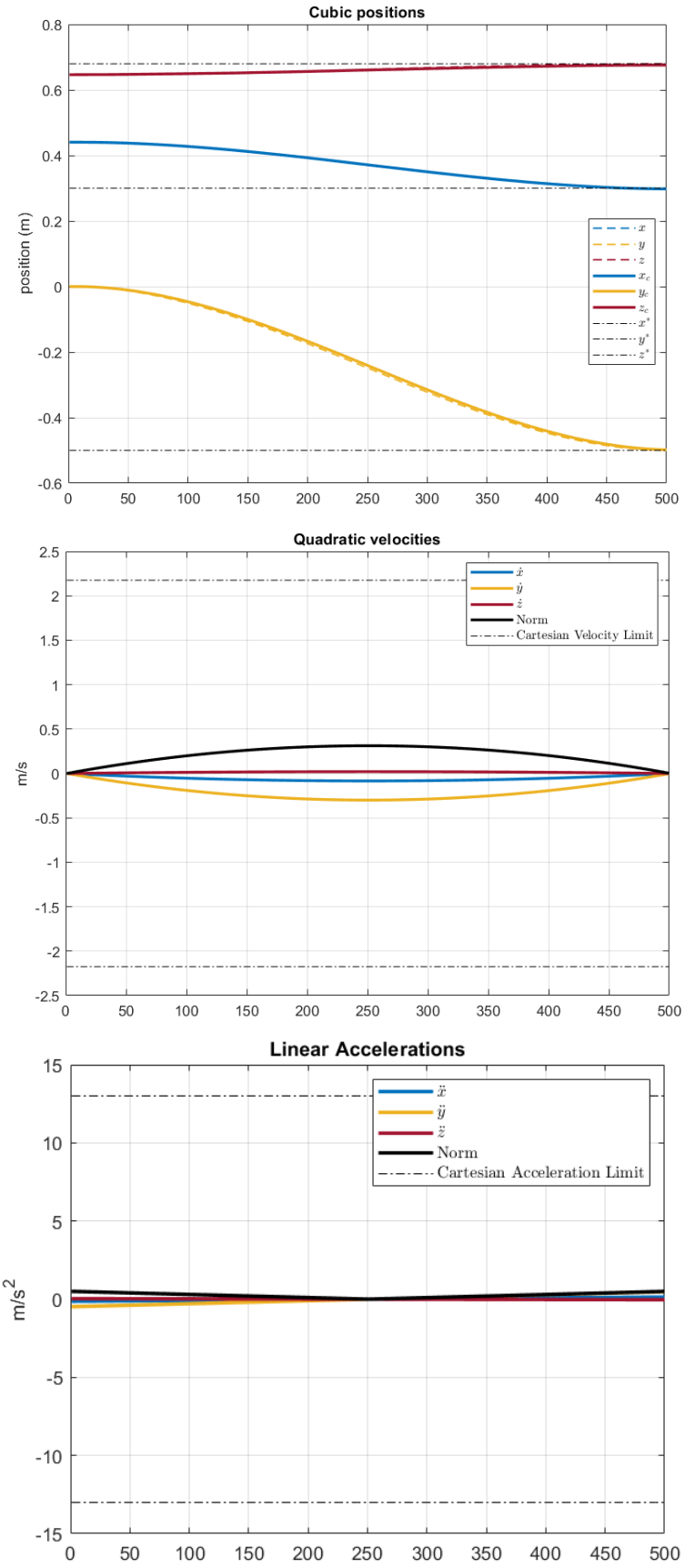


Figure 6.30 Cartesian position, velocity and acceleration for a 2.5 s trajectory.

As shown, the position of the end effector converges to the goal position specified when computing the coefficients, moreover the trajectory is tracked along the entire motion.
Both Cartesian velocity and acceleration are within the limitations provided by Franka [26].

The following results have been obtained through the motion analysis of a trajectory performed in 0.7 s, time which has been pointed out as the theoretical limit for the ball: in practice the catching time is almost always lower because the velocities of the ball on x and z are quite similar.

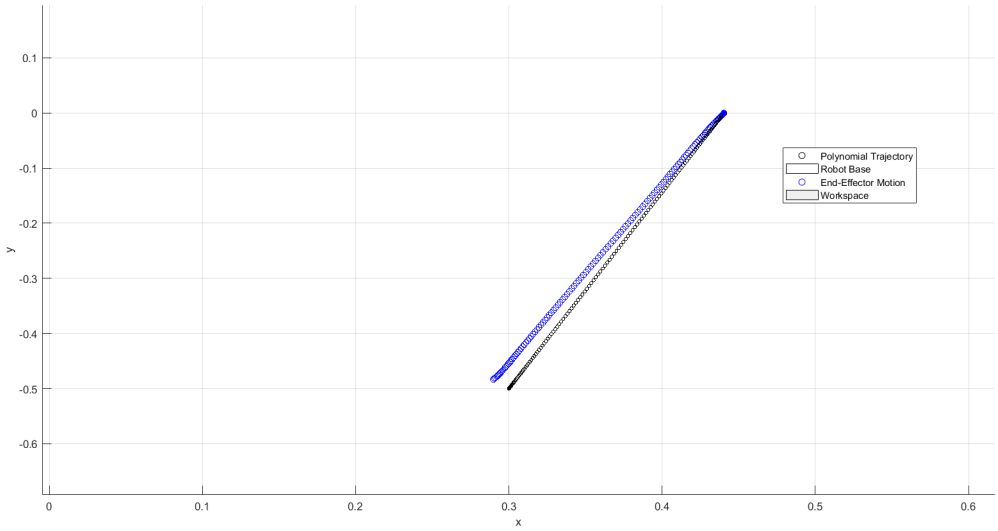


Figure 6.31 End-effector motion on the plane xy for a 0.7 s trajectory.

By observing the end-effector motion it's clear that the required velocity is responsible of a drift in position which causes the robot to miss the catch point, or reaching it after the time deadline when using closed loop control on the trajectory.

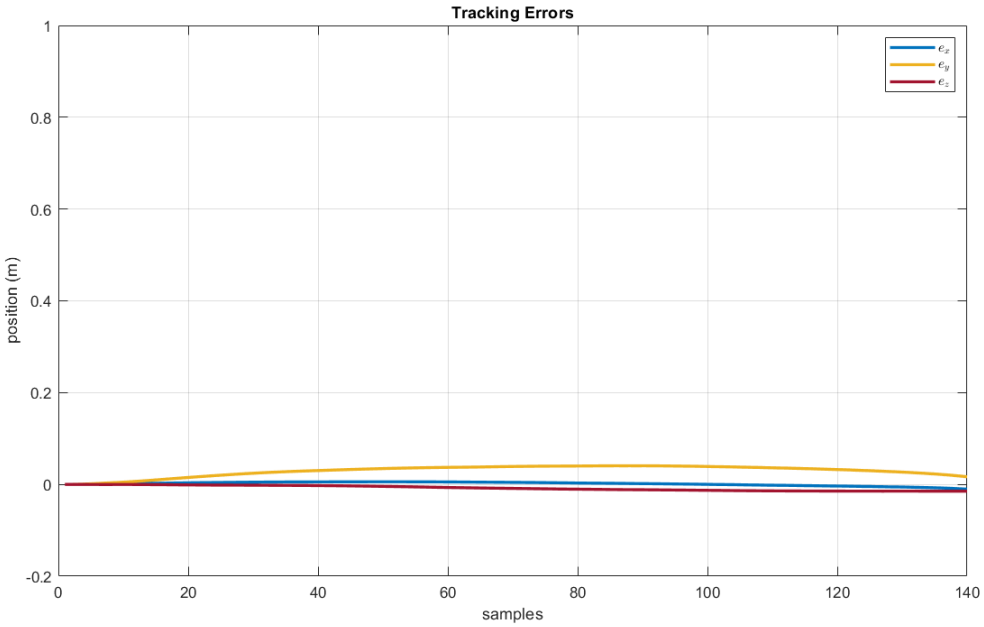


Figure 6.32 Position errors for a 0.7 s trajectory.

The Cartesian position errors computed along the trajectory show that the the y direction is the most difficult to track, which is consistent with the fact that the initial error is greater on y coordinate.

By observing the cubic polynomial trajectory and its derivatives it can be shown that the final convergence to the catch point coordinates is almost complete, but along the motion the end-effector drifts from the desired motion, especially along y.
The limits of the Cartesian motion for the end-effector are met, even if the values have fairly increased from the previous tests of 2.5 s.

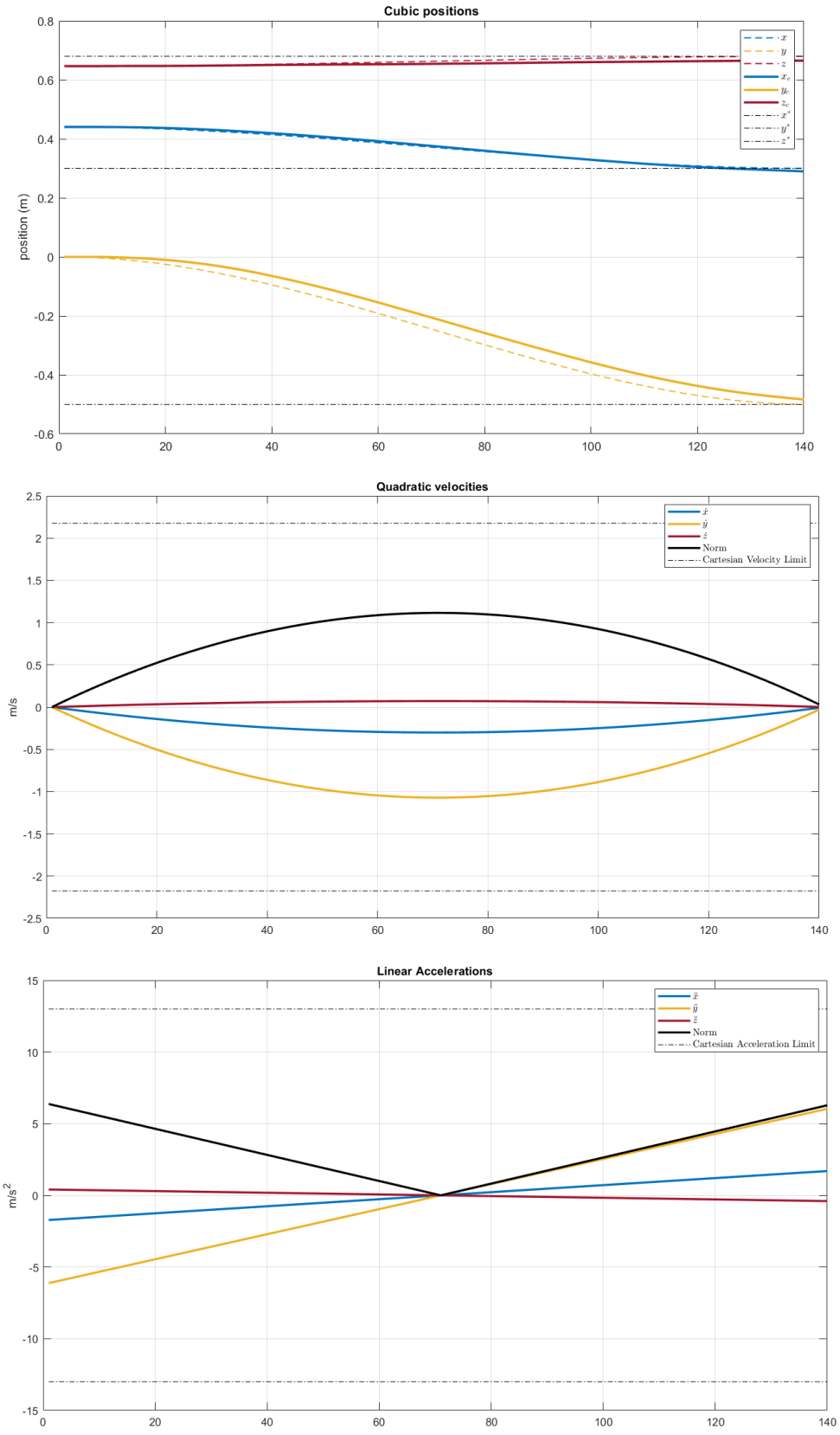


Figure 6.33 Cartesian position, velocity and acceleration for a 0.7 s trajectory.

The cause of the motion drift along the trajectory can be found by analyzing the motion in robot's joints space.
The following plot shows that at approximately 40 samples, which in fact correspond to the instant in which the drift occurs, the velocity command of the third joint is saturated by the controller: this means that the desired motion is assuming a profile which is not completely tracked by the motors.
After about 350 ms even the fourth joint is expected to follow a velocity profile which is beyond its bounds, so even $\dot{q}_7$ saturates.

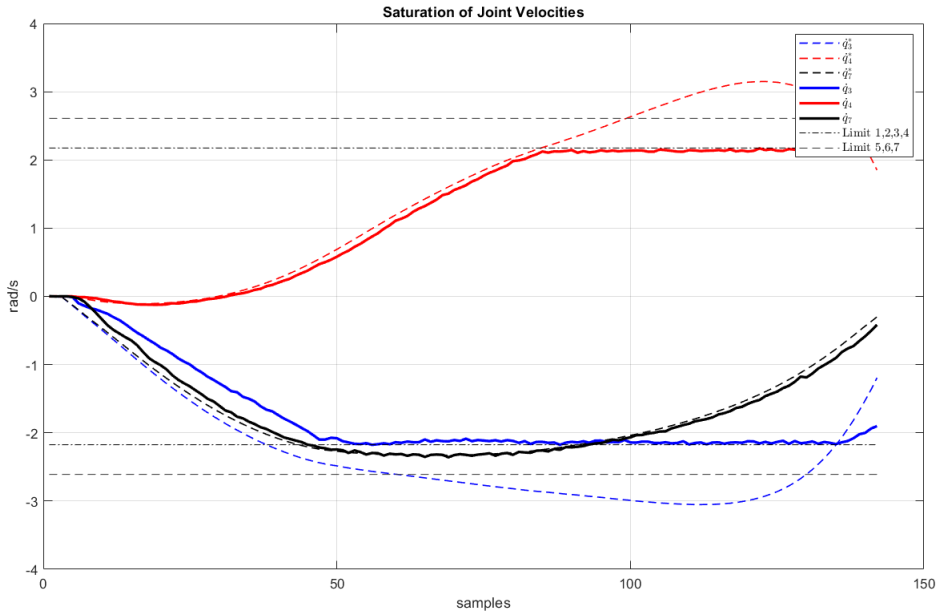


Figure 6.34 Saturation of velocities for joints 3 and 4. The seventh joint is also kept into account because very close to its own velocity negative limit.

By observing the whole sets of desired and commanded joint velocities it can be seen that apart from the two joints which are beyond their threshold, other joints like the seventh and the first are very close to the limits.

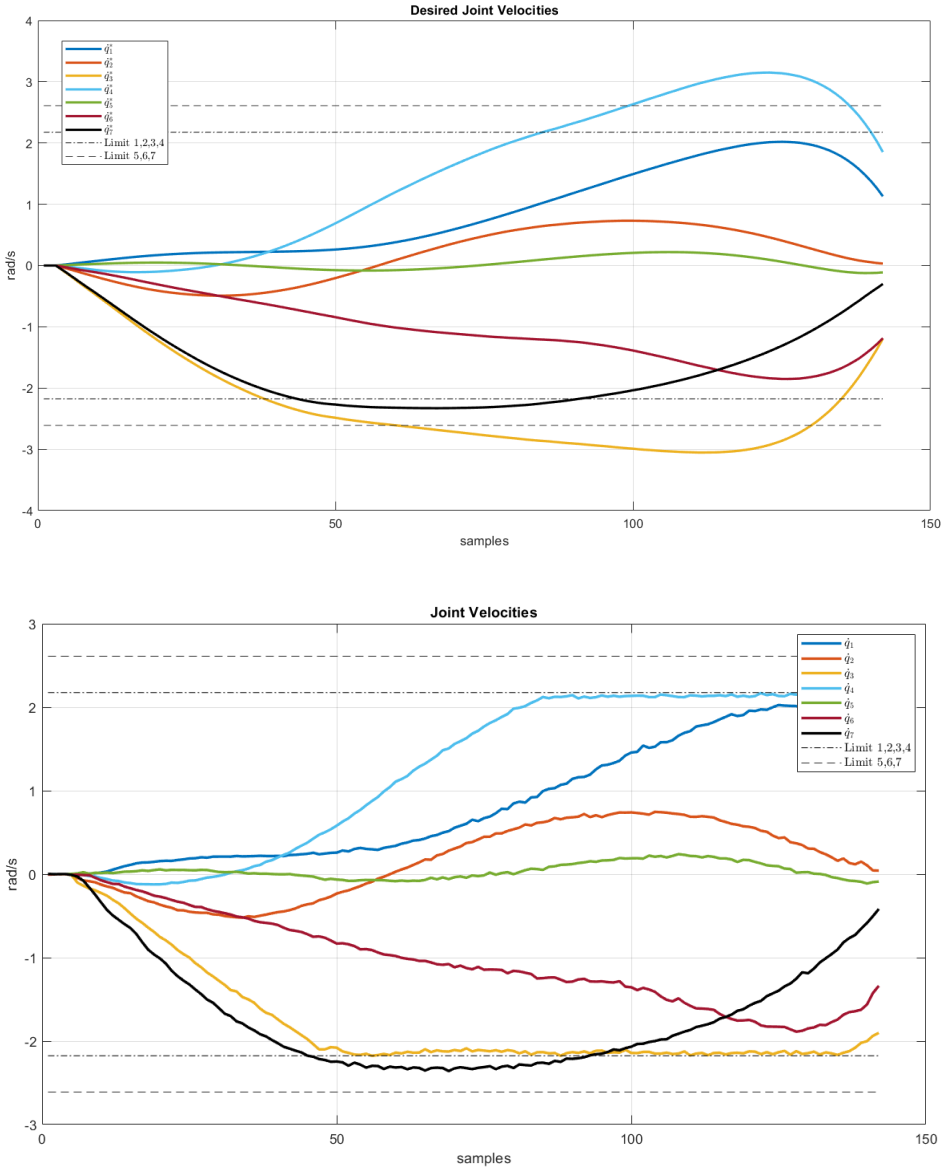


Figure 6.35 Desired and effective joint velocities for a 0.7 s trajectory.

To perform further tests, the final time of the polynomial has been reduced to 0.5 s, which is the average value of a 2 m trajectory, so it's the most common value obtained during the experiments with the ball.

Under this constraint it's immediately clear that the drift in position has increased, and by observing the *xy* plane the visible error is greater than 10 cm.

The profile of the position errors is very similar to the previous case, but the drift on the *y*

coordinate has grown up to almost 20 cm.

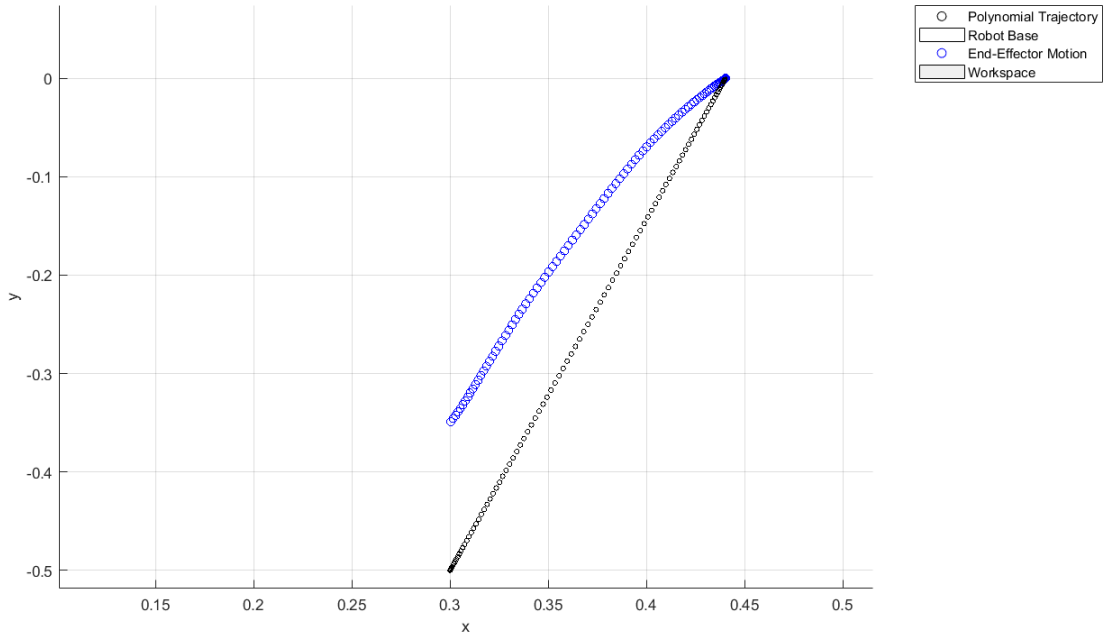


Figure 6.36 End-effector motion on the plane *xy* for a 0.5 s trajectory.

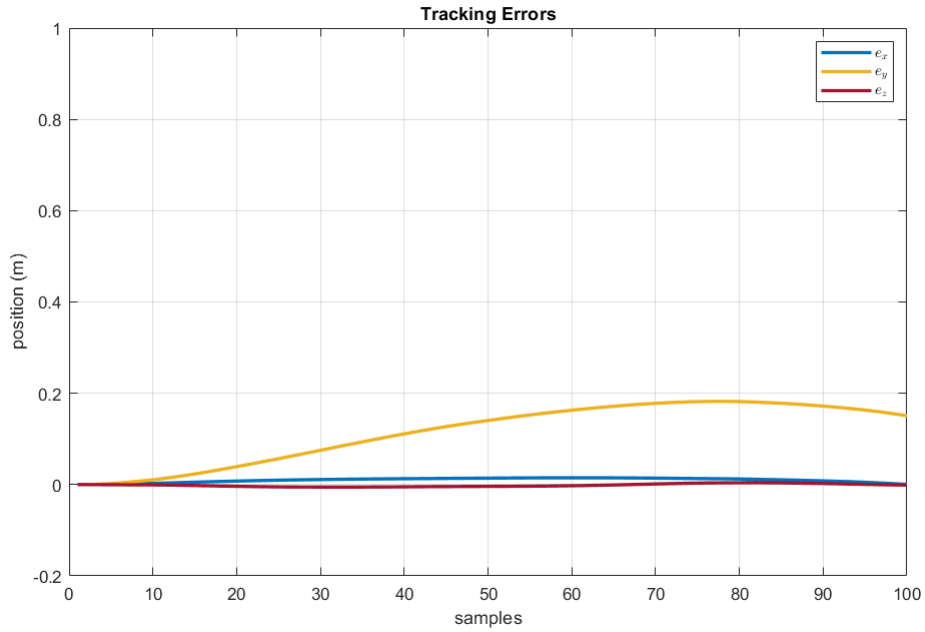


Figure 6.37 Position errors for a 0.5 s trajectory.

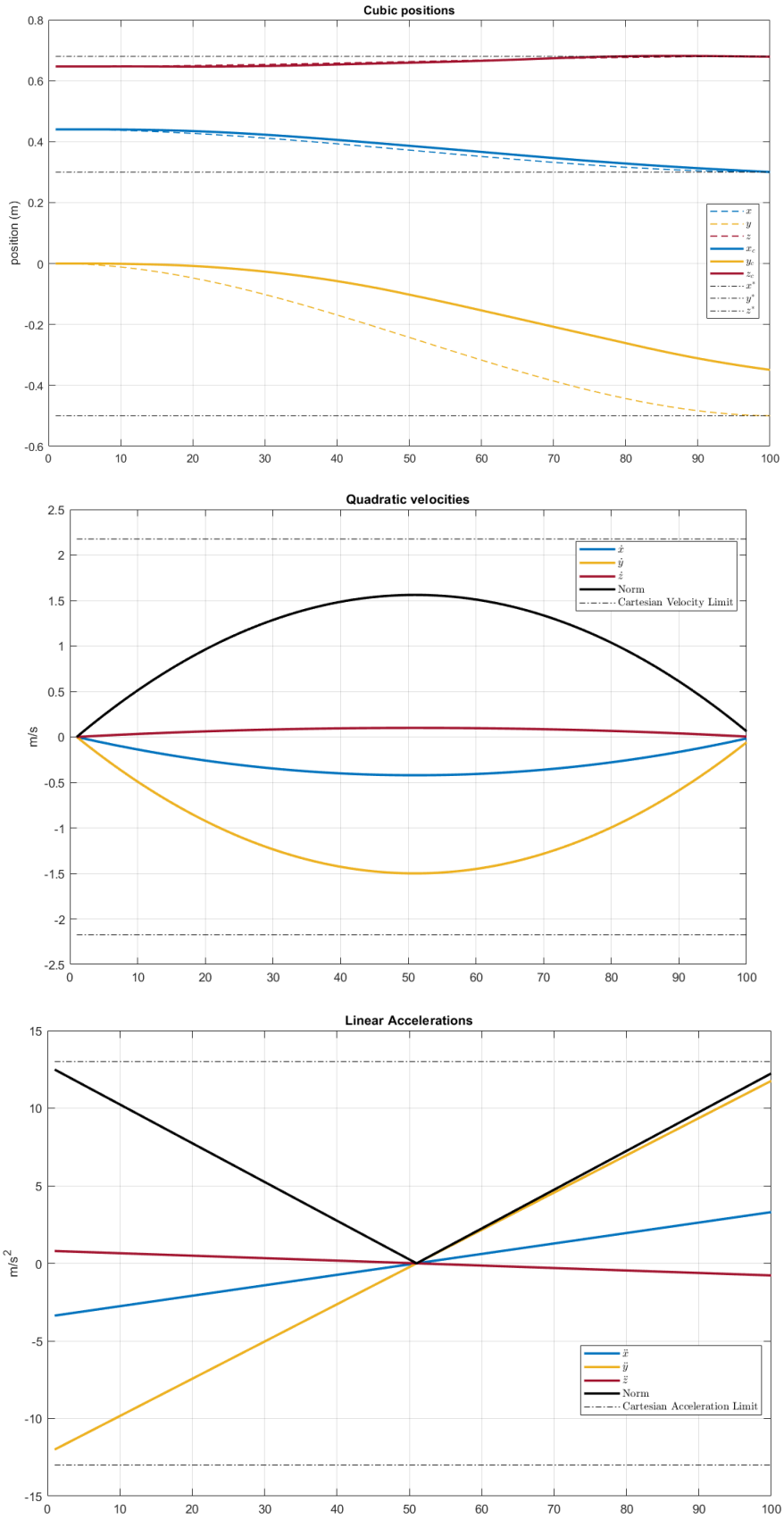


Figure 6.38 Cartesian position, velocity and acceleration for a 0.5 s trajectory.

Now the Cartesian acceleration of the end-effector is very close to the limits provided by the manufacturer: this explains the fact that the performed motion does not follow the trajectory from the beginning, but instead is curved with respect to the desired line.

Again the saturations are quite neat, and now joint 7, which was close to its velocity limit in the previous case, is the first one which saturates, followed by the third and the fourth which reach the limits approximately after the same amount of time already noticed in the 0.7 s test.

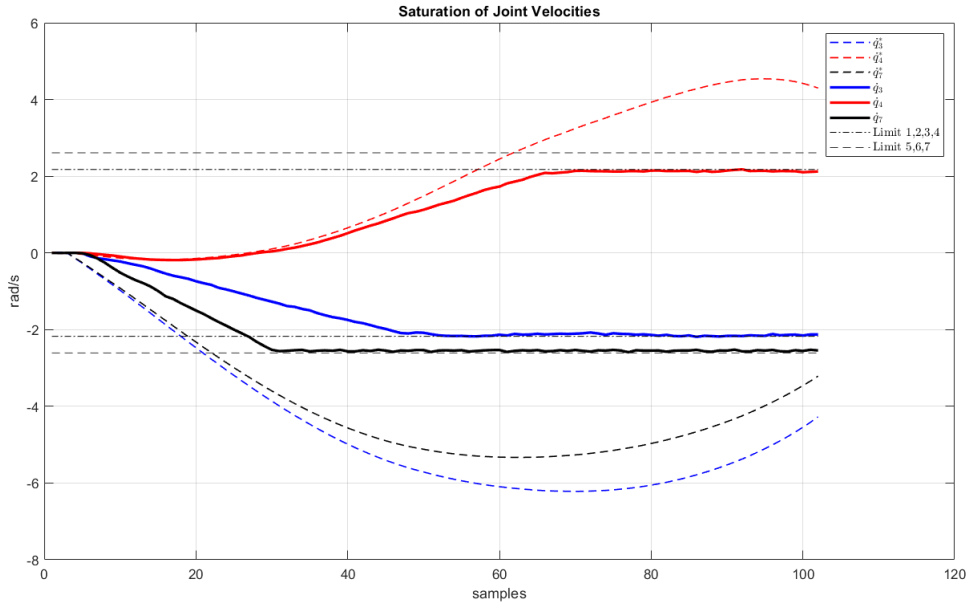


Figure 6.39 Saturation of velocities for joints 3, 4 and 7.

The plot of the desired joint velocities corresponding to the 0.5 s trajectory show that under such time constraint the effort required for the tracking is very high: it can be seen that even the joints which are not saturated are very close to their velocity limits.

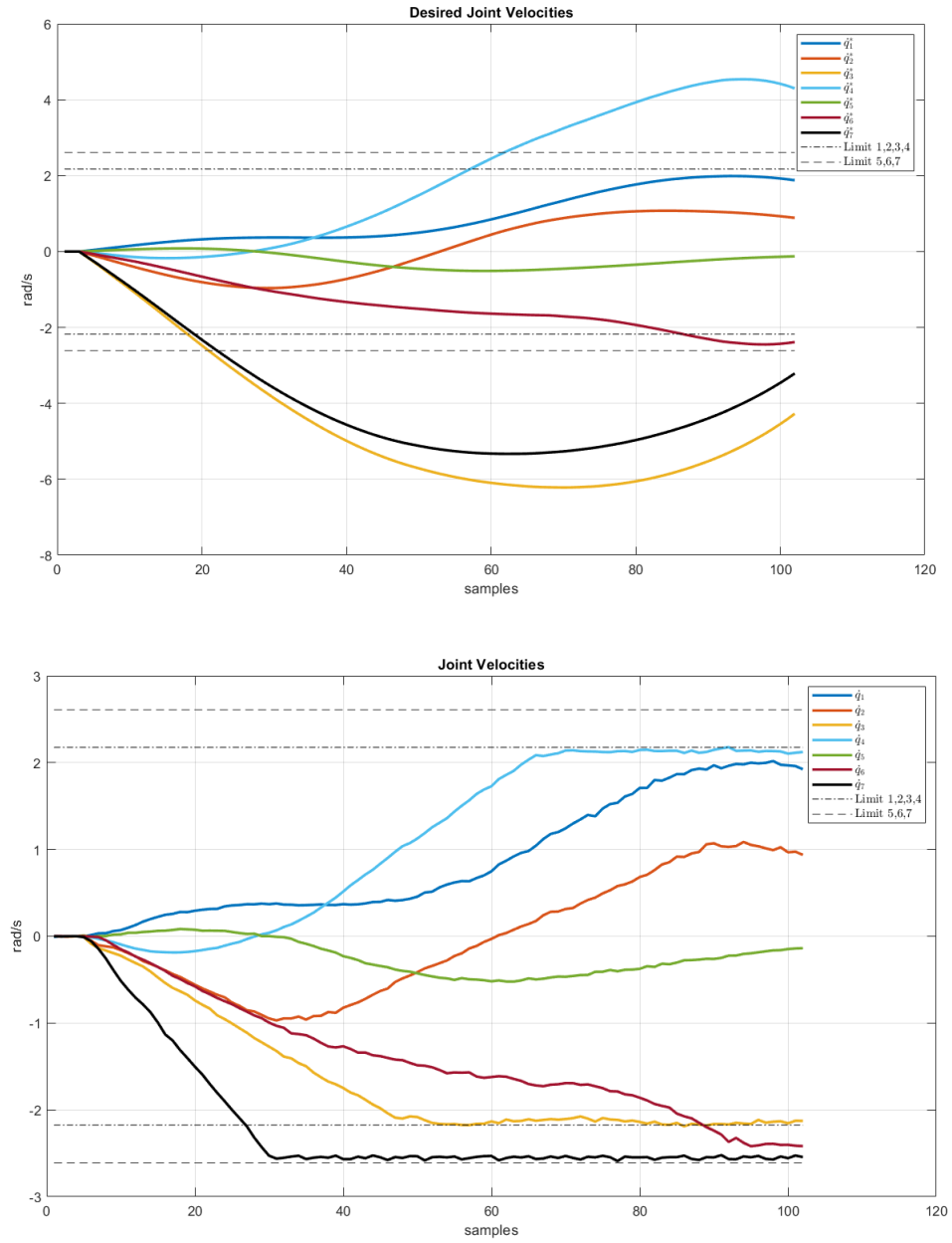


Figure 6.40 Desired and effective joint velocities for a 0.5 s trajectory.

Weighted Jacobian

Some further tests have been performed by applying a weight matrix to the Jacobian, in order to penalize the saturated joints and redistribute the load over the motors.
In particular this technique is implemented in the task which computes the pseudo-inverse Jacobian: before the computation through *singular value decomposition* the matrix W is

defined as a $7x7$ diagonal matrix where each element represents the weight of the i-th joint w_i [9].

$$W = \begin{bmatrix} w_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & w_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & w_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & w_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & w_6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & w_7 \end{bmatrix} \tag{6.8}$$

Then the weight matrix for the joints is built on the elements of W as follows:

$$W_q = \begin{bmatrix} \frac{1}{\sqrt{w_1}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{w_2}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{w_3}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{\sqrt{w_4}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{\sqrt{w_5}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{w_6}} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{w_7}} \end{bmatrix} \tag{6.9}$$

The robot Jacobian is then multiplied by the W_q obtaining J_w, which is the matrix which will be inverted through SVD.
While computing the pseudo-inverse, the matrix obtained by SVD must be again pre-multiplied by W_q.
The values of the weights are assigned at the diagonal elements of W, and it's important to choose them very close to 1: slightly smaller (i.e. 0.8) for adding load to the joint or slightly greater (i.e. 1.2) for reducing the load.
In the further results all the joints which encountered saturation problems are weighted with a value of 1.2 while the other elements of the diagonal are kept to 1.

The following results have been obtained from a 0.7 s trajectory like the one previously discussed, starting from neutral position and lowering the load on joints 3, 4 and 7 through a weight matrix when computing the inverse Jacobian.
The application of weights helps to keep all the desired joint velocities, and so the effective ones, within the saturation limits.

Unfortunately even if this strategy works for a 0.7 s trajectory, some values are very close to the limits, so the workaround won't avoid saturation in case of a trajectory performed within 0.5 s.

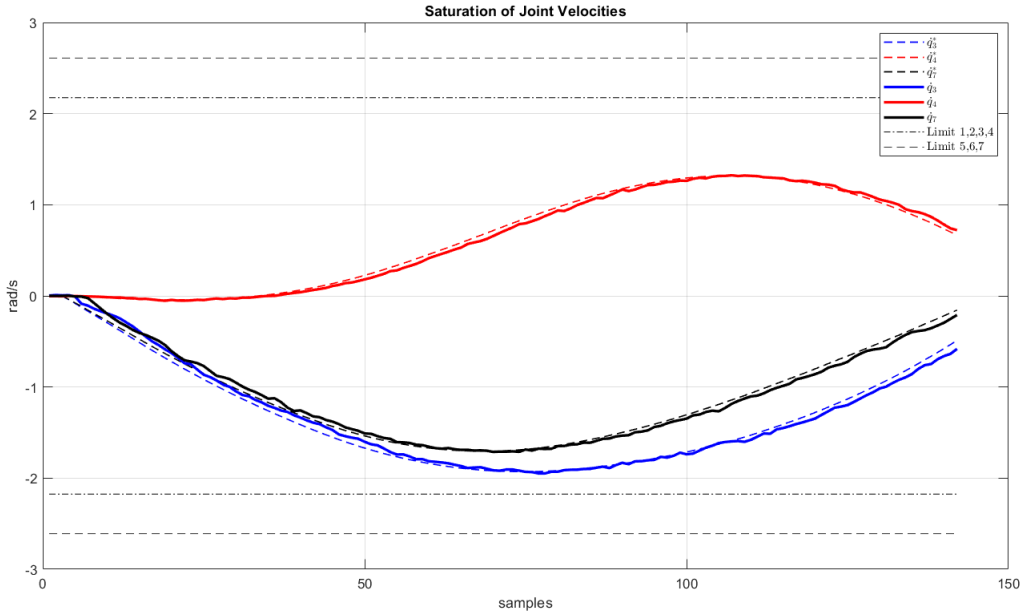


Figure 6.41 The joints are no more saturated, the effort has been distributed to the other joints.

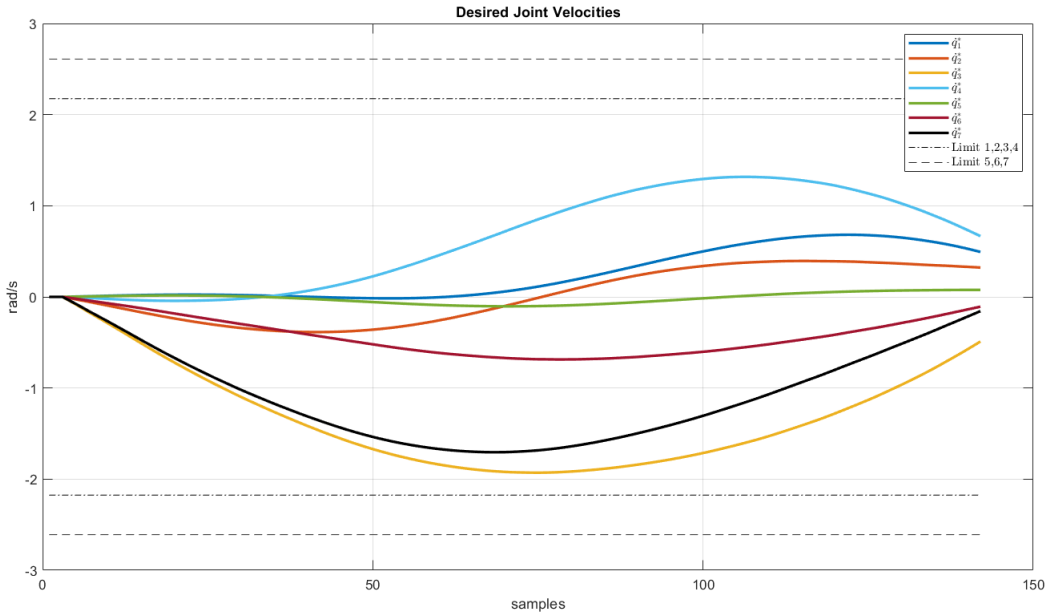


Figure 6.42 Desired joint velocities after the weighting of the Jacobian.

In order to achieve a smooth and fast motion while tracking faithfully the polynomial Cartesian trajectory the effort on the motors must be further reduced.
The desired joint velocities computed from the Cartesian expression of the trajectory must be kept far from the limitations imposed by the controller.
Apart from the weight matrix which has been proven to give some benefits, the proposed solution is to perform an initial approaching of the catch region, in particular along the y coordinate, where the error is greater.
This is done by moving the robot with a simple proportional control law in velocity, with fixed goal coordinates on x and z and the y coordinate of the ball before the throw, when is still holden by the user.
The following plots show how these two precautions help to send feasible joint velocities to the controller even in case of 0.5 s trajectories.

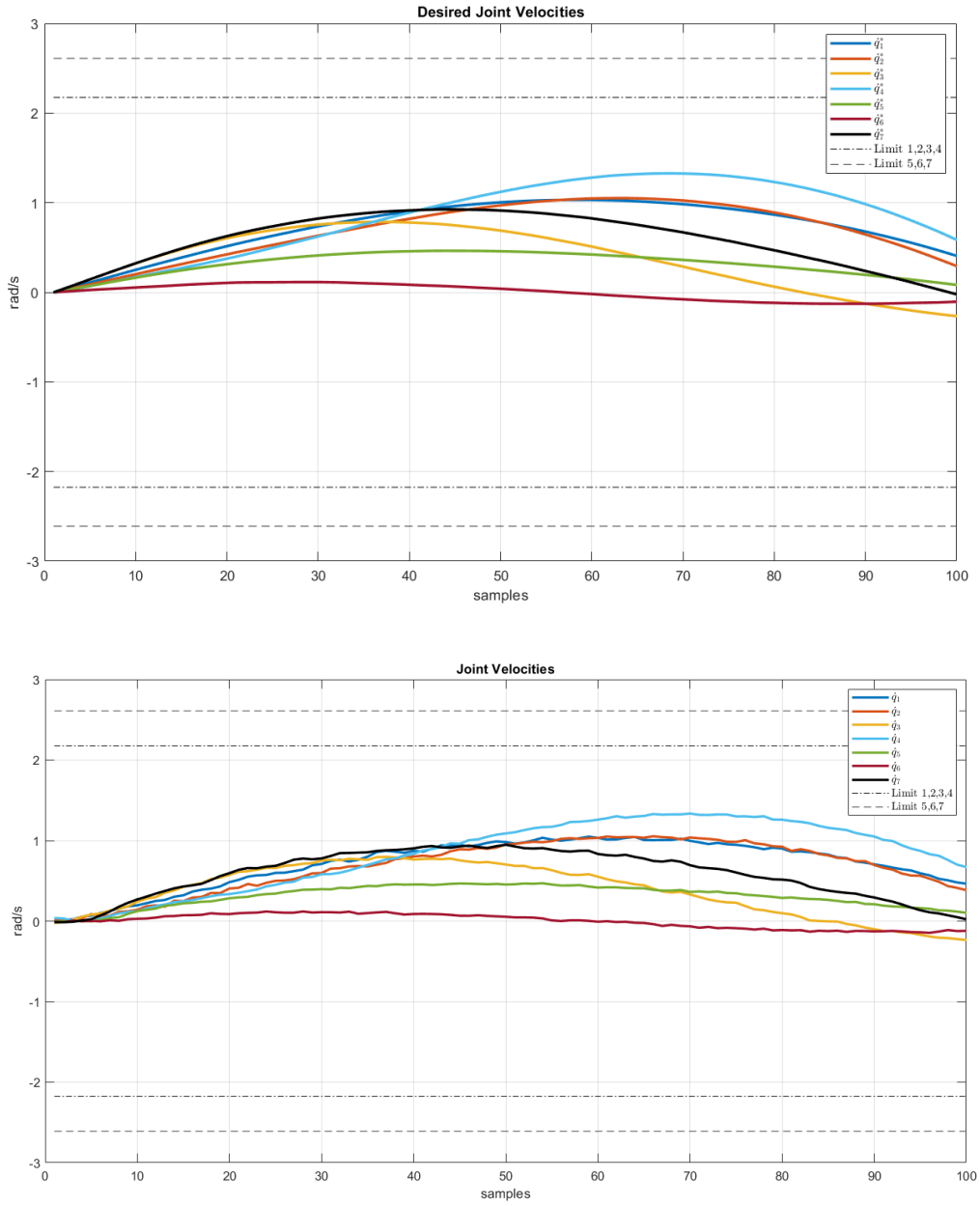


Figure 6.43 Desired and effective joint velocities for a 0.5 s trajectory, with initial approaching of the catch region.

The results of the motion analysis for a prehensile catching problem justifies an initial movement towards the catch region, in order to reduce the effort required from the manipulator.

Moreover the benefits of a weighting technique on the Jacobian matrix have been exploited, and for what concerns the prehensile ball catching problem it must be said that it can be

solved within a feasible amount of time, without drifts caused by robot's motion and with a position accuracy limited only by the precision of the Kalman filter estimation.

6.3.2 Non-prehensile Ball Catching

The motion capabilities of the robot have also been tested for the non-prehensile ball catching scenario, which can yield more difficulties since the polynomial trajectory must be computed in order to comply with non null terminal velocities.
This means that the catch point must be reached at a certain velocity which corresponds to the velocity of the ball, in order to avoid bouncing: this kind of motion is obviously more complex and energy-consuming that the simple point reaching, so for equivalent distance and catch time the effort required to the actuators will be higher.
It must be specified that all the experiments from this point on have been performed with a catch time of 0.5 s and weighted Jacobian.

Low starting point

The first starting point choice is in the lower part of the catch region, as shown in the next figure: the prefixed goal is to achieve a 'back-and-forth' motion in order to damp the motion of the ball at the catch point.

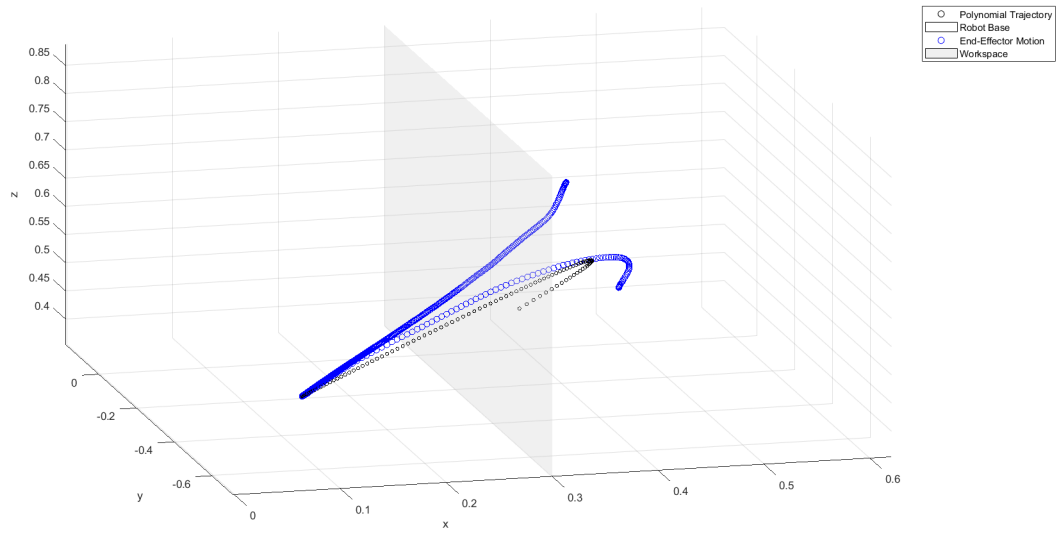


Figure 6.44 End-effector motion in space, with low initial point.

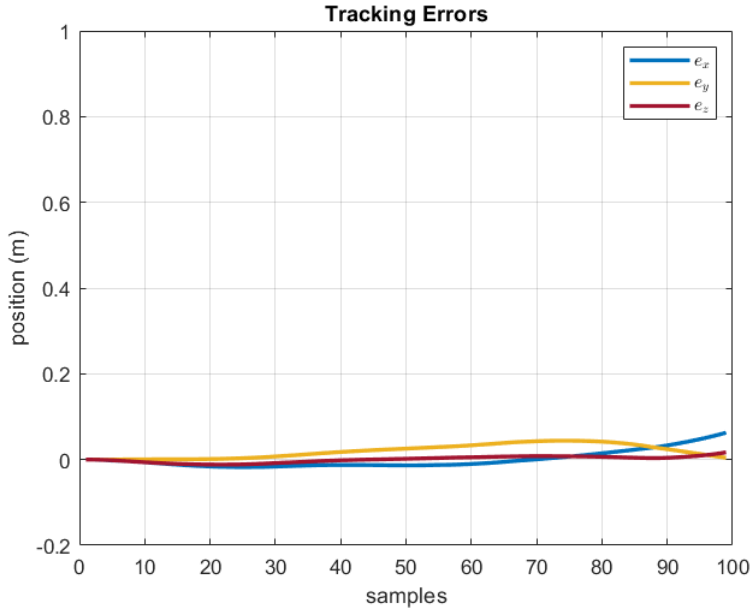


Figure 6.45 Position errors.

What is noticeable from the results is that the tracking of the trajectory is overall good, until the motion inversion happens: the trajectory is driving the end-effector in order to reach the reference point with negative velocities along x and z, corresponding to ball's velocities. The Cartesian linear accelerations computed by deriving the polynomial show that the limitations are violated in correspondence just after the turning point, issue which can be seen even by observing the desired velocities.
This obviously reflects on the positional tracking: apart from the fact that the z coordinate of the catch point cannot be tracked exactly within 0.5 s, it's also interesting to notice that the y position converges to the coordinate of the catch point, but when the turn is performed it drifts away from the desired trajectory due to manipulator's limitations.

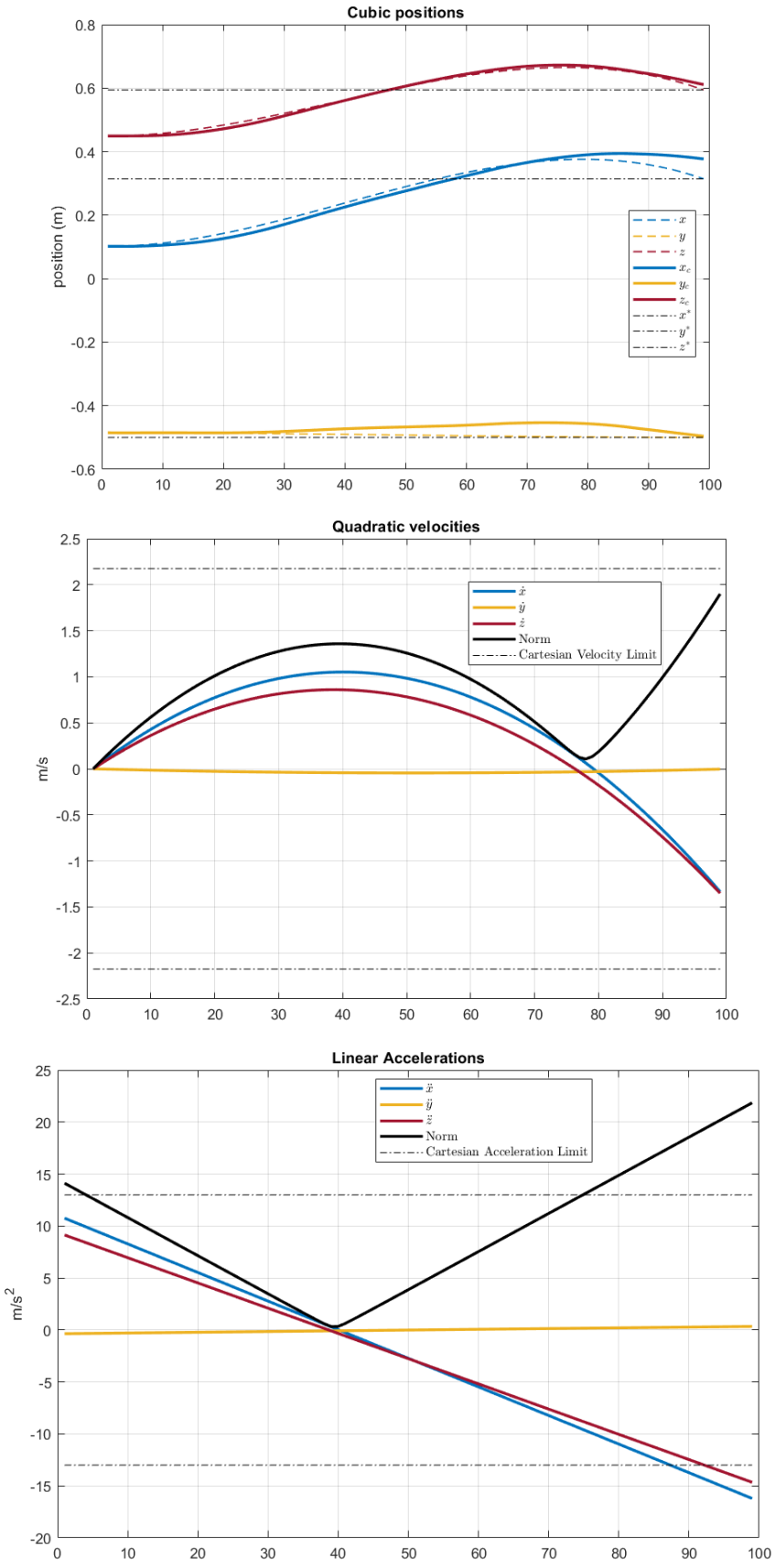


Figure 6.46 Position, velocity and acceleration with a low starting point.

By observing the joint velocities, apart from a saturation of the signals sent to the first two there aren't too much issues along the first phase of the trajectory.
As pointed out before the major issues arise when the backwards motion starts and the Cartesian velocities are inverted: even if the weights on the Jacobian allow a better distribution of the effort, the abrupt change of the motion direction.

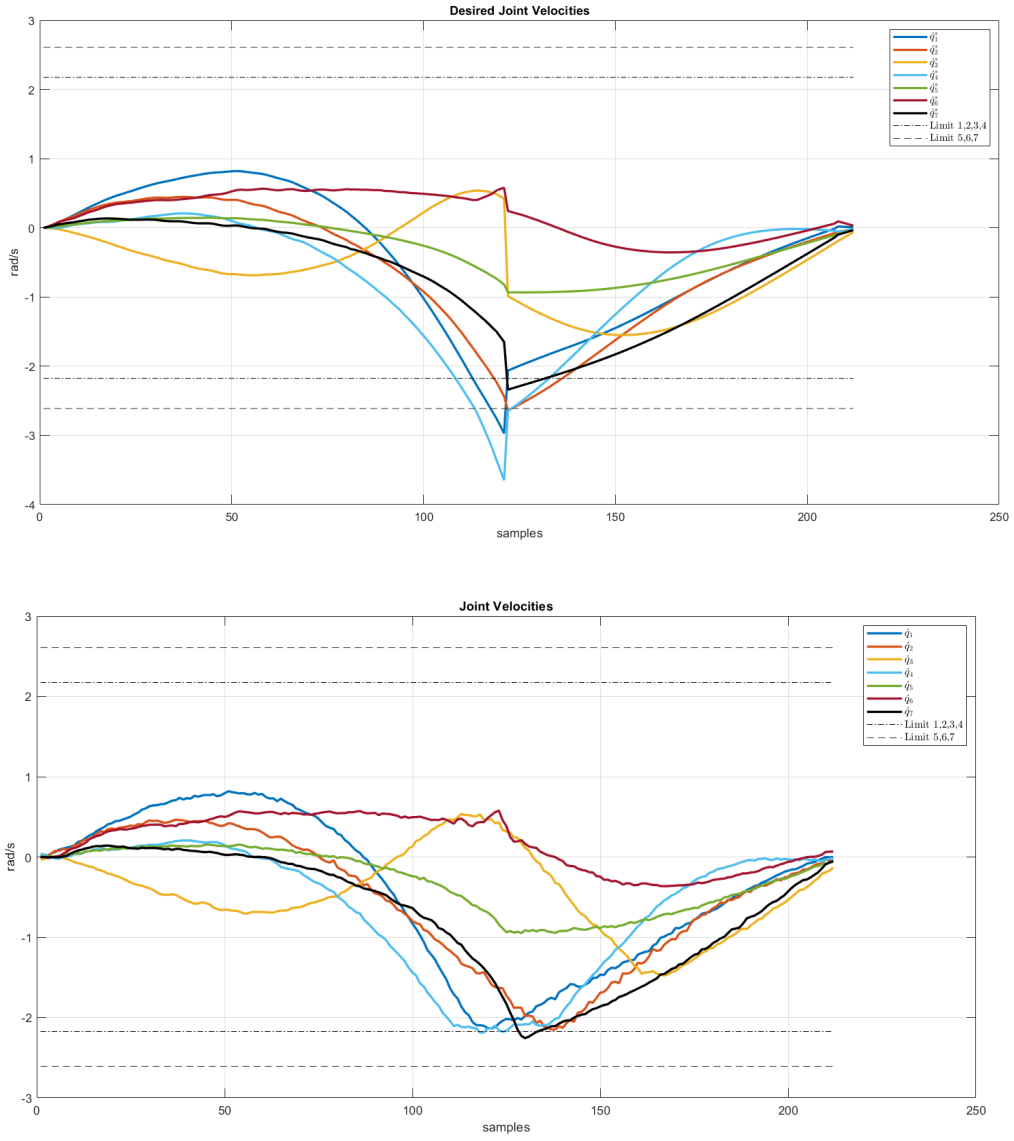


Figure 6.47 Desired and effective joint velocities with a low starting point.

High starting point

The adoption of a low initial point leads to a desired trajectory which performs a narrow curve when attempting to reach the catch point with desired velocities.
In order to keep the same overall behavior but with a smoother motion even when the change of direction occurs, the option of using a high initial point has been considered and implemented.
Basically the manipulator brings the end-effector in the same position specified before on x and y, but with an offset of 70 cm on the z axis.

The scatter plots show that a smoother desired trajectory lead to an improvement of the tracking, even if after the direction change of the end effector the same problems emerged before show up.

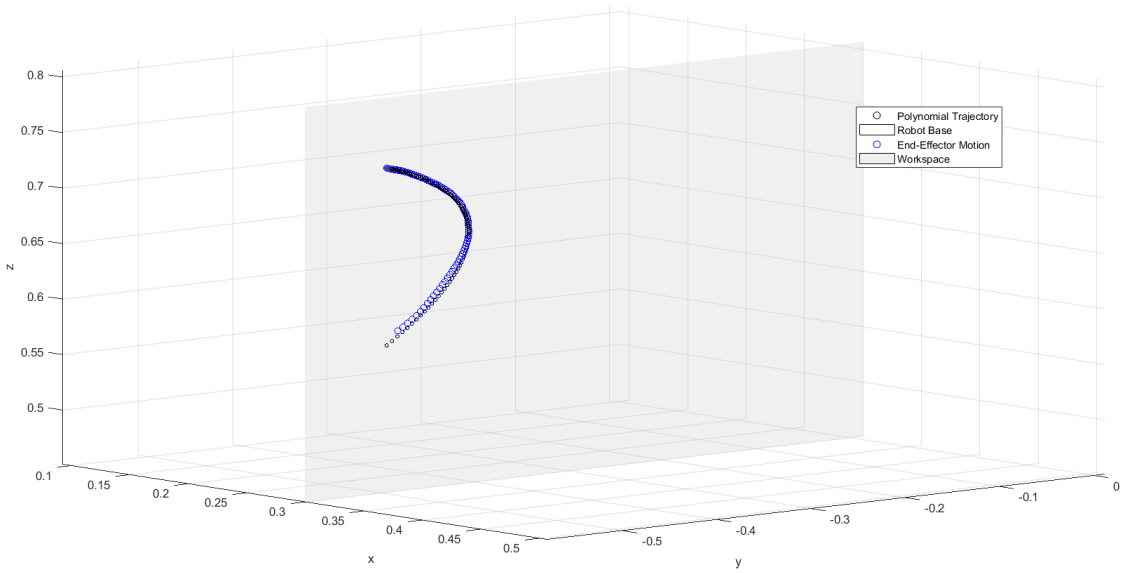


Figure 6.48 End-effector motion in space.

Even if the tracking along the whole trajectory is not achieved, a neat improvement has been reached in terms of the end-effector motion.
By observing the Cartesian position errors, the drift occurs only at the end of the trajectory, and its entity is of 1 cm approximately on each direction.
This behavior is caused, as before, by the accelerations which anyway assume values below

the thresholds even if very close to the limits.

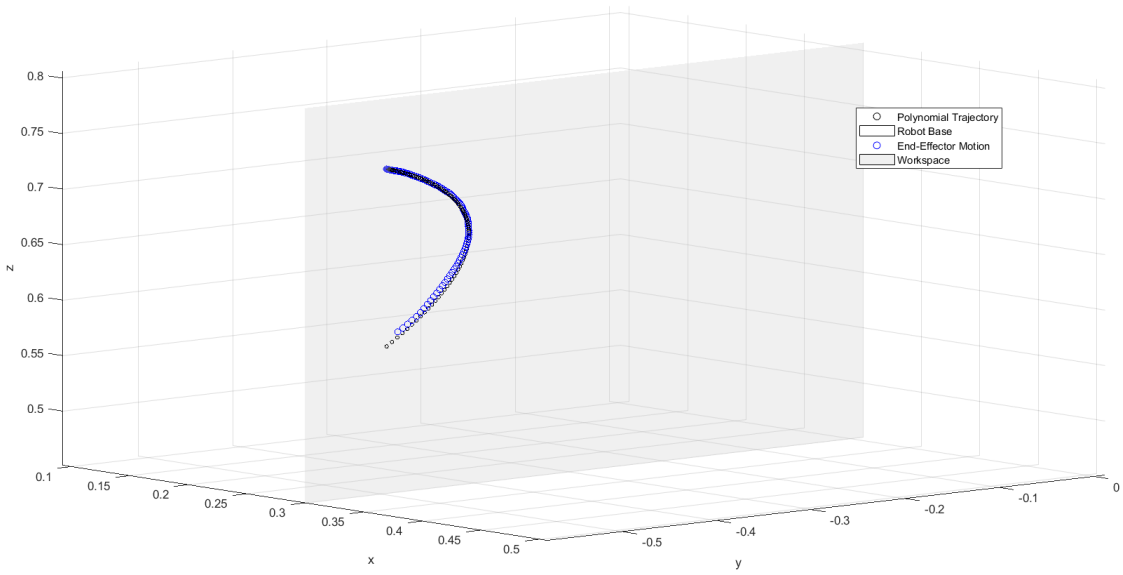


Figure 6.49 End-effector motion on the plane *xy*.

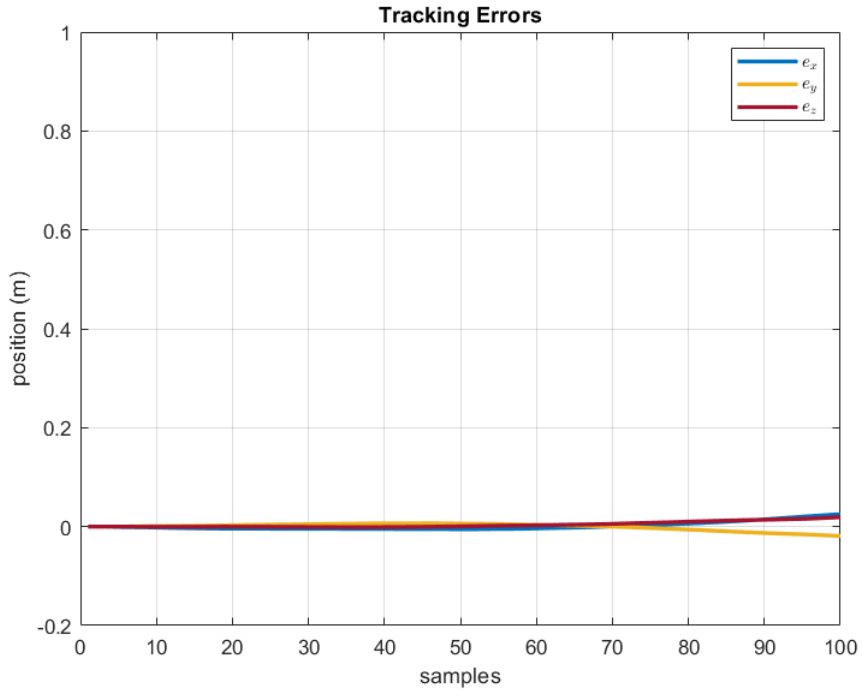


Figure 6.50 Position errors.

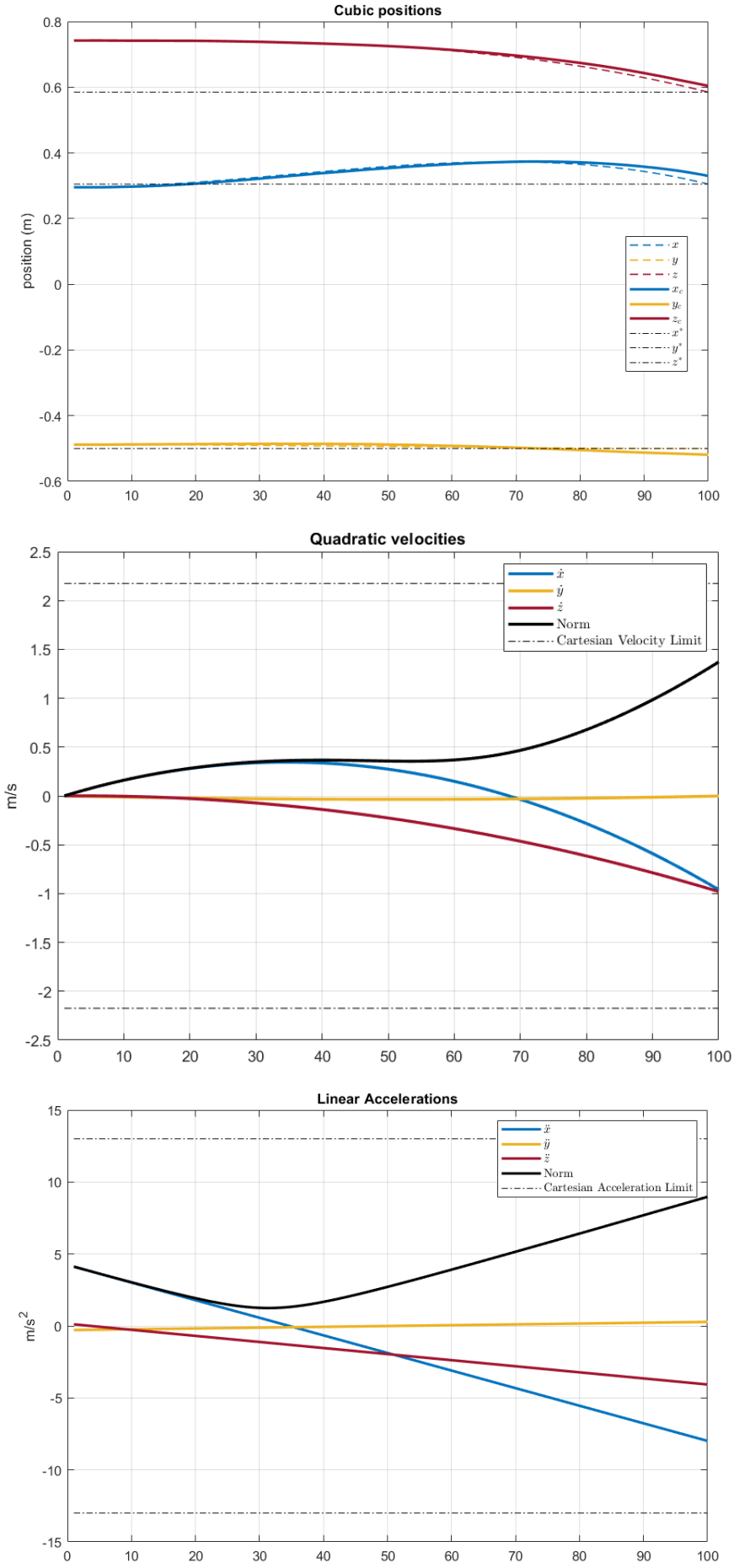


Figure 6.51 Position, velocity and acceleration with high starting point.

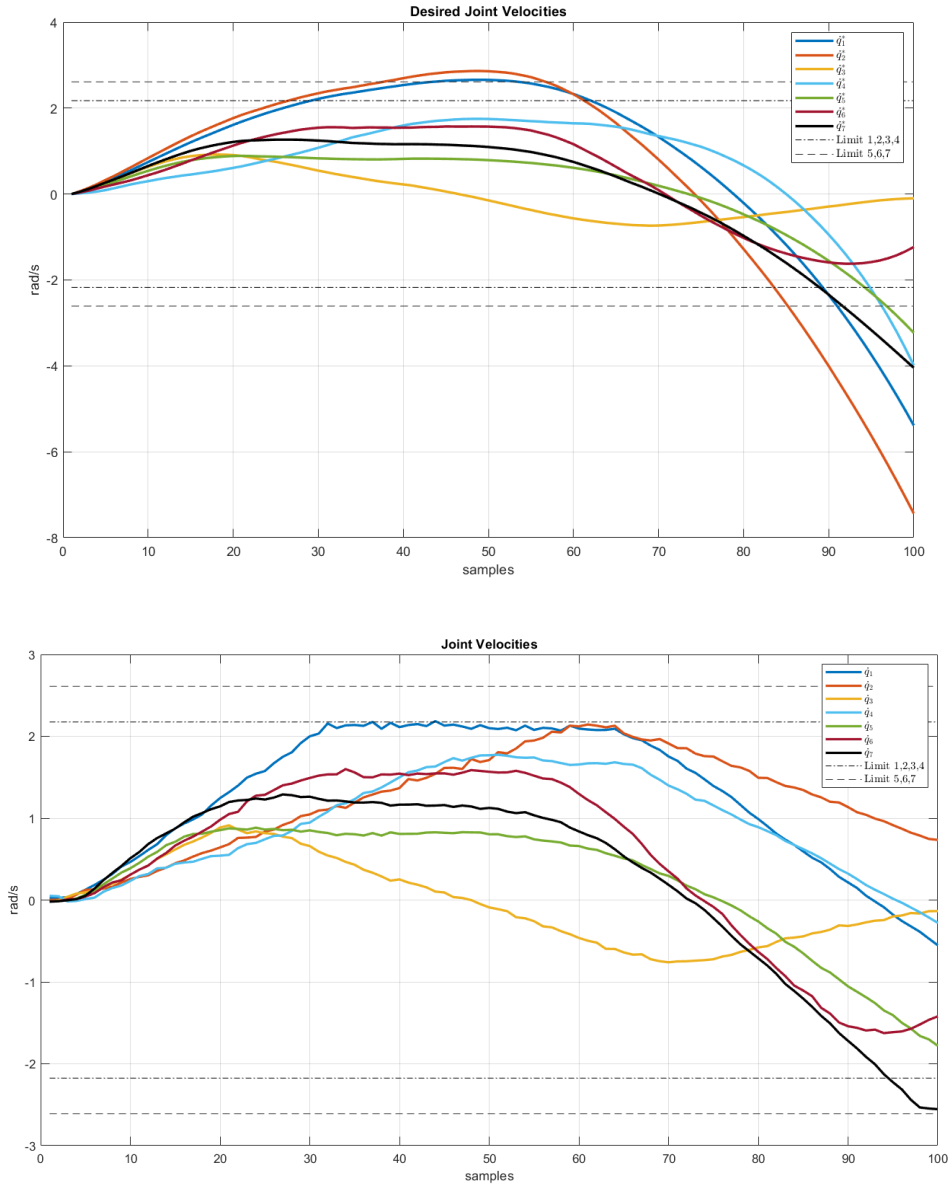


Figure 6.52 Joint velocities with high starting point.

6.4 Ball Interception and Catching

This final section presents the graphical results obtained by complete executions of the ball catching tasks.
Videos of the experiments can be found at https://github.com/francescogrella/RobEng_Thesis_Videos.

6.4.1 Prehensile Ball Catching

The following figure shows the estimation of the ball trajectory along with catch point predictions in the prehensile ball catch experiment.
The mismatch in the beginning is given by the fact that the real throw does not match completely the initial conditions of the Kalman filter. Anyway it can be seen that the predictions are close to the coordinates of the ball, and the robot catches it in 0.45 s.

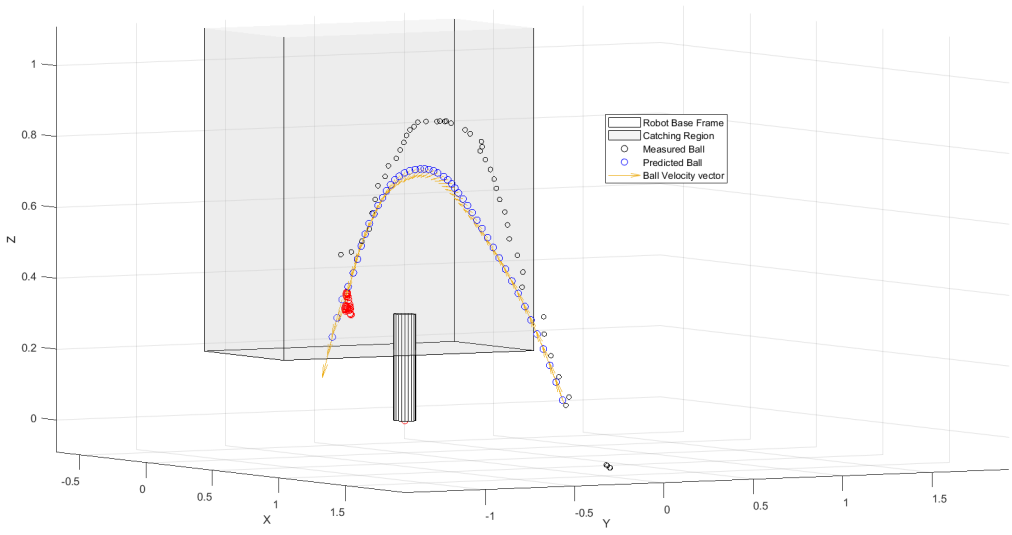


Figure 6.53 3D view of the predictions in the prehensile case.

The mismatch in the initial conditions reflects even on velocity and acceleration estimation, but in the prehensile case their correct estimation is less important than the non-prehensile case, since there is more tolerance on the predicted catch time.

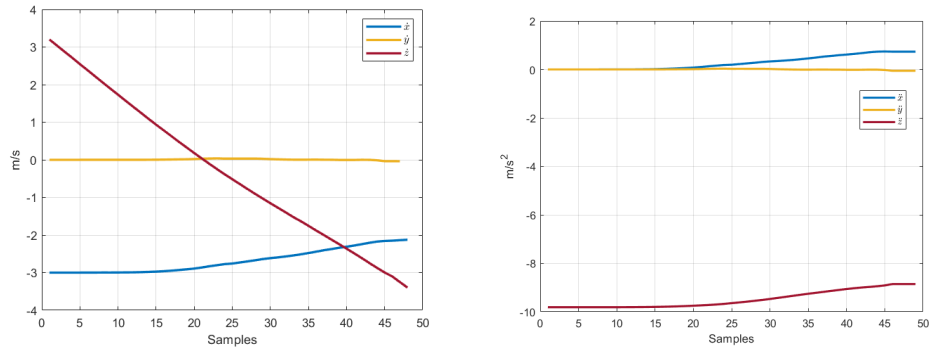


Figure 6.54 Estimated velocities and accelerations.

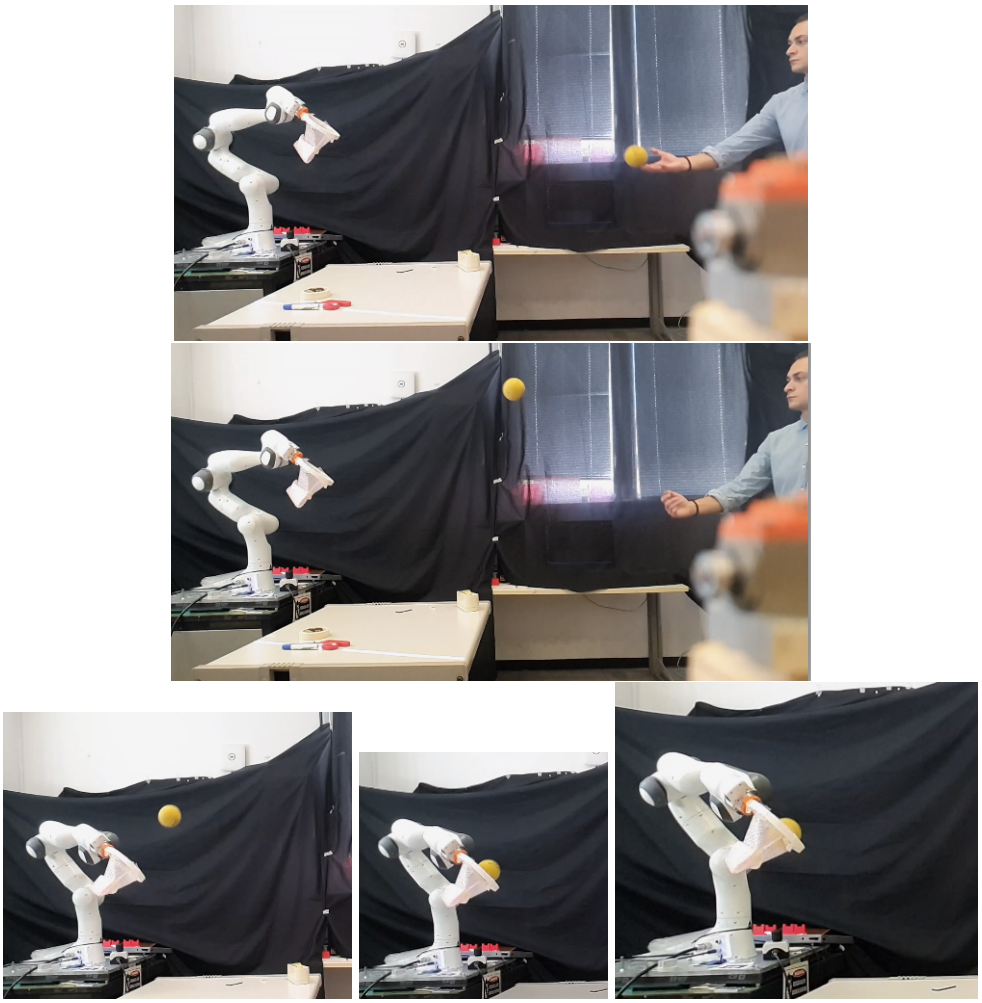


Figure 6.55 Prehensile catching sequence.

6.4.2 Non-prehensile Ball Catching

The experiments related to non-prehensile ball catch also provided very interesting results, even if the task cannot be entirely accomplished.
In particular it has been demonstrated that starting from a point which is above the average throw region, along with weights on the Jacobian matrix, allows to avoid the saturation of the joint velocities: even when performing orientation control the desired trajectory is almost completely tracked.

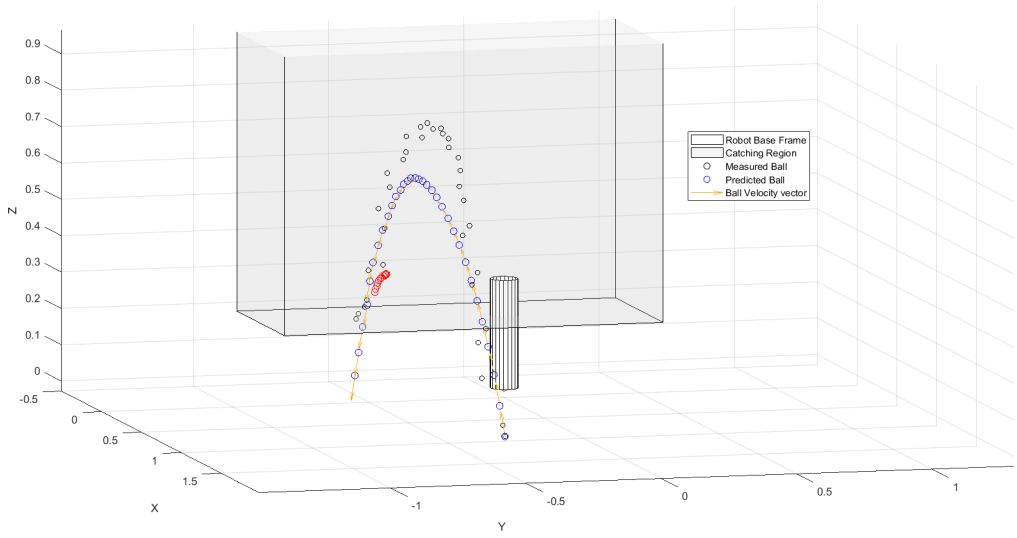


Figure 6.56 3D view of the predictions in the non-prehensile case.

Even in this case a slight mismatch of the initial conditions has occurred, but the predictions were not affected too much.

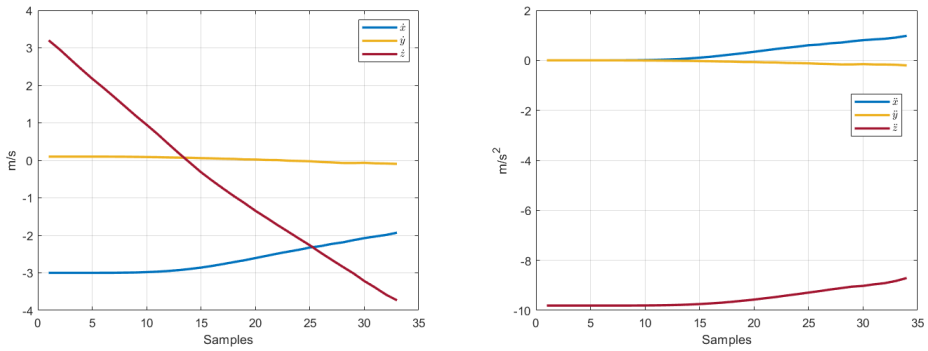


Figure 6.57 Estimated velocities and accelerations.

By observing the end-effector motion in 3D space it can be seen that the catch point is intercepted with an error of approximately 1 cm or smaller on the three axes.
It's important to notice that in the following plots even the deceleration phase is considered, where the final velocities of the trajectory are reduced by a constant value and no desired trajectories are computed.

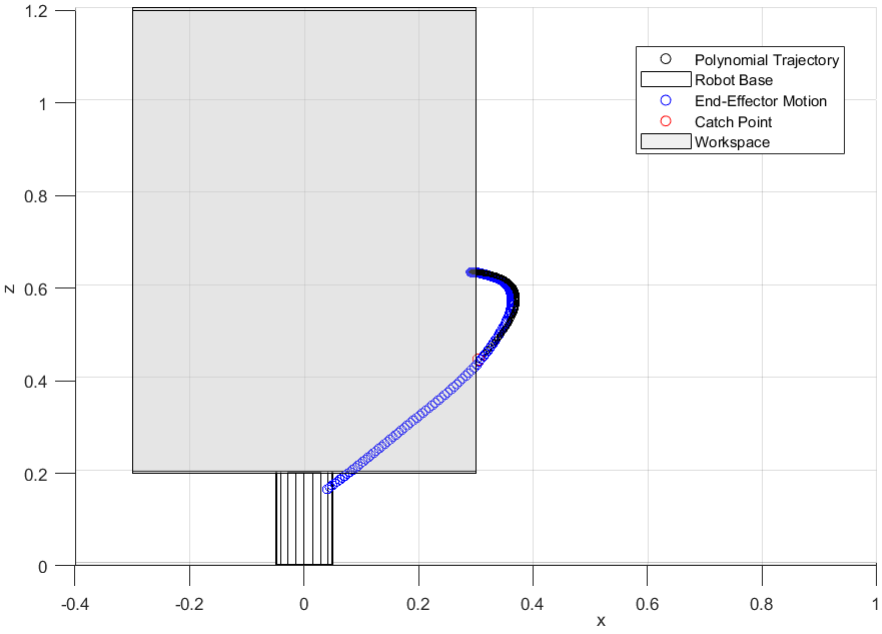


Figure 6.58 XZ plane view of end-effector motion in the non-prehensile case.

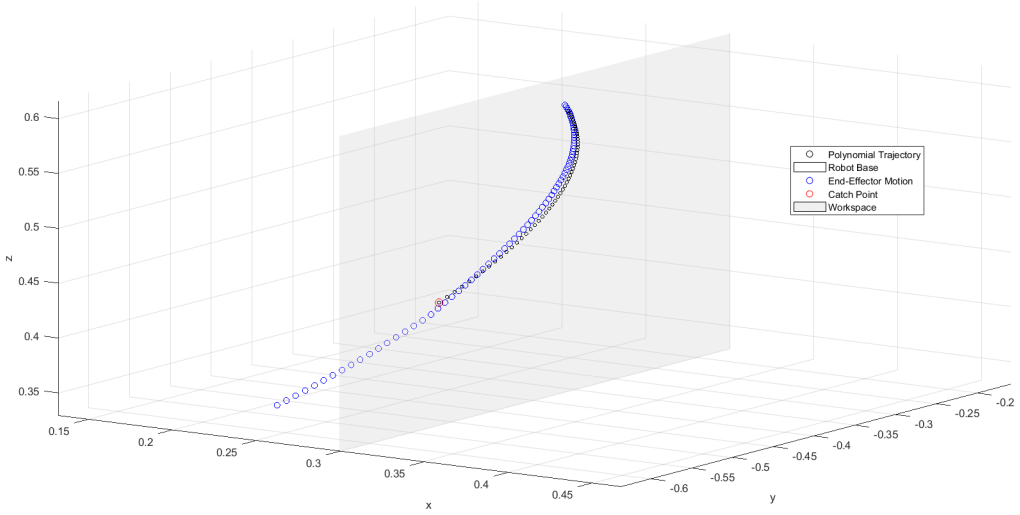


Figure 6.59 Zoom on the predicted catch point.

The accuracy of the result is remarked by the following figures, in particular before the deceleration phase the position errors between the end-effector and the catch points are almost null.
Unfortunately the previously discussed physical limitations effects can be seen by observing the velocities of the end-effector at the end of the trajectory: those values represent the limits of the terminal velocities which can be given for the polynomials, and going beyond results in a complete stop of the manipulator which returns a *constraint violation error*.

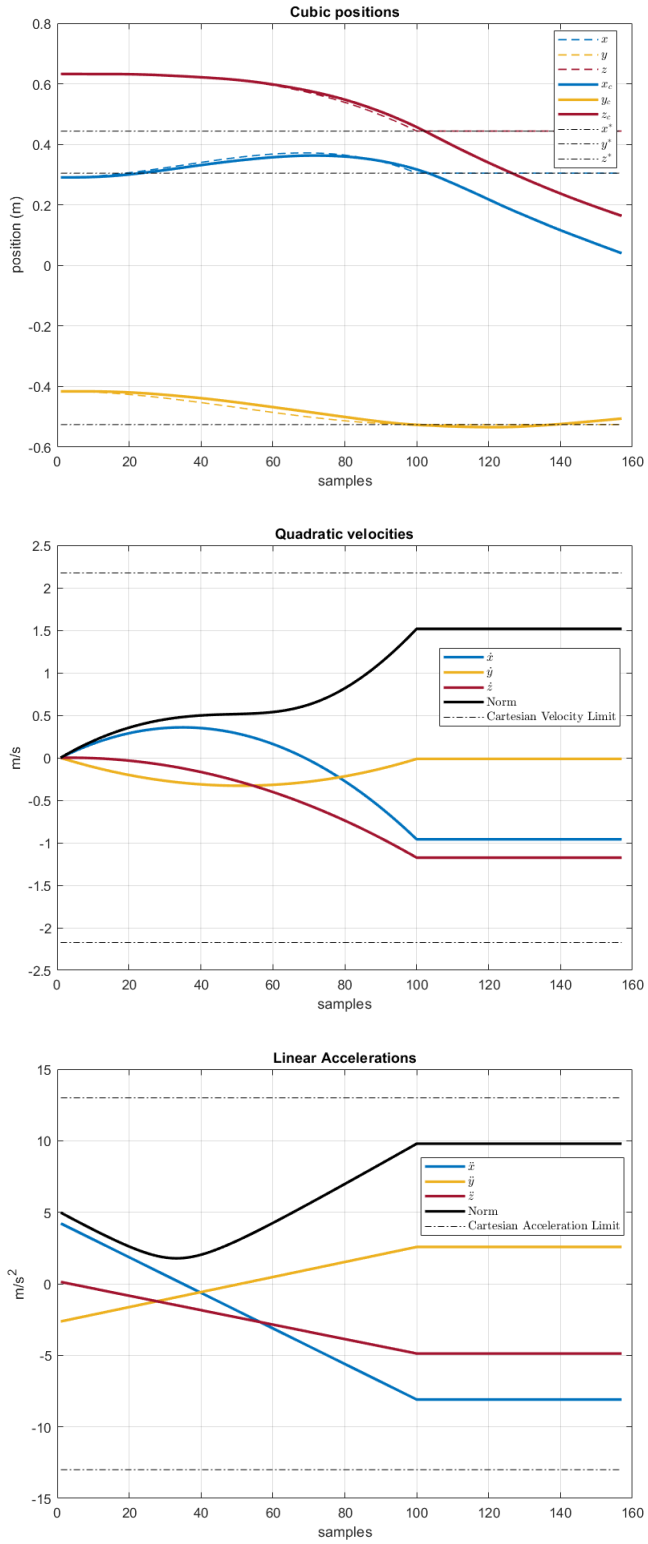


Figure 6.60 Position, velocity and acceleration in the non-prehensile catch.

Finally some considerations can be done for orientation control, which is the other important goal in non-prehensile catch experiments.
As shown in the next figures, the orientation of the end-effector is controlled in order to match the desired one which is computed by the estimation algorithm.
The convergence is not complete due to the already mentioned issues in the change of direction, but the amount of error is quite small.

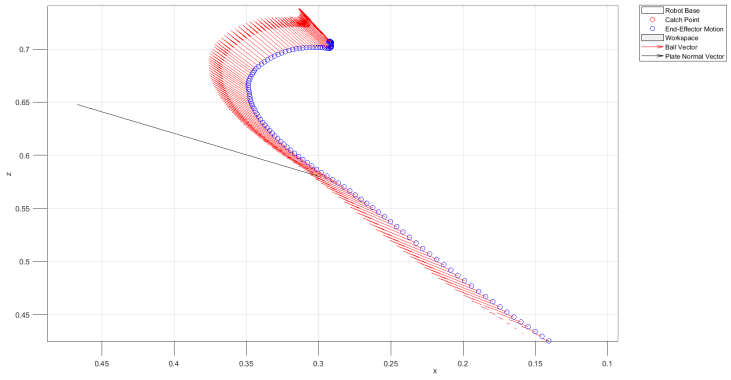


Figure 6.61 3D view of end-effector orientation in the non-prehensile case.

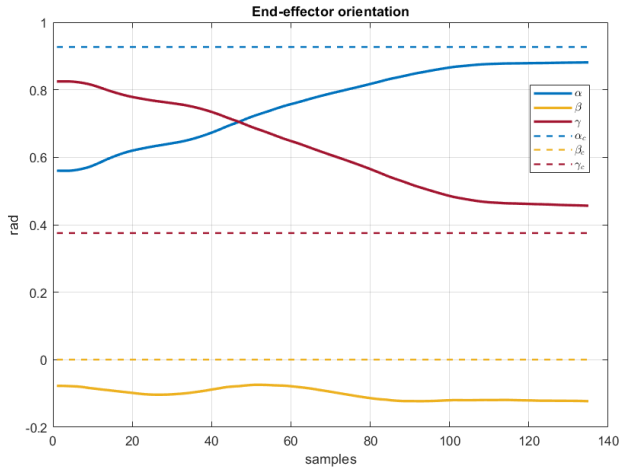


Figure 6.62 End-effector orientation in the non-prehensile case.

For the non-prehensile ball catch experiments it can be shown that the position, time and orientation constraints are almost completely met, but the limited terminal velocities of the trajectories only allow a damping of the bounce, not its total avoidance.

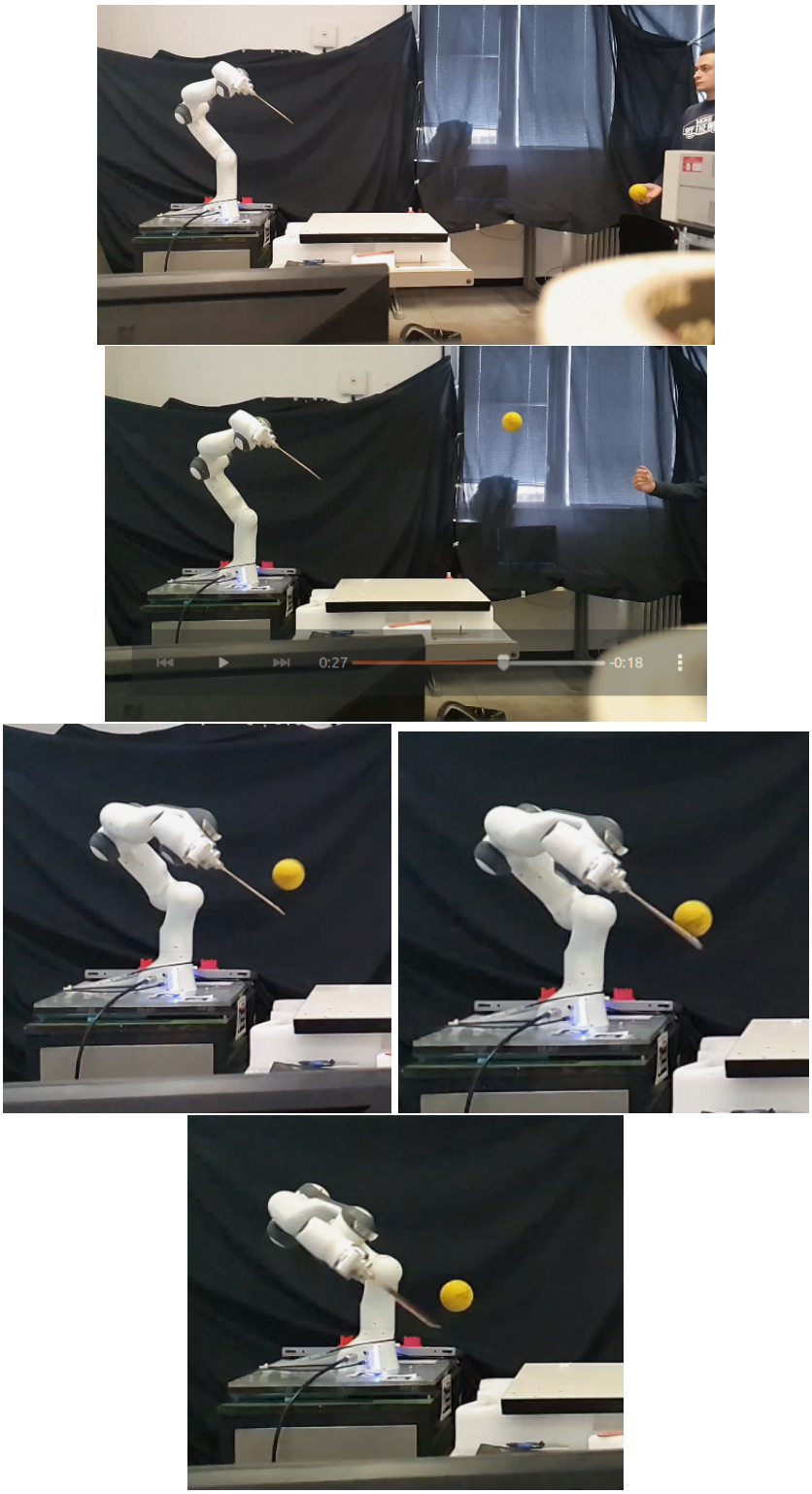


Figure 6.63 Non-prehensile catching sequence.

Chapter 7

Conclusions and Future Work

7.1 Conclusions on Experiments

Regarding the results of the experiments shown and commented along the previous section, the following conclusions can be drawn regarding the whole project.

- For what concerns the detection of the target object, an accuracy of about 3 cm along each direction of the Cartesian space has been achieved with the proposed algorithm. Moreover the presence of a simple GUI raises the robustness of the system with respect to different light conditions and most importantly different configurations of the scene: the capability of adjusting the HSV thresholds by observing both the scene and the binary image is essential for isolating the target object.

- As discussed in the experimental results section, the prediction of the catch point presents some limitations related to the state initialization of the Kalman filter, but when the conditions are matched the time and position accuracy fairly match the expected results.
  In particular the catch points coordinates projected on the yz plane converge to the real trajectory for very small values of the catching time, and within the valid interval the error does not exceeds 4 cm on both axes.
  The whole processing and estimation algorithm has almost no impact on the 60 Hz frequency of the camera: the sampling time of the catch points is approximately 165 ms.

- The results given by the accomplishment of the prehensile ball catch task have shown that with the proposed architecture the goal is achieved.

The designed end-effector can reach the computed catch point within the predicted amount of time, which is enough for intercepting the motion and for stopping the ball. Moreover this task brought to an initial evaluation of the physical limitations of the Franka manipulator, and led to the adoption of some auxiliary techniques in order to properly achieve the goal.

- When performing experiments on the non-prehensile ball catching task, the prefixed goal was to achieve an interception of the object by using a plate mounted on the end-effector avoiding the bounce.
  To pursue this result there have been performed several experiments adopting some measures in order to help the manipulator, since the effect of the limitations which came out before has been amplified by the effort-demanding terminal conditions of the trajectory.
  The experiments have shown that the position error between the catch point and the center of the plate was quite low, similarly to the previous case.
  The same can be said for the catching time and for the orientation of the end-effector, which makes the plate aligned with the orientation of the ball when reaching the catch point.
  The practical problems emerged in the velocity matching part: to avoid the bounce of the ball on the surface the plate should travel through the catch point with velocities very close to those estimated for the ball, but this constraint requires a motor effort which cannot be avoided by the weighting technique or by reaching a better initial position.
  The robot can still perform the motion, but due to this physical limitation the final end-effector velocity cannot match the ball velocity, so the bounce can only be reduced, but not totally avoided.
  A possible future development of the project comprehends the further investigation of the physical limitations of the robot, along with seeking for different techniques to avoid the saturation of the joints in an optimized way.
  The modularity of the control architecture allows to implement other tasks for sending different commands to the joints, so an *Optimal Control* policy can be adopted instead of the actual trajectory planning based on polynomials.
  Moreover different sensing and estimation architectures could be considered, like an eye-in-hand camera configuration with a non-linear trajectory estimator like depicted in [7].

Appendix A

Mathematical Appendix

Intrinsic Camera Calibration

Before starting any kind of image processing operation, the user must ensure that camera's intrinsic parameters are properly calibrated by using ad-hoc techniques and instruments. Even pointed out as *camera re-sectioning*, this is the process of estimating the parameters, usually are represented in a 3×4 matrix called the camera matrix, of a pinhole camera model approximating the camera that produces a given image.

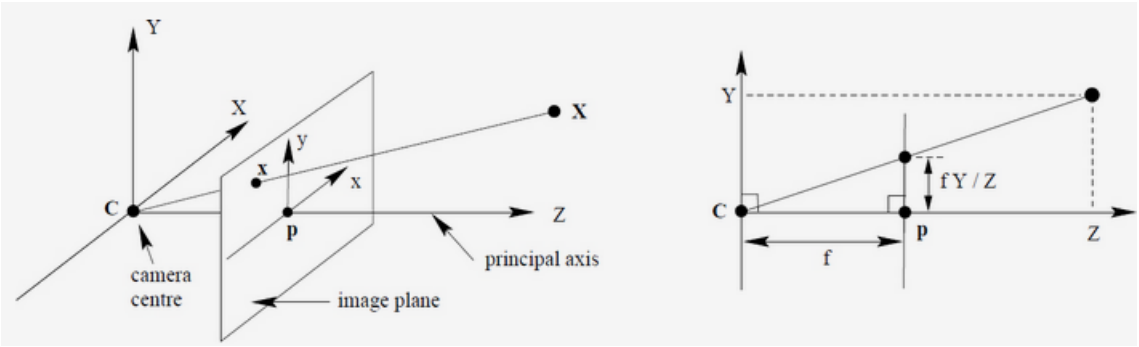


Figure A.1 The Pinhole Camera model.

Assuming to have a generic pixel coordinates vector, represented in homogeneous coordinates as $[u\ v\ 1]^T$ and $[x_w\ y_w\ z_w\ 1]^T$ is used to represent a 3D point position in world coordinates.

The transformation between these two coordinates sets is described as

$$z_c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} R & T \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \tag{A.1}$$

where K is the *Intrinsic parameters matrix*, while R and T are rotation and translation blocks of the *Extrinsic parameters matrix*.
While R and T are computed in the process described in the Experimental Setup section, the matrix K is the following:

$$K = \begin{bmatrix} \alpha_x & \gamma & u_0 & 0 \\ 0 & \alpha_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{A.2}$$

These informations describe focal length, image sensor format, and principal point.
The parameters $\alpha_x = f \cdot m_x$ and $\alpha_y = f \cdot m_y$ represent focal length in terms of pixels, where m_x and m_y are the scale factors relating pixels to distance and f is the focal length in terms of distance [29]. The γ parameter represents the skew coefficient between the x and the y axis, and is often set as 0. Finally u_0 and v_0 represent the principal point, which would be ideally in the center of the image.
Often commercial cameras have a ready-to-use tool for compute intrinsic camera parameters and save them in a configuration file which will be read at any initialization.
The calibration file contains information about the precise position of the right and left cameras and their optical characteristics: is a key feature in the depth estimation process and guarantees its quality.
Usually this file is generated during our manufacturing process, but parameters can be re-computed through the aforementioned tool or even online.

Angle-axis Representation

Given the two unit vectors, where $\mathbf{R_B^{EE}}$ expresses the end-effector frame orientation in the base link frame:

$$\hat{\mathbf{n}} = [R_B^{EE}(0,0), R_B^{EE}(1,0), R_B^{EE}(2,0)] \tag{A.3}$$

$$\hat{\mathbf{h}} = [v_{xb}, v_{yb}, v_{zb}] \tag{A.4}$$

then is always possible to define a plane on which the two unit vectors lie and the rotation is bounded to it: by applying the *wedge product* a vector orthogonal to the plane can be

computed, and is the **rotation vector**.

$$\mathbf{e} = \hat{\mathbf{n}} \wedge \hat{\mathbf{h}} \tag{A.5}$$

From the expression (5.29) the axis and the sine and cosine of the rotation angle can be computed without any effort:

$$sin\theta = \|\mathbf{e}\| \tag{A.6}$$

$$cos\theta = \hat{\mathbf{n}} \cdot \hat{\mathbf{h}} \tag{A.7}$$

Appendix B

Hardware Supports Design

ZED Camera support

Base

The base of the support is designed to be fixed to the wall though two screws with a diameter of 5.5 mm.
On its front side an extrusion of 9 cm ensures that the camera can rotate of approximately 70° both on the left and the right, in order to ensure a full coverage of the room when acting on the orientation of the first joint.
At the end of the extrusion there is a 5.5 mm hole, designed to contain the screw of the joint which will be the vertical rotational axis of the joint.

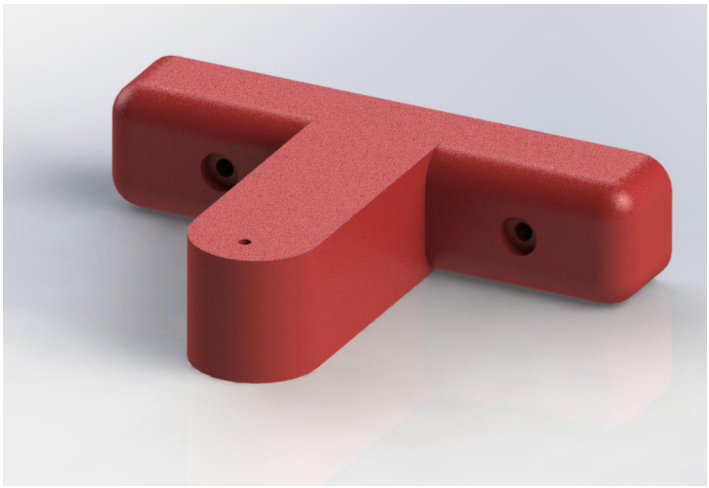


Figure B.1 The base of the camera support rendered with Solidworks tool *Photo360*.

Joint

The joint of the support is designed to provide two rotational degrees of freedom (under specific constraints) to the ZED camera.
Two holes have been practiced in order to fit two 5.5 mm screws: the first one is be the vertical rotational axis and is attached to the base, while the second, lodged in the chamfered part, is designed to be the rotational axis of the lodge for the ZED.

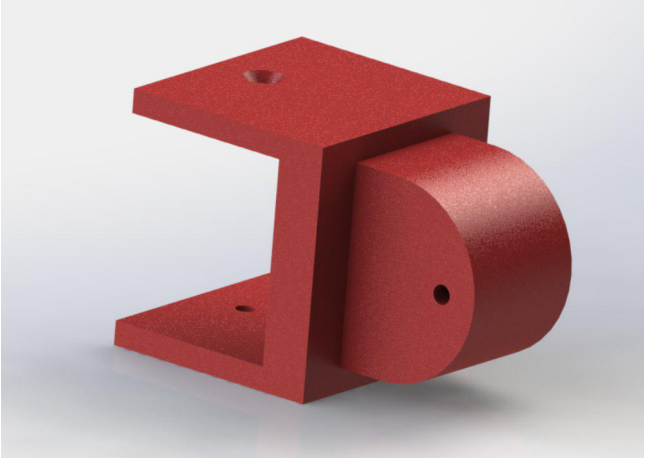


Figure B.2 The joint of the camera support rendered with Solidworks tool *Photo360*.

Table B.1 Approximate Joint Limits of the ZED Support

Joint	Max. Angle	Min. Angle
1	70°	-70°
2	30°	-30°

Slot

The last part of the support is quite important since it must provide mechanical support to the ZED camera as long as the experimental setup is set.
The lodge for the camera has been designed and built by keeping into account the exact dimensions of the camera in order to be safe and to avoid any loosening of the object itself, preventing any fall or undesired displacement.
The camera must be inserted from one side and slided until it stops due to its back USB 3.0 cable.

Two extrusions on the front side can lock the camera even if its orientation is not perfectly horizontal but points towards the ground, anyway due to the structure of the joint this part cannot be tilted too much, fact that increases the safeness of the support.

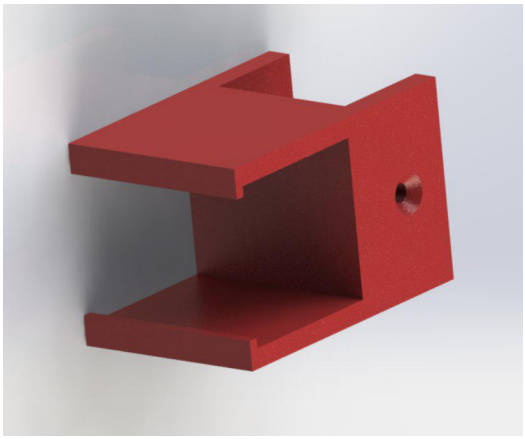


Figure B.3 The camera slot of the camera support rendered with Solidworks tool *Photo360*.

In the following figure a rendered view of the full assembly is provided, where the motion capabilities in terms of orientation of the slot can be visualized.
In its practical use is important to remark that the screws of the joints must be kept very tight in order to avoid angular displacements, and the tightness has been checked approximately every two weeks for the entire period of the experiments.

Figure B.4 The assembly of the camera support rendered with Solidworks tool *Photo360*.

Modular end-effector for ball-catching

In order to perform experiments with the robot, the original gripper provided by the vendor has been replaced with a modular end-effector designed and realized with the same techniques adopted for the camera support.

Base

The base of the end-effector has been sized with the exact measurements of the robot's flange, with two holes for fixing it with the gripper screws.

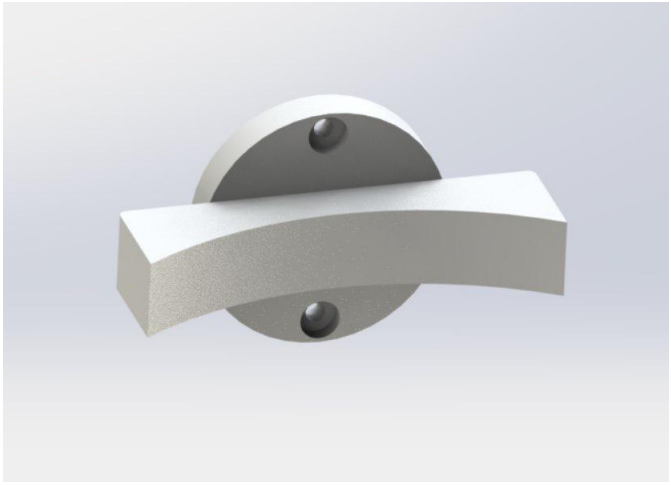


Figure B.5 Render of the end-effector base.

External Frame

The main component of the designed end-effector is a ring with an external diameter of 20 cm (maximum size allowed by the 3D printer) and an internal diameter of 18 cm.
It has been built with the purpose of supporting both prehensile and non-prehensile ball catching experiments, so it can be used alone by attaching a small net around it in order to stop the ball.
Moreover the internal section has two symmetric slots which allow a further assembly of the end-effector in order to perform different experiments.

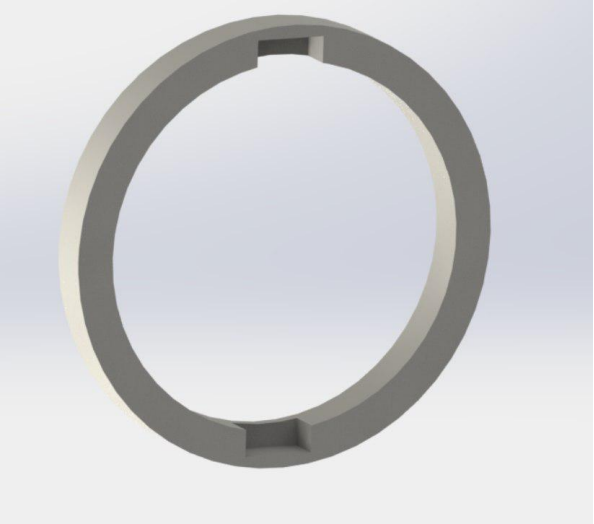


Figure B.6 Render of the external circular frame of the end-effector.

Inner Ring

For assessing and improving the precision of the performance in prehensile ball catching, another ring has been designed with the purpose of reducing the diameter of the first one. The external diameter is 18 cm large while the internal is 14 cm: given a ball with a diameter of 9 cm it represents a valuable obstacle to the task, and most importantly can drive further developments for improving the accuracy of the robot's motion.

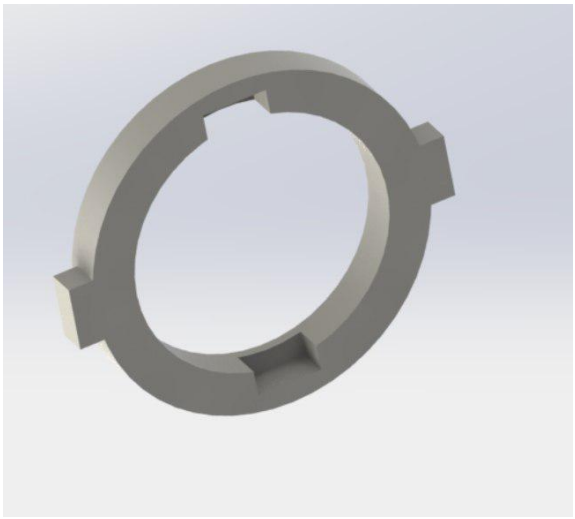


Figure B.7 Render of the inner ring.

Inner Plate

Another assembly has been conceived for performing non-prehensile experiments.
A plate with a diameter of 18 cm which can be assembled with the external ring through the slots has been designed for assessing the performance of the robot with non-null terminal velocities catching and trying to avoid the bounce of the ball on it.

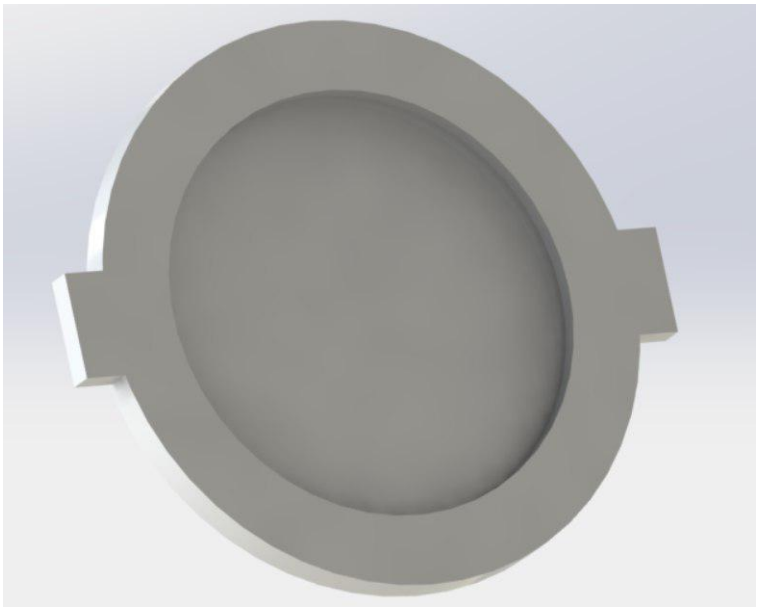


Figure B.8 Render of the plate for non-prehensile ball catching.

Here the assemblies of the two experimental configurations are shown, along with their exploded views.

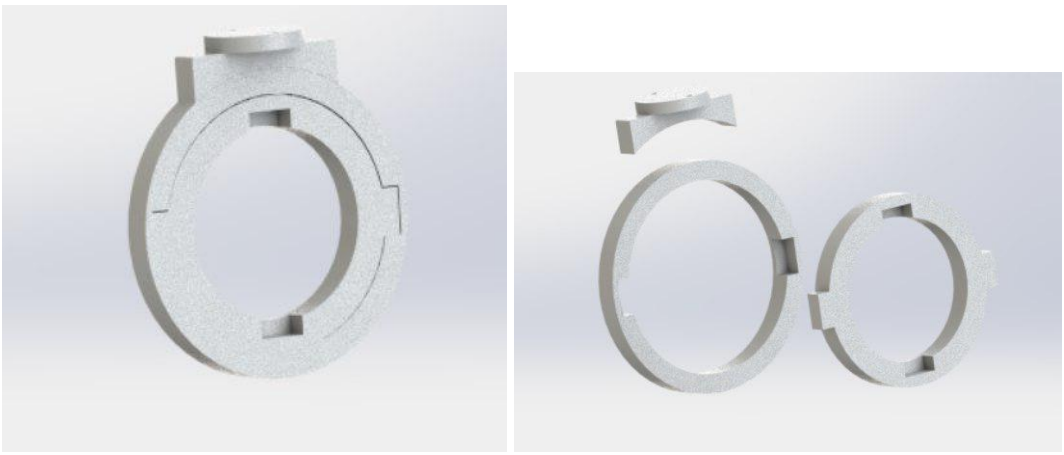


Figure B.9 Assembled and exploded views of the end-effector in prehensile mode.

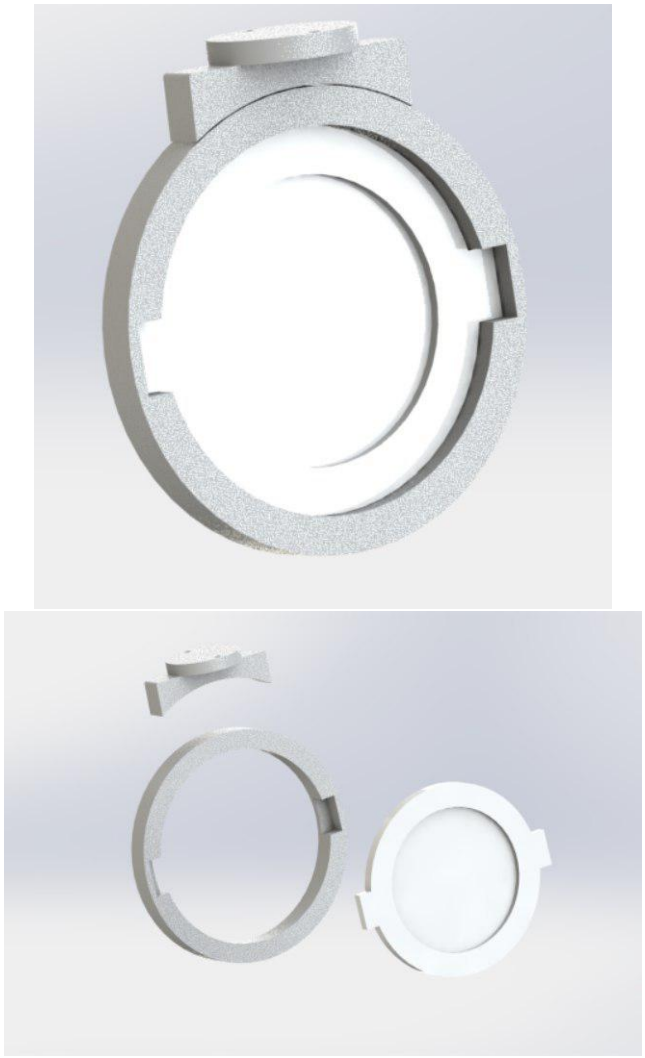


Figure B.10 Assembled and exploded views of the end-effector in non-prehensile mode.

Appendix C

Camera-robot Calibration

Notation

Since this section contains the description of a geometric algorithm, it's due to provide to the reader a meaning of the adopted symbols.

Table C.1 Table of Symbols used in the Section

Symbol	Meaning
$< c >$	Camera Left Optical Frame
$< b >$	Ball Coordinates
$< m >$	Marker Frame
$< ee >$	Robot End Effector Frame
$< r >$	Robot Base Frame / World Frame
${}^{\mathbf{C}}\mathbf{T}_{\mathbf{b}}$	Transformation from <c> to <b>
${}^{\mathbf{C}}\mathbf{T}_{\mathbf{M}}$	Transformation from <c> to <m>
${}^{\mathbf{R}}\mathbf{T}_{\mathbf{M}}$	Transformation from <r> to <m>
${}^{\mathbf{R}}\mathbf{T}_{\mathbf{EE}}$	Transformation from <r> to <ee>
${}^{\mathbf{R}}\mathbf{T}_{\mathbf{b}}$	Transformation from <r> to <b>
${}^{\mathbf{EE}}\mathbf{T}_{\mathbf{b}}$	Transformation from <ee> to <b>

Software Tools

For achieving a reliable calibration the *ROS framework*, *AR Track Alvar* and *Matlab* have been used.

- The ***ROS Framework***, or Robot Operating System, is a well-known open-source middleware for designing and programming robot applications.
  It provides several intuitive tools for implementing control systems or even only validate a robot and acquire informations from data.
  For solving the calibration problem the **tf** ROS package has been used along with the visualization tool RViz: **tf** is a set of geometric tools which allow to visualize and compute transformations between the different coordinate frames which are registered to the ROS_MASTER at the moment.
  Moreover the **franka_ros** package allowed to add to the *tree of frames*, a data structure with which ROS stores and represents all the frames, the coordinate frames of the Panda joints.

- In order to retrieve a transform between <r> and <c> a 'bridge transform' is needed: the common way to achieve this result is to use *Markers*: the ROS software package compatible with ZED camera which has been used is ***ar_track_alvar***, a marker recognition suite which allows, after a proper configuration, to retrieve the coordinates transformation between the <c> frame and the marker frame <m> just by including the marker itself in the field of view of the camera.

Figure C.1 The markers used in the Calibration procedure: using groups of markers can increase the precision in the detection of the single marker.

- The ***MATLAB*** software suite has been used for computing the mean of different transforms, in order to have more accurate parameters.

Calibration Algorithm

The procedure adopted for computing the transformation between the camera frame and the robot base frame, adopted as world frame, is described by the following steps:

1. A target marker is placed near the robot base, by keeping into account the physical distance from its center to the center of the base.
   The size of the base is retrieved by the section drawings provided by the manufacturer, so is possible to compute a rigid transformation which expresses the marker in < r > coordinates: ${}^{\mathbf{R}}\mathbf{T}_{\mathbf{M}}$.

2. The ROS package ar_track_alvar is run, and the detection of the marker is checked through RViz GUI: if the position is not stable due to noise, light variations or excessive distance, adding other markers near the target one can improve the performance.
   Now the transform ${}^{\mathbf{C}}\mathbf{T}_{\mathbf{M}}$ which expresses the marker in the system of coordinates of the camera optical frame (specified as parameter in configuration file) is loaded in the frame tree.

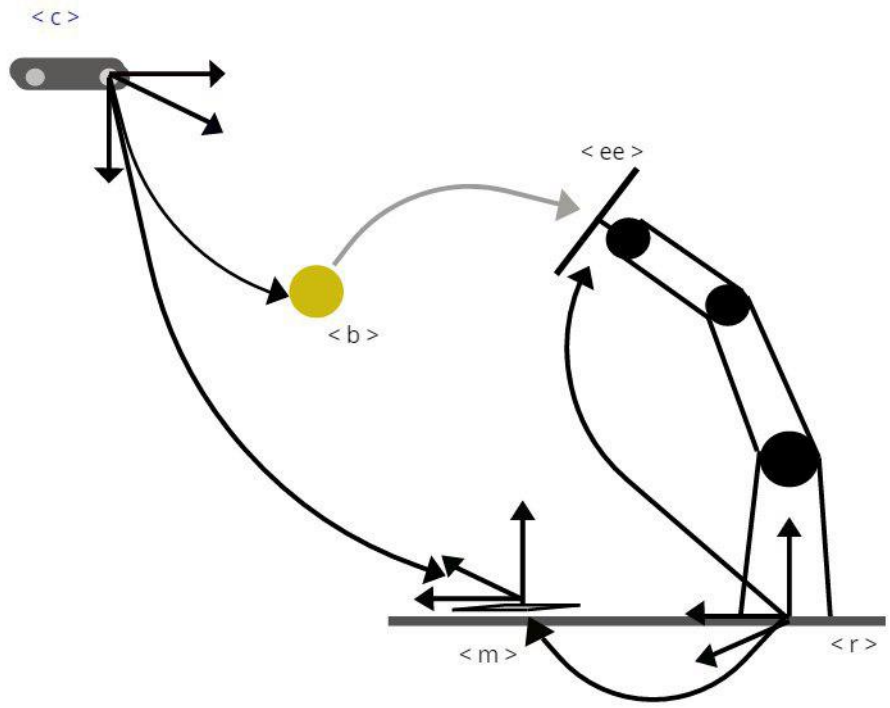


Figure C.2 Schematic representation of the Calibration problem

3. The rigid transformation ${}^{\mathbf{R}}\mathbf{T}_{\mathbf{M}}$ is manually loaded in the tree with the *tf static_transform_publisher* ROS command.
   A link has been created inside the tree due to the common marker object, so by using the *tf echo* $< c > < r >$ command the transform ${}^{\mathbf{R}}\mathbf{T}_{\mathbf{M}}$ is automatically published in the format of a translation vector $[x, y, z]$ and the orientation quaternion $[wxyz]$ which encodes orientation angles informations.

4. Once the two vectors have been saved, the quaternion can be converted in rotation matrix form with MATLAB and then the 4-by-4 transformation matrix can be finally composed.
   Since the transform is published every second and the marker detection could not be steady all the time, is useful to store more ${}^{\mathbf{R}}\mathbf{T}_{\mathbf{M}}$ matrices and compute an average through MATLAB.
   Now the matrix can be used in control algorithms or object detection to express the ball with respect to the base of the robot, which is the coordinate system to which the

manipulator refers for all the computations.

Validation

To graphically assess the correctness and robustness of the transformation matrix retrieved from the procedure RViz can be used.
Is sufficient to manually publish the rigid ${}^{\mathbf{R}}\mathbf{T}_{\mathbf{M}}$ and run an image viewer from inside RViz GUI: by adding the robot frames to the scene a mixed-reality scene will be visible, with the field of view of the camera overlapped to the RViz graphical space.
In this scenario, the correctness of the calibration can be declared if the frames are correctly positioned on the real robot visible in the scene.

Appendix D

Ball Detector Tuning

The binary thresholding operation performed in the image processing pipeline is one of the most important phases of the whole process, since it allows to isolate the exact pixel region corresponding to the target object.
Due to the implemented GUI based on trackbars, the minimum thresholds for the Hue, Saturation and Value can be adjusted in real-time while observing the scene: thanks to this feature some good values have been detected for the conditions of the experimental setup.
Moreover this procedure allows slight changes in order to comply with light variations over the day.

The following sequence of images shows the binary images after the thresholding and binary morphing operations.
When tuning the parameters a good practice is to bring all the minimum thresholds to zero and start to raise one component after another, then when the object remains visible together with only few other blobs a finer tuning can be done until only the target remains in the binary scene.

Table D.1 Table of ball segmentation thresholds

Component	**Minimum value**	**Maximum value**
H	24	45
S	135	256
V	92	217

Figure D.1 Binary scene with all values to 0.

Figure D.2 Binary values with threshold on H channel.

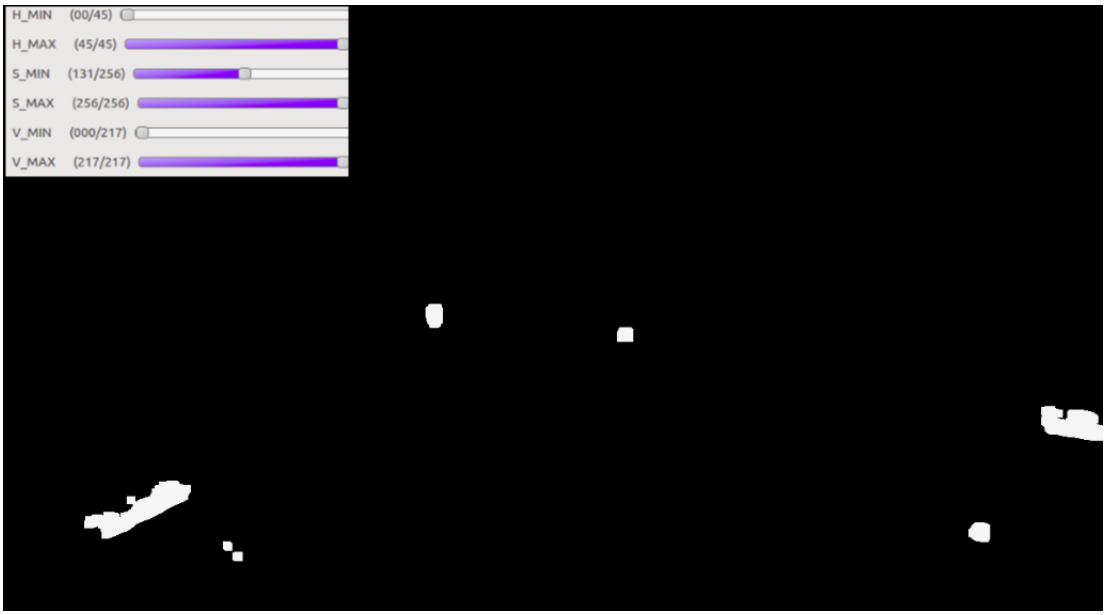


Figure D.3 Binary values with threshold on S channel.

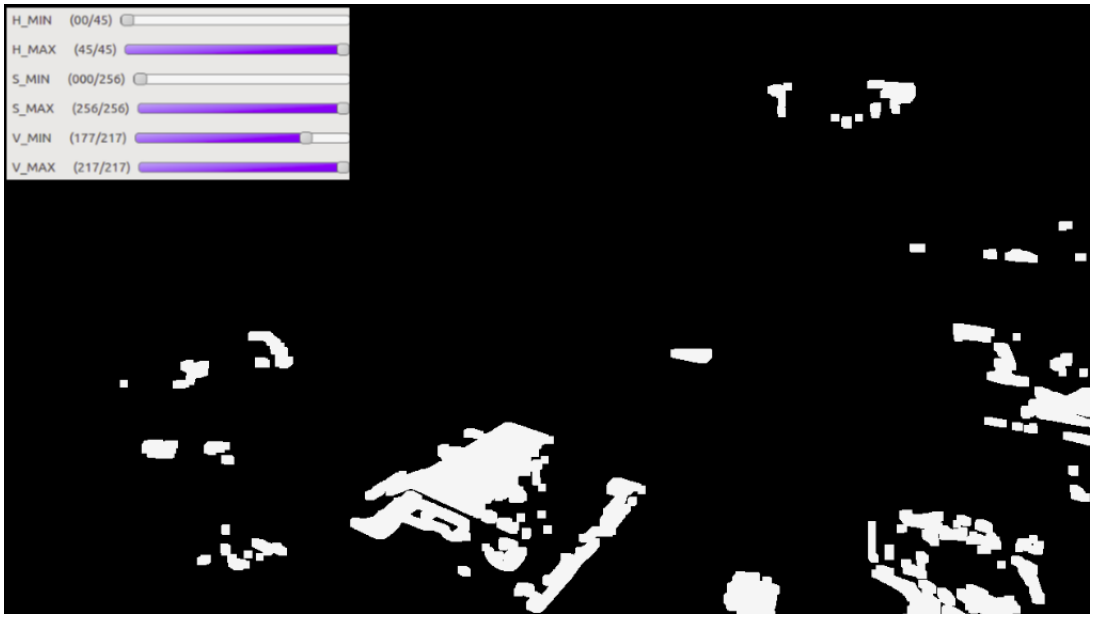


Figure D.4 Binary values with threshold on V channel.

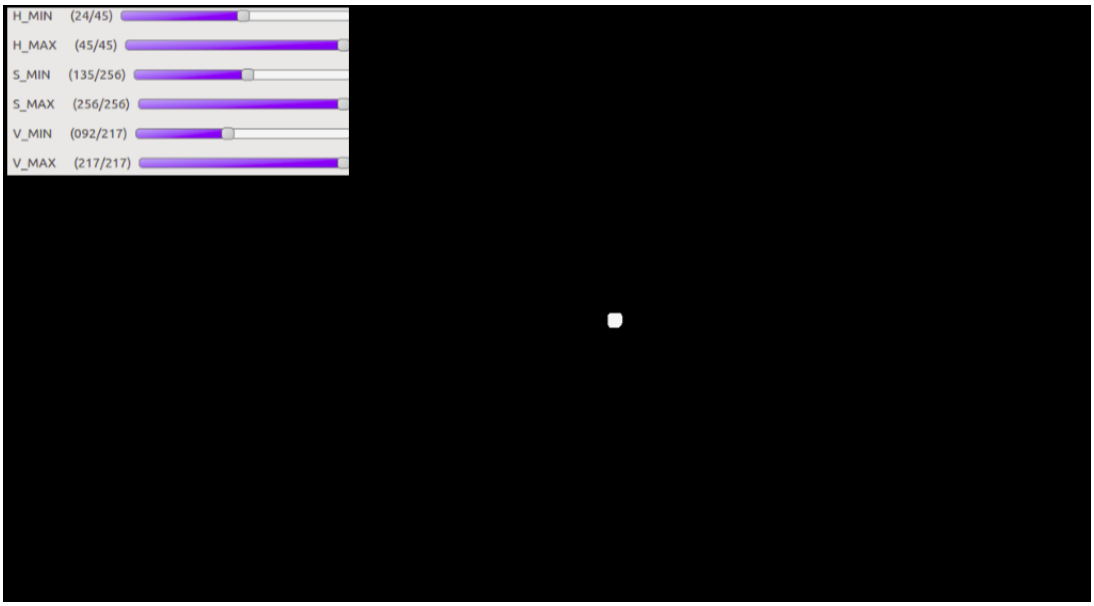


Figure D.5 Binary values with optimal thresholds.

Appendix E

Camera Latency Analysis

The concept of *Camera Latency* is very important when building a real-time system since is an unavoidable source of delay which affects the system, and it should be reduced or when not possible it should be kept into account when designing the control law.
The cause of camera latency is the physical process of digital image formation, which traditionally involves three steps:

- The *Image Sensor*, usually an integrated circuit based on CMOS or CCD and arranged as a matrix of pixels, acquires the brightness values each time the shutter closes.
- The raw image then runs through an *Image Signal Processor*, which is a DSP integrated circuit optimized for working with matrices of pixels and performs several *early vision* operations like preliminar noise filtering directly on the electrical values of the pixels, before any software processing.
  In modern cameras early vision operations are handled at firmware level.
- Finally the pre-processed data are transferred through a BUS (tipically an USB cable) to the connected pc/workstation.

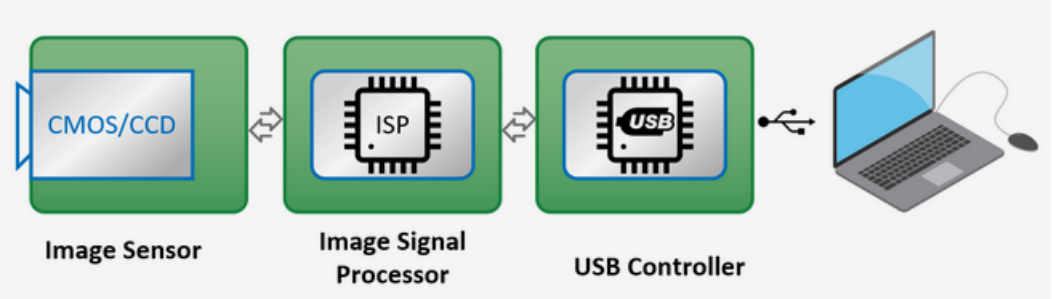


Figure E.1 Electronic pipeline in digital image formation and acquisition is the fundamental cause of camera latency. [8].

Two methods for assessing real camera latency have implemented and tested, and the experimental latency values have been confronted with the nominal values provided by the vendor on technical documentation.

Screen-based Latency Assessment

The first presented method is quite simple to implement: it only requires the camera to be connected to the workstation and of course a working monitor.
The algorithm only consists of a simple acquisition loop which shows on screen the video stream along with a timer updated from the inside of the loop.
The resolution of the timer can be arbitrary, anyway for having reliable and consistent results the resolution should be at least of milliseconds.
The test, described in figure 4.4, consists of recording the screen and compute the difference of the timer values in the concentric views, which is a measurement of the camera latency.

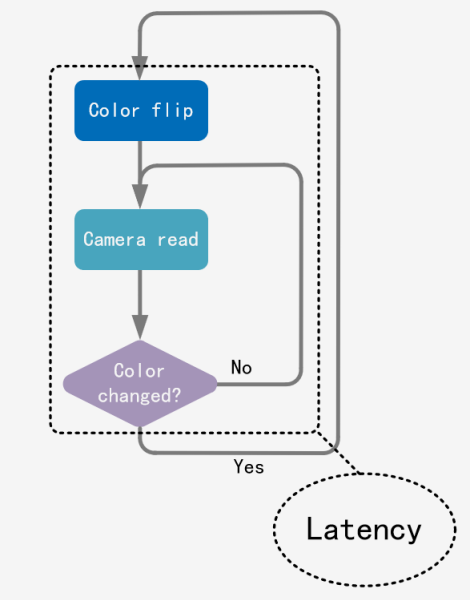


Figure E.2 Logic diagram of the test.

Color-based Latency Assessment

The second test, used as backup for the first one and to remark the correctness of the results, requires a simple image processing loop (color brightness detection) and a few hardware components: an LED circuit and a microcontroller board.
Specifically an Arduino DUE board has been used along with an LED.

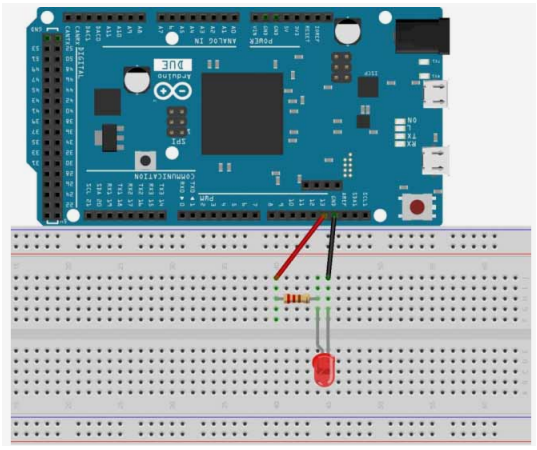


Figure E.3 Scheme of led blinking circuit with Arduino DUE board.

The test is run by a Python script which records the video stream from the camera and blinks the LED at regular time intervals, registers the time at which the LED is turned on and then with a simple image processing algorithm voted to color detection through thresholding; another timestamp is retrieved when the algorithm detects the change and the difference between the two instants is computed.
For this test to be valid is necessary to have a 'barebone' image processing pipeline in order to not introduce additional delays related to computations.

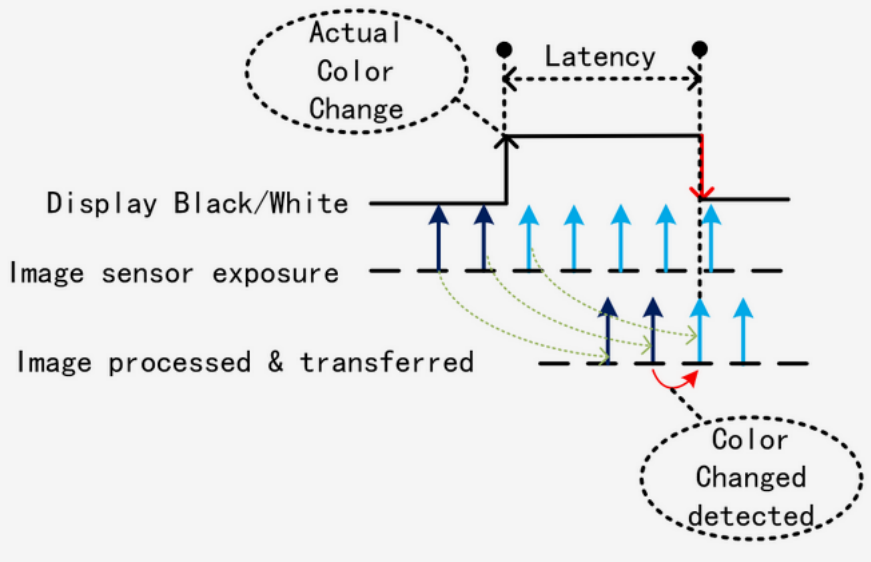


Figure E.4 Timing diagram of the color-based test.

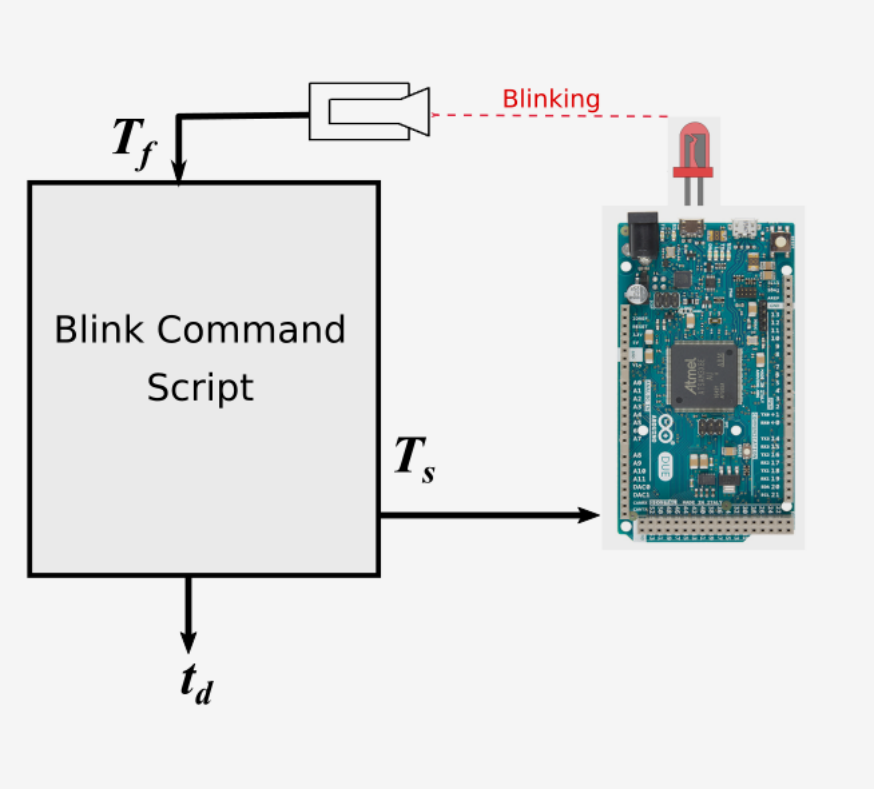


Figure E.5 Architecture of the test.

Conclusions

After collecting empirical values of camera latencies from the two tests, the following statements can be expressed.

- The measured latencies obtained with the two tests are approximately similar, so the measurements are consistent independently from the method of evaluation.
- The measured latencies are within the range provided in Stereolabs table, so the evaluation is consistent with respect to vendor's specifications.
- From the results emerged that the measured latency at 60 fps is approximately 45 ms, so around the upper bound of the interval.

Software Technologies for Communication

In a distributed control architecture typically there are at least two connected workstations for handling different components of the system.
The simplest scheme involves two separate workstations: one voted to compute complex control laws and send commands to the robot controller (or even the robot itself) and the other voted to process sensors data in order to provide a feedback to the control system.
This kind of architectures implicitly suggest that some kind of communication between the workstations is required: each machine must be aware of the computations performed by the other one and the data have to be exchanged at high speed and without losses to guarantee a good overall performance.
In the field of systems engineering several techniques have been developed to interface the components (nodes) of an heterogeneous network, and today many of them are efficiently adopted for distributed robot control applications.
Moreover it must be underlined that applications with an expected real-time behavior still have to rely on cabled connections, because are still the fastest and most affordable transmission media for digital signals if confronted with WiFi or other radio frequency technologies.

The Client/Server model

Client–server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients.
Often clients and servers communicate over a computer network on separate hardware, but in particular applications both client and server may reside in the same system.
The architecture works as follows: a client cannot share any of its resources through the network, but can advance requests for a server's content or service function.
Servers initially await incoming requests from the clients, which can receive the produced data.

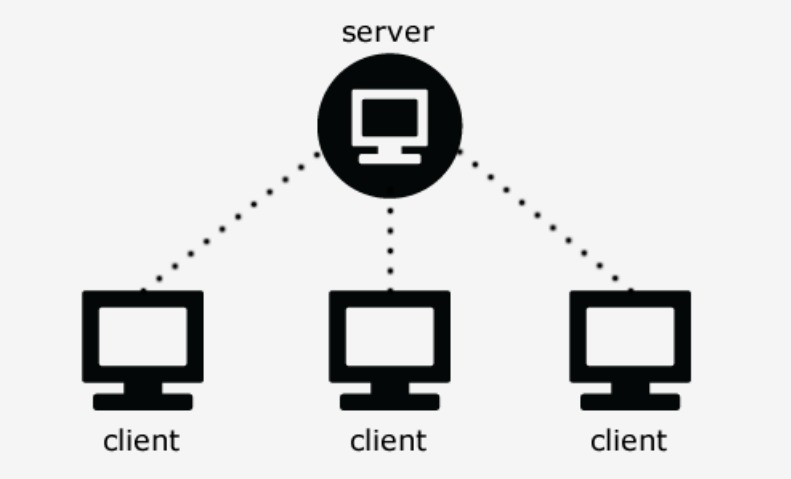


Figure E.6 Schematic model of the Client/Server Architecture.

The server component of the architecture provides a function or service to one or many clients, which initiate requests for data, web pages or files.
Through this model the server can share its software and hardware resources to one or more consumers, and even provide access to its storage devices: this kind of applications is called a 'service'.
Whether a computer is a client, a server, or both, is determined by the nature of the application that requires the service functions.
Client software can also communicate with server software within the same computer.
In the case of a straightforward distributed control application, a common fashion as described in [7] is to use a client/server model in order to communicate setpoints to the workstation which computes the control law to be sent to the robot controller.
The server runs on the workstation adopted for sensors data processing, so while perceptual informations are acquired, it waits for an incoming request to start its transmission.
At user's level the implementation of a client/server model is fully supported by several libraries like UNIX sockets, anyway at a lower level the adopted communications protocols are essentially two, TCP and UDP, defined within the transport layer of the Internet Protocol Suite.

The Publish/Subscribe model

In software architecture, publish–subscribe is a messaging pattern where senders of messages, called publishers, do not program the messages to be sent directly to specific receivers, called

subscribers, but instead categorize published messages into classes without knowledge of which subscribers, if any, there may be.
Similarly, subscribers express interest in one or more classes and only receive messages that are of interest, without knowledge of which publishers, if any, there are.
In fact is a communication system which does not yields a direct interaction between machines but instead adopts a 'blackboard' paradigm.
This pattern provides greater network scalability and a more dynamic network topology, with a resulting decreased flexibility to modify the publisher and the structure of the published data.
In many pub/sub systems, publishers post messages to an intermediary message broker or event bus, and subscribers register subscriptions with that broker, letting the broker perform the filtering and in addition, the broker may prioritize messages in a queue before routing.
The key idea of this model is that publishers are *loosely coupled* to subscribers, and need not even know of their existence: the topic is the focus of communication, so publishers and subscribers are allowed to remain ignorant of system topology.

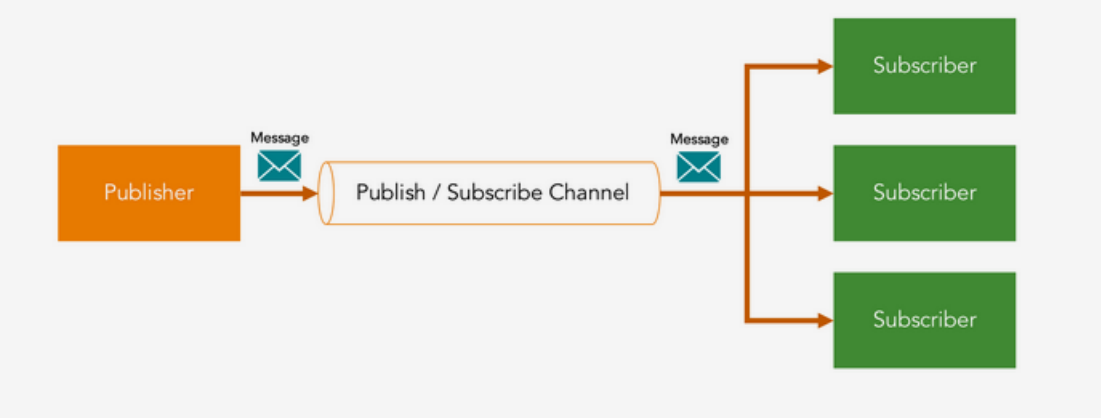


Figure E.7 Schematic model of the Publish/Subscribe Architecture.

In the traditional *tightly coupled* client–server paradigm, the client cannot post messages to the server while the server process is not running, nor can the server receive messages unless the client is running.
Pub/sub provides the opportunity for better scalability than traditional client-server, through parallel operation, message caching, tree-based or network-based routing, etc. However, in certain types of tightly coupled, high-volume enterprise environments, as systems scale up to become data centers with thousands of servers sharing the pub/sub infrastructure, current vendor systems often lose this benefit; scalability for pub/sub products under high load in

these contexts is a research challenge.

In robotics systems design the pub/sub model has been successfully adopted in several framework intended to be general purpose middleware architectures.

In particular ROS implements a local and distributed messaging framework based on publisher and subscriber nodes: this allows to build complex applications with heterogeneous networks of sensors and robots without any effort in communication design.

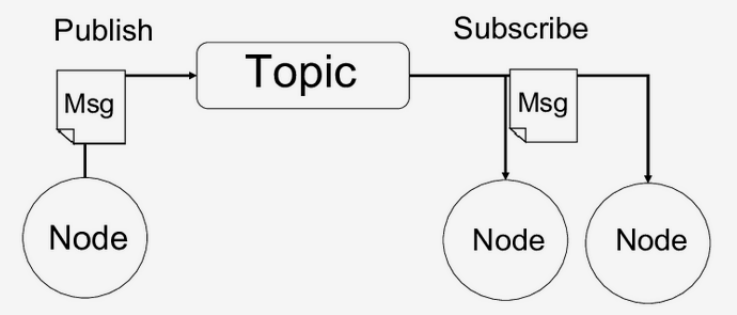


Figure E.8 Schematic model of the Publish/Subscribe Architecture in ROS.

References

[1] L. Weiss, A. Sanderson, and C. Neuman. Dynamic sensor-based control of robots with visual feedback. *IEEE Journal on Robotics and Automation*, 3(5):404–417, October 1987.

[2] Friedrich Lange and Gerhard Hirzinger. Stability preserving sensor-based control for robots with positional interface. volume 2005, pages 1700 – 1705, 05 2005.

[3] Carlos Pérez-Vidal, Luis Gracia, Nicolas Garcia, and Enric Cervera. Visual control of robots with delayed images. *Advanced Robotics*, 23:725–745, 01 2009.

[4] B. Hove and J. E. Slotine. Experiments in robotic catching. In *1991 American Control Conference*, pages 380–386, June 1991.

[5] H. H. Rapp. A ping-pong ball catching and juggling robot: A real-time framework for vision guided acting of an industrial robot arm. In *The 5th International Conference on Automation, Robotics and Applications*, pages 430–435, Dec 2011.

[6] Georg Batz, Arhan Yaqub, Haiyan Wu, Kolja Kühnlenz, Dirk Wollherr, and Martin Buss. Dynamic manipulation: Nonprehensile ball catching. pages 365 – 370, 07 2010.

[7] Vincenzo Lippiello, Fabio Ruggiero, and Bruno Siciliano. 3d monocular robotic ball catching. *Robotics and Autonomous Systems*, 61:1615–1625, 12 2013.

[8] Latency camera measurement, https://www.dlology.com/blog/how-to-measure-the-latency-of-a-webcam-with-opencv/.

[9] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics: Modelling, Planning and Control*. Springer Publishing Company, Incorporated, 1st edition, 2008.

[10] Maria Frucci and Gabriella Sanniti di Baja. From segmentation to binarization of gray-level images. *Journal of Pattern Recognition Research*, 3, 01 2008.

[11] Brian Yandell. Smoothing methods in statistics. *Technometrics*, 39:338–339, 03 2012.

[12] Michael W. Schwarz, William B. Cowan, and John C. Beatty. An experimental comparison of rgb, yiq, lab, hsv, and opponent color models. *ACM Trans. Graph.*, 6(2):123–158, April 1987.

[13] Thierry Carron Patrick Lambert. Symbolic fusion of luminance-hue-chroma features for region segmentation. *Pattern Recognition*, 32(11):1857 – 1872, 1999.

[14] S.E. Umbaugh. *Digital Image Processing and Analysis, 3rd edition, Applications with MATLAB and CVIPtools*. 02 2018.

[15] Ian Young. Image analysis and mathematical morphology, by j. serra. academic press, london, 1982, xviii + 610. *Cytometry*, 4:184–185, 09 1983.

[16] Robert Haralick, Stanley Sternberg, and Xinhua Zhuang. Image analysis using mathematical morphology. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-9:532 – 550, 08 1987.

[17] Emilio Maggio and Andrea Cavallaro. Video tracking: Theory and practice. *Video Tracking: Theory and Practice*, pages 229–245, 12 2010.

[18] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Journal of Fluids Engineering*, 82(1):35–45, 03 1960.

[19] J. Y. Ishihara, M. H. Terra, and J. C. T. Campos. Robust kalman filter for descriptor systems. *IEEE Transactions on Automatic Control*, 51(8):1354–1354, Aug 2006.

[20] M. H. Terra, J. P. Cerri, and J. Y. Ishihara. Optimal robust linear quadratic regulator for systems subject to uncertainties. *IEEE Transactions on Automatic Control*, 59(9):2586–2591, Sep. 2014.

[21] J. Michael McCarthy. Introduction to theoretical kinematics. 1990.

[22] R. Penrose. A generalized inverse for matrices. *Mathematical Proceedings of the Cambridge Philosophical Society*, 51(3):406–413, 1955.

[23] S. J. Julier and J. K. Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, March 2004.

[24] Charles Chui and Guanrong Chen. Kalman filtering: With real-time applications. *Kalman Filtering: With Real-Time Applications*, pages 112–120, 01 2009.

[25] Franka Emika. Website, https://www.franka.de/, 2019.

[26] Franka control interface documentation, https://frankaemika.github.io/docs/, 2019.

[27] Stereolabs website, https://www.stereolabs.com/.

[28] Adrian Kaehler and Gary R. Bradski. *Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library*. 03 2016.

[29] Richard Hartley and Andrew Zisserman. Multiple view geometry in computer vision. 19, 03 2001.