Visual Servoing for Reaching and Grasping Behaviors

Master Thesis

By

PAOLA A. ARDÓN RAMÍREZ

HERIOT-WATT UNIVERSITY

A dissertation submitted to ViBot program in accordance with the regulations of Erasmus Mundus Master in Vision and Robotics.

JUNE 2017

Master Thesis Supervisors: Dr. Mauro Dragone and Dr. Mustafa Suphi Erden

ABSTRACT

Humanoid robots are playing increasingly important roles in real life tasks, especially when it comes to indoor applications. Applications such as grasping simple objects show to be of great interest in the field.

The well-known Aldebaran service robot Pepper shows to be useful for the task. To the date, there is not public application that allows Pepper to grasp objects. Thus, this work rather than presenting new methodologies it implements a combination of existing algorithms to create a new application for Pepper for reaching and grasping behaviors using visual servoing control techniques.

Given the robot's kinematics and dynamic control schemes, a pose based visual servoing (PBVS) is used for reaching and grasping manipulating the robot's right hand as end-effector.

For the implementation, we use a markerless model-based tracker method for detecting the object, and a pattern detection algorithm to add precision to the tracking of Pepper's hand. These two methods combined with an adaptive gain for the control scheme create a robust robotic's PBVS application. The system is implemented using ROS as the main platform along with Aldebaran NaoqiSDK, ViSP and WhyCon libraries. For this robotic system, the method shows better results for grasping than using the conventional motion planing techniques; showing a 48.8% of increment in the grasping success rate.

Due to the modular implementation of the system, it can easily be improved and extended to other humanoid robots by trying different detection/tracking methods, manipulating both end-effectors among others.

Index: Robotics, visual servoing, pose based visual servoing, model-based tracking, Pepper robot,ROS

DEDICATION AND ACKNOWLEDGEMENTS

I would like to thank my project supervisors Dr. Mauro Dragone and Dr. Mustafa Suphi Erden for their helpful guidance and support in the process. As well as Dr. Yvan Petillot for his comments and advice on the implemented system. Especial thanks to Giovanni Claudio for his support on the use of ViSP packages and extra guidance on visual servoing techniques. Additionally, thanks to Bence Magyar and Christian Dondrup for their advice on ROS-Aldebaran node design and framework.

Table of Contents

List of Tables

List of Figures

Figure **Page**

CHAPTER 1

INTRODUCTION

The technological advancements in the last decade have significantly improved the quality of people's life. In an attempt to make life easier a big step has been taken in the field of robotics – the development of humanoid robots.

Humanoid robots are designed to emulate aspects of human behavior in order to accomplish some tasks. Being grasping objects one of them. Grasping is considered to be simple for humans, yet it is not so simple for a robot. It requires many skills. On one hand, a planning and reaching approach. On the other, the target's perception and model reconstruction. Studies demonstrate that for humans most of this information is obtained by vision [27]. Hence, the importance of visual control techniques in robotics. These are techniques that allow to control the motion of a robotic system with the information extracted from vision sensors [20].

1.1 Problem Statement and Solution

Because of the many robotics applications in the household, there is a growing interest of implementing visual control techniques that allow a humanoid robot to reach and grasp objects.

The path planning techniques used in industry with the purpose of grasping are not suitable for humanoid robots. These techniques cannot react quickly to changes in everyday environment and do not adapt to the low repeatability and imprecise kinematics of service robots.On the other hand, visual servoing control schemes offer the flexibility to

build the system in a way that fits the robot's morphology and environmental conditions. For the purposes of this project we focus the implementation of our visual control scheme on Pepper robot, shown in fig.1.1.

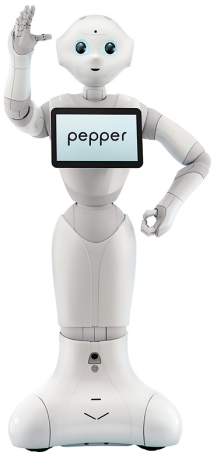

FIGURE 1.1. Pepper – humanoid robot created by Aldebaran-Softbank with the purpose of being a human companion. Image courtesy of [1]

1.1.0.1 Project Objectives

The project's aim is to investigate existing methods to build a visual servoing technique robust enough to be applied on a new robotic platform which end-effectors have less than 6-DOF (degrees of freedom) with the final purpose of grasping an object.Specifically, in this project the following is to be accomplished:

- Study visual servoing approaches and control strategies developed over the last years.
- Design a visual servoing controller and a reaching and grasping strategy for the humanoid robot Pepper (building on existing control libraries).
- Implement a system demonstration on the real robot.

1.2 Project Management

In order to accomplish the objectives of the project the schedule detailed in fig.1.2 and table 1.1 was closely followed.

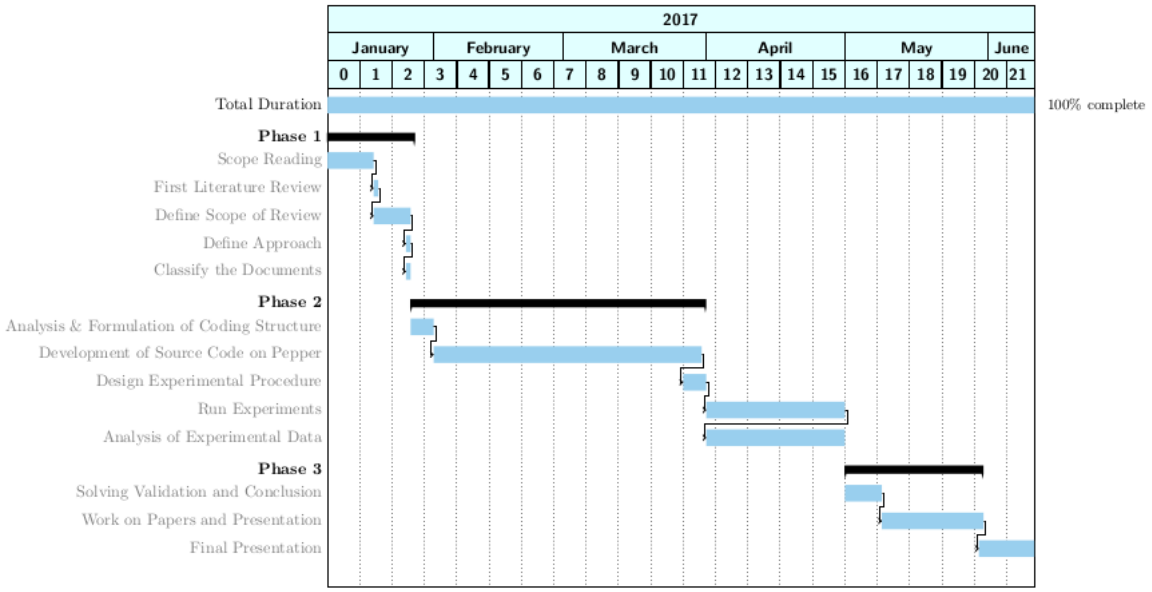

Figure 1.2: Project Management for PBVS on Pepper

We were given a total of 21 weeks to finish the project. As seen in table 1.1 the schedule had a couple of setbacks. Working with hardware can bring some extra difficulties that included the handling of the robot's morphology, unplanned software update of the robot's image system, among others. All together brought a delay on the dates marked in red on the schedule.

Despite the many delays, the system integration was successful.

Table 1.1: Project Management

Task Name	Start	End	Duration
Reading Scope	Jan 01	Jan 08	8 days
1st Literature Review	Jan 08	Jan 08	1 day
Define Scope of Review	Jan 08	Jan 12	4 days
Define Approach	Jan 12	Jan 14	2 days
Classify Documentation	Jan 14	Jan 15	1 day
Software Structure	Jan 15	Jan 30	15 days
Coding	Feb 01	March 31	59 days
Design Experiments	April 01	April 03	3 days
Run Experiments	April 04	May 01	27 days
Analysis on Experiments	May 02	May 08	6 days
Solving/Conclusion	May 08	May 12	6 days
Documentation	May 12	May 30	17 days

1.3 Contributions and Outline

This work does not present new methodologies but rather an implementation with already existing algorithms that together build a new and robust application for grasping an object using Pepper robot. Showing a success rate of over 70% for grasping, outperforming by 48.8% the performance of motion planning techniques for this robotic plaform. To the knowledge of the author, there are not reported results of using visual servoing techniques with the purpose of grasping on Pepper. Thus, we present a robust pose based visual servoing (PBVS) technique that demonstrates to be reliable and suitable for real time applications.

The code of the implemented method is available to download in an open repository in *bitbucket*, along with the guidelines for the installation of the needed modules. Additionally, given the structure of the implementation, it can easily be extended to different objects and other humanoid robots.

This paper is organized as follows: chapter 2 explains the state-of-the-art methods for visual servoing, chapter 3 shows the implemented control technique and the software used for the implementation, chapter 4 shows the pattern tracking algorithm to obtain the position of the robot's end-effector, chapter 5 explains a maskerless model base tracker algorithm that gives the object's position, chapter 6 expands on how to control Pepper joints in velocity and chapter 7 shows the final results of the implementation.

CHAPTER 2

STATE OF THE ART

This chapter aims to introduce the general concepts of the most common control techniques and their requirements to achieve a successful visual servo system. Regardless of the control scheme we look to reduce the error $e(t)$ [20], defined as:

$$e(t) = \boldsymbol{s}(\boldsymbol{m}(\boldsymbol{t}), \boldsymbol{a}) - \boldsymbol{s}^* \tag{2.1}$$

where:

- $\boldsymbol{m}(\boldsymbol{t})$ is the vector representing the set of image measurements. These are the coordinates of the interest points [20]. Once these measurements are obtained, they are used to compute a vector of k visual features $\boldsymbol{s}(\boldsymbol{m}(\boldsymbol{t}), \boldsymbol{a})$.
- $\boldsymbol{a}$ is the set of parameters that represent any potential additional data. This can be the camera intrinsic parameters or the 3D model of the object to track [26].
- $\boldsymbol{s}^*$ is the vector that stores the desired values of the features or desired final position. The setup of $\boldsymbol{s}^*$ is what defines the servoing scheme and we will see it in detail in section 2.2.

Once $\boldsymbol{s}^*$ is chosen the control scheme is simplified [22], and all we need to do is to apply a velocity controller v_c. Now, we relate $\dot{\boldsymbol{s}}$ and $\boldsymbol{v_c}$ as:

$$\dot{\boldsymbol{s}} = \boldsymbol{L_s} \boldsymbol{v_c} \tag{2.2}$$

with $\boldsymbol{L}_s$ being a $k \times 6$ interaction matrix [27]. We obtain the time variation by defining:

$$\dot{\boldsymbol{e}} = \boldsymbol{L_e} \boldsymbol{v_c} \tag{2.3}$$

where $\boldsymbol{L_e} = \boldsymbol{L_s}$ [20]. Now, we need to input $\boldsymbol{v}_c$ to the robot's controller to ensure the exponential decrease of the error. Therefore $\boldsymbol{v}_c$ is defined as:

$$\boldsymbol{v_c} = -\lambda \widehat{\boldsymbol{L_e^+}} \boldsymbol{e} \tag{2.4}$$

where $\widehat{L_e^+}$ is the approximation of the pseudo- inverse of the interaction matrix L_e, assuming L_e is full rank [20]. And λ is the proportional gain [27] or a complex function to regulate s to s^*. Thus, the real behavior of the closed loop system error is calculated by:

$$\dot{\boldsymbol{e}} = -\lambda \boldsymbol{L_e} \widehat{\boldsymbol{L_e^+}} \boldsymbol{e} \tag{2.5}$$

The following sections will go into how to build the interaction matrices depending on the control techniques and how to build s^* depending on the robot configuration.

2.1 Control Techniques

There are mainly three types of control techniques:

2.1.1 Image Based Visual Servoing – IBVS

Image based visual servoing is built upon the selection of a visual feature set $\boldsymbol{s}$ in the image that need to reach the desired $\boldsymbol{s}^*$ value [23]. Where, $\boldsymbol{s}$ is usually composed of the image coordinates of different points belonging to the target.And the desired set of features can be obtained by computing the projection in the image of the target's 3D model for the desired camera pose. Therefore for this approach the camera calibration plays a key role [23].

A 3D point projected in the camera frame as a 2D point, $x = (x, y)$, is defined as [20]:

$$\begin{cases} x = X/Z = (u - c_u)/f_\alpha \\ y = Y/Z = (v - c_v)/f \end{cases} \tag{2.6}$$

where $m = (u, v)$ gives the coordinates of the image point expressed in pixel units, and $a = (c_u, c_v, f, \alpha)$ represent the intrinsic camera parameters [20]. Being c_u and c_v the coordinates of the principal point, f the focal length and α the ration of the pixel dimension.

For this scheme the interaction matrix is represented as:

$$\boldsymbol{L}_x = \begin{bmatrix} -1/Z & 0 & x/Z & xy & -(1+x^2) & y \\ 0 & -1/Z & y/Z & 1+y^2 & -xy & -x \end{bmatrix} \tag{2.7}$$

Where Z represents the depth of the corresponding point in camera frame. Therefore Z must be estimated before hand. It is a good practice to use an approximation $\hat{L_x^+}$, given the involvement of the camera parameters.

As a summary, for IBVS is important to face camera calibration errors, noisy measurements of the image and the unknown value of Z. It is also essential to ensure the asymptotic stability and convergence of the system [22].

2.1.2 Pose Based Visual Servoing – PBVS

Position based visual servoing extracts the position from the 3D model of the object which directly depends on the camera intrinsic parameters [20]. For this approach is common to define $\boldsymbol{s}$ in terms of the parameters that represent the camera pose. Therefore it is safe to consider three camera frames [20]:

- The current camera frame, which is denoted as F_c,
- The desired camera frame, F_{c*}, and
- The reference frame, F_o, which is attached to the object.

Usually, the PBVS is designed by using $\boldsymbol{s} = ({}^{c*}t_c, \theta_u)$ in which case $s^* = 0,\ e = s$ and the interaction matrix is:

$$\boldsymbol{L}_e = \begin{bmatrix} \boldsymbol{R} & 0 \\ 0 & \boldsymbol{L}_{\theta_u} \end{bmatrix} \tag{2.8}$$

where: t represents the translation vector and θ_u gives the angle/axis for the rotation [20]. ${}^{c}t_o$ represents the coordinates of the origin of the object frame expressed relative to the current frame and ${}^{c*}t_c$ represents the coordinates of the current frame expressed relative to the desired frame [20]. Additionally, let $\boldsymbol{R} = {}^{c*}\boldsymbol{R}_c$ represent the rotation matrix that determines the orientation of the current camera frame with respect to the desired frame [23].

If the pose parameters are perfectly estimated, the camera trajectory is a straight line as seen in fig.2.1 [23].

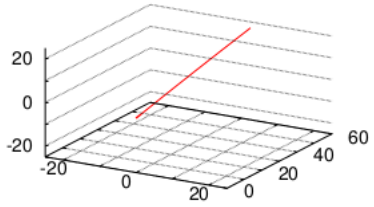

Figure 2.1: System behavior using PBVS when $s = ({}^{c*}t_c, \theta_u)$ approach. Image Courtesy of [20].

The greatest advantage of PBVS above IBVS is the stability of the system. This means it has better response to large transnational and rotational camera motions On the other hand, there are some disadvantages that need to be considered:

- As most of the inverse problems, it can result in an ill-posed problem sensitive to perturbations [22].
- Small errors in the image measurements may lead to very different results, especially in the case of planar targets. In these cases, the control law can be completely unstable.
- It can be computational expensive and non-linear depending on the choice of s [22].

As a general picture, most of the problems in this control scheme result from the target's pose estimation algorithm. Certainly, small errors in getting the points position in the image can significantly affect the accuracy and stability of the system.

2.1.3 Hybrid Visual Servoing – HVS

For the third controlling scheme the purpose is to put together the advantages of both image based and pose based visual servoing approaches. Considering a feature vector s_t and an error e_t devoted to control the translational degrees of freedom [20], we can split the interaction matrix as:

$$\begin{aligned} \dot{s} &= L_{s_t} v_c \\ &= \begin{bmatrix} L_v & L_\omega \end{bmatrix} \begin{bmatrix} v_c \\ \omega_c \end{bmatrix} \\ &= L_v v_c + L_\omega \omega_c \end{aligned} \tag{2.9}$$

where v_c and ω_c represent the linear and angular camera velocities. The most common method for the hybrid approach is the $2-1/2$D Visual servoing explained in [34]. Where s_t is selected as the coordinates of an image point

This approach offers more advantages such as [20]:

- In terms of stability, it is also clear that this scheme is asymptotically stable in perfect conditions.
- The only unknown parameter involved in this scheme is Z^* which can be calculated online using adaptive techniques.
- It does not required full pose estimation
- The Cartesian camera motion and image plane trajectory can be controlled simultaneously.
- It can accommodate large translational and rotational camera motions.

On the downside, it also has some disadvantages [22]:

- There is always the possibility of the features leaving the camera frame.
- It is noise sensible and affected by the reference point. Therefore, it can also be computational expensive.

2.2 Configurations for Robot End Effector

There are two main configurations for the robot end effector: **Eyen-in-hand**, which is when the camera is attached to the moving hand, thus moves with the end-effector. This configuration offers higher precision since the camera is closer to the moving manipulator [17]. **Eye-to-hand** is when the camera observes the target and the moving hand from a fixed position. This lowers the precision due to the farther positioning of the camera from the workspace. Figure 2.2 shows both configurations, the first one being eye-to hand and the second one representing eye-in-hand.

These configurations are particularly useful when the robot has less degrees of freedom and therefore the control scheme needs to be expressed in joint space [20]. In the joint space, for both configurations, the system equation is as follows:

$$\dot{\boldsymbol{s}} = \boldsymbol{J_s}\dot{\boldsymbol{q}} + \frac{\boldsymbol{\delta_s}}{\boldsymbol{\delta_t}} \tag{2.10}$$

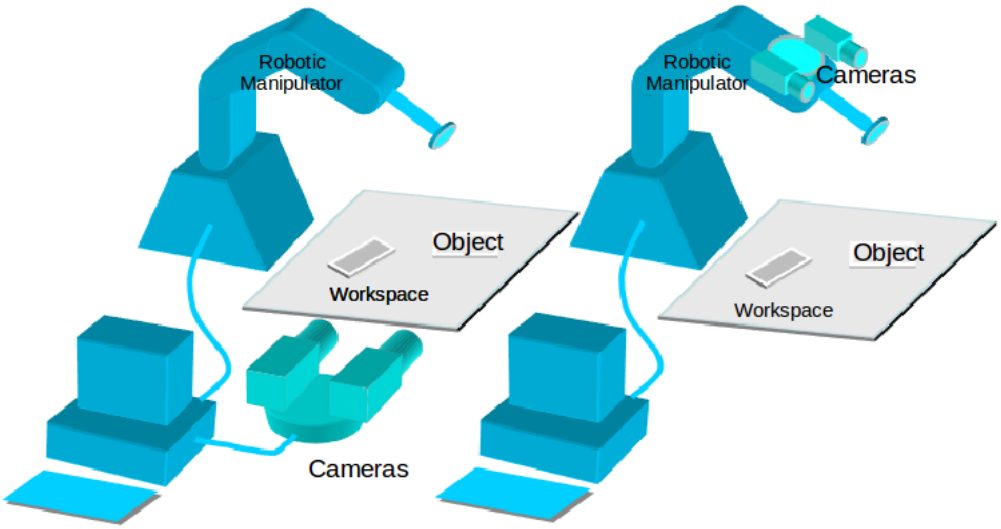

Figure 2.2: Eye-Hand Configurations for Grasping. Image Courtesy of [17].

Where, $J_s \in R^{k \times n}$ represents the feature Jacobian matrix which is usually linked to the interaction matrix, and n represents the number of robot joints [20]. Now:

- **For an eye-in-hand** $\frac{\delta_s}{\delta_t}$ represents the time variation of s due to the object's motion [20], and $\boldsymbol{J_s}$ is given by:

  $$\boldsymbol{J_s} = \boldsymbol{L_s}\ {}^{\boldsymbol{c}}\boldsymbol{X_N}\boldsymbol{J(q)} \tag{2.11}$$

  where, ${}^{\boldsymbol{c}}\boldsymbol{X_N}$ represents the spatial motion transformation matrix from the camera frame to the end effector frame. And, $\boldsymbol{J(q)}$ is the robot Jacobian expressed in the end effector frame.

- **For an eye-to-hand** $\frac{\delta_s}{\delta_t}$ is the time variation of s due the camera motion [20], and J_s is:

  $$\boldsymbol{J_s} = -\boldsymbol{L_s}\ {}^{\boldsymbol{c}}\boldsymbol{X_o}\ {}^{\boldsymbol{o}}\boldsymbol{J_q} \tag{2.12}$$

  where, ${}^{\boldsymbol{o}}\boldsymbol{J_q}$ is the robot Jacobian expressed in the robot reference frame. And ${}^{\boldsymbol{c}}\boldsymbol{X_o}$ is the spatial motion transformation matrix from the camera frame to the reference frame [20].

Once the modeling is done we need to design a control scheme expressed in the joint space. To ensure the stability of this control scheme it is necessary to consider again $e = s - s^*$ and the exponential decoupled decrease of e [20], therefore now the control

scheme is represented as:

$$\dot{\boldsymbol{q}} = -\lambda \widehat{\boldsymbol{J}_{\boldsymbol{e}}^{+}} \boldsymbol{e} - \widehat{\boldsymbol{J}_{\boldsymbol{e}}^{+}} \frac{\widehat{\delta_s}}{\delta_t} + \boldsymbol{P}_{\boldsymbol{\lambda}} \boldsymbol{g} \tag{2.13}$$

Where:

- $\widehat{\boldsymbol{J}_{\boldsymbol{e}}^{+}}$ is an estimation of the Moore-Penrose pseudo-inverse of the task Jacobian. Which is a combination of the interaction matrix and the articular Jacobian of the robot [35]. Where $\boldsymbol{J}_e$ depends on the chosen visual servoing task.
- $\boldsymbol{P}_{\boldsymbol{\lambda}}$ is the large projection operator that allows the system to perform a secondary task despite if the main task is full rank [35]. It is defined in [20] as: $P_\lambda = \overline{\lambda}(||e||)P_{||e||} + (2 - \overline{\lambda}(||e||))P_e$. Being $P_e = (I_n - J_e^+ J_e)$ the classical projector and $P_{||e||}$ the new projection operator that imposes the exponential decrease of the norm of the error instead of each term of the error vector. Therefore, the use of the sigmoid function $\overline{\lambda}(||e||)$ to switch from $P_{||e||}$ to P_e [35].
- $\boldsymbol{g}$ is a vector that defines the secondary task. It can be designed to avoid joint limits, obstacles or singularities. In our case we will use it to avoid the joint limits so that the velocity controller is more reliable.

To ensure the global asymptotic stability of the system we consider the following [20]:

- If $k = n$ then the condition is given by:

$$\boldsymbol{J_e} \widehat{\boldsymbol{J_e^{+}}} > \boldsymbol{0} \tag{2.14}$$

- If $k > n$ then:

$$\widehat{\boldsymbol{J_e^{+}}} \boldsymbol{J_e} > \boldsymbol{0} \tag{2.15}$$

In both cases the extrinsic camera parameters reside in J_e and the estimated ones in $\widehat{J_e^{+}}$

2.3 Related Work

Many approaches have been directed towards the integration and implementation of robust visual servoing techniques for reaching and grasping behaviors. However, recent studies lean towards more daring techniques involving sophisticated 3D model reconstruction and machine learning algorithms such as the ones presented in [38], [33]

and [30]. For the purposes of this paper, the related work is limited to conventional techniques using visual servoing and object tracking models.

One of the first and most significant works in the area is the one presented in [19]. They achieved real time interaction with vision and their system is capable of tracking moving objects [19]. Their focus was to implement the hand-eye configuration for dynamic grasping task on a moving conveyor system. Another work is [18]. It shows good results and overcomes three problems for grasping behaviors: computational time for 3D motion parameters, predictive control of the robotic arm and grasp planning.

In [29] besides achieving the grasping task, they solve the problem of aligning the end effector of the robot with the object to grasp. By emphasizing the importance of the online image Jacobian they show how to plan the grasping between two solids in 3D projective space using uncalibrated stereo vision [29].On the downside of this approach, the process requires various visual processes such as object recognition and stereo matching which results in a quite computational expensive algorithm.

Closer to humanoid robots in [43] and [24] they present a hybrid visual servoing control scheme for grasping that proves to be robust for real-time applications. [43] shows the robustness of the system with ARMAR III arm robot. Where their control scheme robustness lies on estimating the hand position in case of failed visual hand tracking with the combination of visual, force and motor encoder data sensors.

On the other hand [24] shows a similar work, however it does not rely on force sensors for the reaching and grasping task but only on visual data. In [39] they use the same strategy but this time their purpose is to measure task error convergence time with Nao robot.

Contrary to [24], [41] shows a combination of pose based visual servoing with a robust laser scanner that grabs objects features such as color in an indoor environment. This is combined with stereo measurements which ensures the efficiency of the grasping action even if the object is unknown. A more sophisticated technique used for PBVS without the aid of extra sensory data is defined in [37]. They locate the right position for the object to be grasped with local visual descriptors. In a later stage, they learn the good grasping points and generalize this acknowledge to new objects based on experience.

In the following sections we will see how this work combines object recognition and tracking techniques with pose based visual servoing on Pepper.

CHAPTER 3

System Architecture

Visual Servoing techniques offer a way to control the robot's motion depending on the received feedback from the environment. As we saw in chapter 2 there are three main choices to make when designing a visual servoing application: The control technique, the configuration of our end-effector and how we want to describe the object that we want to manipulate. In this chapter we focus our attention on the first two: how we combine the visual servoing control scheme with the end-effector configuration for Pepper robot.

As for any design, we need to evaluate our system disadvantages before jumping into the implementation. In terms of hardware, the grasping task with Pepper robot has the following constraints:

- Pepper's 2D cameras are very limited. They have 5fps (frames-per-second) operating at $10Hz$. This can affect the object recognition and tracking tasks of the implementation.
- Pepper can only lift 300 *grams*, therefore the choice of the object to grasp should be as light as possible and it should fit in Pepper's hands.
- The arms have 5-DOF. There are some positions and orientations that the robot cannot reach, thus the grasping task is compromised unless an extra DOF is added to the servo.

- There is no control over each of the joints of the hand nor tactile/force/pressure sensors that would give any feedback about grasping. The hand only has the ability to open and close.
- Unlike Romeo and NAO, robots of the same company, Pepper does not have the needed modules to control the joints in velocity (methods of ALMotionProxy), which is a key task for servo techniques.

Given these limitations we implement a combination of PBVS and eye-hand configuration that even though it is proven to be computational expensive, it is also shown to be more stable and to find a more accurate solution. In our case, the expected risk of choosing precision above computational efficiency is worth to be taken.

For this chapter we divide the information in the following manner: Section 3.1 shows the details of our design for the control technique implementation and section 3.2 shows the software options that we considered for the application .

3.1 PBVS for Pepper

It is time to focus on visual servoing for humanoid robots, especially for Pepper. In this part of the document we will see the mathematical basis for the design of the control scheme. In our implementation, we use the bottom 2D camera for the image extraction which is located in the mouth, and the robot's right hand is the chosen manipulator.

During the control task the position of the hand and the object are obtained at every iteration using the techniques explained in chapters 4 and 5 respectively. Figure 3.1 shows the frame references for the implementation. The frame $\{o\}$ is attached to the object; $\{h\}$ to the hand marker and $\{h^*\}$ to the desired pose . The transformation matrix ${}^{o}M_{h}^{*}$ is constant. And, at the end of the PBVS the frames $\{h\}$ and $\{h^*\}$ should be congruent.

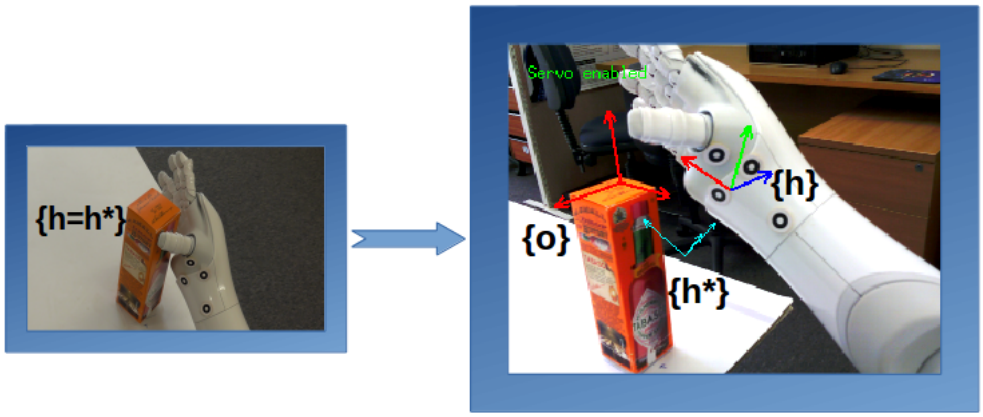

Figure 3.1: Frames Definition for Grasping an Object

Let's start with the goal frame, ${}^{c}\boldsymbol{M}_{h*}$. This frame is equal to the position of the object multiplied by a constant transformation ${}^{o}\boldsymbol{M}_{h*}$ [24]. This constant transformation is learned either by placing the hand at the desired position or by saving the desired object's offset with respect to the camera, as illustrated on the left side of fig.3.1. The definition of this transformation is in the following form:

$$ {}^{c}\boldsymbol{M}_{h*} = \left({}^{h*}\boldsymbol{M}_{c}\ {}^{c}\boldsymbol{M}_{o} \right)^{-1} \tag{3.1} $$

Now that we have the goal position we need to reduce the error e_h. In order to do so, we define the pose between h and $h*$ in the following way:

$$ {}^{h*}\boldsymbol{M}_{h} = {}^{o}\boldsymbol{M}_{h*}^{-1}\ {}^{c}\boldsymbol{M}_{o}^{-1}\ {}^{c}\boldsymbol{M}_{h} \tag{3.2} $$

From this matrix the error tasks can be extracted by taking the translation and orientation vectors that represent the axis. Leaving the error representation as:

$$ \boldsymbol{e}_{h} = \left({}^{h*}\boldsymbol{t}_{h}, {}^{h*}\theta\boldsymbol{u}_{h} \right) \tag{3.3} $$

The interaction matrix used in our approach is the one defined in [20]:

$$ \boldsymbol{L}_{=} \begin{bmatrix} {}^{h*}\boldsymbol{R}_{h} & \boldsymbol{0}_{3} \\ \boldsymbol{0}_{3} & \boldsymbol{L}_{\theta_u} \end{bmatrix} $$
$$ \boldsymbol{L}_{\theta_u} = \boldsymbol{I}_{3} - \frac{\theta}{2}[\boldsymbol{u}]_{\times} + \left(1 - \frac{sinc\theta}{sinc^{2}\frac{\theta}{2}} \right)[\boldsymbol{u}]_{\times}^{2} \tag{3.4} $$

And, we use the joint velocity vector $\dot{\boldsymbol{q}}_h$ with $\widehat{\frac{\delta s}{\delta t}} = 0$ for the eye-hand configuration. In the successful cases, this set-up produces an ideal straight trajectory in the cartesian

space. This trajectory represents an exponential decay in the error measurement as we will see in chapter 7.

The implementation of the PBVS for Pepper is represented in fig.3.2 where the features and the desired position of the object are obtained from the MBT. The hand tracker instead of coming from the robot's odometry comes from a roundels tracker. And, the joints control in velocity is through a joint velocity control package deployed on Pepper.

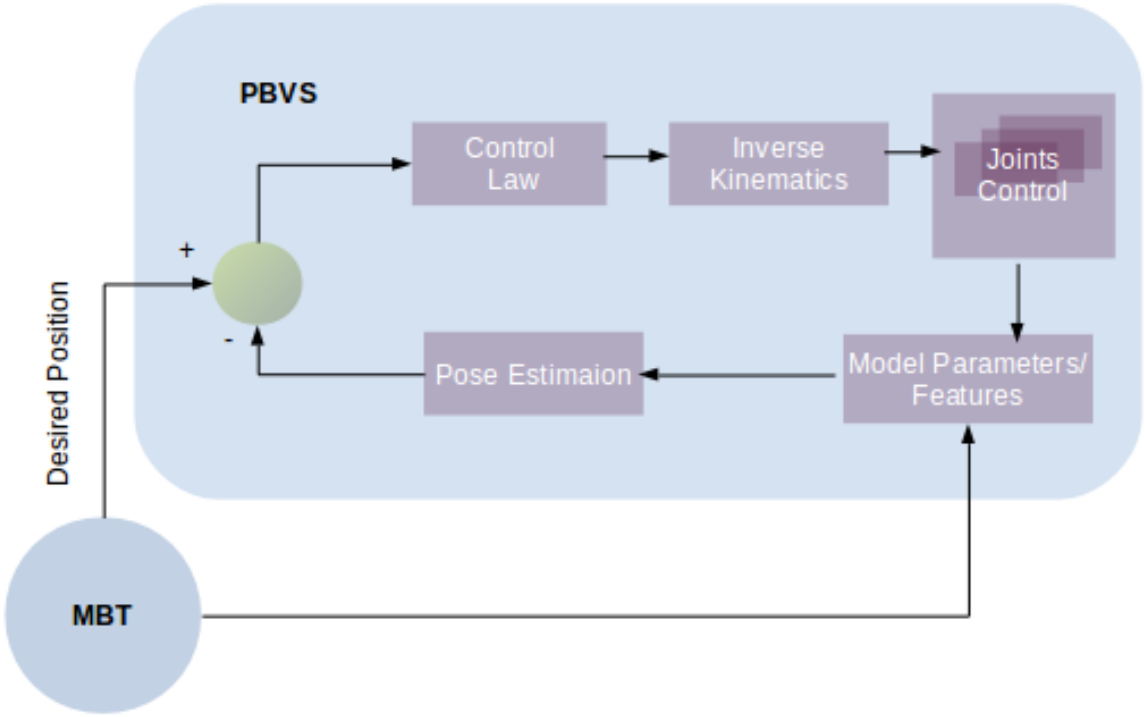

Figure 3.2: Closed Loop – PBVS Control Scheme for Pepper

There are two key aspects that are taken into account in our implementation:

1. *The robot's Kinematics and Dynamic Characteristics*: Because Pepper's arm has only 5-D0F, in our implementation we decided to add 1-DOF to the servo. We do so by extending the PBVS implementation to the robot's base. In this way we manipulate 6 joints for the servo.

   For the first stage, we apply a combination of PBVS with eye-in-hand configurations. Where now our current position is defined by the transformation matrix between the torso and the headPitch joint, and the goal position is the one defined by the

box. Once the robot arrives to the desired position, the second stage consists of a combination of PBVS with an eye-to-hand configuration. Where now the current position is the one given by the right hand and the desired position is where the hand should arrive to grasp the box.

2. *Tunning Lambda Gain:* As described in chapter 2, an adaptive gain λ is used to reduce the time of convergence in order to speed up the servo. This parameter is arbitrary and manually set in the system. If its value is too high there is a risk for oscillation at the time of converge, which compromises the precision of the system. By using ViSP libraries we set the adaptive gain λ in the following way [21]:

$$\lambda(||e||) = (\lambda_0 - \lambda_\infty) e^{\frac{-\lambda_0' \times ||e||}{\lambda_0 - \lambda_\infty}} + \lambda_\infty \tag{3.5}$$

Where:

- $\lambda_0 = \lambda(0)$ is the gain in 0, which is for very small values of $||e||$,
- $\lambda_\infty = \lambda_{||e|| \to \infty} \lambda(||e||)$ is the gain to infinity, which is for high values of $||e||$,
- And, λ_0' is the slope of λ at $||e|| = 0$

In our case for the final PBVS we have $\lambda_0 = 0.02, \lambda_\infty = 0.07$ and $\lambda_0' = 3$.

3.2 Software

For our implementation two options were considered, both of them including the NaoqiSDK of Aldebaran-Softbank and ViSP libraries. NaoqiSDK is the set of tools that facilitates the creation of applications for Pepper robot [1], and ViSP stands for Visual Servoing Platform which eases the prototyping and development of applications that involve visual servoing techniques [7].

The first proposed design, besides the NaoqiSDK and ViSP libraries, required a set of third party libraries and a bridge to connect all of them. Figure 3.3 shows the first implemented approach. Some of the relevant third party libraries were:

- **Metapod libraries:** which were used to compute the kinematics of the robot. Metapod stands for META-Programming Optimized Dynamics library [12]. Metapod solves the robot dynamics algorithms by using R.Featherstone's Spatial Algebra [28] to describe forces, motions and inertias.

- **OpenCV:** is the Open Source Computer Vision library [13] which is used to develop real-time computer vision algorithms. Its main purpose for this version of the implementation was for object recognition.

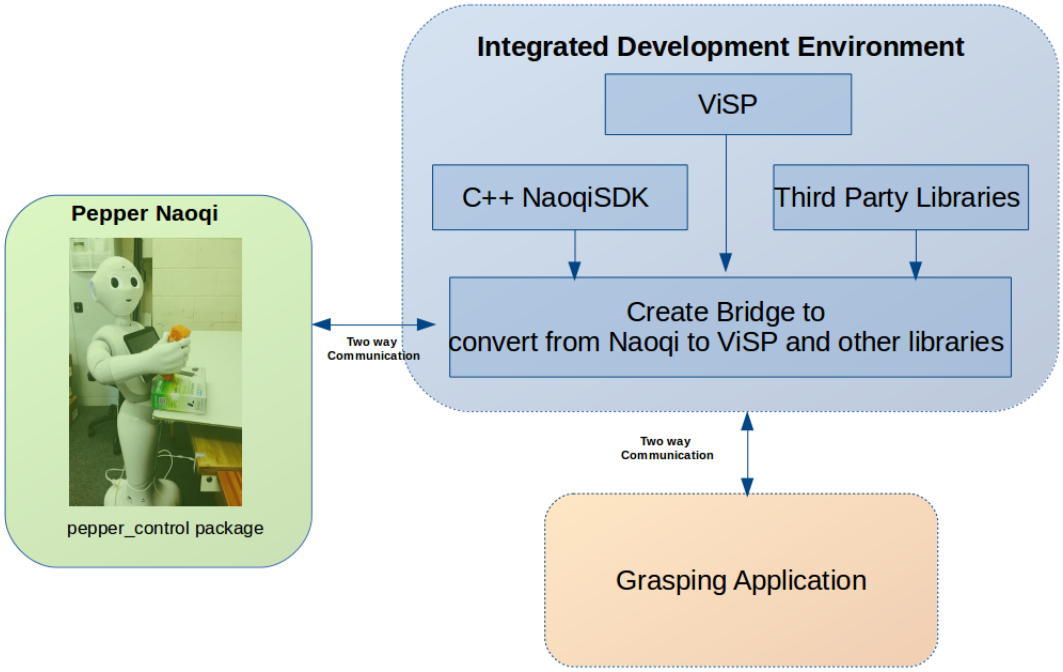

Figure 3.3: First Proposed Solution for the System

A bridge was needed to manage the tools and conversions between the third party libraries and the NaoqiSDK. There is one bridge that accomplishes these purposes and has been thoroughly tested on Romeo robot. The implementation is done by Inria-ViSP labs, *visp_naoqi* [9]. The *visp_naoqi* bridge manages all the tools for Romeo robot, specially for grabbing the images and estimating the camera parameters (intrinsic and extrinsic). Additionally, it controls the robot in velocity and gets the Jacobians [9]. However, when adapting this bridge to Pepper many compatibility issues appeared with the URDF of the robot (Unified Robot Description Format) and the integration of the external velocity control package. Because this approach did not ease the implementation of the grasping application for us, it was discarded at an early stage.

Figure 3.4 shows the communication diagram of the final system. In this approach we use ROS, which is the Robot Operating System that facilitates libraries and tools to help the development of robotic applications [15]. The greatest advantage of using

ROS is that it already has the needed tools to manipulate Pepper robot so no bridge implementation is needed.

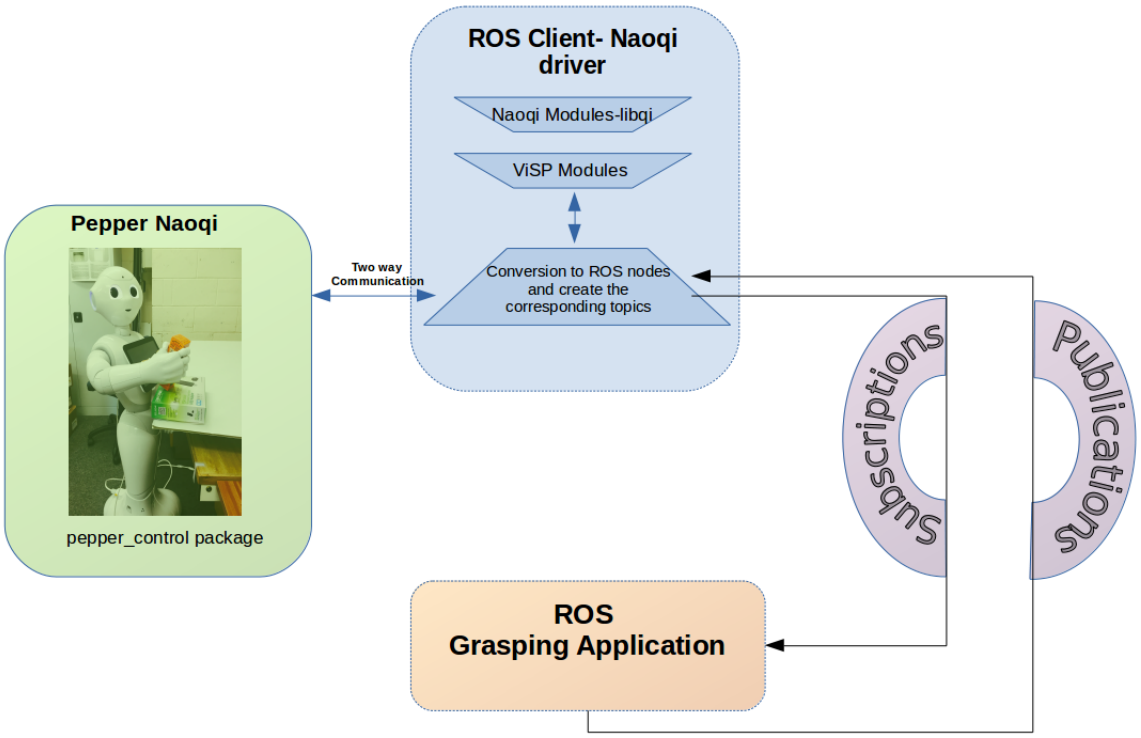

Figure 3.4: Functional Design for Grasping Application

Some of the libraries used for this architecture are:

- ***naoqi_driver*** is a driver module between Aldebaran's NAOqiOS and ROS. It publishes all sensor and actuator data which we need to handle the behavior and grasping tasks [14].
- Because the calibration of the kinematics of the robot are very inaccurate, we will see in chapter 4 that a tracking technique is needed. For it, **WhyCon** libraries are used. WhyCon offers vision-based localization capabilities specially for low frame rate cameras, just as Pepper's, achieving millimiter precision with very high performance [11].
- ***vision_visp*** is a ROS node that provides ViSP algorithms as ROS components, so it facilitates the conversion of data, especially the images format [16].

Figure 3.5 is the graphical representation of the implemented system shown in fig.3.4. It illustrates the different nodes for the application. The red block, pepper_grasping_pbvs, shows where we implement the core of the application, presented in section 3.1. The nodes in purple, where we implement/adapt the end-effector and object tracking algorithms. The rest of the blocks represent the used libraries and borrowed applications required for the implementation. Where, block 1 shows topics used from the NaoqiSDK, block 2 the ones from naoqi_driver, from which we access the camera of the robot. Block 3 represents the topics from WhyCon Libraries and block 4 the ones from vision_visp.

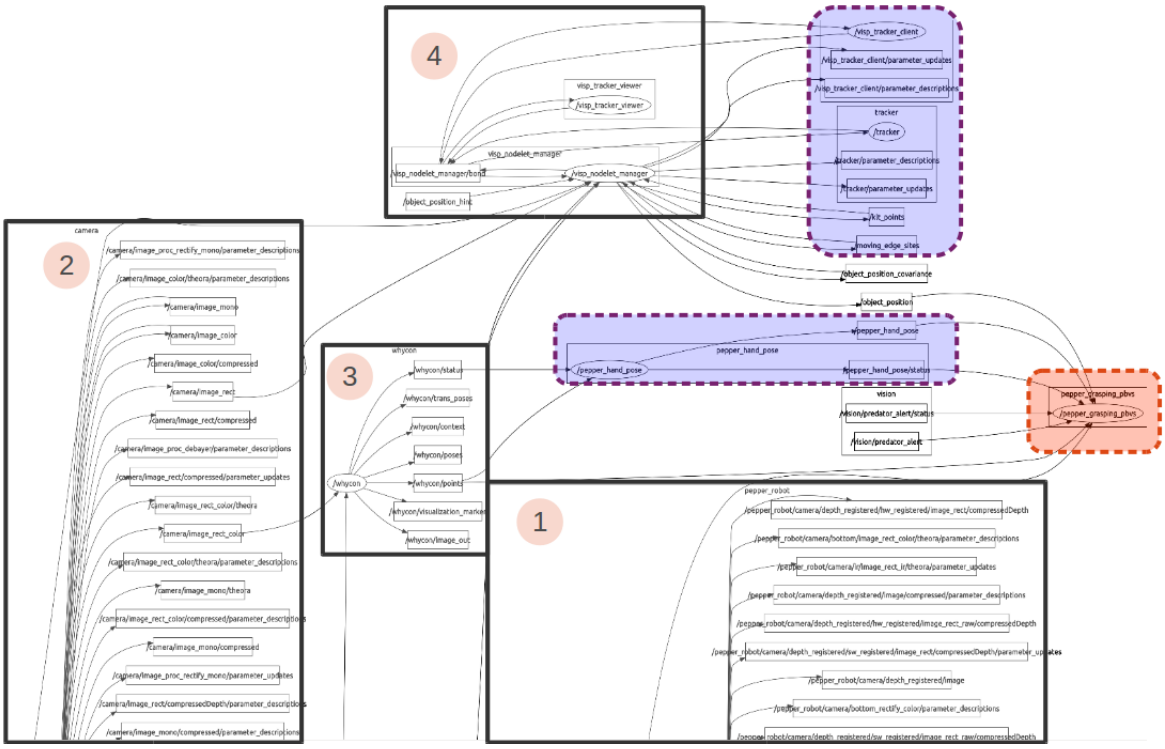

Figure 3.5: RQT Map for the Final Implemented System

Additional to these nodes, it is worth mentioning the use of *MoveIt!* package in ROS. This is used to test the grasping behavior on Pepper robot and to quantitative measure its limitations. *MoveIt!* offers simple but useful takes as inputs for pose vector (postition and orientation). As a result, it generates a large number of potential grasp approaches and directions taking into account the robots kinematic capabilities. Thanks to *MoveIt!* we are able to compare our approach against the conventional motion planning techniques.

Before jumping into the comparison of the methods we will see in detail the tracking algorithms that allowed us to get to the final PBVS application. Which leads us to the following two chapters.

CHAPTER 4

TRACKING THE ROBOT'S MANIPULATOR

In order to achieve a successful visual servoing getting the accurate position of Pepper's arm is crucial. Due to the inaccuracy of the Pepper's kinematic model this task is achieved with the help of vision. We place markers on its hand so that we can compute the hand pose at every iteration of the process. As for any method where the pose of a target is to be extracted, the very first step is to calibrate the camera.For the hand tracking two methods were tested:

1. Quick Response Codes (QR codes) tracking from OpenCV libraries. Figure 4.1 shows the setup for the QR codes.

2. Roundels detection from WhyCon libraries. Explained in the following section.

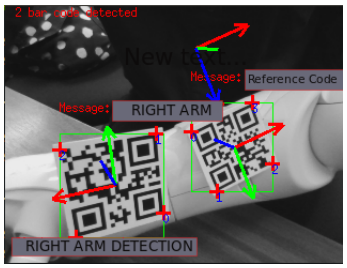

Figure 4.1: QRCode Detection for Pepper

Both targets were automatically detected and gave the needed pose of the hand with respect to the last arm joint, WristYaw. However, due to the low image resolution the

detection and tracking was more robust using the roundels from WhyCon. Nonetheless by the end of this chapter we will compare both results. Figure 4.2 shows a simplified map for the relevant nodes in this part of the implementation.

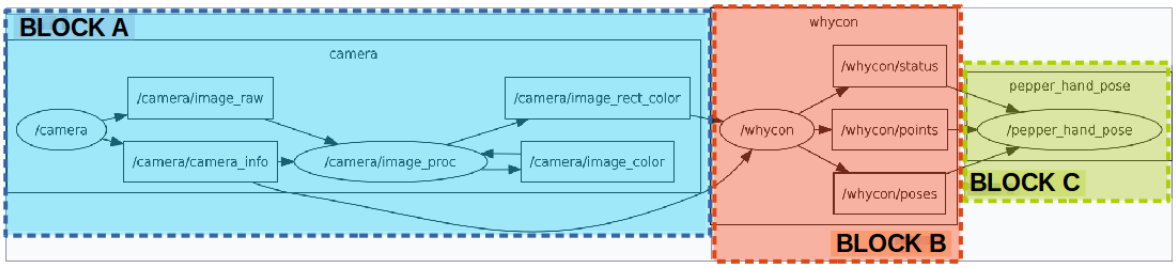

Figure 4.2: Simplified Map for the Implementation of the Hand Tracker Node in ROS

From this figure, **BLOCK A** represents the camera information obtained from the naoqi_driver, **BLOCK B** represents the relevant topics from WhyCon libraries necessary for the implementation, and **BLOCK C** represents the focus of this chapter, the tracking application for the hand. The node is pepper_hand_pose, being the modified version of [10] . In the following sections we will go into the details of the algorithm that allows us to track Pepper's right hand.

4.1 Pose Estimation using WhyCon Libraries

This method is solely based on the efficient detection of black and white roundels such as the one shown in fig.4.3. For this section we use WhyCon libraries that facilitate the detection and tracking of the pattern. However, we need to feed the system with the position of the target. For this task we use the localization method recommended in [40]. Once we have the 3D position of the patterns we build the transformation matrix that describes the pose of the hand. In our approach we track Pepper's hand using 4 patterns, as shown in fig.4.4 to add robustness to the method. For this approach the roundels dimensions need to be known, inner and outer diameter. Once the pattern has been detected and localized the process starts over to track the roundels position.

Figure 4.3: WhyCon Roundels. Image Courtesy of [40]

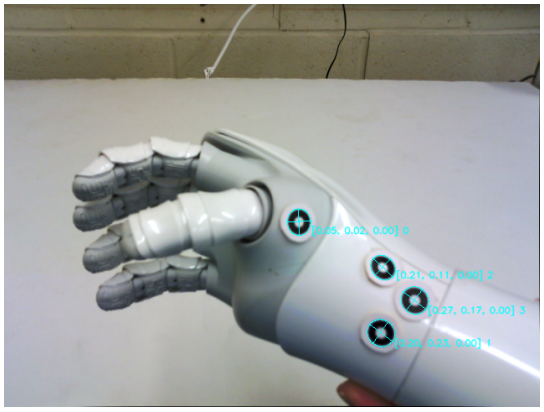

Figure 4.4: WhyCon Roundels Placed on Pepper

In summary, the detection combines a flood-fill segmentation algorithm with an efficient thresholding technique [40]. The main advantage of this technique is that it does not processes the entire image but only the area occupied by the pattern [31], which guarantees its fast computational performance.

Before going into the localization step it is worth reviewing the algorithms concerning the pattern detection.

- The *fast flood-fill* looks for segments of dark pixels using a bounding box of the size of the input diameter [40]. They test the found roundel using the efficient thresholding technique and applying a roundness test. The process is repeated but now looking for bright pixels. Once the search is done it is considered that the algorithm has found two circles, and so it compares the ratio of their area with the previously inputed diameter to the system [40]. The pseudo algorithm can be found in appendix A.3.1.

- The *efficient thresholding* consists on adapting the threshold value. It is defined as τ and it can be sensitive to lightning conditions. In [40] they adapt it in a granularity manner according to the binary search scheme. When the pattern is successfully detected, the threshold is updated so that it can be used for the next iteration, improving the precision of the segmentation [40]. τ is defined as:

$$\tau = \frac{\mu_{outer} + \mu_{inner}}{2} \tag{4.1}$$

where μ_{outer} and μ_{inner} correspond to the mean brightness of the inner and outer segments.

Now that we have the detection done by WhyCon we need to determine the position of each roundel. As recommended in [40] we localize the roundels in the following manner:

1. First the relative pattern position to the camera module needs to be calculated. At this point it is assumed that the camera distortion does not affect the measurements and that the intrinsic camera parameters are already known [31]. Following this assumption, we extract from the covariance matrix eigenvectors the ellipse center and the semiaxes to then transform them into a canonical camera coordinate system [40]. Where the canonical form is a pinhole camera model with unit focal length and no distortion.

2. These extracted parameters are used to get the coefficients of the characteristic equation, which should result in a bilinear matrix of the form $X^T\ Q\ X$, where Q is called the conic and X represents each point lying on the ellipse [31].

3. The orientation and position within the camera coordinate is obtained by eigen analysis [31]. The eigenvalues and eigenvectors of Q are represented by $\{\lambda_0, \lambda_1, \lambda_2\}$ and $\{q_0, q_1, q_2\}$ respectively [40]. Then the position of the circle can be calculated as:

$$\boldsymbol{x_c} = \pm \frac{d_0}{\sqrt{-\lambda_0 \lambda_2}} \left(\boldsymbol{q_0} \lambda_2 \sqrt{\frac{\lambda_0 - \lambda_1}{\lambda_0 - \lambda_2}} + \boldsymbol{q_2} \lambda_2 \sqrt{\frac{\lambda_1 - \lambda_2}{\lambda_0 - \lambda_2}} \right) \tag{4.2}$$

   where d_0 is the circular pattern diameter.

Now we have the position x_c represented in a camera coordinate frame. But, we are interested on getting the 3D position of Pepper's hand. To achieve this we use four circular patterns that will help us define the transformation between a global $\boldsymbol{x}$ and the camera coordinate system $\boldsymbol{x}_c$, represented as:

$$\boldsymbol{x} = \boldsymbol{T}(\boldsymbol{x_c} - \boldsymbol{t_0}) \tag{4.3}$$

where $\boldsymbol{T}$ is the similarity transformation matrix [40]. And $\boldsymbol{t}_0$ represents the coordinate system origin. In our case we have $\boldsymbol{x_0}, \boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{x_3}$ to define $\boldsymbol{t_0}$ as the center of mass of all the patterns. Then, the application calculates the transformation between the vector $\boldsymbol{t}_0$ (camera coordinate frame) and matrix $\boldsymbol{T}$ (global coordinate system), which is the matrix that we are interested on. Figure 4.5 shows an example of the tracking with WhyCon roundels.

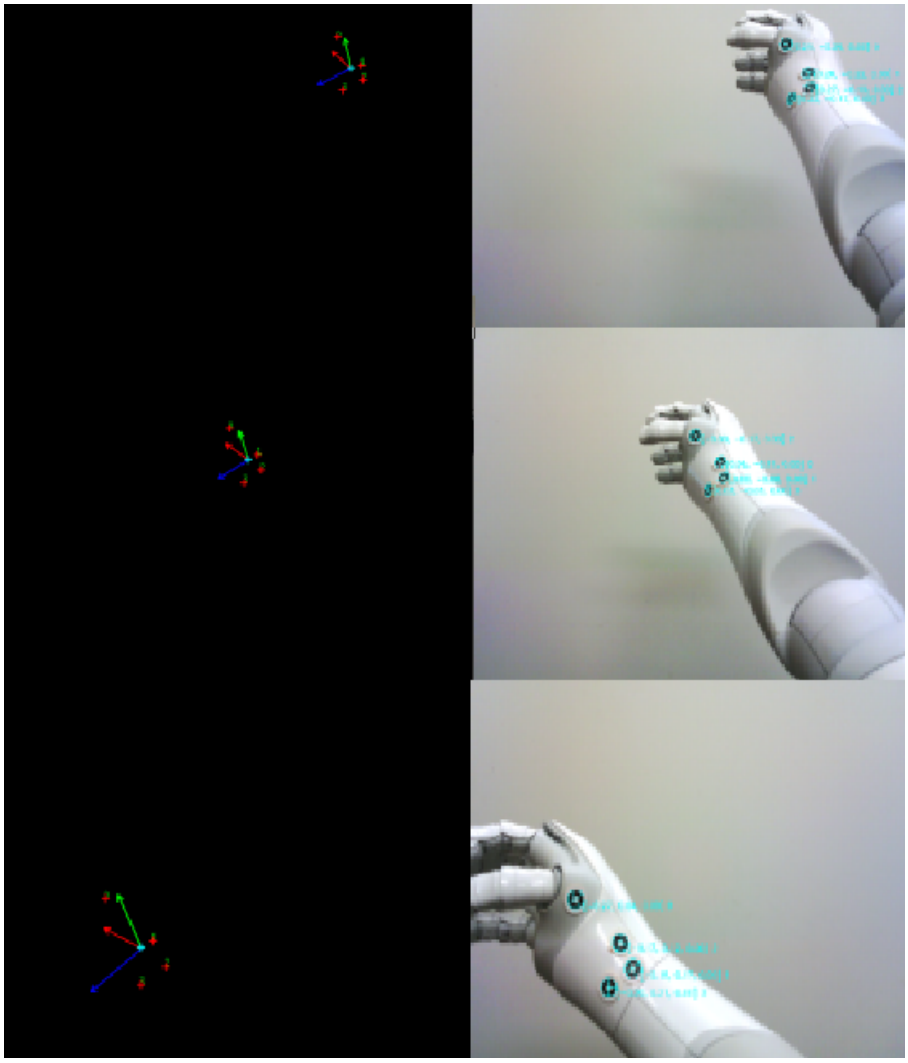

Figure 4.5: WhyCon Roundels Placed on Pepper Following Hand's Trajectory

From this setup we also obtain the transformation matrix from the torso to the camera frame and from the torso to the WristYaw frame.

From the different tests the advantages of using this algorithm are:

- The detection shows to be robust to outliers. This allows us to have a better precision in the tracking system
- And because the search looks at pixels that actually belong to the pattern, the computational efficiency is notorious compared to the QR codes, making it ideal for Pepper's robot camera.

4.2 Comparison of Methods

In order to validate that the method offered by WhyCon indeed gives better results than the QR codes we tested both markers on Pepper's hand. For the tests we placed first the QRcodes on the end-effector and run the QR code tracking application. This application is based on the OpenCV libraries merged in ViSP. On a second stage, we placed the WhyCon roundels on the end-effector and tried the ROS node. Figure 4.6 and fig.4.7 show the results of the performance of each method we tried.

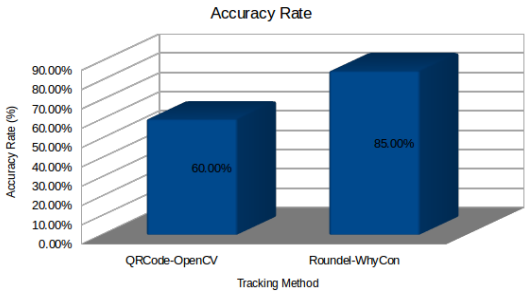

Figure 4.6: Comparison recognition Performance, Accuracy Rate for Both Methods.

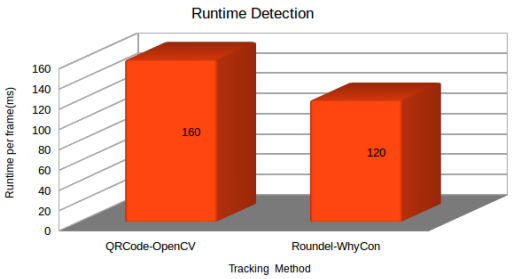

Figure 4.7: Comparison recognition Performance, Runtime Detection for Both Methods.

We can see that both methods have an acceptable updating time frame, under $200ms$. However, the roundels detection update is slightly faster than the QR codes ($120ms$ vs $160ms$).

Furthermore, the accuracy rate of both detection methods was measured. The WhyCon roundels method shows to be 25% higher than the QR codes, meaning that it is less likely to fail at detecting the pattern. We can see that the tracking with WhyCon roundels outperforms in every way the QR codes in OpenCV, reassuring that it is the right tracking method to track Pepper's hand.

CHAPTER 5

TRACKING THE OBJECT TO GRASP

Tracking the object to grasp at every iteration is as important as tracking the end-effector. As we saw in chapter 3 for our approach we assume the object to be static. Because we decided to apply a PBVS scheme our vector of desired features need to be built after the 3D model of the target object. In ViSP they propose a hybrid 3D model-based tracker (MBT) that allows the tracking of a markerless object as long as the CAD model is provided and the camera is previously calibrated [25]. The hybrid MBT is defined as the following twofold process:

1. *Extract the features of the object*. For which we use the Kanadae-Lucas-Tomasi method for feature extraction. These features reside inside the moving edges of the object. In order to track the edges, they use a classical approach for edge detection and tracking.

2. *Determine the pose of the camera to match the features* . In this step they use a virtual visual servoing to match the tracked features with the obtained 3D model.

Figure 5.1 shows a simplified version of the borrowed hybrid MBT from ViSP. Where **BLOCK A** represents the camera information extracted from the robot, **BLOCK B** represents the node that manages the communication between the naoqi_driver topics and the ones in ViSP for the detection and the tracking of the features. **BLOCK C** and **BLOCK D** take care of the update and description of the object for the tracking. In order to initialized the MBT we need to input the description of the object to grasp through a

CAD file. This CAD file feeds the description to these two blocks. And, fig.5.1 (1) and (2) show where the implementation of the KLT and edge tracker methods reside.

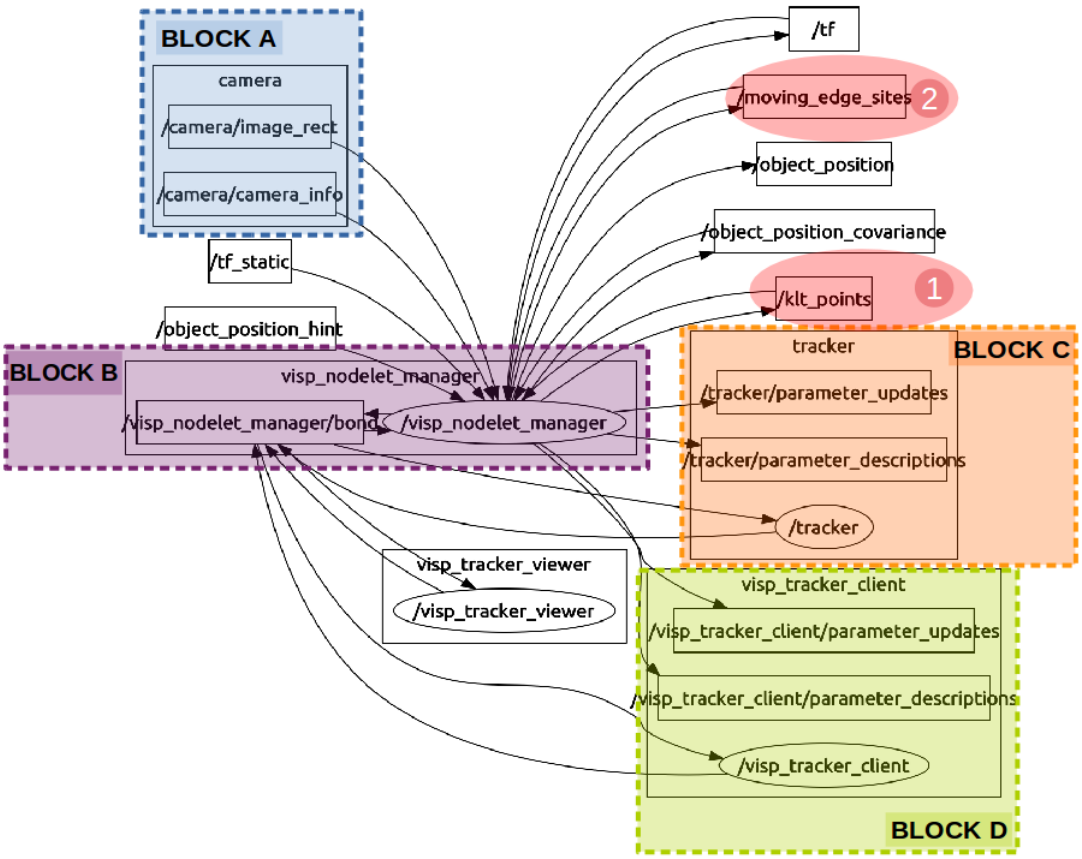

Figure 5.1: Simplified MBT Implementation in ROS

The 3D model input files needed by blocks **C** and **D** are built following the guidelines in [21]. These files provide the points coordinates and faces of the object to be tracked, along with the camera parameters. These descriptions are required to start the tracking after a couple of mouse clicks. Once initialized, the model is automatically detected and tracked. This task allows us to instantly determine the homogeneous transformation matrix of the object with respect to the robot's camera frame. Figure 5.2 shows the markerless hybrid MBT structure.

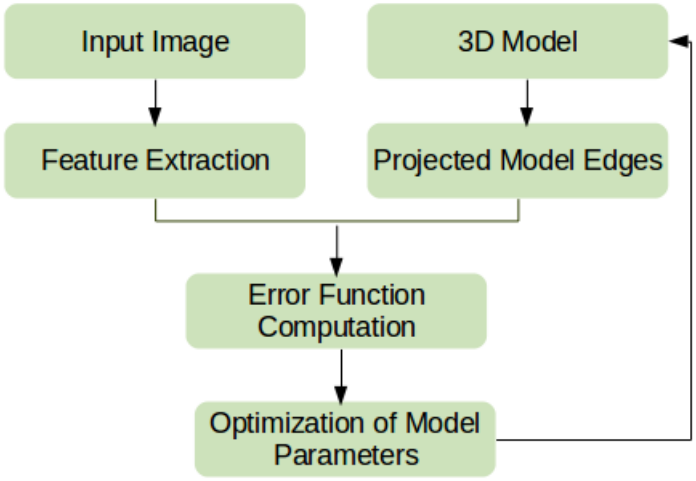

Figure 5.2: Model Base Tracker KLT + Edge Detection

The following sections give a summary on how this method works. We also show some of the tests we ran that helped us decide on this technique.

5.1 Extract the Features of the Object

As we previously saw the hybrid method concerns the features of the object and the tracking of edges, this combination adds robustness to the tracking.

5.1.1 Kanade-Lucas-Tomasi (KLT) Feature Extraction

This is a relative new algorithm that extracts the features of a textured object [42]. The algorithm consists of: 1) selecting features and 2) tracking these features from frame to frame. They define an image sequence as:

$$I(x+y+t+\tau) = I(x-\xi, y-\eta, t) \tag{5.1}$$

Where the first image is taken at $t+\tau$ and is obtained by moving every point in the current image t. And, the displacement motion is denoted as $\boldsymbol{d} = (\xi, \eta)$ of the point $x = (x, y)$ between time instants t and τ [32]. Instead of tracking pixels, in [42] they track windows that enclose enough textures. In order to handle the velocities they model an affine map, so that the different velocities are associated to different points in the window as shown in fig.5.3.

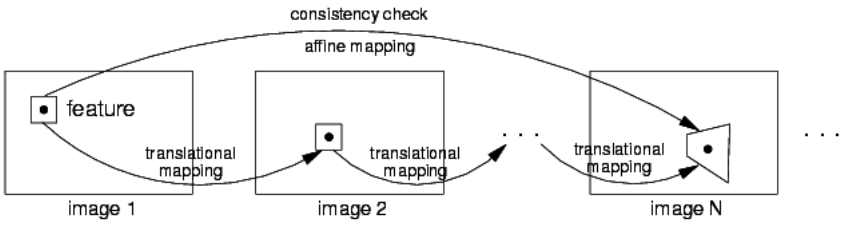

Figure 5.3: KLT Affine Map. Image courtesy of [42]

Now, redefining the left term in eq.5.1 as $J(x)$ and the right one as $I(x-d)$ we can express the new image model as :

$$J(x) = I(x-d) + n(x) \tag{5.2}$$

where now we introduce the noise term n [42]. Therefore, the displacement vector d is chosen to reduce the error defined as:

$$\epsilon = \int_W [I(x-d) - J(x)]^2 \omega dx \tag{5.3}$$

where ω can be:

1. In the best case scenario set to 1,
2. Be a Gaussian function with the emphasis on the central area of the window, or
3. Depend on the image intensity.

Because we are looking for regions with enough textures, in [42] they agreed that the richer regions are the ones located in the corners, edges or simply those that show high second-order derivatives. These regions are the ones enclosed by the window, for which we apply the theory of Harris corner detector.

The algorithm is summarized in fig 5.4. Where, to initialize the process for a template image, $J(x)$, we extract the neighborhood of Harris point and set the initial displacement as (0,0), then iterate in the following steps:

1. Take a patch and extract the neighborhood of x Harris point with current shift d, $I(x-d)$
2. Estimate the noise $n(x)$,

3. Compute the gradients ∇I at the translated coordinates

4. Estimate the displacement, and update the the translation estimate ($\boldsymbol{d} \leftarrow \boldsymbol{d} + \nabla d$),

5. Test the convergence $||\nabla d|| \leqslant \epsilon$ and iterate again,

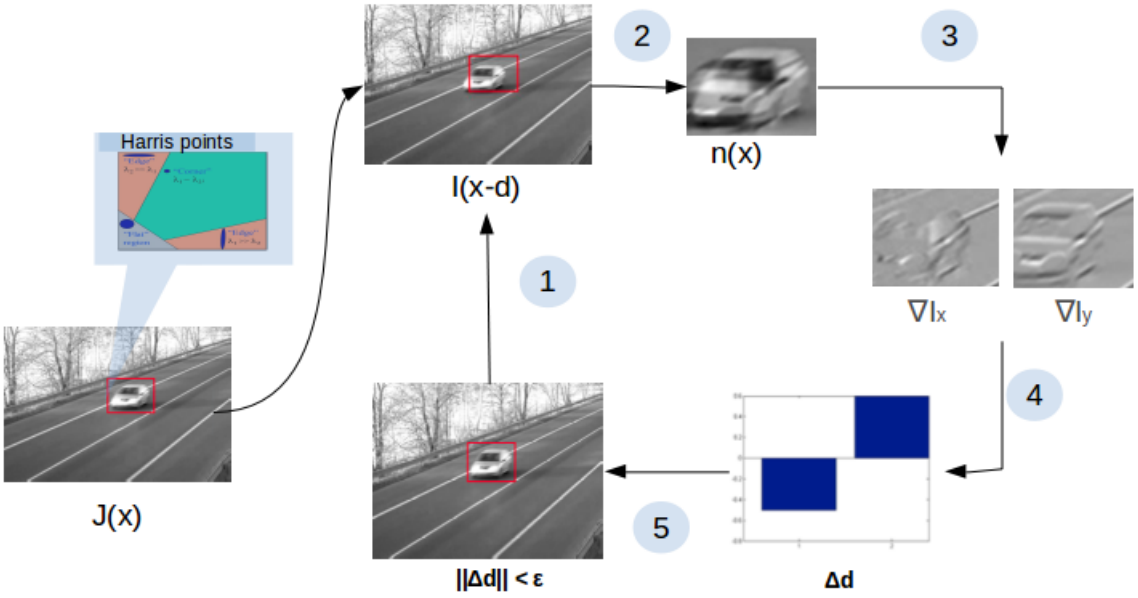

Figure 5.4: KLT Process Summary. Image courtesy of [42]

5.1.2 Edge Tracker

For this part, ViSP libraries offer an edge detection with gradient filter method. This is one of the most popularly used on edge detection.

The simplest approach is to apply the central differences from the input image or its smoothed version [36], and then apply a Sobel filter to it.

The edge detection process is summarized as:

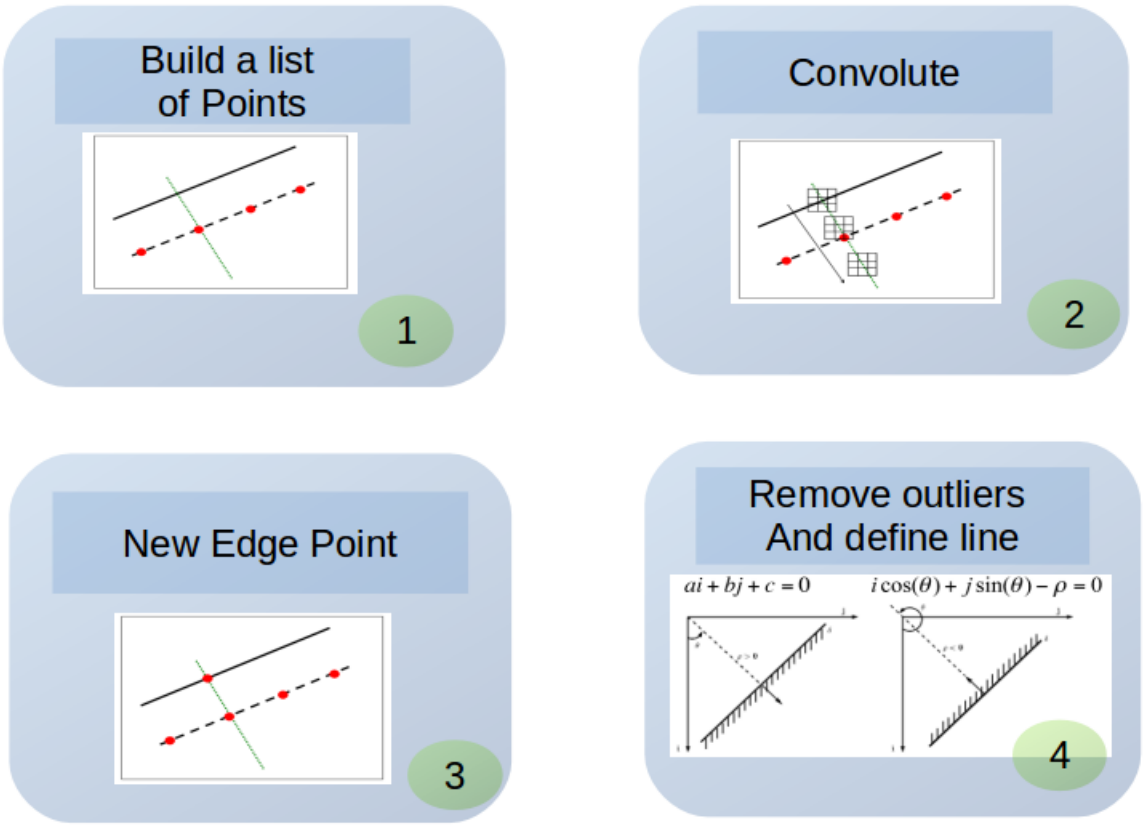

Figure 5.5: Edge Detection Method. Images courtesy of [8]

Where, from the illustration on fig.5.5:

1. A list of points is built along the normal to the edge,fig 5.5 (1).
2. For each point the convolution with the filter is calculated, fig 5.5 (2).
3. If the convolution result is high enough and close to the previous one then is considered a new edge point, fig 5.5 (3).
4. Finally, regarding the moving edges method we remove the outliers and define the line (as in fig 5.5 (4)) with the following equations:

$$ai+bj+c=0 \;\; and \;\; i\,cos(\theta)+j\,sin(\theta)-\rho=0 \tag{5.4}$$

The hybrid scheme is achieved by stacking the features extracted from the KLT and the edge tracker into the vector $\dot{s}$ adding them into a larger interaction matrix of size $nd \times 6$ that relates the object to the camera. Where n belongs to the number of

features and d represents the features dimension. These features are composed of a set of distances between local point features and the contours of the global 3D model [25].

To initialize the tracker the parameters from the CAD and XML files are needed. Once these parameters are input into the system the actual pose of the object can be calculated. A detail of the parameters can be found on appendix A.4.

5.2 Getting the Pose with Respect to the Camera

For this part of the algorithm they obtain the point transfer from the image homography. If we use the point transfer to ${}^2\boldsymbol{p} = {}^2\boldsymbol{H}_1 \, {}^1\boldsymbol{p}$, where ${}^2\boldsymbol{H}_1$ is the image homography, we obtain (x_i, y_i). With which we can represent the interaction matrix that links the variation of the point position to the camera motion, usually denoted as $L({}^2e_1)$ [25]. This iteration matrix has the form of the one calculated using the IBVS approach.

In order to know the pose between the camera and the world coordinate we can follow this definition:

$$\widehat{{}^nM_W} = \widehat{{}^nM_{n-1}} \, \widehat{{}^{n-1}M_W} \tag{5.5}$$

where ${}^1M_W$ is obtained from the 3D model files input in sec 5.1 [21]. This approach is only possible if all the tracked points remain in the field of view and if the number of tracked points remain constant. For example, if only 50% of the initial points remain then a new reference image is needed for the points to be extracted [25]:

$${}^nM_W = {}^n M_{R_k} \, {}^{R_k}M_{Rk-1} \ldots {}^{R_1}M_{R_0}^{R_0} M_W \tag{5.6}$$

5.3 Testing the Method

In order to test this algorithm for our implementation we compare it with the tracking method explained in chapter 4 expecting to obtain similar results. For this tests rounds we care about comparing the translation and rotational vectors obtained from the transformation matrices.

Since the hybrid MBT needs the 3D model of the object, we created the corresponding files that describe a tabasco box of $8 \times 3 \times 3$ *inches*.

For the tests, the WhyCon markers explained in chapter 4 were placed on the box and the translational and rotational information was extracted. Simultaneously, the model based tracker proposed by ViSP was put to run so that the same information could be obtained.

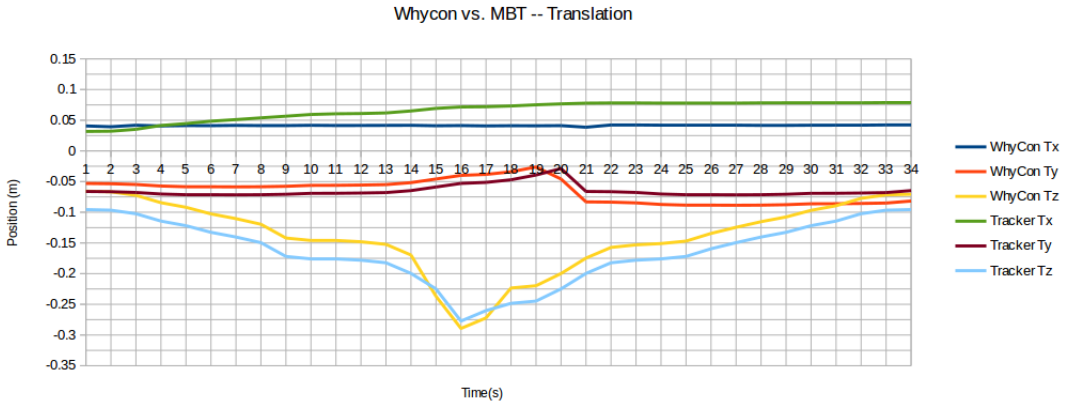

Figure 5.6: Comparison on the Tracking- Translation Vectors

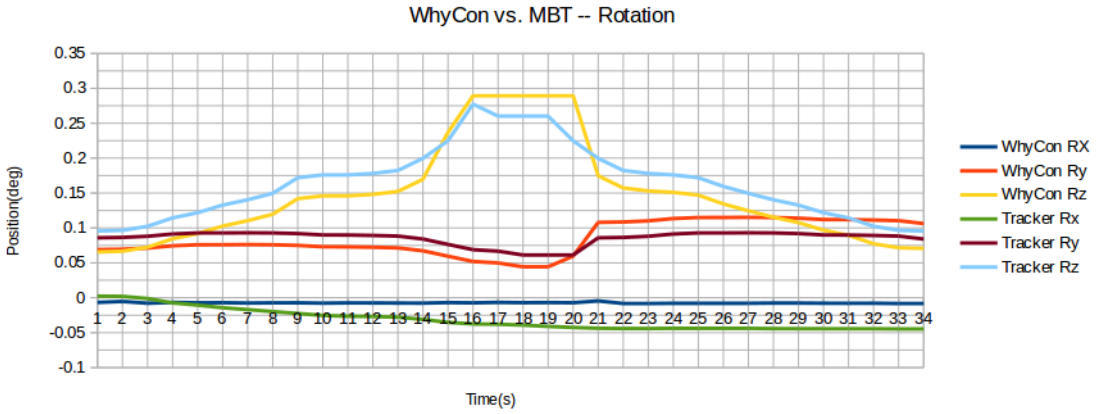

Figure 5.7: Comparison on the Tracking- Rotation Vectors

As seen in figures 5.6 and 5.7 the results are similar, both trackers show the same behavior. One of the greatest advantages noticed during the tests was the invariant behavior of the hybrid MBT to occlusions on the box features. As previously explained, we placed the roundels on the box.Fortunately, the roundels did not affect the extraction of features. Additionally, the edge tracking served to add robustness to the process.

Table 5.1 shows the error measurements for the different components of the translational and rotational vectors. From the table we see that the errors are very small for both vectors.

Table 5.1: Accuracy of the Tracker in Comparison with Whycon (meters and degrees)

	Tx	Ty	Tz	Rx	Ry	Rz
Mean	0.053	-0.066	-0.152	-0.017	0.088	0.15
Std	0.012	0.016	0.0554	0.018	0.019	0.023
Max	0.061	-0.038	-0.083	-0.001	0.100	0.28

On the downside, during the different testings we noticed the following disadvantages:

- The initialization needs to be as accurate as possible, otherwise the tracker fails.
- If there is not enough contrast around the contours or the occlusions are too large then it is most likely to fail. Because of the importance on the features amount, the choice of the object to grasp is important for the performance of the algorithm.
- It is very sensitive to illumination changes.
- It is less computational efficient than the WhyCon method. In comparison it has a delay of around 1 *second*. This delay can be handle by using a higher speed communication channel between the external processor and the robot. Or, by deploying the algorithm as a package inside the CPU of the robot. On the other hand, the tracker works well with small motions of the object. Because for our purposes the box is to be static we assume this risk in our implementation. Figure 5.8 shows the tracking results.

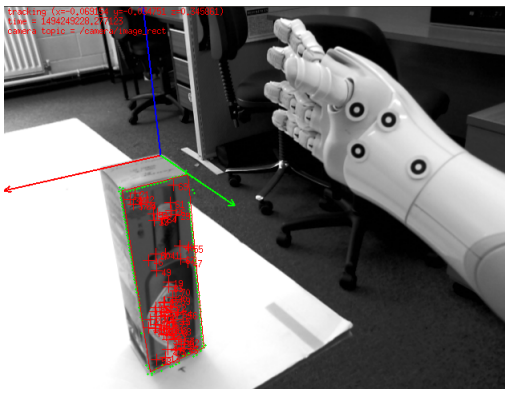

Figure 5.8: Final result for the Box Tracking

CHAPTER 6

CONTROLLING THE JOINTS IN VELOCITY

It is necessary for a successful visual servoing to control the robot's joints in velocity. Aldebaran offers a series of methods for Cartesian control on Romeo and Nao but not for Pepper robot [3]. However, Inria labs as part of their visual servoing platform offer a package that can be deployed on Pepper called *pepper_control* [6].

6.1 Pepper Velocity Control

As any other package that controls the joints in velocity, *pepper_control* manipulates the end-effector of the robot by using a inverse kinematics solver. The end-effectors can be controlled individually or in parallel.

In general, the geometric model of the robot allows to get the positions of an effector such as: $X = [P_x, P_y, P_z, P_{w,x}, P_{w,y}, P_{w,z}]$, where the general expression of the geometric model is defined as:

$$X = f(q) \tag{6.1}$$

Being the direct kinematic model its derivative [3]:

$$\begin{aligned} \dot{X} &= \frac{\delta}{\delta t} f(q)\dot{q} \\ &= J(q)\dot{q} \end{aligned} \tag{6.2}$$

where we know that $J(q)$ is the Jacobian matrix. Because the main purpose is to control any end-effector and deduce the joint position the inverse kinematic is defined as:

$$\dot{q} = J^{-1}\,\dot{X} \tag{6.3}$$

It is usual for J not to be invertible because is not a square matrix. In the case of *pepper_control* they use the Moore-Penrose pseudoinverse to solve this problem.

While testing the package *pepper_control* we noticed some aspects that need to be taken into consideration:

- Before using the package make sure that this is running. One of the disadvantages is that since it is not one of Aldebaran modules this needs to be manually started before being used.
- If a robot singularity configuration is found then a great velocity could be sent to the robot, letting Pepper to lose balance or if the safety is disabled, to crash.
- In the best case scenario the package should be directly connected to the sensors and actuators, just as the Aldebaran DCM module does for Romeo and Nao. Instead, the package works as a service for the ALMotion module. This extra connection means a small delay in the sending/reading data from the sensors. Fortunately, this delay is small enough that can be neglected and does not represent a great error for the servoing performance.

In order to test *pepper_control* we implemented a simple program to send and receive velocities in a sinusoidal form. In fig.6.1 the sent velocity is represented in green. And, the retrieved velocity from the sensors in represented in blue.

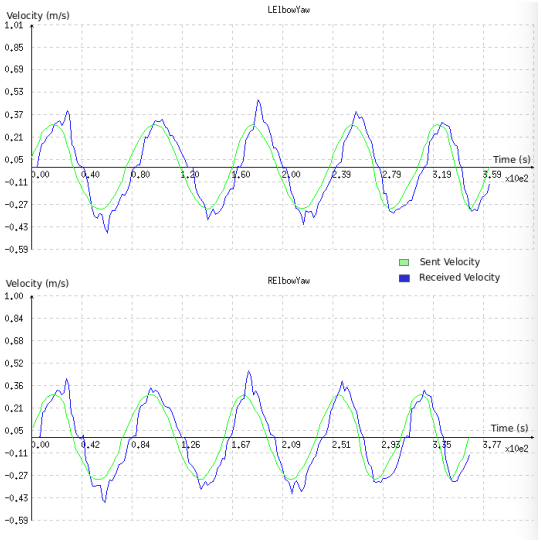

Figure 6.1: Pepper Velocity Control for Left and Right Elbow

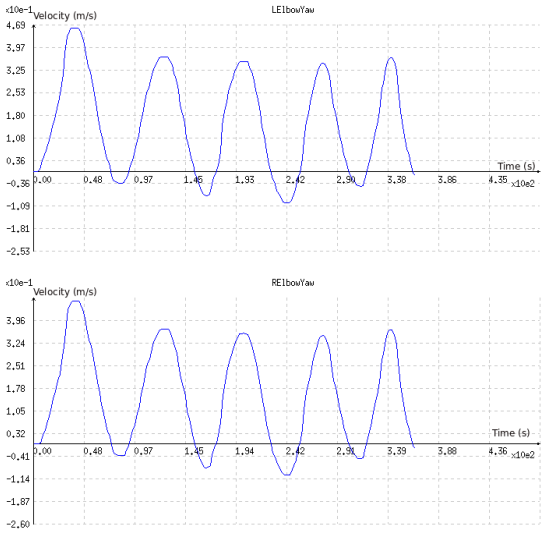

Figure 6.2: Pepper Velocity Control Error Measurement

From fig.6.2 we can see that the error is really small. Resulting in an average error of $3.24 \times 10e^{-1}m/s$. Because of this outcome we rely on the package to be used to control Pepper's joints in velocity in our implementation. The example shown in fig.6.1 is set to manipulate the XlbowYaw joints. For the servoing task, first we control the base and then in parallel the five joints of the right arm: RShoulderPitch, RShoulderRoll, RElbowYaw, RElbowRoll, RWristYaw, as shown in fig.6.3.

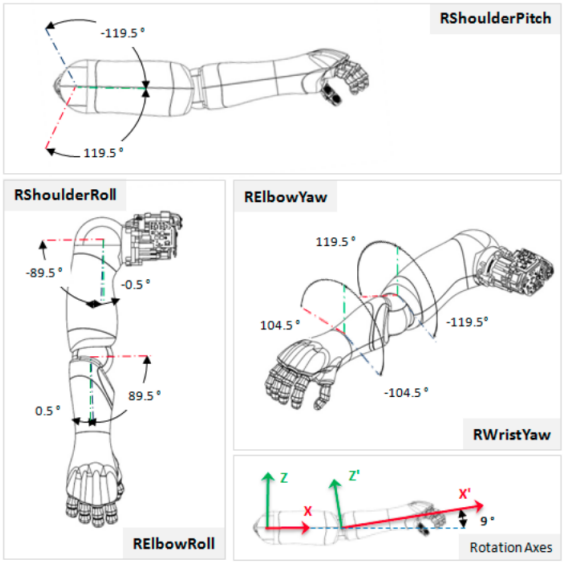

Figure 6.3: Pepper's Right Arm Joints. Image courtesy of [1]

CHAPTER 7

RESULTS FOR PBVS ON PEPPER

Up to this point we have shown the different modules that lead us to achieve a PBVS control technique for Pepper. We have seen the method that allows us to track Pepper's hand using roundels detection, the markerless hybrid MBT method to detect and track the box, how we control Pepper joints in velocity and the mathematical model to implement the core of PBVS.

In this chapter we present the final results of the unified system. A demonstration video can be found in:

- https://www.dropbox.com/s/dv3slax949tnllu/PepperVS.mp4?dl=0

And the code can easily be accessed and downloaded from:

- https://paolaArdon@bitbucket.org/paolaArdon/master_thesis_vs_pepper.git

In order to test the implementation a Lenovo Flex 2-14 with Ubuntu 14.04 LTS, core i7 was used. The Aldebaran version for the NaoqiSDK and the system image inside the robot was 2.4.3.

As we previously saw in chapter 3, we implement a closed loop technique combining PBVS and eye-hand configurations.

For our vision sensor we are using Pepper's bottom 2D camera. Both cameras have a vertical field of view (FOV) of 44.30°and a horizontal FOV of 57.20°[1], as observed in figure 7.1. Section 7.1 shows the results of implementing the PBVS by manipulating only the arm joints.

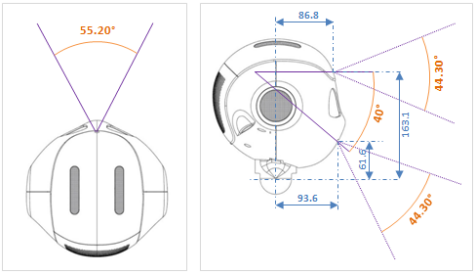

Figure 7.1: Pepper 2D Cameras Specifications. Image courtesy of [1]

These results are then compared to the ones obtained by using conventional path planning techniques. And section 7.2 describes the outcome of adding an extra joint to the servo, which represents the final version for our system.

Figure 7.2 shows the steps to run the final implemented PBVS. These steps are:

1. To initialize the MBT by clicking on the corners of the box. These are the points that we defined in the 3D model file. Once the MBT is started it calculates the error from the actual position of the robot to the previously saved desired position of the box, defined as *cMdBox* homogeneous matrix,

2. Using this matrix the velocities are calculated and applied to the base,

3. Once the error is small enough or 0 the base stops. An open loop motion process is applied to the right arm, so it goes inside the camera frame,

4. Now Pepper is able to see both the arm and the box. The hand position node is started and the matrix *eMh* that describes its 3D pose is extracted,

5. Once the position of the hand and the box are known, we extract the previously saved desired offset of the hand with respect to the box, *dhM*. The PBVS servo is started to take the arm as close to the desired position as possible,

6. Once the error is beneath the threshold Pepper closes its hand and grasps the box. Raises the arm and tracks a human using modules from Aldebaran for people perception+tracking (ALPeoplePerception + ALMovementDetction + ALEngagementZones) and delivers the box. Please go to appendix A.5 for a summary on how Pepper follows people.

Note: a detail of the matrices can be found in appendix A.2.

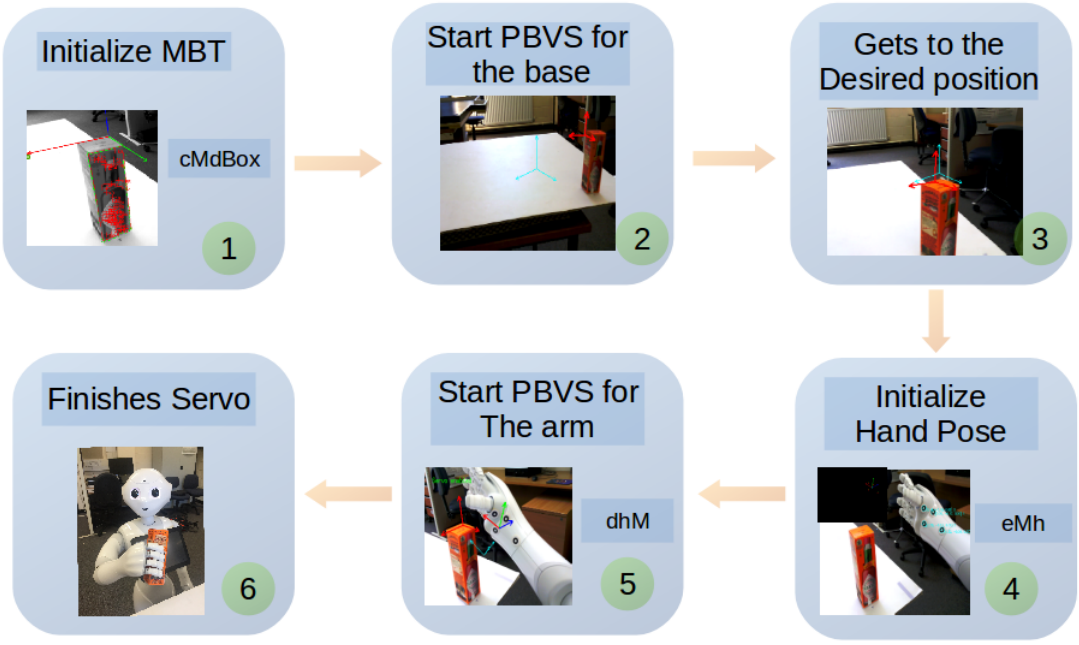

Figure 7.2: Process to Run PBVS application on Pepper

7.1 PBVS with 5-DOF

As we remember, Pepper has a total of 20-DOF. From which 5-DOF belong to the arm. In most cases it is impossible to completely satisfy the desired position and orientation of a 6-DOF end-effector with a 5-DOF manipulator. This does not mean the task is impossible, yet it will not be able to grasp the box at all times.

From chapter 3 we learned that *MoveIt!* easily allows us to test motion planning and analyze the inverse kinematics (IK) of a given robotic system. For these purposes *MoveIt!* offers an easy to integrate solution that allows the user to input the robotic description files along with a series of actions to carry. In our case we are interested on the IK analysis on Pepper's arm. In this manner we can predict the positions to which Pepper can potentially grasp by using 5 joints. Moreover, *MoveIt!* gives us a basis to compare our PBVS application results against the ones given by motion planning techniques.

For the simulated environment in *MoveIt!* we use Pepper's URDF (Unified Robot Description Format) along with a model of a box, so that we can reproduce the grasping. Figure 7.3 shows a comparison of the real world scenario versus the one in the simulator.

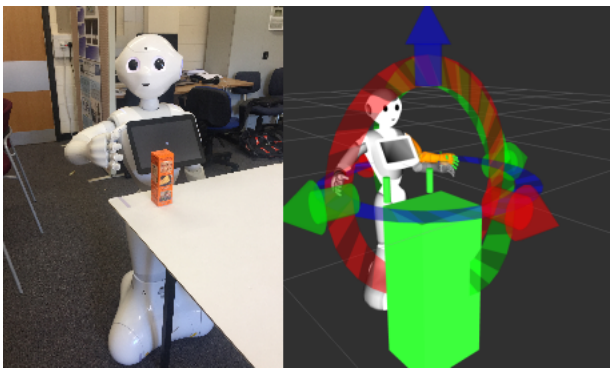

Figure 7.3: Real robot and MoveIt! Simulator to calculate Affordable Positions for the Box to be Grasped

Before going into the evaluation process it is worth mentioning that we do not consider grasping quality nor quality of the movement of the object while being lift-up.

For this part of the process, we count a successful grasp when from the full set of different simulations done in *MoveIt!* the simulated robot is able to reach and lift-up the box. On the other hand, a grasp is considered to be failed if the simulated robot is not able to reach the desired position. The simulated tests are done following these steps:

1. Generate a random reachable position for the box,
2. Start the IK analysis on Pepper to determine if the box can be grasped or not,
3. If the box can be grasped, generate the possible kinematics solutions that would allow Pepper to successfully reach and grasp the box
4. Otherwise, generate another position/orientation for the box and iterate on the process,

In total we generated 144 different positions for the box, only for 33 of these positions the system was able to find a solution. This represents a 23.20% of success rate (fig.7.4). As expected, the positions where the system came with a positive outcome were the ones where the box was placed close to Pepper's arm.

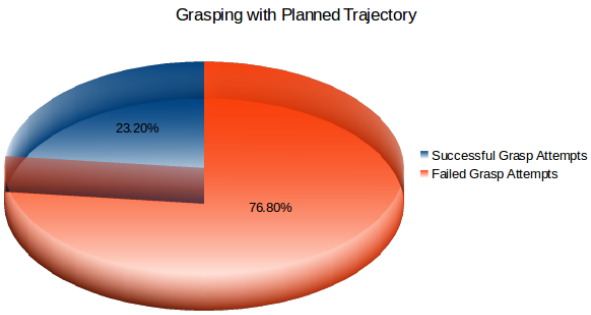

Figure 7.4: Graphical Representation of Success vs Failed Attempts for Grasping with Motion Planning

The average time for the system to find a solution was 2.46 *seconds*, as seen in fig 7.5. On the other hand, when *MoveIt!* could not solve the system it tried for as long as 23.14 *seconds* ending up with 76.80% of failed attempts.

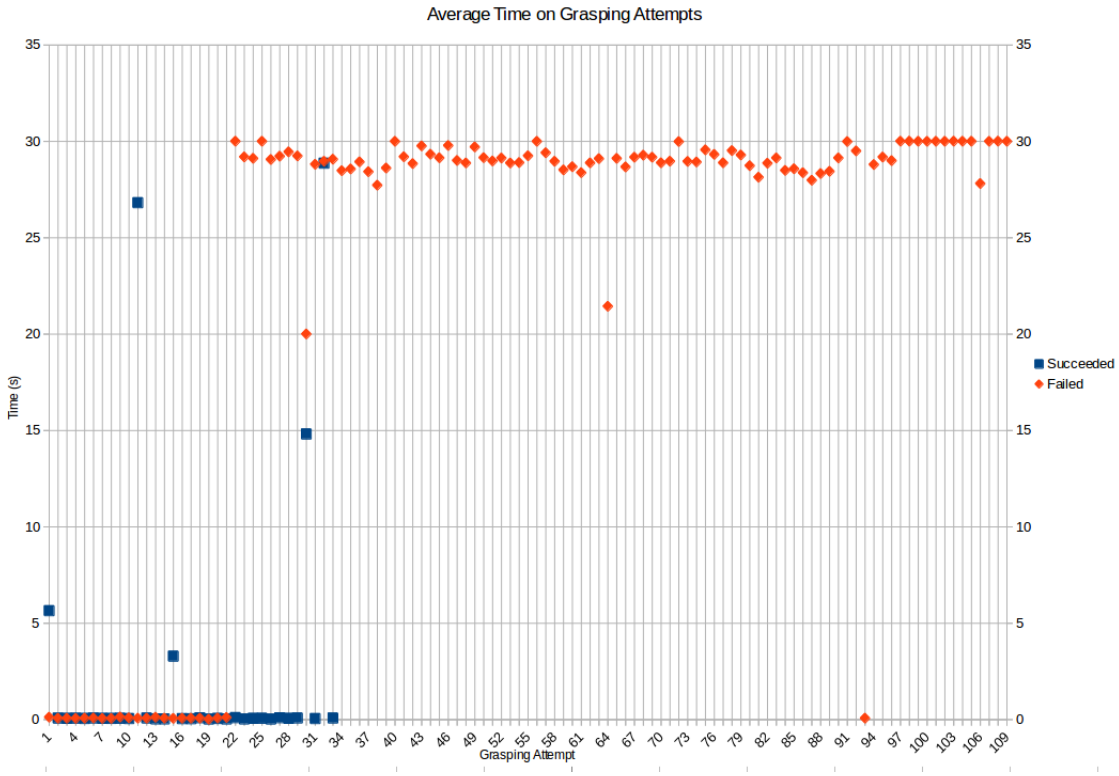

Figure 7.5: Time Taken to Succeed or Fail for Grasping

We now have an estimation of what Pepper can do by only manipulating the arm and so we can test our, so far implemented, PBVS with eye-to-hand configuration approach.

In the real scenario, we do not consider to measure quality of grasp or movement of the object while being lift-up, but the number of successful attempts.

For our purposes we consider a successful grasp if the box is not dropped by the end-effector during the reaching process and can be lift-up. And, we consider it to be failed if the box was dropped in the process or if it was approached correctly but the robot did not lift it up.

	Predicted as No	Predicted as Yes	
Actual No	TN=7	TP=2	9
Actual Yes	FN=3	FP=3	6
	10	5	

Figure 7.6: Results of Grasping Attempts with Pepper – 5-DOF

Out of 15 cases with different box positions only on 3 of them Pepper was able to successfully grasp the box. For these successful cases the initial positions of the box w.r.t the camera frame were: a) $x = 0.1099, y = -0.053, z = 0.347$ b) $x = -0.069, y = -0.034, z = 0.3458$ c) $x = 0.0363, y = -0.0634, z = 0.525$, which are close to the Pepper's arm. Figure 7.6 help us visualize the performance of the implemented control technique.

For this version of the PBVS we obtain a sensitivity rate of only 40%, which means that Pepper is most likely to fail at grasping the box. Only 20% of the time Pepper was able to accurately predict and grasp it. As expected, the servo is not as efficient as the motion planning technique. In this case, the average time for Pepper to reach the desired position (when possible) was 33 *seconds* while using a static gain $\lambda = 0.04$. Figure 7.7 shows an example of a successful grasp done with Pepper using 5-DOF.

Figure 7.7: Successful Grasp with 5-DOF.

Figure 7.8 shows the error decrement of one of the successful cases. As it can be seen, the error never reaches 0 for the rotational and the translational staying rather in steady state. For this implementation version, the threshold error in the translational was set to 11 *mm* and the rotational threshold to 14 *degrees*, which are high threshold values for the task.

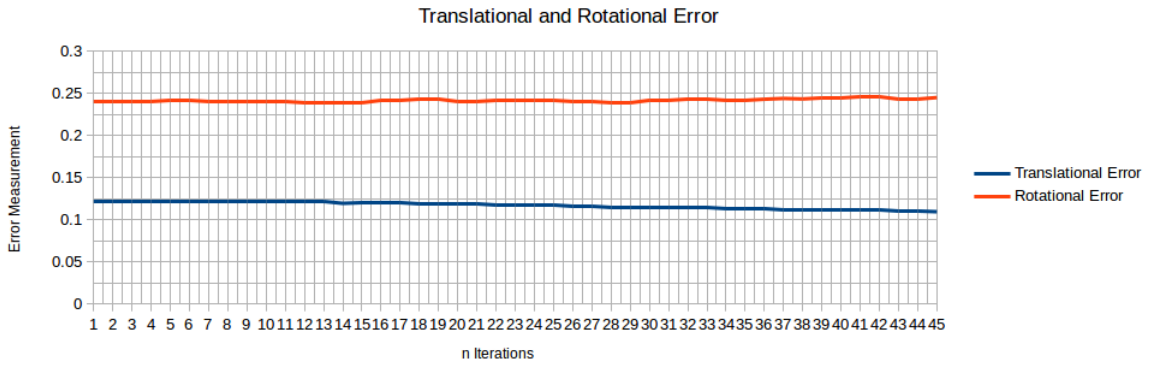

Figure 7.8: Mean Square Error for 5-DOF

Moreover, fig 7.9 shows the applied velocities to the joints. Where, the system arrives to a convergence, however the last joint *RWristYaw* never converges. This is because it stays in the calculation state trying to reach a position is physically impossible for the joint, which clearly influences on the big threshold for the errors.

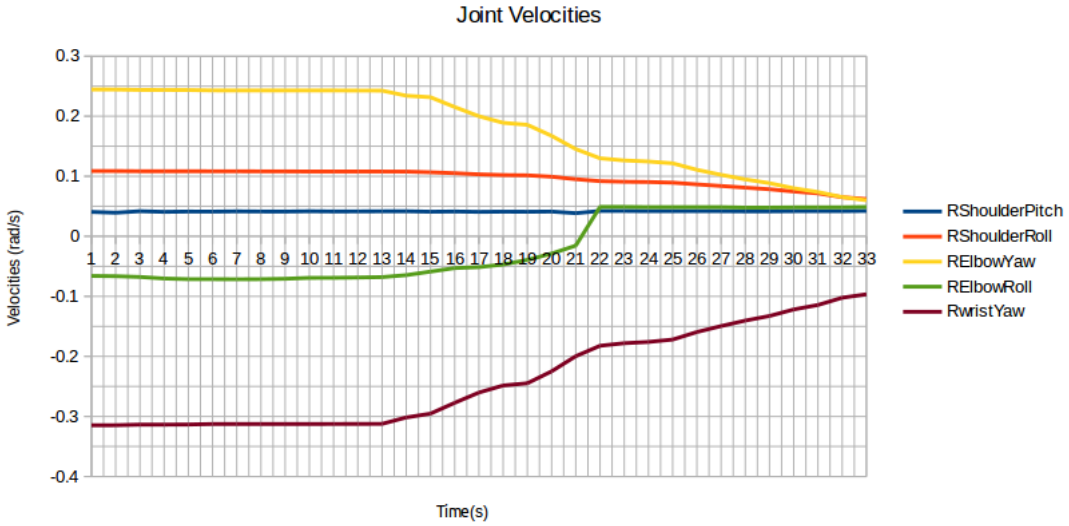

Figure 7.9: Right Arm Joints Velocities

In literature many analytical solutions can be applied for inverse kinematics that do not fulfill the required DOF, as in our case. These techniques vary depending on the missing DOF, ranging from optimization techniques to satisfying the orientation vectors.However, these techniques are proven to work on motion planning techniques and can be computational expensive for real time applications.

7.2 PBVS with 6-DOF

Instead of manipulating the orientation vectors we decided to satisfy the number of DOF required for a successful grasping. We do this task by extending the PBVS to the base of the robot. Given our modular implementation, we simply add a function that implements a PBVS with eye-in-hand configuration. In this way, the robot successfully travels to a position where is sure that the offset between the box and the camera is small enough for process in section 7.1 to start.

Now we take into account the transformation matrix between the HeadPitch and the torso of the robot (which is for now our $\{h\}$ frame) to calculate the velocities that

need to be applied to the base in order to get to the desired position. Contrary to the previous approach were we set a static gain for the control scheme, for this version of the implementation we use an adaptive gain for both PBVS tasks. Figure 7.10 shows the behavior of the velocities applied to the base.

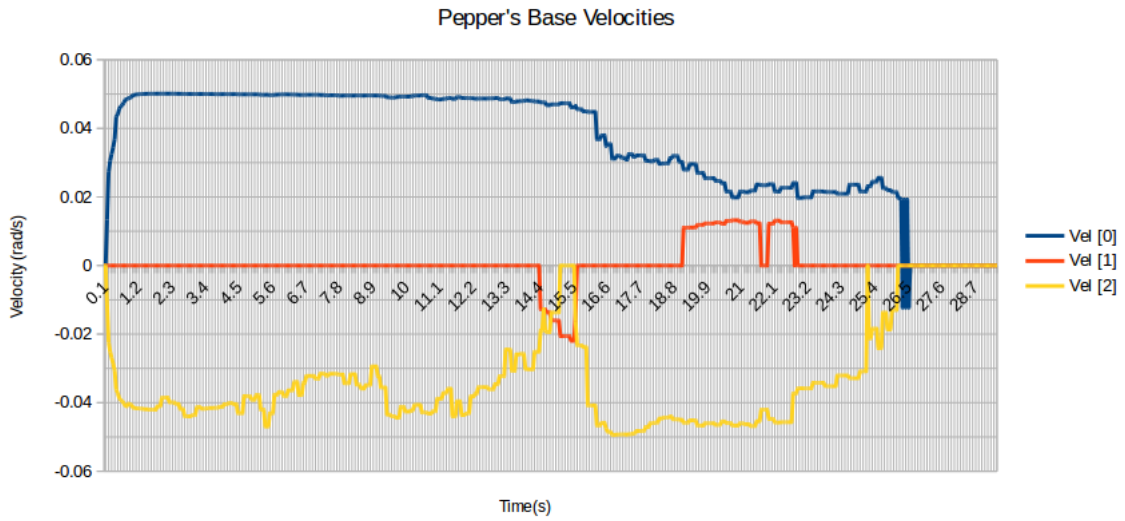

Figure 7.10: Velocities for Pepper's Base

To test the base PBVS, the robot was placed 50cm away from the box, getting to the desired position in 27 *seconds* while using an adaptive gain of $\lambda_0 = 0.02$, $\lambda_\infty = 0.03$ and $\lambda'_0 = 3$ for the base task.

Now we can test both PBVS tasks. For this experiment we used $\lambda_0 = 0.02$, $\lambda_\infty = 0.08$ and $\lambda'_0 = 3$. Figure.7.11 shows the new confusion matrix for 6-DOF with a total of 25 test runs. With the addition of 1-DOF to the servo, out of 25 trials Pepper successfully grasped 17. Improving the sensitivity rate from 40% to 80%. Which means that now we rarely miss the box.

	Predicted as No	Predicted as Yes	
Actual No	TN=3	TP=4	7
Actual Yes	FN=1	FP=17	18
	4	21	

Figure 7.11: Results of Grasping Attempts with Pepper – 6-DOF

During the different tests it was noticed that the attempts where Pepper miss-predicted the goal position were the ones where the hybrid MBT fails. Either it gets lost because of illumination, misses features and/or is not correctly initialized (remember we manually initialize the MBT with a couple of clicks). Figure 7.12 shows the error measurements in cm and rad. Where now we have an average error of 1 mm in and 5 deg. Versus the 11 mm and 14 $degrees$ of section 7.1. Furthermore, now we see the exponential decay of the error.

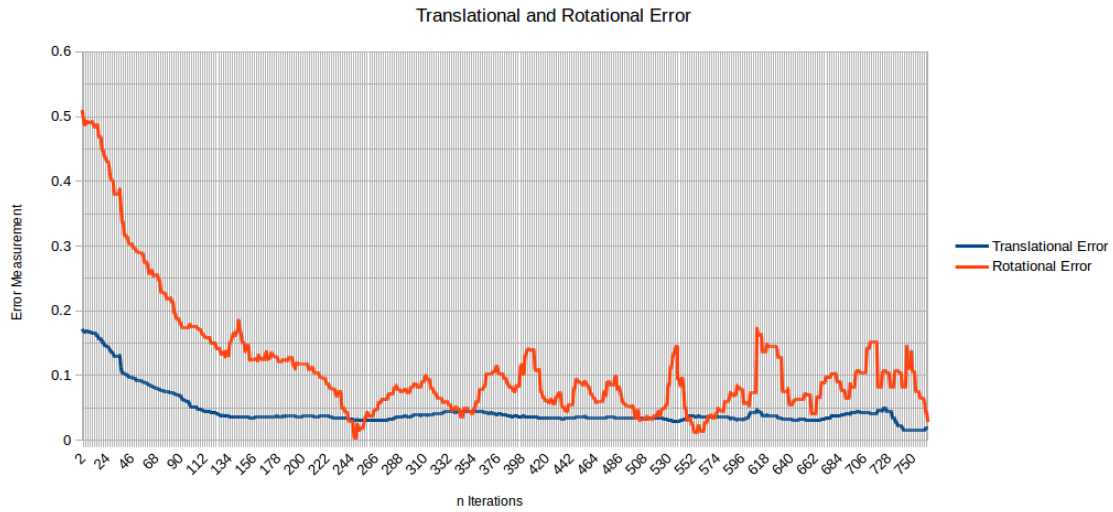

Figure 7.12: Mean Square Error with 6-DOF

It is worth noticing that each iteration is done at $20Hz$, so we process the signal every 0.05 $seconds$. In this case, the task error arrives to convergence in 17 $seconds$, which is approximately half of the time it took in section 7.1. A better detailed image of the decay on the tasks error for the arm PBVS can be seen on fig 7.13.

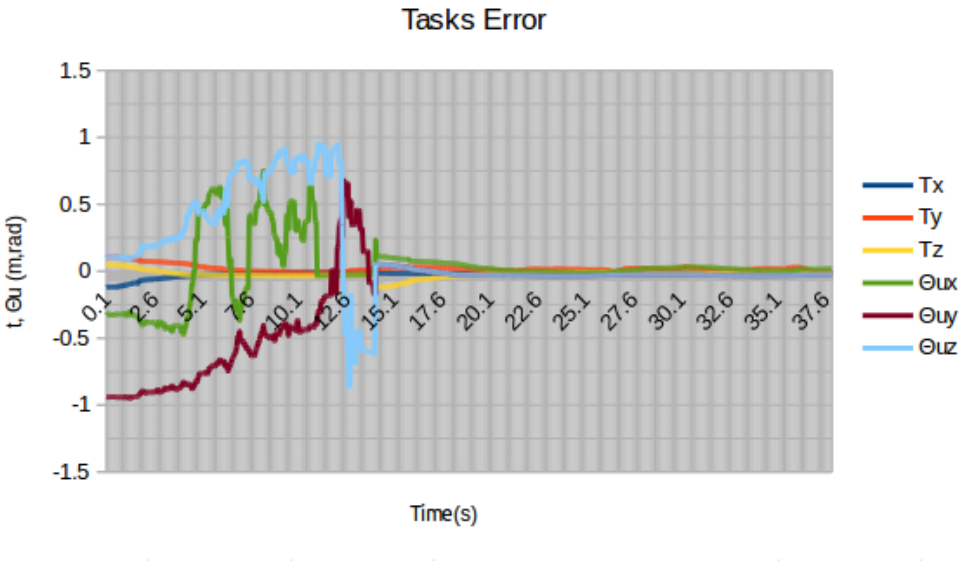

Figure 7.13: Task Error for the PBVS on Pepper

Now that we have improved our performance we want to know the right tunning of the parameters so that our approach is as efficient as possible. Given our implementation we care about tunning the adaptive gain values. Figure 7.14 shows the output of the applied velocities when $\lambda_0 = 0.06$, $\lambda_\infty = 0.07$ and $\lambda_0' = 3$. The convergence time in this case is approximately 25 *seconds*. In order to achieve the exact position of the box, we consider a 4mm and 3 *deg* threshold error.

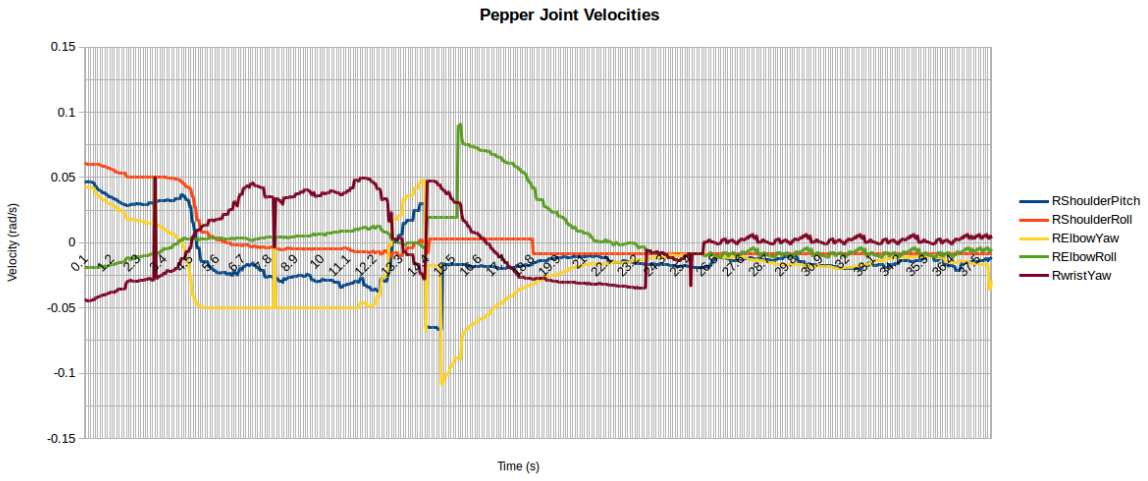

Figure 7.14: Joints Velocities for $\lambda_\infty = 0.07$

Now let's see what happens when we use : $\lambda_0 = 0.06$,$\lambda_\infty = 0.1$ and $\lambda_0' = 3$ As seen

in fig.7.15 the convergence time is around 12 *seconds*. In this case the error threshold increased to 6mm and 8 *deg* so that Pepper reached to the box without dropping it.Remember that we do not have tactile sensors to get feedback.

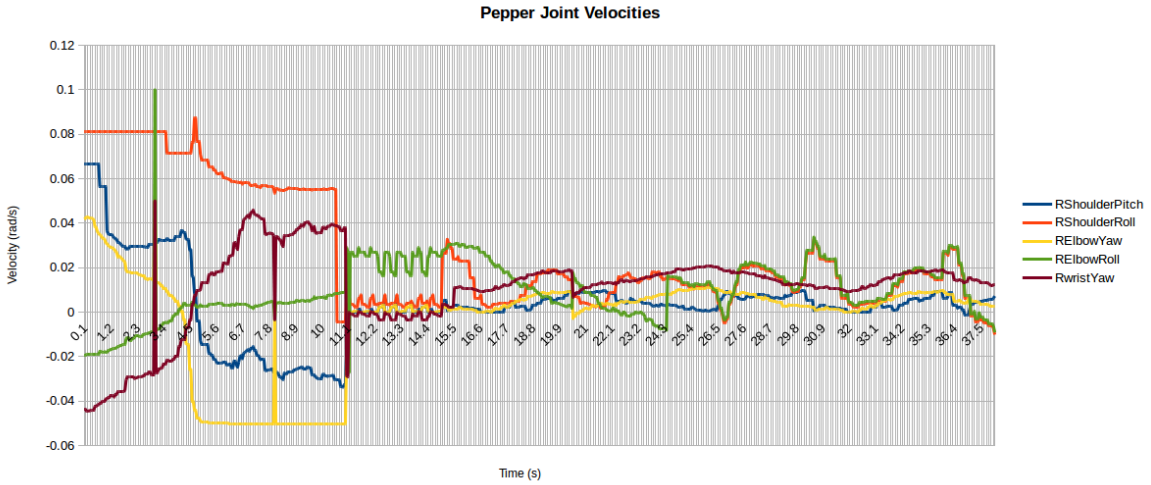

Figure 7.15: Joints Velocities for $\lambda_\infty = 0.1$

On the other hand, if the value is set as low as $\lambda_\infty = 0.02$ the convergence time increases to 30 seconds. However, the error threshold is set to 1mm and 1 *deg*, meaning that we have a more accurate result.

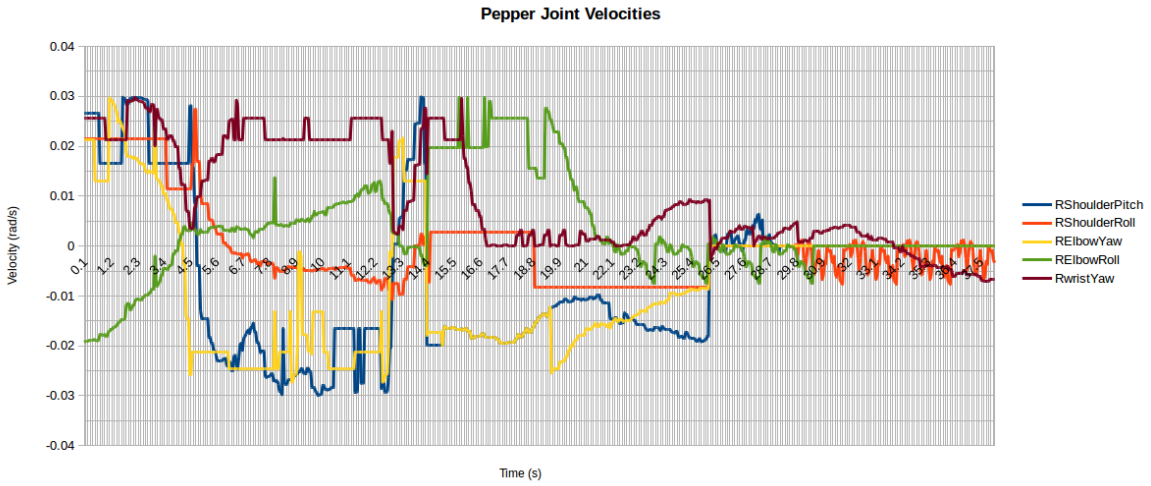

Figure 7.16: Joints Velocities for $\lambda_\infty = 0.02$

In all the cases of changing our adaptive gain the system arrives to the goal position traveling a relative straight line in the Cartesian space, as shown in fig.7.17. For the case

shown in the figure the adaptive gain was :$\lambda_0 = 0.6$, $\lambda_\infty = 0.07$ and $\lambda'_0 = 3$. This trajectory demonstrates that the method follows the optimal solution to the goal position while keeping stability.

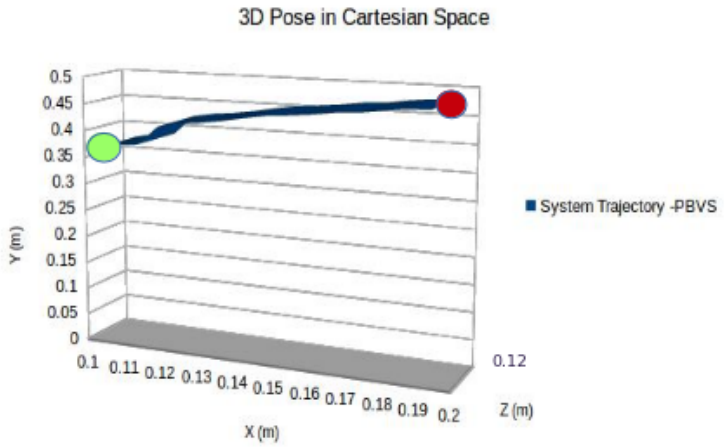

Figure 7.17: System Behavior for PBVS . Goal position: $x = 0.104;\ y = 0.340; z = 0.0670$

Table 7.1 shows a summary of the results from varying the λ_∞ parameter from the adaptive gain. Where we can compare the threshold values chosen for the translational and rotational errors , $||e_{u_t}||$ and $||e_{u_\theta}||$ respectively and their convergence time.

Table 7.1: Summary for Tunning Parameters

λ_∞	$\|\|e_{u_t}\|\|$	$\|\|e_{u_\theta}\|\|$	**Conv. Time**
0.02	1 mm	1 deg	30 sec
0.07	4 mm	3 deg	25 sec
0.1	6 mm	8 deg	12 sec

CHAPTER 8

CONCLUSIONS

This work presents a new application for Pepper robot: grasping objects using visual servoing techniques. To the date, there are not reported results of visual servoing techniques implemented on Pepper with the purpose of grasping objects. For our implementation we do not measure the grasping quality based on the object movement while being lift-up, nor on how the hand approaches the object. Rather, we count a successful grasp if Pepper was able to reach, grasp and lift up the box without dropping it.

From our implementation, we see that the combination of the hybrid markerless MBT for the box, the roundels tracker for Pepper's hand along with an adaptive gain for the control scheme allows the final system to be notably robust and efficient. Given the modularity the implementation it can adapt easily to other humanoid robots and objects. Despite the system limitations, our application demonstrates to be reliable and suitable for real time tasks.

8.1 Discussion

For our purposes we discuss the different modules that build our application: the two tracking systems and the core of the control scheme. Regarding the hand tracking system, on one side, they are discrete looking on Pepper's hand. On the other, the method adds precision to the calculations of the pose of the hand while not representing a computational burden for the system. Which is key for real time applications. Additionally,

thanks to the adaptive thresholding the tracker does not get affected by light changes.

Moreover, the hybrid markerless MBT that we use to track the object shows to be adequate for the application despite our low frame rate . Where the most important advantage is its robustness to camera occlusions by the robot's hand when approaching/grabbing the object. Furthermore, with the right conditions and communication channel between the robot and the external processor, this MBT shows to be invariant to the small movements provoked by the hand-object collision.

Thanks to the adaptive gain the implemented PBVS shows relatively small converging times. Where the whole task, meaning approaching and grasping the box, takes an average of one minute to complete. By comparing the results of chapter 7 section 7.2 with the ones presented in section 7.1 we clearly see difference for the velocities discontinuities and oscillations. From the results of the varying λ_∞ we observe that the applied velocities are high when the error exponential decrement is high and small when the error decrement is small. Where the oscillations are directly related to the λ_0 or λ_∞ gain value.

From the summary table 7.1 in chapter 7 the difference in the system performance when varying the gain is notorious. The greater the λ_∞ value the faster the system arrives to a convergence time. However, also the greater the threshold for the error so that the robot does not drop the box during the reaching process. In the example given in table 7.1 we see that the system achieves a precision of 1 mm and 1 deg when λ_∞ is really small, however it takes around half a minute to grasp the box. For our purposes we care more about precision than computational time. Therefore we use a combination of $\lambda_\infty = 0.07$ but a $\lambda_0 = 0.02$. In this manner, we achieve a relative small error slightly faster but reducing the oscillations when the error is small so that we have a higher precision when grasping ,taking 28 *seconds* to complete the task. Contrary to the results in section 7.1 where we took 33 *seconds* to complete the grasping with big threshold values.

After the different tests we can assure the system is stable for real time applications, specially taking into account the possible perturbations caused by these varying gains. This stability is mainly demonstrated when performing the PBVS for the base. For which, there is a high gain at the beginning of the task that then switches to a low gain when the robot gets closer to the box. This change produces some oscillations, that if not correctly applied, may affect the object tracking due to the motion of the camera. Hence, the importance of parameter tuning to avoid instability in the system.

This work also demonstrates the advantages of using a visual servoing technique

above traditional path planning algorithms for grasping purposes. Specially on a system which end-effector degrees of freedom are less than 6. For our system, the PBVS approach shows a success rate of 48.8% higher than the conventional motion planning methods.

As a summary, table 8.1 compares the results from using motion planning with *MoveIt!*, our 5-DOF PBVS and the final 6-DOF PBVS.

Table 8.1: Comparing *MoveIt!*, 5-DOF PBVS and 6-DOF PBVS

	MoveIt	**5-DOF PBVS**	**6-DOF PBVS**
Time Performance (sec)	**2.6**	33	28
Success Rate	23.20%	20%	**72%**
Sensitivity Rate	1	0.4	**0.8**

As observed in this table, the final PBVS outperforms the other two except in computational time. As we can remember from both approaches, 5-DOF PBVS and 6-DOF PBVS, besides adding a joint to the servo we modified our control scheme from a static gain to an adaptive one. This change helped us achieve a faster converging time regarding the PBVS for the arm. It is worth noticing that this timing does not count the PBVS for the base, in the case of 6-DOF PBVS, where it takes an extra 30 *seconds* to arrive to the box position. Nonetheless for our purposes we choose precision over computational efficiency.

8.1.1 Disadvantages of the Implemented Method

Despite the good performance of the final implementation, the system has some disadvantages that need to be taken into account:

1. Unfortunately, because we are relying on the patterns to locate the hand, the end-effector position/orientation is limited to an angle where the camera can retrieve the image of the roundels.

2. The MBT is sensitive to illumination changes, which causes inaccurate calculations for the position of the box.

3. The MBT does not work properly under 5GHz frequency for the communication channel. There is a great tendency to loose features, therefore to miscalculate the position of the box.

4. As expected, the larger the distance from the box the harder it is for the tracker to be initialized. After many testing rounds we found that 45 *cm* represents a safe distance for Pepper to start the PBVS for the base.

5. Because we do not have control of the fingers in Pepper's hand, in order to save the goal position we need to take into account the length and the current position of the fingers, so that when Pepper is approaching the box the robot does not drop it.

6. The current implementation only takes boxes as input 3D models.

7. Additionally, the current implementation is proven to work with the 2.4.3 Naoqi version but not on 2.5. Among different factors,the most important is that the package that controls the joints in velocity is not compatible with the newest release of the NaoqiSDK.

8.1.2 Comparison with Other Approaches

To the date there are not reported results of any method applied on this robotic platform that tries to achieve a grasping behavior. However, for the sake of evaluating the validity of our application, we can check the results with similar methods applied on similar robots. In [39] we find a similar work for Nao robot. They also implement a closed loop PBVS with the purpose of grasping a box. For the implementation in [39]: Nao has 4-DOF on its arms, only receives feedback from the vision sensors and has the same camera specifications as Pepper. They achieve the servo task by manipulating the head and arm joints,giving them a total of 6-DOF. To track their target object they use a moving-edge MBT that tracks solely the edges of the box. In their approach they use the kinematics of the robot to get the joint positions. Moreover, they use a static gain for the control. On their final results, they report a convergence time of 54.67 *seconds* with a threshold error of 5*cm* and 3 *degrees*, achieving a grasping success rate of 52.6%.

Even though they achieve a faster convergence time than our approach on a different robot, our combination of algorithms allows us to have a better precision and a higher success rate with Pepper.

8.2 Future Work

We consider our application to have a great potential, giving room to numerous extensions. Some of them being:

- The biggest setback in terms of processing is the tracking of the box features, which is also the main cause of failure in the system when grasping. The communication channel has a key role in this task. Therefore it would be nice to integrate this tracker into the robot's CPU instead of having it running on the external computer.
- At the moment Pepper is able to grasp only boxes that fit in its hands. In the same manner for future extensions, cylindrical objects could also be grasped by implementing a parallel module.
- Instead of only manipulating one hand, both hands could be integrated into the servo. Hence, Pepper would be able to lift greater objects in terms of surface but always taking into consideration the weight constraints.
- Another nice fix would be to remove the visual markers on the hand and integrate a MBT to track its position instead.
- For the first step: the PBVS for the robot's base, the box needs to be inside the camera frame. A nice extension would be to add a TLD (Tracking Learning Detection) tracker so that it looks for the box in the room before initializing the MBT.
- When is going from the base PBVS to the hand PBVS,it would be good to implement a parallel module that applies IBVS to keep track of the hand while making sure the box is still in the camera frame (instead of just applying an open-loop motion).
- In terms of implementation, the modules are triggered manually. It would be good for demo purposes to add speech recognition so that Pepper follows a voice command to search and grasp the box.

APPENDIX A

APPENDIX A - PARAMETERS DETAILS

This chapter shows in detail the camera calibration and the different homogeneous matrices used in the implementation.

A.1 Camera Calibration Parameters

Camera parameters for perspective projection without distortion:

$$p_x = 273.4914551 \quad p_y = 275.7431335$$
$$u_0 = 155.1124547 \quad v_0 = 126.0573575 \tag{A.1}$$

Where (u_0, v_0) are the coordinates of the principal point in pixel; and (p_x, p_y) are the ratio between the focal length and the size of a pixel.

A.2 Matrices Details

A.2.1 Homogeneous matrix eMh

Transformation matrix between the hand the RWristYaw:

$$\begin{bmatrix} 0.9904602583 & 0.1154415606 & -0.07524442084 & 0.065964 \\ -0.1185634218 & 0.9922029043 & -0.03842019991 & -0.041977 \\ 0.07022244505 & 0.04697491713 & 0.9964246913 & 0.063426 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.2}$$

A.2.2 Homogeneous matrix $dhMoffset$

Transformation matrix of the desired offset between the hand the Box:

$$\begin{bmatrix} 0.9904602583 & 0.1154415606 & -0.07524442084 & 0.065964 \\ -0.1185634218 & 0.9922029043 & -0.03842019991 & -0.041977 \\ 0.07022244505 & 0.04697491713 & 0.9964246913 & 0.063426 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.3}$$

A.2.3 Homogeneous matrix $cMdbox$

Transformation matrix of the desired offset between the box and the camera frame for the base PBVS:

$$\begin{bmatrix} -0.8081692083 & 0.5869575788 & -0.04840796853 & 0.041674 \\ 0.2710923161 & 0.2977708672 & -0.9153368051 & -0.004123 \\ -0.5228493922 & -0.7528700494 & -0.3997689355 & 0.40209 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.4}$$

A.3 Pseudo Algorithms

A.3.1 Fast Flood-Fill Algorithm

Algorithm 1: Flood-fill segmentation

Data: $(p, \rho_e xp, class)$: p– starting pixel position; $\rho_e xp$ – expected area to bounding box dimensions ratio; $class$– searched segment type (white or black)

$s_{id} \leftarrow s_i d + 1$ // increment segment ID

$q_{old} \leftarrow q_{end}$ //store previous queue end

$pixel.class$[p] $\leftarrow p$ //push its position to the queue

//#1 perform the flood fill search

while $q_{end} > q_{start}$ **do**

  $q \leftarrow queue[q_{start}++]$ // pull pixel from the queue and check its neighbours;

  foreach *offset* $\in \{+1, -1, +\omega, -\omega\}$ **do**

    $r \leftarrow q+$ offset;

    if *pixel.claass[r] = unknown* **then**

      pixel.class[r] $\leftarrow$ classify(Image[r],τ)

    if *pixel.class[r] = class* **then**

      queue[$q_{end}++$]$\leftarrow r$;

      pixel.class[r] $\leftarrow s_{id}$;

      update $v_{min}, v_{max}, \nu_{min}, \nu_{max}$ fro r_u, r_v

  go back to the beginning of current section;

valid $\leftarrow$ false;

//#2 test for the pattern size and roundness;

if $s > min.size$ **then**

  $v \leftarrow (v_{max} + v_{min})/2$ //segment center x axis;

  $\nu \leftarrow (\nu_{min} + \nu_{max})/2$ //segment center y axis;

  $b_v \leftarrow (v_{max} - v_{min}) + 1$ // estimate segment width;

  $b_\nu \leftarrow (\nu_{max} - \nu_{min}) + 1$ // estimate segment height;

  $\rho \leftarrow \rho_{exp} \pi b_v b_\nu / 4s - 1$ // calculate roundness;

  if $\rho_{tol} < \rho < \rho_{tol}$ **then**

    $\mu \leftarrow \frac{1}{s} \sum_{j=q_{old}}^{q_{end}^{-1}} Image[j]$ // mean brightness;

    valid $\leftarrow$ true // mark segment valid

Result: $(v, \nu, b_v, b_\nu, \mu, valid)$: (ϵ, ν) – segment center; (bv, b_ν) – bounding box ; v – average brightness ; valid – validity

A.4 KLT and Edge Tracker Parameters

Tables A.1 and A.2 show the initialization parameters from the XML and CAD files for the 3D model tracker:

Table A.1: KLT Settings

	KLT Settings
Window Size	5 X 5 pixels
Mask Border	5 pixels
Max num of features	1000
Max dist between points	5 pixels
Harris free parameter	0.01
Block Size	3x3 pixels

Table A.2: Edge Tracker Settings

	Moving Edges Settings
Convolution Mask	5 X 5 pixels
Likelihood test ratio	1000
Contrast tolerance	50% and 50%
Sample Step	4 pixels
Good moving threshold	40%

A.5 Pepper Follow People

This implementation was done following the model of the FollowPeople from *visp_naoqi* [9]. In order to achive the implementation we need to combine three modules of Aldebaran: ALMovementDetection, ALPeoplePerception and ALEngamenZones.

A.5.1 ALMovementDetection

This module is designed to help detect movement. This is achieved by obtaining the needed data from the 3D sensor in the robot. Frames are collected at a regular interval and each new frame is compared with the previous one. The pixels for which the difference is above a threshold are identified as moving pixels. Then all the moving pixels are clustered using both their physical proximity and their value difference [4]. Each time a movement is detected the memory is updated and an event is raised.

The main disadvantage of the module is that it does not update the memory is the robot is moving. Thus, it only detects movement if it is in steady position.

A.5.2 ALPeoplePerception

This module works as an extractor keeping track of the people around the robot. It gathers the visual information from RGB cameras and a 3D sensor if available. The algorithm raises a flag when it detects a new human and updates the memory. The robot keeps two lists: visible humans and not visible humans. When somebody gets out of the robots camera field of view, he/she is not immediately removed from the people list, as this disappearance may be temporary and the result of the robot movements [5]. Which takes us to the main limitation of this module which is the low precision in the pose estimation of the human. Additionally, it does not work properly on high light conditions or if another human is standing too close to the previous recognized one.

A.5.3 ALEngagementZones

Once we have recognized the human we need to approach him/her in a safe way. Which is why we use ALEngagementZones. This module allows to classify detected people and/or movements using their position in space with respect to the robot [2]. This module divides the space in front of the robot into configurable zones. Thus, making it easy to adapt the behavior of the robot depending on the zone a person or a movement is detected in [2].

Bibliography

[1] *Aldebaran aldebaran – pepper robot specifications*. http://doc.aldebaran.com/2-0/family/juliette_technical/. Accessed: 2017-05-05.

[2] *Aldebaran alengagementzones*. http://doc.aldebaran.com/2-4/naoqi/peopleperception/alengagementzones.html. Accessed: 2017-04-13.

[3] *Aldebaran cartesian control*. http://www.bx.psu.edu/~thanh/naoqi/naoqi/motion/control-cartesian.html. Accessed: 2017-02-03.

[4] *Aldebaran movement detection*. http://doc.aldebaran.com/2-4/naoqi/vision/almovementdetection.html#almovementdetection. Accessed: 2017-04-13.

[5] *Aldebaran people pereception*. http://doc.aldebaran.com/2-4/naoqi/peopleperception/alpeopleperception.html. Accessed: 2017-04-13.

[6] *Inria peppercontrol*. https://github.com/lagadic/pepper_control. Accessed: 2017-02-03.

[7] *Inria visp*. https://visp.inria.fr/. Accessed: 2017-02-03.

[8] *INRIA visp edge tracking*. http://visp-doc.inria.fr/manual/visp-2.6.0-tracking-overview. Accessed: 2017-02-03.

[9] *Inria visp naoqi bridge*. http://jokla.me/software/visp_naoqi/. Accessed: 2017-02-01.

[10] *INRIA whycon tracking*. https://github.com/lagadic/pepper_hand_pose. Accessed: 2017-02-03.

[11] *Irse whycon*. https://github.com/lrse/whycon. Accessed: 2017-02-02.

[12] *Laas metapod*. https://github.com/laas/metapod. Accessed: 2017-02-03.

[13] *OpenCV opencv team*. http://opencv.org/. Accessed: 2017-02-02.

[14] *ROS naoqi driver*. http://wiki.ros.org/naoqi_driver. Accessed: 2017-02-03.

[15] *ROS ros.org*. http://wiki.ros.org/. Accessed: 2017-02-02.

[16] *ROS vision visp*. https://github.com/lagadic/vision_visp. Accessed: 2017-02-03.

[17] P. ALLEN, B. YOSHIMI, A. TIMCENKO, AND P. MICHELMAN, *Hand-eye coordination for grasping moving objects*, 2005.

[18] L. BELUSSI AND N. S. T. HIRATA, *Fast qr code detection in arbitrarily acquired images*, in SIBGRAPI, 2011.

[19] K. BENAMEUR AND P. R. BÉLANGER, *Grasping of a moving object with a robotic hand-eye system*, in IROS, 1993.

[20] F. CHAUMETTE, *Handbook of robotics chapter 24: Visual servoing and visual tracking*, 2007.

[21] F. CHAUMETTE, E. MARCHAND, F. SPINDLER, R. TALLONNEAU, AND A. YOL, *Visp 2.6.2: Visual servoing platform*, 2011.

[22] F. CHAUMETTE, P. RIVES, AND B. ESPIAU, *The task function approach applied to vision-based control*, in Fifth International Conference on Advanced Robotics 'Robots in Unstructured Environments, 1991.

[23] F. C. CHAUMETTE, *Springer-verlag copyright notice potential problems of stability and convergence in image-based and position-based visual servoing*, 1998.

[24] G. CLAUDIO, F. SPINDLER, AND F. CHAUMETTE, *Vision-based manipulation with the humanoid robot romeo*, in Humanoids, 2016.

[25] A. I. COMPORT, É. MARCHAND, M. PRESSIGOUT, AND F. CHAUMETTE, *Real-time markerless tracking for augmented reality: the virtual visual servoing framework*, IEEE Trans. Vis. Comput. Graph., 12 (2006), pp. 615–628.

[26] D. DEMENTHON AND L. S. DAVIS, *Model-based object pose in 25 lines of code*, International Journal of Computer Vision, 15 (1992), pp. 123–141.

[27] B. ESPIAU, F. CHAUMETTE, AND P. RIVES, *A new approach to visual servoing in robotics*, in Geometric Reasoning for Perception and Action, 1991.

[28] R. FEATHERSTONE AND D. E. ORIN, *Robot dynamics: Equations and algorithms*, in ICRA, 2000.

[29] R. HORAUD, F. DORNAIKA, AND B. ESPIAU, *Visually guided object grasping*, IEEE Trans. Robotics and Automation, 14 (1998), pp. 525–532.

[30] D. KRAFT, R. DETRY, N. PUGEAULT, E. BASESKI, J. H. PIATER, AND N. KRUGER, *Learning objects and grasp affordances through autonomous exploration*, in ICVS, 2009.

[31] T. KRAJNÍK, M. A. NITSCHE, J. FAIGL, P. VANEK, M. SASKA, L. PREUCIL, T. DUCKETT, AND M. MEJAIL, *A practical multirobot localization system*, Journal of Intelligent and Robotic Systems, 76 (2014), pp. 539–562.

[32] B. D. LUCAS AND T. KANADE, *An iterative image registration technique with an application to stereo vision*, in IJCAI, 1981.

[33] Z. MACURA, A. CANGELOSI, R. ELLIS, D. BUGMANN, M. H. FISCHER, AND A. MYACHYKOV, *A cognitive robotic model of grasping*, 2009.

[34] E. MALIS, F. C. CHAUMETTE, AND S. BOUDET, *2 1/2 d visual servoing*, 1999.

[35] M. MAREY AND F. CHAUMETTE, *A new large projection operator for the redundancy framework*, 2010 IEEE International Conference on Robotics and Automation, (2010), pp. 3727–3732.

[36] D. MARR AND E. HILDRETH, *Theory of edge detection.*, Proceedings of the Royal Society of London. Series B, Biological sciences, 207 1167 (1980), pp. 187–217.

[37] L. MONTESANO, M. LOPES, A. BERNARDINO, AND J. SANTOS-VICTOR, *Learning object affordances: From sensory–motor coordination to imitation*, IEEE Trans. Robotics, 24 (2008), pp. 15–26.

[38] A. MORALES, E. CHINELLATO, A. H. FAGG, AND A. P. D. POBIL, *An active learning approach for assessing robot grasp reliability*, in 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566), 2004.

[39] A. A. MOUGHLBAY, E. CERVERA, AND P. MARTINET, *Error regulation strategies for model based visual servoing tasks: Application to autonomous object grasping with*

nao robot, 2012 12th International Conference on Control Automation Robotics and Vision (ICARCV), (2012), pp. 1311–1316.

[40] M. NITSCHE, T. KRAJNIK, P. VCIZEK, M. MEJAIL, AND T. DUCKETT, *Whycon: An efficent, marker-based localization system*, 2015.

[41] G. TAYLOR AND L. KLEEMAN, *Grasping unknown objects with a humanoid robot*, 2002.

[42] C. TOMASI AND T. KANADE, *Shape and motion from image streams: a factorization method | part 3 detection and tracking of point features*, 1991.

[43] N. VAHRENKAMP, S. WIELAND, P. AZAD, D. I. GONZALEZ-AGUIRRE, T. ASFOUR, AND R. DILLMANN, *Visual servoing for humanoid grasping and manipulation tasks*, in Humanoids, 2008.